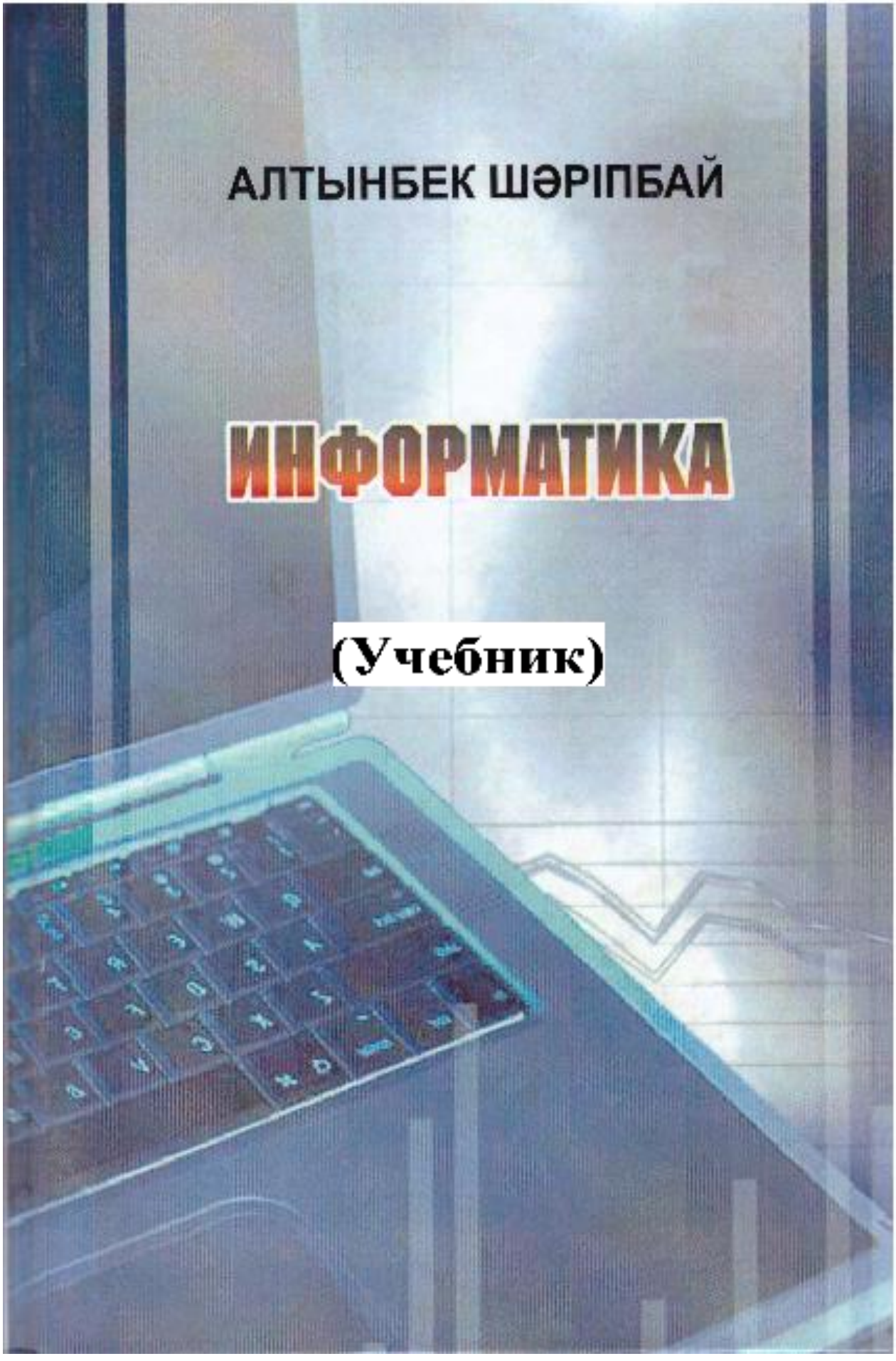

АЛТЫНБЕК ШӘРІПБАЙ
ИНФОРМАТИКА
(Учебник)

ШӘРІПБАЙ А.А.

ИНФОРМАТИКА
(Учебник)

Алматы 2015
Эверо

УДК 004.4(075.8)
ББК 32.973.26-018.1я73
Ш-25

Рецезенты:

Адамов А.А. – заведующий кафедрой Математического и компьютерного моделирования, ЕНУ имени Л.Н.Гумилева, д.т.н., профессор.

Данаев Н.Т. – директор Института математики и механики КазНУ имени Аль-Фараби, д.ф.-м.н., профессор, лауреат государственной премии РК, член-корр. НАН РК, академик НИА РК.

Ускенбаев Р.К. – проректор Международного университета информационных технологий, д.т.н., профессор.

Автор:

Ш-25 Шарипбай А.А – директор НИИ «Искусственный интеллект», профессор кафедры «Информатики и Информационной безопасности» ЕНУ имени Л.Н. Гумилева, доктор технических наук, профессор, лауреат государственной премии РК, академик МАИ, академик АПН РК.

РЕКОМЕНДОВАНО
Ученым советом ЕНУ имени Л.Н.Гумилева

Ш-25 Шарипбай А.А. Информатика – Учебник, -Алматы, 2015, - 276 с.
ISBN 978-601-240-913-0

Учебник предназначен для изучения основных разделов самой быстроразвивающей в настоящее время науки Информатики.

Содержание учебника составлено в соответствии с Государственным общеобязательным стандартом образования по специальности 5В060200-Информатика. Учебный материал организован так, чтобы можно было применить кредитную технологию, он состоит из трех уровней: модуль, блок, урок. Это позволит обучать и проводить контроль знаний на четырех уровнях: на уровне урока – текущий контроль, на уровне блока – промежуточный контроль, на уровне модуля – рубежный контроль, на уровне учебника – итоговый контроль. Оценка знаний осуществляется с помощью тестов, составленных в соответствии с этими уровнями.

Учебником могут пользоваться студенты, магистранты, докторанты, преподаватели, ученые по специальностей Информатика, Вычислительная техника и программное обеспечение, Информационные системы, Автоматизация и управление, Математическое и компьютерное моделирование, Системы информационной беопасности, а также все, кто желает самостоятельно изучить информатику.

1404000000
Ш–25
ISBN 978-601-7454-76-0

УДК 004.4(075.8)
ББК 32.973.26–018.1я73

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

В настоящее время Информатика (Computer Science) является самой нужной и самой важной наукой, потому что нет ни одной области интеллектуальной деятельности человека, в которой не применяются результаты информатики. Например, с помощью компьютерной программы решаются различные задачи, создаются проекты, принимаются решения, управляются процессы, рисуются картины, доказываются теоремы, исполняется музыка, играются игры и совершаются многие другие действия.

С информатикой связывают новую революцию общества, после революции освоения вещества и энергии, влияющую не только на материальные производства, но и на интеллектуальные и духовные жизни, коренным образом преобразуя их с помощью компьютерных программ, позволяющих автоматизировать решения задач сбора, обработки и передачи информации.

Информатика занимается проблемами решения различных задач из интеллектуальной жизни человека и создания информационных технологий для автоматизации отдельных процессов с помощью программ на языке компьютера.

Информатика состоит из *Технического обеспечения – Hardware* (*Твердые средства*), *Программного обеспечения – Software* (*Мягкие средства*) и *Интеллектуального обеспечения – Brainware* (*Умные средства*).

Предлагаемый учебник предназначен для обучения основ этих частей информатики и в нем имеются одноименные с ними три модуля. В каждом модуле в соответствии с его темой даются ключевые слова, показывается онтологическая (структурная и смысловая) модель основного понятия, излагается учебный материал и вместе с ними решаются следующие основные задачи обучения: 1) для ознакомления (презентации) учебного материала дается теоретическая информация; 2) для осмысления (семантизации) учебного материала разбираются примеры; 3) для закрепления (валидации) учебного материала предлагаются задания;

4) для проверки (верификации) ознакомленного, осмысленного, закрепленного учебного материала ставятся вопросы; 5) для получения оценки (рейтинга) по полученному знанию рассматриваются тесты. В конце учебника даются ответы на задания, вопросы и тесты.

В модуле объекты информатики рассмотрены блоки информация, величины, структуры данных, где даны следующие уроки: информация – определение информации, сообщение и содержание, объемные единицы и количество информации, вероятностные единицы и количество информации; величины – виды величин, виды числовых величин, числовые операции и их свойства, системы счиления чисел, виды символьных величин, символьные операции и их свойства, виды логических величин, логические операции и их свойства; структуры данных – классификация структур данных, множества, операции над множествами, строки, таблицы и хеш-таблицы, списки, графы и деревья, стеки, очереди и деки;

В модуле твердое обеспечение информатики рассмотрены блоки компьютеры, представление информации в компьютере, где даны уроки компьютеры - состав и структура компьютера, принцип работы и рабочий цикл компьютера, поколение компьютеров, персональные компьютеры, представление информации в компьютере – представление символов, представление чисел.

В модуле мягкое обеспечение информатики рассмотрены блоки алгоритмы, алгоритмические языки, структуры управления, языки программирования, где даны уроки алгоритмы – понятие обработки, понятие алгоритма; алгоритмические языки - вербальный алгоритмический язык, графический алгоритмический язык; структуры управления - структуры управления и их значения, последовательность, простое ветвление, альтернативное ветвление, многозначное ветвление, повторение с предусловием, повторение с постусловием, параметрическое повторение; языки программирования – понятие программы и языков программирования, классификация языков программирования.

В модуле умное обеспечение информатики есть блок методы сортировки, где даны уроки понятие сортировки, сортировка включением, сортировка выбором, сортировка обменом.

В списке литературы приводятся общие литературные источники, которые были использованы при формировании учебного материала учебника. Среди них имеются широко распространенные монографии, учебники, учебные пособия, государственные стандарты и др. Поэтому в тексте настоящего учебника не указаны прямые ссылки на них.

Автор благодарит рецензентов А.А.Адамова, Н.Т. Данаева, Р.К. Ускенбаеву за ценные замечения и предложения, выражает свою признательность преподавателям кафедры информатики и информационной безопасности ЕНУ имени Л.Н.Гумилева, которые ведут занятия по дисциплине «Информатика», З.К. Кадеркеевой и А.Н. Исаиновой за техническую редакцию учебника и доценту этой же кафедры, к.т.н, моей ученице А.С. Омарбековой за помощь при подборе материала по объектно-ориенттированному языку программирования Java.

Автор:
Алтынбек Амирович ШАРИПБАЙ,
доктор технических наук, профессор,
лауреат Государственной премии Республики Казахстан.

I. ОБЪЕКТЫ ИНФОРМАТИКИ

I.1. Информация

В этом модуле будут рассматриваться понятие «информация» («величина»). Определяются их виды, типы, структуры и способы представления информации. Задаются операции, определенные над величинами и их свойства. Рассматриваются структуры данных и их виды. Задаются операции, определенные над структурами данных и их свойства. Рассматриваются способы представления структур данных.

Ключевые слова: *информация*, *сообщение*, *содержание*, *передатчик*, *приемник*, *канал связи*, *непрерывная* (*аналоговая*) *информация*, *дискретная* (*цифровая*) *информация*, язык общения, *величина*, *имя*, *значение*, *символьная величина*, *буквы*, *цифры*, *специальные символы*, *числовая величина*, *целое число*, *вещественное число*, *комплексное число*, *логическая величина*, *истина*, *ложь*, *постоянная величина*, *переменная величина*, *идентификатор*, *числовая операция*, *символьная операция*, *логическая операция*, *коммутативность*, *ассоциативность*, *дистрибутивность*, *единица измерения информации*, *количество информации*, *объемное количество информации*, *вероятностное количество информации*, *множество*, *строка*, *таблица*, *хеш-таблица*, *список*, *граф*, *дерево*, *очередь*, *стек*, *дек*.

Цель: *определить понятие информации*; *расмотреть ее виды*, *структуры и типы*, *ознакомиться с типами величин*, *узнать свойства операций*, *научиться определять значения выражений*, *определить единицы измерения и количество информации*, *рассмотреть структуры данных*, *операции над структурами дванных и свойства этих операций*, *а также способы представления структур данных*.

Структура: Структура понятия информации показана на рисунке 1

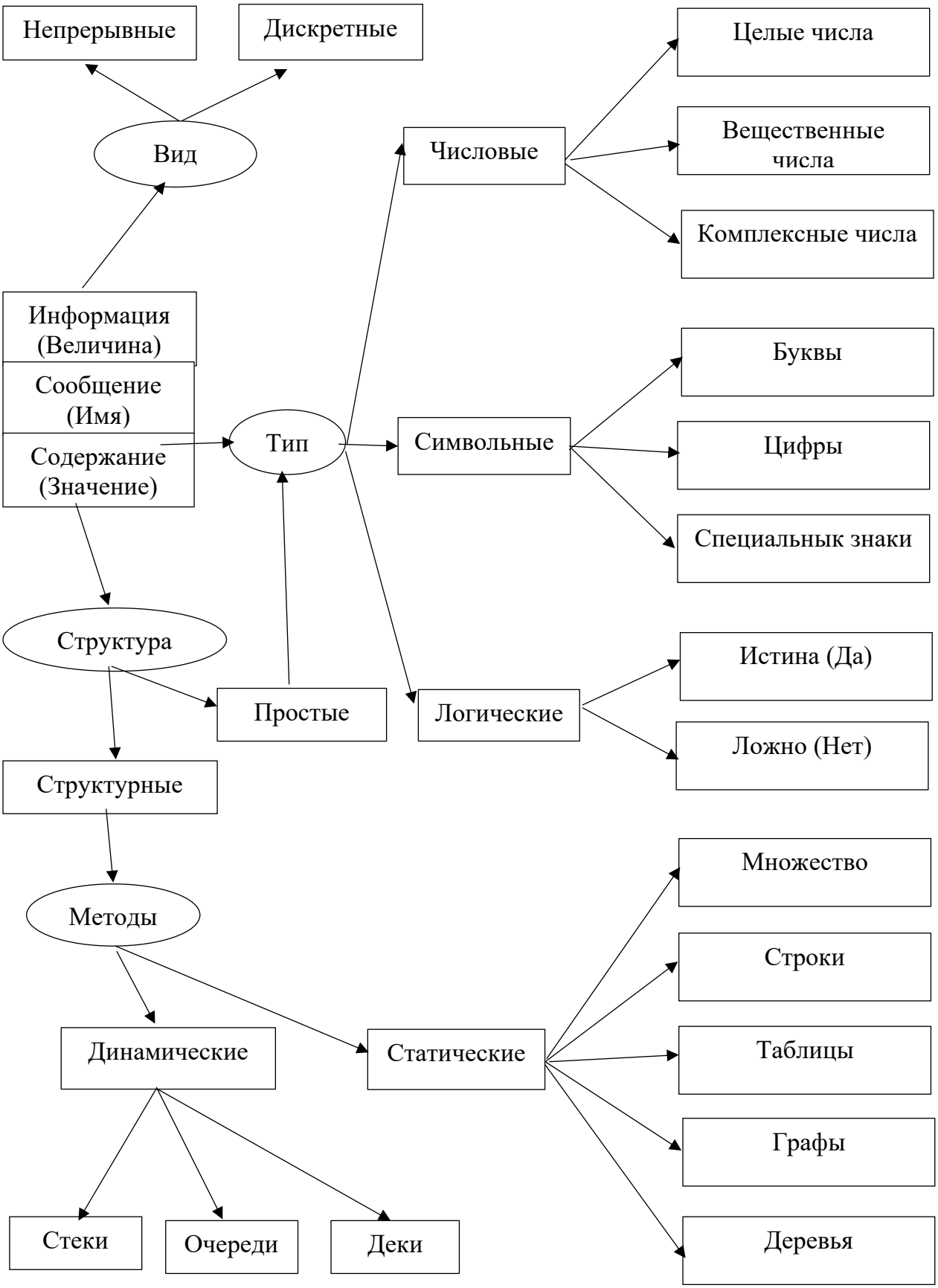


Рисунок I.1. Структура объектов информатики.

I.1.1. Определение информации

Информация является основным понятием информатики и происходит от латинского слово «*Information*», которое означает передать сведения, сообщить о чём–либо (лице, предмете, факте, событии, явлении, процессе), независимо от формы его представления.

Понятие «информация» не имеет точного (математического) определения. Если мы попытаемся ему дать точное определение, то опять придем к другому неопределенному понятию. Однако мы, не зная его точного определения, можем принимать, понимать, сохранять, обрабатывать (преобразовывать) и если надо передавать (дать) другому. С учетом сказанных можно дать следующее интуитивное (не точное) определение понятия «информация».

Информация – представление (обозначение) свойств и отношений материальных и нематериальных объектов и субъектов окружаюшего мира независимо от сознания (восприятия) человека.

Информацию можно представлять (обозначать) и понимать. Следовательно, каждая информация должна иметь свою *форму* и свое *содержание*.

Форму информации называют *сообщением*. Сообщение дается в материально–энергетическом виде (например: света, звука, движения, символа и т.п.). Иначе говоря, сообщение есть выражение (предложение) некоторого языка. К таким языкам относятся:

- естественные языки (казахский язык, русский язык, английский язык и др.);

- математический язык (множество выражений, содержащих условные обозначения свойств и отношений математических объектов);

- музыкальный язык (ноты – множество выражений, содержащих условные обозначения свойств и отношений звуков в звуковом ряде);

- язык глухонемых (множество выражений, содержащих условные движения лица и рук);

- искусственные языки (языки программирования, языки спецификации, языки проектирования и др.);

- и другие.

Вообще, когда говорим о сообщении надо помнить о существовании его *передатчике* и *приемнике*. Ими могут быть живые организмы (люди, животные, птицы, рыбы, насекомые и др.) и не живые (технические) устройства. Например, если приемником является человек, то он принимает сообщение своими органами чувств. А в технических устройствах для этого применяются различные приборы.

Сообщение от передатчика к приемнику передается посредством специальной среды, называемой *каналом связи.* Например, в качестве такой среды для звукового сообщения можно взять воздух, в котором могут распространяться звуковые волны, а каналом связи для письменного сообщения может быть бумага, на которой можно написать текст. Схема передачи информации показана на рисунке I.1.1.А.

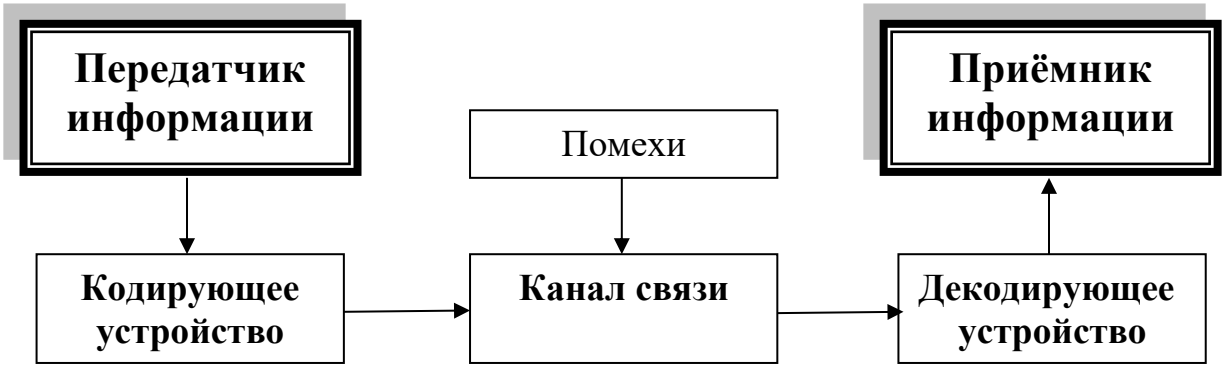


Рисунок I.1.1.А. Схема передачи информации.

При передаче (приеме) сообщения состояние передатчика (приемника) изменяется в зависимости от времени. Поэтому можно рассматривать описание материально–энергетического состояния передатчика (приемника) как функцию $\boldsymbol{x}(\boldsymbol{t})$, зависящую от времени $\boldsymbol{t}$. Эта функция $\boldsymbol{x}(\boldsymbol{t})$ может быть и непрерывной, и дискретной (разрывной). В зависимости от этого информация может быть *непрерывной* (*аналоговой*) и *дискретной* (*цифровой*). Например, к непрерывной информации можно отнести температуру среды, изменяющейся в зависимости от времени, а к дискретной информации – передачу сообщения с помощью знаков некоторого языка общения (примеры, показанные выше).

Любую непрерывную информацию можно превратить в дискретную информацию, это называется *дискретизацией*.

Для дискретизации функции среди её бесконечно многих значений берется ограниченное число значений, которые могут характеризовать остальные значения. Суть этого метода в следующем:

1) ось абциссы графика (область определения) функции разбивается на равные отрезки с помощью конечного числа точек $\boldsymbol{t_1}$, $\boldsymbol{t_2}$,..., $\boldsymbol{t_n}$ и считается, что в каждом отрезке значение функции постоянно, например, считается равным среднему значению в этом отрезке;

2) Спроектировав значение функции в каждом отрезке на ординатную ось графика (область изменения) функции, нужно найти точки $\boldsymbol{x_1}$, $\boldsymbol{x_2}$,..., $\boldsymbol{x_n}$.

Найденные таким образом точки $\boldsymbol{x_1}$, $\boldsymbol{x_2}$,..., $\boldsymbol{x_n}$ будут считаться дискретно приближенным представлением непрерывной функции $\boldsymbol{x}(\boldsymbol{t})$. Его точность можно неограниченно улучшать, уменьшая отрезки в области определения функции, до удовлетворения необходимого требования.

Графики непрерывной функции $x(t)$ и её дискретизации показаны на рисунке I.1.1.B.

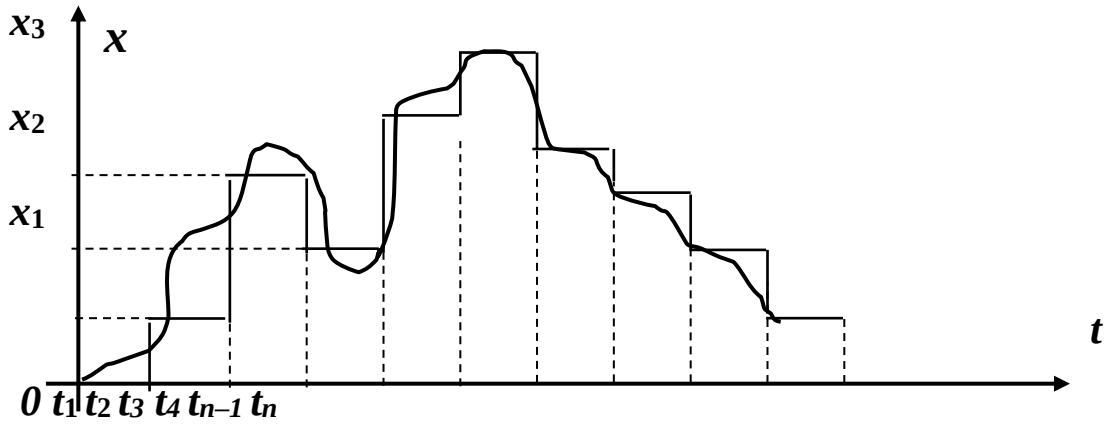


Рисунок I.1.1.B. Дискретизация непрерывной функции

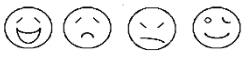
Возможность дискретизации непрерывной информации позволяет представлять сообщение этой информации с помощью символов алфавита некоторого языка общения. Это очень важно для *приема*, *представления*, *хранения*, *обработки и передачи* информации с помощью компьютера.

Примеры I.1.1.

1. A+B = B+A – предложение математического языка, характеризующее свойство операции сложения.
2. ☺ – радость человека на языке мимики.
3. ➲ – знак правила дорожного движения.
4. *ЕСЛИ* **B** *ТО* **A1** *ИНАЧЕ* **A2** – инструкция в искусственном языке, указывающая на то, что если выполняется условие **B**, то нужно выполнить действие **A1**, иначе – действия **A2**.
5. Телефонная связь: передатчик и приемник информации – абоненты. Кодирующее устройство – микрофон, преобразует звуковые колебания в электрические сигналы. Канал связи – кабель. Декодирующее устройство – наушник (динамик). На канал связи могут действовать помехи, что ухудшает качество связи.

Задание I.1.1. Какому языку относятся эти выражения:

1. Қазақстан Республикасы егеменді ел.

2. $a + (b+c)=(a+ b)+ c$.
3.  .

Помощь

1. Казахское предложение, показывающее название города.
2. Показывает свойство операции сложения.
3. Мимика, показывающая настроение человека.

Вопросы I.1.1.

1. Что такое информация?
2. Какие есть виды языков общения?
3. К чему относятся языки программирования?

Тесты I.1.1.

1. Слово «Information» произошло от какого языка?

A) Греческий;
B) Латинский;
C) Английский;
D) Французский;
E) Немецкий.

2. К какому языку относится данное выражение (a+b+c)?

A) Естественный;
B) Логический;
C) Мимика;
D) Математический;
E) Химический.

3. Что сделать для компьютерной обработки сообщения?

A) Дискретизация;
B) Группировка;
C) Структуризация;
D) Автоматизация;
E) Информатизация.

I.1.2. Сообщение и содержание

Из вышеприведенных примеров замечено, что сообщение является носителем содержания информации, т.е. содержание информации определяется вместе с её сообщением. Поэтому между передатчиком и приемником должно быть предварительное соглашение о виде (представлении) и содержании сообщения. Такое соглашение называется правилом интерпретации.

***Замечание I.1.2*:**

1. Разноинтерпретированное одно и тоже сообщение является носителем разного содержания. Например, читатели в зависимости от своих точек зрения могут воспринять одну и ту же журнальную статью по разному.

2. Одно и тоже содержание может быть передано разным сообщением. Например, доклад одного и того человека на одну и ту же тему представленный, на разных языках.

Интерпретация заданного сообщения берется из общего правила, которое может применяться к множеству сообщений, построенных по единому закону. Например, если сообщение задано в виде предложения на известном естественном языке, то правило интерпретации такого предложения можно взять из общего правила интерпретации, применяемого ко всем предложениям этого языка. Итак к общему правилу интерпретации относится и правило определения значений чисел по их форме записи. Также часто используются в качестве сообщения утверждения, имеющие значения «истино» или «ложь» и вопросы, ответы которых могут быть «*да*» или «*нет*». Например, «*Снег имеет белый цвет*», «*Человек никогда не умирает*», «*Интересно ли изучать информатику*?», «*Понятно ли сказанное*?». Интерпретация таких сообщений зависит наряду от вышесказанных, также от жизненного опыта и сознания приемника.

Таким образом, способности интерпретации сообщений передатчика и приемника зависят от их интеллектов.

Примеры I.1.2.

Чтобы заметить связи между сообщением и содержанием рассмотрим несколько примеров. Они приведены в таблице I.1.2.

Таблица I.1.2. Примеры сообщения.

№	1–ое сообщение	2–ое сообщение
1.	Мен жақсы оқимын.	Я учусь хорошо.
2.	Из пламени получается лед.	1>3
3.	Ты меня понял?	X=0?
4.	XXI	21
5.	Завтра пойдет снег.	Лампа погасла.
6.	☼♀♂♥	© ®
7.	I speak.	Мен сөйлеймін.

В примерах 1,2,3,4 показано, что двумя разными сообщениями можно передать одно и то же содержание: в 1-м примере предложения на казахском и русском языках имеют одно и то же содержание, во 2-м примере значение «ложь» имеют два сообщения. В 3-м примере приведены вопросы, требующие ответы «да» или «нет». В 4-м примере одно и то же число представлено двумя способами.

В 5-м примере показана возможность передачи нескольких содержаний одним и тем же сообщением: по первому сообщению понимается, что завтра похолодает или можно покататься на санях.

В 6-м примере показана возможность общаться по предварительным условным обозначениям. Значения таких сообщений доступны только тем, кто знает значение этих знаков.

В 7-м примере одно значение передано предложениями двух естественных (английского и казахского) языков.

Задания I.1.2.

Найдите связь между сообщением и содержанием:

1. 2+2=4.
2. Завтра будет жарко.
3. Я студент. Мен студентпін.

Помощь:

Если сообщение задано в виде предложения на естественном языке, то правило интерпретации такого предложения можно взять из общего правила интерпретации, применяемого ко всем предложениям этого языка.

Вопросы I.1.2.

1. Что такое правило интерпретации?
2. Какая связь имеется между сообщением и содержанием?
3. Как между собой соотносятся сообщение и содержание?

Тесты I.1.2.

1. Между передатчиком и получателем какое предварительное соглашение имеется о виде (представлении) сообщения и о его значении?

A) Интерполяции;
B) Интеграции;
C) Интерпретации;
D) Информатизации;
E) Интервенции.

2. Какая связь между сообщениями, представленные римскими цифрами XIII и арабскими цифрами 13 и их значениями?

A) Одно и тоже значение числа написано двумя способами;
B) Многозначное сообщение;
C) Однозначное сообщение;
D) Одно значение и одно сообщение;
E) Нет никакой связи.

3. Что является носителем содержания информации?

A) Интерпретация;
B) Канал;
C) Передатчик;
D) Приемник;
E) Сообщение.

I.1.3. Объемные единицы и количество информации

Определение понятия «Количество информации» очень трудно. Для этого сначала нужно определить единицу измерения информации. Её можно определить двумя способами: *объемный* и *вероятностный*. Оба этих способа стали известны одновременно в 40-х годах XX века. Их создали ученые США, одни из основателей науки информатики Джон фон Нейман и Клод Шеннон.

Джон фон Нейман самый первый показал возможность построения компьютеров, это привело к определению меры информации объемным способом, а Клод Шеннон определил меры информации вероятностным способом.

Минимальная неделимая единица информации называется ***bit* – *бит***, происходит от двух английских слов ***bi****nary digi****t*** – двоичная цифра**.** Причиной этого стало удобство для разработчиков компьютера работать двоичными числами при сохранении и обработки информации в компьютере: физический элемент, имеющий только два устойчивых состояния, может быть реализован многочисленными устройствами, два устойчивых состояния обозначаются двоичными цифрами 0 и 1. Например, легко создать устройство, показывающее наличие или отсутствие электрического тока, измеряющее низкий или высокий уровень напряжения, обнаруживающее полярность намагничивания и др.

Объемом информации, записанной двоичными цифрами 0 и 1, называется число двоичных цифр, применяемых в этой записи, это число всегда будет целым.

Следующую объемную единицу измерения информации называют *byte – байт*, она состоит из 8 битов, т.е. 1 байт = 2^3 бит. В одном байте можно раздельно записать 2^8 = 256 различных друг от друга символов, т.е. емкость одного байта составляет 256 битов. Это означает, что с помощью одного байта можно представить 256 различных (не повторяющихся) сообщений.

Объемное количество информации является очень большим числом. Поэтому для удобства применения определены крупные объемные единицы измерения. Эти единицы измерения кратны двум, т.е. они должны быть степенями двойки. Список объемных единиц измерения информации приведен в таблице I.1.3:

Таблица I.1.3. Список объемных единиц измерения информации.

№	**Название**	**Символ**	**Степень**
1	*byte* – байт	Б	10^0
2	*kilobyte* – килобайт	Кб	10^3
3	*megabyte* – мегабайт	Мб	10^6
4	*gigabyte* – гигабайт	Гб	10^9
5	*terabyte* – терабайт	Тб	10^{12}
6	*petabyte* – петабайт	Пб	10^{15}
7	*exabyte* – эксабайт	Эб	10^{18}
8	*zettabyte* – зеттабайт	Зб	10^{21}
9	*yottabyte* – йоттабайт	Йб	10^{24}

Примеры I.1.3.

1. Если в книге 400 страниц, в каждой странице 50 строк, а в каждой строке 50 символов, то объем книги в байтах будет 400*50*50 = 1 000 000 Байт = 1 Мб, т.е. в диске объемом 1 Гб можно сохранить 1000 таких книг.
2. Если размер электронной книги составляет 10 мегабайт и емкость электронной библиотеки составляет 100 гигабайт, то в ней можно хранить 100*1024/10 = 102400 электронных книг.
3. Если размер электронной книги составляет 10 мегабайт и емкость электронной библиотеки составляет 100 терабайт, то в ней можно хранить 100*1024*1024/10 = 104857600 электронных книг.

Задания I.1.3.

1. Назовите способы измерения количества информации.
2. Найти объемный размер в слове «РИМ».
3. Укажите емкость одного байта.

Помощь:

1. Способы Джон фон Неймана и Клода Шеннона.

2. Сохраняемая в компьютере информация (слово, число, рисунок, компьютерная программа) записывается в двоичных цифрах.

3. Нужно указать количество битов в одном байте.

Вопросы I.1.3.

1. Как называется минимальная единица информации?

2. Кто определил объем информации?

3. Чему должны быть кратны крупные единицы измерения информации?

Тесты I.1.3.

1. Сколько битов в одном байте?

A) 2;
B) 10;
C) 8;
D) 0;
E) 17

2. Сколько мегабайтов будет один ГБ?

A) 1040;
B) 1024;
C) 10024;
D) 124;
E) 102400.

3. Сколько гигабайтов будет один ТБ?

A) 10024;
B) 1040;
C) 1024;
D) 124;
E) 102400.

I.1.4. Вероятностные единицы и количество информации

До введения и обсуждения понятия «Вероятностное количество информации», рассмотрим один опыт, относящий к теории вероятности. В качестве примера можно взять бросание *N гранной игральной кости* (самая распространенная $N = 6$). Результатом считается выпадание грани, с надписью цифр *1*, *2*,..., *N*.

Введем числовую меру для измерения неопределенности, назвав её ***энтропией*** и обозначим через *H*. Величины *N* и *H* будут между собой в следующем функциональном отношении:

$$H = f(N), \qquad (1.1)$$

где функция *f* является неотрицательной и возрастающей для наших $N = 1, 2, \ldots, 6$.

Рассмотрим более подробно бросание игральной кости:

1) подготовка к бросанию кости: её исход неизвестен, т.е. есть какая-то неопределенность, обозначим её через *H1*;

2) игральная кость брошена: получена информация об исходе опыта, количество этой информации обозначим через *I* ;

3) обозначим неопределенность этого опыта после его осуществления через *H2*;

В качестве количества информации в ходе осуществления опыта можно взять разницу неопределенностей, полученных до опыта и после опыта:

$$I = H1 - H2. \qquad (1.2)$$

Очевидно, что в случае получения конкретного результата, ранее полученная неопределенность исчезает, т.е. $H2 = 0$. Таким образом, количество информации после опыта будет совпадать с начальной энтропией, т.е. $I = H1$. Говоря по-другому, неопределенность в опыте совпадает с информацией об исходе этого опыта. Здесь значение *H2* может не равняться нулю, например, в ходе опыта надпись следующей грани выпадания больше 3.

Следующим важным обстоятельством является определение вида функции *f* в формуле (1.1). Если через *N* обозначить число

граней, а через M – число бросаний игральной кости, то общее число исходов, определяемое векторами длины M и состоящих из N знаков, будет равно N в степени M

$$X=N^M. \qquad (1.3)$$

Например, в случае двух бросаний кости с шестью гранями имеем: $X = 6^2 = 36$. Фактически каждый исход X является некоторой парой (X1, X2), где $X1$ и $X2$ – соответственно исходы первого и второго бросаний, а X – общее число таких пар.

Ситуацию с бросанием M раз кости можно рассматривать как некую сложную систему, состоящую из независимых друг от друга подсистем - "однократных бросаний кости". Энтропия такой системы в M раз больше, чем энтропия одной системы (принцип аддитивности энтропии):

$$f(6^M) = M \cdot f(6).$$

Данную формулу можно распространить и на случай любого N:

$$f(N^M) = M \cdot f(N). \quad (1.4)$$

Теперь прологарифмируем левую и правую части формулы (1.3):

$$\ln X = M \cdot \ln N,$$

далее M находится так:

$$M = \frac{\ln X}{\ln N}.$$

Подставляем полученное для M значение в формулу (1.4):

$$f(X) = \frac{\ln X}{\ln N} \cdot f(N).$$

Обозначив через K положительную константу $\frac{f(N)}{\ln N}$, получим:

$$f(X) = K \cdot \ln X,$$

или, с учетом (1.1), $H=K \cdot \ln N$.

Обычно принимают $K = \frac{1}{\ln 2}$**.** Отсюда получается формула **Хартли**:

$$H = \log_2 N. \qquad (1.5)$$

Важным при введении какой-либо величины является вопрос о том, что принимать за единицу ее измерения. Очевидно, *H=1* при *N* = 2. Иначе говоря, в качестве единицы принимается количество информации, связанное с проведением опыта, состоящего в получении одного из двух равновероятных исходов (примером такого опыта может служить бросание монеты, при котором возможны два исхода: "орел", "решка"). Такая единица количества информации называется "*бит*".

Все *N* исходов рассмотренного выше опыта являются равновероятными и поэтому можно считать, что на "долю" каждого исхода приходится одна *N*-я часть общей неопределенности опыта:

$$\frac{\log_2 N}{N}.$$

При этом вероятность *i*-го исхода P_i равняется, очевидно, *1/N*. Таким образом, э**нтропия** находится по **формуле Шеннона** так:

$$H = \sum_{i=1}^{N} P_i \cdot \log_2(\frac{1}{P_i}). \quad (1.6)$$

Та же формула (1.6) принимается за меру энтропии в случае, когда вероятности различных исходов опыта **неравновероятны** (т.е. *P*. могут быть различны).

Замечание I.1.4:

Соотношение между объемным и вероятностным количеством информации неоднозначное. Не всякий текст, записанный двоичными символами, допускает измерение объема информации в вероятностном смысле, но заведомо допускает его в объемном. Далее, если некоторое сообщение допускает измеримость количества информации в обоих смыслах, то они не обязательно совпадают, при этом вероятностное количество информации не может быть больше объемного.

Примеры I.1.4.

1. Рассмотрим алфавит, состоящий из двух знаков 0 и 1. Если считать, что со знаками 0 и 1 в двоичном алфавите связаны

одинаковые вероятности их появления ($P(0) = P(1) = 0,5$), то по формуле (1.5) количество информации на один знак при двоичном кодировании будет равно

$$H = 1\log_2 2 = 1 \text{ бит.}$$

Таким образом, количество информации (в битах), заключенное в двоичном слове, равно числу двоичных знаков в нем.

2. Определим количество информации, связанное с появлением каждого символа в сообщениях, записанных на русском языке. Будем считать, что русский алфавит состоит из 33 букв и знака "пробел" для разделения слов. По формуле (1.5)

$$H = \log 34 = 5 \text{ бит.}$$

Однако в словах русского языка (равно как и в словах других языков) различные буквы встречаются неодинаково часто. По формуле (1.6): *H= 4,72 бит.*

Задания I.1.4.

1. Пусть в указанном ящике имеются 16 разноцветных шаров. Нужно вычислить объем информации, используя энтропию, чтобы вытащить белый шар.

2.Определите количество информации, связанное с появлением каждого символа в сообщениях, записанных на казахском языке в 42-х буквенном алфавите.

3.Определите количество информации, связанное с появлением каждой двоичной цифры в сообщениях, записанных в одном мегабайте.

Помощь:

1. $H = \sum_{i=1}^{N} P_i \cdot \log_2(\frac{1}{P_i})$.

2. Для определения количество информации используйте формулу (1.5).

3. Надо учесть, что в одном мегабайте находятся 10^6 битов.

Вопросы I.1.4.

1. Какими свойствами обладает функция для вычисления энтропии?
2. В чем заключается принцип аддитивности энтропии?
3. Как называется величина, измеряющая неопределенность?
4. Отношение между объемными и вероятностными единицами измерения информации однозначно?
5. Зачем нужна формула Хартли?

Тесты I.1.4.

1. Что является минимальной неделимой единицей объемного измерения информации?

A) Терабайт (tera***byte***);
B) Байт (Byte);
C) Мегабайт (mega***byte***);
D) Гегабайт (giga***byte***);
E) Бит (bit).

2. Как называется величина измерения неопределенности по Шенону?

A) Эктропия;
B) Энтропия;
C) Величина;
D) Информация;
E) Байт.

3. Указать формулу Хартли?

A) $H = \log_2 N$;
B) $H = \lg N$;
C) $H = \ln N$;
D) $H = \ln (N–1)/2$;
E) $H = \ln (N–1)^2$.

I.2. Величины

I.2.1. Виды величин

В информатике понятие информации часто заменяют понятием величина, при этом сообщение информации расматривается как имя величины, а содержание информации – как значение величины.

Величины – объекты, используемые при обработке информации.

Величины, в зависимости от способов придания значений своим именам, подразделяются на *постоянные величины* и *переменные величины*.

Если значения величин определяются во время построения общих *правил интерпретации* (обьяснения) используемого языка общения, то такие величины будут постоянными. Иначе говоря, при именовании простой величины одновременно определяется ее значение. Вместе со значением станет известным его тип. Например, если цепочку 135, образованную из цифр 1, 3 и 5, рассматривать как имя постоянной величины, то её значение будет целое число «сто тридцать пять». Если из этих же цифр построить другую цепочку 315, то она будет именем другой постоянной величины со значением «триста пятьнадцать», являющимся целым числом. Отсюда следует следующее утверждение:

Имя, значение и тип постоянной величины не изменяются, они определяются одновременно.

Теперь определим *переменные величины*. Если значения величин изменяются во время их обработки, то такие величины называются переменными величинами.

Обычно, тип значения переменной величины не должен изменяться. В противном случае во время обработки этой величины возможно появятся много трудностей с неправильными решениями.

В информатике для именования переменной величины используется понятие «идентификатор».

Идентификатор – цепочка с ограниченной длиной, которая начинается с буквы и состоит из букв и цифр.

Значение величины описывается своим типом, который однозначно определяет:

а) допустимые значения, которые может иметь объект описываемого типа;

б) допустимые операции, которые могут быть применимы к объекту описываемого типа.

Вообще, значения величины подразделяются на *численные*, *символьные* и *логические* типы.

Для обработки заданных величин нужно применить операции к этим величинам. А для того чтобы применить операции нужно знать их определения, обозначения и свойства.

Операции, в зависимости от того, что над какими типами они определены подразделяются на *числовые операции*, *символьные операции* и *логические операции*.

Следует отметить, что любая из этих операций определена и исследована в различных разделах математики. Например, все символьные операции определены в разделе «Математическая лингвистика», которая исследует состав и свойства языков, все логические операции определены в разделе «Математическая логика», а все числовые операции – в других разделах математики (арифметика, алгебра и др.).

Можно определить свойства операций, определенных на каждом типе величин. Эти свойства группируются и образуют *аксиоматику* касательно этих величин.

В предлагаемых аксиоматиках, предназначенных для величин различных типов, имеются много сходства. Они показывают эквивалентность, в том числе имеются закономерности такие как *коммутативтность*, *ассоциативность* и *дистрибутивность*.

Такие закономерности упрощают сложное выражение, сокращая количество операций, и облегчают его вычисление.

Примеры I.2.1.

1. N12B – идентификатор.
2. 7X – не идентификатор, потому что начинается с цифры 7.
3. A+B – не идентификатор, так как в нем есть знак «+».

Задания I.2.1. Будут ли идентификаторами такие величины?

1. A–B;
2. XY;
3. C6R7.

Помощь:

Идентификатор начинает с буквы и состоит из букв и цифр.

Вопросы I.2.1.

1. Какие типы имеются у величин?
2. Зачем нужен идентификатор?
3. Чем различаются постоянные и переменные величины?

Тесты I.2.1.

1. Каким понятием можно заменить информацию?

A) Сообщение;
B) Идентификатор;
C) Представление;
D) Величина;
E) Определение.

2. У какой величины не изменяются имя, значение и тип?

A) Определенная величина;
B) Переменная величина;
C) Не точная величина;
D) Постоянная величина;
E) Неопределенная величина.

3. Как подразделяются величины в зависимости от способов присваивания значения?

A) Постоянная и переменная величины;
B) Постоянная и непостоянная величины;
C) Переменная и непостоянная величины;
D) Непостоянная и точная величины;
E) Действительная и непостоянная величины.

I.2.2. Виды числовых величин

Числовой тип образуется из множества чисел, операции над этими числами и свойствами этих операций. Они подразделяются на три: *целые числа*, *вещественные числа и комплексные числа*.

Замечание I.2.2:

В информатике обработка числовых величин может потребовать различные системы счисления чисел. Основаниями таких систем могут быть 2,3,4,... Разница между ними только в методах обозначения значений чисел, а виды операции над числами и их свойства будут одинаковыми. Поэтому сначала рассмотрим хорошо знакомую нам запись чисел в десятичной системе счисления, операции над ними и свойства этих операций, так как все сказанное, связанное с десятичной системой счисления, подходят и к числам в другой системе счисления.

Целые числа представляются арабскими цифрами, перед их отрицательными значениями записывается знак «–», а перед положительными значениями может быть записан знак «+».

Действительные числа, в зависимости от способа представления, подразделяются на две группы: *действительные с фиксированной точкой* и *действительные с плавающей точкой*.

Представление *действительных с фиксированной точкой* состоит из целой и дробной частей. Целая часть размещается перед (в левой стороне) дробной частью и они разделяются между собой знаком «.», называемым десятичной точкой. Перед ними для указания положительности или отрицательности их значений записывается «+» или «–». Обе части представляются арабскими цифрами.

Представление действительных с плавающей точкой состоит из частей, называемых *мантиссой*, основой *системы счисления* и *порядка*. Значения мантиссы, основания системы счисления и порядка могут быть положительными или отрицательными. Для их обозначения перед значениями записываются «+» или «–». Порядок представляется как целое число, а мантисса – как действительное число с фиксированной точкой.

Если обозначить мантиссу через ***M***, порядок через ***p***, основу системы счиления через ***q***, то действительные числа будут следующими:

$$M * q^{p}.$$

Чтобы понять сказанное в таблице I.2.2 расмотрены примеры действительных чисел с плавающей точкой.

Таблица I.2.2. Примеры действительных с плавающей точкой.

№	Пример	Мантисса	Порядок	Значение
1.	$-12.*10^{3}$	–12	+3	–12000
2.	$0.3*10^{+2}$	0.3	+2	30
3.	$254*10^{-2}$	254	–2	2.54
4.	$1.5*10^{1}$	1.5	+1	15
5.	$+2.17*10^{2}$	2.17	+2	217

Одно и то же число в форме с плавающей точкой может быть представлено многими записями. Например, одно и то же число 3.14 может иметь следующие записи:

$$314.*10^{-2} = 31.4*10^{-1} = 3.14*10^{0} = 0.314*10^{1} = 0.0314.*10^{2} = \text{K}$$

Чтобы иметь единственную запись для представления действительного числа с плавающей точкой нужно нормализовать его. Для этого должно выполняться следующее условие:

$$q^{-1} \le |M| < 1,$$

где $|\text{M}|$ – абсолютная величина.

Например, действительные числа с плавающей точкой $13.64*10^{2}$ и $0.00617*10^{-5}$ в нормализованном виде будут такими:

$$0.1364*10^{4} \text{ и } 0.617*10^{-7}.$$

Комплексные числа представляются в виде алгебраической суммы своих частей: первая (левая) слагаемая – действительная часть, второя (правая) слагаемая – комплексная часть. Действительная часть и комплексная часть комплексного числа записываются в виде действительных чисел. Чтобы их различить

после комплексной части ставится строчная латинская буква «*i*» в качестве её признака.

Примеры I.2.2.

1. 3.14 – положительное действительное число с фиксированной точкой, целая часть 3, а дробная часть 14.
2. +5 – положительное целое число 5.
3. 0.2 – положительное действительное число с фиксированной точкой, целая часть 0, а дробная часть 2.
4. -1.001 – отрицательное действительное число с фиксированной точкой, целая часть 1, а дробная часть 001.
5. 0.0 – положительное действительное число, целая часть 0 и дробная часть 0.
6. 0 + 3*i* – положительное комплексное число, действительная часть 0, а мнимая часть 3.
7. -3.14 + 2*i* – отрицательное комплексное число, действительная часть -3.14,, а мнимая часть 2.
8. $1.7*10^2$- 0.12*i* – положительное комплексное число, действительная часть $1.7*10^2$, а мнимая часть 0.12.

Задания I.2.2.

Определить действительные числа с плавающей точкой и комплексные числа:

1) 40.23;
2) $1.21*10^2+5i$;
3) $3.3*10^{-2}$.

Помощь:

1) Действительное число с фиксированной точкой состоит из целой и дробной частей.

2) Комплексное число имеет действительную часть и мнимую часть, в конце которой записывается строчная латинская буква *i*.

3) Действительное число с плавающей точкой имеет основание системы счисления, мантиссу и порядок.

Вопросы I.2.2.

1. Какие типы имеются у числовых величин?

2. На какие группы подразделяются действительные числа в зависимости от формы их представления?

3. Из каких частей состоят комплексные числа?

Тесты I.2.2.

1. Что нужно использовать при нормализации действительных чисел?

A) $q^{+1} \le |M| < 1$;

B) $q^{-1} = |M| < 1$;

C) $q^{-1} \le |M| < 1$;

D) $q^{-1} > |M| < 1$;

E) $q^{-1} \le |M| < \infty$.

2. Что является для 0 + 3*i* действительной частью, а что мнимой частью?

A) Действительная часть *i*, мнимая часть 3;

B) Действительная часть 0, мнимая часть 3;

C) Действительная часть 3, мнимая часть i;

D) Действительная часть *i*, мнимая часть 0;

E) Действительная часть 0, мнимая часть *i*.

3. Какими цифрами обозначается любое целое число?

A) Латинскими цифрами;

B) Греческими цифрами;

C) Казахскими цифрами;

D) Русскими цифрами;

E) Арабскими цифрами.

I.2.3. Числовые операции и их свойства

Операции, определенные над числовыми типами, нам известны со школы как *арифметические операции*: *сложение*, *вычитание*, *умножение*, *деление*. Они обозначаются знаками «+», «-», «*», «/» соответственно. Применяя эти операции, можно построить числовые выражения. Обычно, для записи числовых выражений мы используем *инфиксную запись*, в которой знаки операций записываются между операндами (аргументами). Например, если для любых чисел a и b ставится в соответствии третье число, являющееся суммой этих чисел, то её мы запишем как $a+b$.

При вычислении значений числовых выражений нужно учитывать приоритеты (порядки выполнения) этих операций: сначала выполняются высокоприоритетные операции умножение и деление, затем – низкоприоритетные операции сложение и вычитание. Иногда для изменения порядка выполнения операций используются скобки. В одном выражении можно использовать несколько скобок и их можно записывать внутри друг друга при этом операция в самой внутренней скобке и в самой левой скобке выполняется первой.

Существуют и бесскобочные способы записи числовых выражений, называемые *префиксной записью* и *постфиксной записью*. В префиксной записи знаки операций записываются перед своими операндами, а в постфиксной записи знаки операции – после своих операндов. Например, числовое выражение в инфиксной записи (*x*+3)*(y-2) записывается в префиксной записи как * + *x* 3 - *y* 2, а в постфиксной записи – как *x* 3 + *y* 2 - *.

Замечание I.2.3:

1. Правила (но не порядок) выполнения этих операций на различных типах будут разными, несмотря на то, что они обозначаются одинаково. Например, правило суммирования целых чисел не годится для сложения действительных чисел.

2. Префиксная запись и постфиксная запись арифметического выражения, в честь их автора польского математика Яна

Лукашевича, названы *прямой польской записью* и *обратной польской записью* соответственно.

Пусть a, b и c – любые числа и a^{-1} – обратное к a число. Тогда операции над числами «+» – *сложение* и «*» – *умножение* имеют свойства, показанные в таблице I.2.3.

Таблица I.2.3. Свойства числовых операций.

№	Аксиома	Описание
1	$a+b=b+a$	Закон коммутативности
2	$a*b=b*a$	
3	$a+(b+c)=(a+b)+c$	Закон ассоциативности
4	$a*(b*c)=(a*b)*c$	
5	$a*(b+c)=a*b+a*c$	Закон дистрибутивности
6	$(a+b)*c=a*c+b*c$	
7	$a+0=0+a=a$	Свойства сложения
8	$a+(-a)=(-a)+a=0$	
9	$a*1=1*a=a$	Свойства умножения
10	$a\neq 0, a*a^{-1}=a^{-1}*a=1$?	

Аксиома 8 гласит, что для каждого числа a существует противоположное к нему число $-a$, а аксиома 9 – для каждого ненулевого числа a найдется обратное ему число $a^{-1}=1/a$.

Примеры I.2.3.

1. $5*(b+c)=5*b+5*c$;
2. $3+(-3)=(-3)+3=0$;
3. $(4+8)/2+(5+3)*2=12/2+8*2=6+16=22$.

Задания I.2.3.

1. Найдите обратное число к числу 6;
2. Вычислить $2*a+3*b$ при $a=5$ и $b=8$;
3. Вычислить $(a+5)*(3*b)$ при $a=10$ и $b=2$.

Помощь:

1. $a * a^{-1} = a^{-1} * a = 1$;

2. Учитывать порядок выпонения операций;

3. Учитывать порядок выпонения операций и скобки.

Вопросы I.2.3.

1. Что такое префиксная запись?
2. Что такое инфиксная запись?
3. Что такое постфиксная запись?

Тесты I.2.3.

1. Найти закон коммутативности?

A) $a + b = b + a$;

B) $a + (b+c) = (a + b) + c$;

C) $a + 0 = 0 + a = a$;

D) $a*1 = 1* a = a$;

E) $a * (b+c) = a*b + a* c$.

2. Найти закон дистрибутивности?

A) $a + (b+c) = (a + b) + c$;

B) $a * (b+c) = a*b + a* c$;

C) $a + 0 = 0 + a = a$;

D) $a*1 = 1* a = a$;

E) $a + b = b + a$.

3. Найти закон ассоциативности?

A) $a * (b+c) = a*b + a* c$; B) $a + (b+c) = (a + b) + c$;

C) $a + (-a) = (-a) + a = 0$;

D) $a*1 = 1* a = a$;

E) $a + b = b + a$.

I.2.4. Системы счисления чисел

Вообще, *системой счисления* называется способ записи чисел с помощью цифр и множества правил. Есть несколько способов записи чисел с помощью цифр.

Любая система счисления должна удовлетворять следующие правила:

– наличие возможности записи значений чисел в заданном диапазоне (интервале);

– каждая последовательность цифр определяет только одно числовое значение;

– простота выполнения операций.

Все системы счисления подразделяются на: *позиционные системы счисления* и *непозиционные системы счисления*.

1. В позиционной системе счисления значения цифр зависят от их места (позиции) в записи числа. Если в записи числа одна и та же цифра встречается несколько раз, то она определяет разное значение. Например, запись числа арабскими цифрами: в трехзначном (трехразрядном) числе 333 самая левая цифра 3 определят три сотни, средняя цифра 3 – три десятки, а самая правая цифра 3 – три единицы.

2. В непозиционной системе счисления значения цифр не зависят от их места в записи числа. Например, запись числа с помощью римских цифр: в записи числа LXXXVIII (восемьдесят восемь) цифра L обозначает пятьдесят, X – десять, V – пять, I – единицу.

Позиционная система счисления характеризуется своей основой счисления. Основа определяет число цифр, применяемых в этой системе. Например, число цифр в десятичной системе равно десять, в восьмеричной системе – восемь, а в двоичной системе – два и т.п.

Возможности всех позиционных систем счисления одинаковы. Однако среди них десятичная система счисления является наиболее распространенной. Тому причина – в обеих руках человека имеются десять пальцев, что позволяет легко и удобно посчитать до десяти.

Досчитав до десяти и исчерпав все возможности «вычислительного средства», разумно для следующей позиции (второго разряда) взять число 10 (десять) как новую единицу. Далее десятичное число десять будет единицей следующей позиции (третьего разряда) и так, продолжая вычисления, появилась десятичная система счисления.

Однако десятичная система счисления не сразу заняла преимущественное место в вычислениях. В разных исторических периодах многие народы использовали не десятичные системы счиления. Например, у древних тюрков основа системы счисления равно семи (1 неделя = 7 дней, 1 обхват = 7 пяди), у древних вавилонцев – шестьдесят (1 минута = 60 секунд, 1 час = 60 минут, 1 градус = 60 минут), у англичан – двенадцать (1 год = 12 месяцев, 1 фут = 12 дюйм, 1 шиллинг = 12 пенс).

В любой позиционной системе счисления с основой ***q*** заданное число ***A*** можно представить следующим образом:

$$A_{(q)} = a_{n-1}q^{n-1} + \Lambda + a_1 q^1 + a_0 q^0 + a_{-1}q^{-1} + \Lambda + a_{-m}q^{-m} \quad (\boldsymbol{1})$$

где a_i – число цифр, применяемых в системе счисления, n – число разрядов в целой части, m – число разрядов в дробной части ($i=n-1,..,1,0,-1,..,-m$).

Замечания I.2.4.1:

1)Число, являющееся основой самой маленькой системы счисления равно двум, она называется двоичной системой счисления. Число в двоичной системе счисления состоит только из цифр 0 и 1.

2)Самой распространенной системы счисления является десятичная система счисления, которая сотоит из десяти арабских цифр: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Среди этих систем счисления нам нужны *десятичная система счиления* и *двоичная система счисления.*

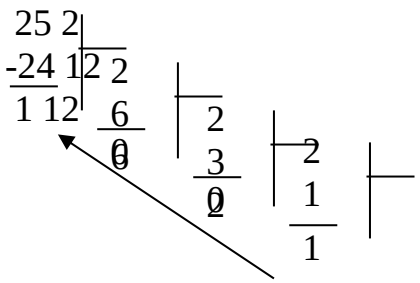
В таблице I.2.4 для каждой десятичной цифры дается двоичный эквивалент.

Таблица I.2.4. Двоичный эквивалент десятичных цифр.

Десятичная цифра	Двоичная цифра
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

Из этой таблицы можно заметить, что при записи одного и того же числового значения в различной системе счисления потребуется различное число знаков (разрядов). Например, двухзначное число 16 в десятичной системе счисления в двоичной системе счисления будет 10000, т.е. потребует пять знаков.

Для перевода заданного целого числа с основой ***p*** на основу ***q*** нужно несколько раз делить это число на ***q*** пока остаток не станет меньше, чем ***q.*** Полученное частное взять в качестве самого старшего разряда числа с основой ***q***, а в качестве значений остальных разрядов нужно взять остатки в направлении, начиная с последнего остатка до первого остатка, и образовать цепочку слева направо. Например, перевод числа 25 в десятичной системе счисления на двоичную систему будет так:



Собирая полученные остатки в обратном порядке получим 11001, что означает $25_{(10)} = 11001_{(2)}$.

Для проверки правильности число 11001 разложим в соответствии с выражением (1) следующим образом:

$$11001_{(2)} = 1*2^4 + 1*2^3 + 0*2^2 + 0*2^1 + 1*2^0 = 16 + 8 + 0 + 0 + 1 = 25_{(10)}$$

Для перевода заданного правильного дробного числа с основой ***p*** на основу ***q*** нужно несколько раз умножать это число на ***q***, пока значение разряда дробной части не будет равно нулю или до получения заданной точности. В качестве значения разрядов правильной дроби с новой основой ***q*** нужно образовать цепочку слева направо в направлении, начиная с первой появившейся целой части до последней появившейся целой части. Например, перевод дробного числа 0.625 в десятичной системе счисления в двочную систему счисления будет выглядеть так:

0,	625
	* 2
1,	250
	* 2
0,	500
	* 2
1,	000
	* 2
0,	000

Теперь, собирая сверху вниз, полученные единицы перехода на старшие разряды от результатов выполнения цепочки операций умножения дробной части заданного числа на основание переводимой системы счисления (в нашем случае 2), получим 1010, что означает $0.625_{(10)} = 0.1010_{(2)}$.

Замечания I.2.4.2:

1) Правила выполнения операций в двоичной системе счисления и в десятичной системе счисления похожи.

2) Свойства операций над двоичными числами одинаковы со свойствами операций над десятичными числами.

Примеры I.2.4:

1. Четырехзначное число ***1952*** десятичной системы выражается так:

$$1952_{(10)} = 1*10^3 + 9*10^2 + 5*10^1 + 2*10^0.$$

2.Число десятичной системы с трехзначной целой частью и трехзначной дробной частью ***596.174(10)*** выражается так:

$$596.174_{(10)} = 5*10^2 + 9*10^1 + 6*10^0 + 1*10^{-1} + 7*10^{-2} + 4*10^{-3}.$$

3. Число двоичной системы с четырехзначной целой частью и трехзначной дробной частью ***1010.101(2)*** выражается так:

$$1010.101_{(2)} = 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 + 1*2^{-1} + 0*2^{-2} + 1*2^{-3}.$$

Задания I.2.4.

1. Переведите заданное число 10000001 двоичной системы в десятичную счисления.

2. Переведите заданное число 129 десятичной системы в восьмеричную систему счисления.

3. Переведите заданное число 7A0 шестнадцатеричной системы в десятичную систему счисления.

Помощь:

1) Число ***A*** в позиционной системе с любой основой ***q*** можно выразить как:

$$A_{(q)} = a_{n-1}q^{n-1} + \Lambda + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + \Lambda + a_{-m}q^{-m}$$

2) Для перевода заданного целого числа с основой ***p*** на основу ***q*** нужно делить несколько раз на ***q,*** пока остаток не станет меньше, чем ***q.*** Частное – старший разряд, остатки – остальные разряды.

3) Можно также использовать формулу

$$A_{(q)} = a_{n-1}q^{n-1} + \Lambda + a_1q^1 + a_0q^0 + a_{-1}q^{-1} + \Lambda + a_{-m}q^{-m}$$

Вопросы I.2.4.

1**.** Что является системой счисления?

2. На какие группы подразделяются все системы счисления?

3. Можно ли одно и то же числовое значение представить в различной системе счисления?

Тесты I.2.4.

1. Какая зависимость имеется сежду цифрами в позицонной системе счисления?

A) Значение цифр зависит от их записи;

B) Значение букв зависит от их места в записи;

C) Значение числа зависит от записи;

D) Место цифр зависит от их значения;

E) Значение цифр зависит от их места в записи.

2. Какая зависимость имеется в непозицонной системе счисления?

A) Значение цифр не зависит от их места в записи;

Б) Значение букв не зависит от их места в записи;

C) Значение числа не зависит от записи;

D) Не зависит от записи заданного числа;

E) Значение цифр зависит от их места в записи.

3. Что является основой позиционной двоичной системы счисления?

A) 2;

B) 0 и 1;

C) 1 и 2;

D) 10;

E) 0.

I.2.5. Виды символьных величин

Символьный тип образуется из множества цепочек знаков алфавита заданного языка общения. В информатике алфавиты состоят из группы: *букв*, *цифр* и *специальных знаков*.

Среди букв могут быть строчные (маленькие) и прописные (заглавные) буквы. Иногда в алфавит одного языка могут добавляться буквы алфавита другого языка. Например, в алфавите современного казахского языка дополнительно имеются буквы русского языка *ё, э, ю, я, ц, ч, щ, ь, ъ*.

В качестве цифр берутся не римские, а арабские цифры *0,1,2,3,4,5,6,7,8,9*.

К специальным знакам относятся *знаки используемых операций*, *знаки препинания*, *знаки группировки* (*скобки*), *знак пробела* и т.п.
Значения символьных величин заключаются в символьных скобках. В качестве символьной скобки применяется знак апострофа « ' ». Например, 'ABC', '2015', 'X+1 > 0', '(3,14)'**.**

Итак, значение символьной величины представляется цепочкой букв, цифр или специальных знаков, заключенных в символьные скобки. Можно определить длину каждой цепочки: она равна количеству символов в этой цепочке. Длину обозначают между двух вертикальных линий « | ». Например, длины выше приведенных символьных величин представляются так:

$$|ABC| = 3,\ |2015| = 4,\ | X+1 > 0| = 5,\ (3,14)| = 6.$$

В множество символьных величин включается абстрактная величина «*пустая цепочка*». В составе пустой цепочки нет ни одного символа. Обычно пустая цепочка обозначается знаком «ε». Длина пустой цепочки равна нулю, т.е. $| \varepsilon | = 0$.

Замечание I.2.5.

Пробел не пустая цепочка, он считается реальным символом. Длина пробела равна единице, т.е. | | = 1.

Примеры I.2.5:

1) ‘ABCDE’ – цепочка, образованная из начальных букв латинского алфавита.

2) ‘X+Y = 100’ – цепочка, образованная из смешанных знаков.

3) ‘A1’– цепочка, образованная из латинской буквы и цифры.

Задания I.2.5.

Определить тип следующих цепочек:

1. ‘ABC’;
2. ‘2013’ ;
3. ‘X+Y > 1’.

Вопросы I.2.5.

1. Что означает длина символьной величины?
2. Чему равна длина символьной величины |ABCD|?
3. Чему равна длина пустой цепочки?

Тесты I.2.5.

1. Из каких групп состоят алфавиты языков общений, используемых в информатике?

A) Состоит из группы букв, цифр и специальных знаков.
B) Не состоит из группы букв, цифр и специальных знаков.
C) Состоит из группы цифр и специальных знаков.
D) Состоит только из группы специальных знаков.
E) Состоит только из группы букв и цифр.

2. Какая цепочка состоит из смешанных знаков?

A) ‘210’ ;
B) ‘ABC’ ;
C) ‘X+1 > 0’;
D) ‘( )’;
E) ‘96’.

3. Чему равна длина цепочки |ABCDFHTRY| ?

A) 0;
B) 1;
C) 10;
D) 9;
E) 11.

I.2.6. Символьные операции и их свойства

Все операции над символьными величинами в основном подразделяются на две группы:

1) Операции *конструирования*, позволяющие из заданных двух символьных цепочек получить одну символьную цепочку;

2) Операции *деления*, позволяющие из заданной символьной цепочки удалить символ или часть этой цепочки в зависимости от определенного условия.

Самой простой операцией конструирования является операция *конкатенация* (*сцепление*), она обозначается знаком «·». Для цепочек X и У она определяется так: после сцепления справа к значению X значения У получится цепочка Z и записывается в виде X•Y=Z. Например, если значение X есть ‘ҚАЗАҚ’, а значение Y есть ‘СТАН’, то значение Z будет ‘ҚАЗАҚСТАН’. В некоторых языках общения знак этой операции не записывается.

К операции конструирования относится операция *дизъюнкция* (*выбор*), обозначаеиая знаком «|». Например, если A=‘АЛМАТЫ’, B=‘АСТАНА’, то C = A|B = ‘АЛМАТЫ’ | ‘АСТАНА’.

Еще одной операцией конструирования является операция *итерация*, она обозначается знаком «*». Эта операция производная (сложная), которая определяется через операции *конкатенация* и *дизъюнкция*. Например, для любой символьной величины X можно определить:

$$X^* = \varepsilon \mid X \mid X^2 \mid X^3 \mid \ldots \mid X^n \mid \ldots,$$

где $X^n = \underbrace{X \cdot X \cdot \ldots \cdot X}_{n}$.

Для правильного построения символьного выражения с помощью операций конструирования нужно знать порядок их выполнения. Установлен следующий порядок: первая – операция итерация «*», вторая – операция конкатенация «·», третья – операция дюзъюнкция «|». Иногда для указания порядка выполнения операций используются скобки. Например, у этих двух выражений *0|10** и (*0*|(*1*(*0**))) значения одниковы.

Самой простой операцией деления является операция удаление из заданной цепочки определенного количества символов, начиная с указанного места. Если имя операции деление обозначим через ***ДЕЛ***, то можно записать удаление символов из цепочки X в количестве M, начиная с позиции K в виде следующей функции:

ДЕЛ (X, K, M) = У,

где зачение цепочки У получается после удаления M символов из значения X, начиная с позиции K, то можно писать

Теперь рассмотрим свойства символьных операций.

Пусть α, β и γ – произвольные непустые символьные величины и ε – пустая цепочка. Тогда свойства операций конструирования, определенных над символьными величинами, «•» – *конкатенация*, «|» – *дизъюнкция* и «*» – *итерация* показываются в таблице I.2.6.

Таблица I.2.6. Свойства символьных операци.

№	Аксиома	Описание
1	$\alpha\cdot\varepsilon = \varepsilon\cdot\alpha = \alpha$	Закон коммутативности для пустых цепочек
2	$\alpha \mid \beta = \beta \mid \alpha$	Закон коммутативности для непустых цепочек
3	$\alpha\cdot\beta \neq \beta\cdot\alpha$	Закон некоммутативности для непустых цепочек
4	$\alpha\cdot(\beta\cdot\gamma) = (\alpha\cdot\beta)\cdot\gamma$	Закон ассоциативности
5	$\alpha \mid(\beta \mid \gamma)=(\alpha \mid\beta) \mid \gamma$	Закон ассоциативности
6	$\alpha\cdot(\beta \mid \gamma)=\alpha\cdot\beta \mid \alpha\cdot\gamma$	Закон дистрибутивности
7	$(\alpha \mid\beta)\cdot\gamma=\alpha\cdot\gamma \mid \beta\cdot\gamma$	Закон дистрибутивности
8	$\alpha\mid\alpha = \alpha$	Закон редукции
9	$(\alpha^*)^*=\alpha^*$	Закон редукции
10	$\alpha^*= \varepsilon\mid\alpha^2\mid\alpha^3\mid\ldots\mid\alpha^n\mid\ldots$	Закон итерации
11	$\alpha^+ =\alpha \mid\alpha^*$	Закон итерации

1. Цепочка α является *префиксом* (*началом*) цепочки β, если найдется цепочка ξ такая, что выполняется равенство β = αξ. Если через α⊂β обозначить отношение «цепочка α является префиксом цепочки β», то его формальное определение можно записать как

$$\alpha \subset \beta \rightleftharpoons \exists \xi(\beta = \alpha\xi).$$

2. Цепочка α является *суфиксом* (*концом*) цепочки β, если найдется цепочка ζ такая, что выполняется равенство β = ζα. Если через β⊃α обозначить отношение «цепочка α является суфиксом цепочки β», то его формальное определение можно записать как

$$\alpha \supset \beta \rightleftharpoons \exists \zeta(\beta = \zeta\alpha).$$

3. Цепочка α является *подцепочкой* (*частью*) цепочки β, если найдутся цепочки ζ и ξ такие, что выполняется равенство β = ζαξ. Если через α⊆β обозначить отношение «цепочка α является *подцепочкой* цепочки β», то его формальное определение можно записать как

$$\alpha \subseteq \beta \rightleftharpoons \exists \zeta \exists \xi(\beta = \zeta\alpha\xi).$$

Примеры I.2.6.

1. Если значение X есть ‘АЛТЫНБЕК’, К=1, а М=5, то
   ДЕЛ (X, К,М) = ***ДЕЛ*** (‘АЛТЫНБЕК’, 1,5) = ‘БЕК’

2. Если значение X есть ‘АЛА’, а значение Y есть ‘ТАУ’, то X•Y = ‘АЛАТАУ’ ≠ Y• X = ‘ТАУАЛА’.

3. Если значение X есть ‘ТАУГҮЛ’, К=4, а М=3, то
   ДЕЛ (X, К,М) = ***ДЕЛ*** (‘ТАУГҮЛ’, 4,3) = ‘ТАУ’.
4. ε⊂*abcd*, *a*⊂*abcd*, *ab*⊂*abcd*, *abc*⊂*abcd*, *abcd*⊂*abcd*.
5. *abcd*⊃ε, *abcd*⊃*d*, *abcd*⊃*cd*, *abcd*⊃*bcd*, *abcd*⊃*abcd*.

Задания I.2.6.

1. Если A=’АЛА’, B= ’ТАУ’, то A·B=?,
2. Если X= ‘ЖЕРОРТА’, К=4 и М=4, то ***ДЕЛ*** (X,К,М)=?
3. Если X= ’БУРАБАЙ’, Y =’БУРА’, то X ⊃Y или Y ⊂X?

Помощь:

1. Нужно знать операции конструирования;
2. Нужно знать операции деления;
3. Нужно знать отношения над цепочками.

Вопросы I.2.6.

1. Сколько групп операци определены над символьными величинами?
2. С помощью каких операций определяется операция итерация над символьными величинами?
3. Что такая пустая цепочка?

Тесты I.2.6.

1. Если A=’ШЫМ’, B= ’КЕНТ’, то A·B=?
   A) ШЫМ;
   B) ШЫМКЕНТ;
   C) КЕНТ;
   D) КЕНТШЫМ;
   E) ШЫН.

2. Если X= ‘ЖЕЗКАЗГАН’, К=4 ; и М= 6, то ДЕЛ (X,К,М)=?
   A) ГАН;
   B) КАЗ;
   C) ЖЕЗКАЗ;
   D) ЖЕЗ;
   E) КАЗГАН.

3. Какому виду аксиомы относится $\alpha \mid \beta = \beta \mid \alpha$?
   A) Некоммутативность для непустых цепочек;
   B) Коммутативность для пустых цепочек;
   C) Закон дистрибутивности для пустых цепочек;
   D) Закон ассоциативности для пустых цепочек;
   E) Коммутативность для непустых цепочек.

I.2.7. Виды логических величин

Логический тип состоит из логических значений. К логическим значениям относятся «*истина*» и «*ложь*».

Логические значения задаются в следующем виде:

1) «*истина*» или «*да*» или «*1*» или «+»;

2) «*ложь*» или «*нет*» или «*0*» или «–».

Эти значения являются результатами определенных условий. К таким условиям относятся утверждения, принимающие значения «*истина*» или «*ложь*», и вопросы, требующие ответы «да» или «нет». Примерами логических значений могут служить значения сообщений из строк 1, 2 и 3 в таблице I.1.2.

В различных литературных источниках вместо логического значения (логической константы) «***ложь***» и «***нет***» используют «***false***», «***non***» или «**0**», а вместо логического значения (логической константы) «***истина***» и «***да***» – «***true***», «***yes***» или «**1**». Поэтому в дальнейшем для удобства в качестве логических значений можно использовать только **0**, **1.** Тогда будем считать, что любые логические переменные принимают значения из множества {**0**, **1**}.

Самое простое условие образуется с помощью операции сравнения (отношения): «<» – «*меньше*»; «=» – «*равно*»; «>» – «*больше*»; «≤» – «*меньше или равно*»; «≥» – «*больше или равно*».

Для построения сложного условия нужно применять логические операции, определенные над логическими значениями. Виды таких операций и их свойства будут рассматриваться ниже в разделе «Логические операции и их свойства».

Примеры I.2.7.

1. Простое условие «2 *меньше* 5» имеет значение «истина».
2. Простое условие «1 равно 3» имеет значение «ложь».
3. Простое условие «А меньше или равно 7» будет иметь значение «истина» или «ложь» в зависимости от значения числовой переменной величины А.

Задания I.2.7.

Определить значение «*истина*» или «*ложь*»:

1. 15=20;

2. 2>12;

3. 7<14.

Помощь:

Логическим значением является результат определенного утверждения, принимающего значения «*истина*» или «*ложь*», или вопросы, требующие ответы «да» или «нет».

Вопросы I.2.7.

1. Значения, определяющие логический тип?

2. Как задаются логические значения?

3. Какое значение будет иметь утверждение «Неверно, что компьютер умнее человека»?

Тесты I.2.7.

1. Определите логическое значение 45=60?

A) Ложь;

B) Истина;

C) Приблизительно;

D) Равно;

E) Мнимо.

2. Определите логическое значение 17<48?

A) Истина;

B) Ложь;

C) Приблизительно;

D) Равно;

E) Мнимо.

3. Определите логическое значение -12≥0?

A) Истина;

B) Ложь;

C) Приблизительно;

D) Равно;

E) Мнимо.

I.2.8. Логические операции и их свойства

Логические операции разнообразны. Мы среди них рассмотрим самые простые. Это операция *нет*, *и, или*. Используя эти операции можно определить любую сложную логическую операцию, т.е. они позволяют построить любое сложное условие.

В зависимости от языка общения *логическое значение*, *операции сравнения* (*отношения*) и *обозначения логических операций* могут быть разными. Например,

1) *Операции сравнения* (*отношения*):
«<» – «*меньше*»;
«=» – «*равно*»;
«>» – «*больше*»;
2) *Логические операции*:
«¬» – «*инверсия*» или «*нет*»;
«&» или «∧» – «*конъюнкция*» или «*и*»;
«|» или «∨» – «*дизъюнкция*» или «*или*».

В любом логическом выражении логические отношения должны выполняться раньше логических операций, а среди логических операций самой первой выполняется операция «*инверсия*», затем операция «*конъюнкция*», а в конце операция «*дизъюнкция*».

Значения (модели) логических операций можно определить с помощью таблиц истинности. Например, если заданы логические переменные A и B, то значения логических операций можно определить следующим образом:

1. *Инверсия* (*нет*):

A	¬A
0	1
1	0

Если значение логической переменной A есть «*ложь*», то его отрицание будет «*истина*» и наоборот.

2. *Конъюнкция* (*и*):

A	B	A∧B
0	0	0
1	0	0
0	1	0
1	1	1

Значение результата будет «*истина*» тогда и только тогда, когда обе переменные A и B принимают значение «*истина*», в противном случае результат имеет значение «*ложь*».

3. *Дизъюнкция* (*или*):

A	B	A∨B
0	0	0
0	1	1
1	0	1
1	1	1

Значение результата будет «*ложь*» тогда и только тогда, когда обе переменные A и B принимают значение «*ложь*», в противном случае результат имеет значение «*истина*».

Перед обсуждением свойств логических операций рассмотрим операции сравнения, которые определяются над любыми типами величин. Операцию сравнения можно определить между количествами или измерениями величин и их различаемыми друг от друга характеристиками.

Операции сравнения имеют много видов. Но, несмотря на это, результаты всех операций сравнения можно представить с помощью логических значений «истина» и «ложь». Кроме того, у всех операций сравнения имеются общие свойства.

Пусть задана операция сравнения ***R***, определенная над величинами *a*, *b* и *c*. Тогда операция ***R*** имеет следующие свойства:

1) *рефлексивность*, если для каждого a выполняется $a\boldsymbol{R}a$;

2) *транзитивность*, если для каждого a, b и c от выполнения $a\boldsymbol{R}b$ и $b\boldsymbol{R}c$ следует выполнение $a\boldsymbol{R}c$;

3) *симметричность*, если для каждого a и b от выполнения $a\boldsymbol{R}b$ следует выполнение $b\boldsymbol{R}a$.

Теперь вместо $\boldsymbol{R}$ будем использовать хорошо нам известные обозначения «<» – меньше, «=» – равно и «>» – больше и запишем следующие свойства:

1. Для любых величин a и b операция $a < b$, $a = b$ или $a > b$ будет только в одном сравнении:

2. Если $a>b$ и $b>c$, то $a>c$.

3. Если $a= b$ и $b= c$, то $a=c$.

4. Если $a< b$ и $b<c$, то $a<c$.

Пусть p, q, r – произвольные логические величины и $\boldsymbol{0}$ – ложь, $\boldsymbol{1}$ – истина. Свойства логических операций приведены в таблице I.2.8.

Таблица I.2.8. Свойства логических операций.

№	Аксиома	Описание
1	$p\wedge q \equiv q\wedge p$	Закон коммутативности
2	$p\vee q \equiv q\vee p$	Закон коммутативности
3	$p\wedge(q\wedge r) \equiv (p\wedge q)\wedge r$	Закон ассоциативности
4	$p\vee(q\vee r) \equiv (p\vee q)\vee r$	Закон ассоциативности
5	$p\wedge(q\vee r) \equiv (p\wedge q)\vee(p\wedge r)$	Закон дистрибутивности
6	$p\vee(q\wedge r) \equiv (p\vee q)\wedge(p\vee r)$	Закон дистрибутивности
7	$\neg(p\vee q) \equiv \neg p\wedge\neg q$	Закон Де Моргана
8	$\neg(p\wedge q) \equiv \neg p\vee\neg q$	Закон Де Моргана
9	$\neg(\neg p) \equiv p$	Закон двойного отрицания
10	$p \equiv p$	Закон идентичности

11	$p \vee \neg p \equiv 1$	Закон исключения третьего
12	$p \wedge \neg p \equiv 0$	Закон противоречия
13	$p \wedge p \equiv p$	Свойство конъюнкции
14	$p \wedge 1 \equiv p$	
15	$p \wedge 0 \equiv 0$	
16	$p \wedge (p \vee q) \equiv p$	
17	$p \vee p \equiv p$	Свойство дизъюнкции
18	$p \vee 1 \equiv 1$	
19	$p \vee 0 \equiv p$	
20	$p \vee (p \wedge q) \equiv p$	

Примеры I.2.8.

1. В выражении A$\vee\neg$B $\wedge$ C выполняется сначала $\neg$B, затем $\neg$B $\wedge$ C, а в конце A $\vee\neg$B $\wedge$ C.

2. В выражении A = 0 $\vee$ B>1 выполняется сначала операция сравнения A = 0 и B>1, затем логическая операция $\vee$.

3. В выражении $\neg$(A=0) $\wedge$ B=1 выполняется сначала операция сравнения A=0 и B=1, затем логические операции $\neg$, $\wedge$.

Задания I.2.8.

В этом выражении выполнить операции и определить его логическое значение.

1) (1 < 2) $\wedge$ 1=3 $\vee$ 2 <5;

2) 1>3 $\vee$ 1<2;

3) 1<3 $\vee$ 5=7;

Помощь:

Среди логических операций самой первой выполняется операция «инверсия», затем операция «конъюнкция», а в самом конце операция «дизъюнкция».

Вопросы I.2.8.

1. Какую логическую операцию обозначают знаки «&», «∧»?
2. Какие значения имеют пустые места в этой таблице?

A	B	A∨B
0	0	
0	1	
1	0	
1	1	

Тесты I.2.8.

1. Какое логическое значение (0 или 1) будет иметь логическое выражение 2>5 ∨ 2<6?

A) 2;
B) 1;
C) 5;
D) 6;
E) 0.

2.Какой порядок выполнения операций в выражении D∨¬F ∧ G?

A) сначала ¬F, затем ¬F ∧ G, а в конце D ∨¬F ∧ G.
B) сначала ¬F ∧ G, а в конце D ∨¬F ∧ G.
C) сначала ¬F, а в конце D ∨¬F ∧ G.
D) сначала ¬F, затем ¬F ∧ G.
E) сначала ¬G, затем ¬G ∧ D, а в конце D ∨¬F ∧ G.

3. Какой из них закон Де Моргана?

A) $\neg(\neg p) \equiv p$;
B) $p \equiv p$;
C) $\neg(p \vee q) \equiv \neg p \wedge \neg q$;
D) $p \wedge \neg p \equiv 0$;
E) $p \vee \neg p \equiv 1$.

I.3. Структуры данных

I.3.1. Классификация структуры данных

Выше в I.2 мы рассмотрели простые (базовые) величины, которые характеризуются неделимой (атомарной) хранимой единицей информации, состоящей из набора однотипных (целочисленных, вещественных, логических, символьных) данных. Однако многие задачи в жизни требуют *составные величины*, которые образованы из величин-элементов. Причем в качестве элементов могут служить не только простые величины, но и составные величины.

В некоторых литературных источниках вместо словосочетания «составные величины» используют *структурные данные* или *структуры данных*. Считая, что они имеют одинаковые значения, в дальнейшем мы будем пользоваться последним словосочетанием, т.е. структурой данных.

Под структурой данных будем понимать множество элементов данных и множество связей между ними. Существует много разновидностей структуры данных. Их классификация может быть выполнена по различным признакам:

1) по наличию связей между элементами подразделяют на *несвязные структуры данных* и *связные структуры данных*. Несвязные структуры данных характеризуются отсутствием связей между элементами структуры. К ним относятся векторы, массивы, строки (записи), стеки и очереди. Связные структуры данных характеризуются наличием связи. К ним относятся односвязные списки, двухсвязные списки и многосвязные списки.

2) по изменчивости размера (количества элементов) подразделяют на *динамические структуры данных*, *статические структуры данных*. В динамической структуре данных количество элементов может уменьшаться или увеличиваться и в ней в одном из этапов обработки может не оказаться ни одного элемента. К этим структурам данных относятся линейные связные списки, стеки,

очереди, деки и разветвленные связные списки, графы, деревья. В статической структуре количество элементов (мощность структуры) не изменяется, она определяется один раз во время обработки. К статической структуре данных относятся строки, массивы, таблицы, множества.

3) по организации взаимосвязей между элементами подразделяют на *линейные структуры данных* и *нелинейные структуры данных*. В свою очередь, линейные структуры данных в зависимости от характера взаимного расположения элементов в памяти подразделяются на *структуры с последовательным расположением элементов в памяти* (векторы, строки, массивы, стеки, очереди, хеш-таблицы) и *структуры с произвольным связным распределением элементов в памяти* (односвязные линейные списки и двусвязные линейные списки).

4) в зависимости от видов связи (отношений) между элементами, нелинейные структуры данных подразделяются на *иерархические структуры данных* (многосвязные списки, двоичные деревья, n-арные деревья, иерархический список), *сетевые структуры данных* (многосвязные списки, граф, ориентированный граф), *табличные структуры данных* (двумерный массив, многомерная таблица реляционной базы данных). В иерархической структуре данных элемент характеризуется ссылкой на вышестоящий в иерархии элемент или ссылками на нижестоящие элементы и (необязательно) порядковым номером в линейной последовательности своего уровня (иерархические списки). В сетевой структуре данных элемент характеризуется набором связей с соседними элементами и ни один элемент явно не выделяется.

5) по способу представления подразделяют на *физические структуры данных и логические структуры данных*. Физическая структура данных – это способ физического представления данных в памяти компьютера. *Логическая* (*абстрактная*) *структура данных* – это рассмотрение структуры данных без учета его представления в памяти компьютера. В общем случае между

физической и соответствующей ей логической структурами существует расхождения, степень которого зависит от самой структуры и особенностей той среды, в котором она должна быть отображена. Вследствие этого расхождения существуют процедуры, осуществляющие отображение логической структуры в физическую, и, наоборот, физической структуры в логическую.

6) по виду памяти, используемой для сохранности данных, подразделяют на *структуры данных для оперативной* и *для внешней памяти. Структуры данных для оперативной памяти*– это данные, размещенные в статической и динамической памяти компьютера. Все вышеприведенные структуры данных – это структуры для оперативной памяти. *Структуры данных для внешней памяти* называют файловыми структурами или файлами. Примерами файловых структур есть последовательные файлы, многораздельные файлы, двоичные деревья.

Приведенная классификация далеко не полная. Есть структуры данных, не вошедшие в эту классификацию.

Существуют два способа физического представления элементов структуры данных в памяти компьютера:

- *смежное представление;*
- *связное представление*.

В смежном представлении для вычисления адреса любого элемента во всех случаях достаточно знать значение номера (индекса) элемента и информации, содержащейся в дескрипторе структуры. При этом адрес любого элемента может быть прямо вычислен из адреса начального или предыдущего элемента. Это позволяет обеспечить очень быстрый доступ к элементам структуры данных. Однако при изменении логической последовательности элементов структуры требуется перемещение данных в памяти. Иначе говоря, смежное представление является очень эффективным, но не гибким способом.

Смежное представление применяется только для статической структуры данных, в которой все элементы однотипны и образуют упорядоченную последовательность. Элементы таких структур можно пронумеровать целыми числами. Иногда номера элементов называют индексами. К таким структурам данных относятся *строки*, *векторы*, *массивы*.

В связном представлении для установления связи между элементами структуры данных используются указатели. При этом каждый элемент структуры снабжается двумя типами полей:

- *поле данных*, в котором содержатся те данные, ради которых и создается структура (в общем случае поле данных само является интегрированной структурой: записью, вектором, массивом и т.п.);
- *поле связок*, в котором содержатся один или несколько указателей, связывающий данный элемент с другими элементами структуры.

Связное представление обладает следующими достоинствами:

– имеет возможность обеспечения значительной изменчивости структур данных;

– размер структуры данных ограничивается только доступным объемом памяти компьютера;

– при изменении логической последовательности элементов структуры требуется не перемещение данных в памяти, а только коррекция указателей.

Связное представление имеет следующие недостатки:

– работа с указателями требует, как правило, более высокой квалификации от программиста, чем работа с индексами элементов;

– на поля связок расходуется дополнительная память компьютера, её объем зависит от количества указателей и может стать больше, чем память для поля данных;

– адрес элемента не может быть вычислен из исходных данных, поэтому после входа в структуру с помощью внешних указателей, содержащиеся в её дескрипторе, поиск требуемого элемента

структуры выполняется следованием по цепочке указателей от элемента к элементу, что требует значительного времени.

Следовательно, связное представление является менее эффективным, но гибким способом. Поэтому связное представление практически никогда не применяется в задачах, где логическая структура данных имеет вид строки, вектора или массива с доступом по номеру элемента, но часто применяется в задачах, где логическая структура требует другого способа доступа к элементам (таблицы, списки, графы и т.д.).

Примеры I.3.1:

1. Строки относятся к несвязанным и линейным структурам.
2. К связным структурам относятся односвязные списки, двухсвязные списки и многосвязные списки.
3. К статическим структурам относятся строки, массивы, таблицы, множества.
4. К структуре данных с последовательным расположением элементов в памяти относятся векторы, строки, массивы, стеки, очереди, хеш-таблицы.
5. К иерархическим структурам данных относятся многосвязные списки, двоичные деревья, n-арные деревья.
6. Стеки относятся к линейным и динамическим структурам.
7. Графы относятся к динамическим и нелинейным структурам.

Задания I.3.1:

1. Перечислите статические структуры данных.
2. Перечислите динамические структуры данных.
3.
4. Перечислите несвязные структуры данных.

Помощь

1. Статические структуры данных не изменяют свои размеры.
2. Динамические структуры данных изменяют свои размеры.
3. В несвязной структуре данных отсутствуют связи.

Вопросы I.3.1:

1. На какие виды подразделяются структуры данных по организации?

2. На какие виды подразделяются нелинейные структуры данных по взаимосвязи между элементами?

3. На какие виды подразделяются структуры данных по способу представления?

Тесты I.3.1:

1. Какими признаками обладает строка?

A) последовательность;

B) иерархичность;

C) динамичность;

D) связность;

E) нелинейность.

2. Что такой массив?

A) множество однотипных упорядоченных элементов;

B) множество однотипных не упорядоченных элементов;

C) множество разнотипных упорядоченных элементов;

D) множество односвязных элементов;

E) множество несвязных элементов.

3. Что такое структура данных?

A) множество элементов данных для обмена между внутренней и внешней памяти;

B) множество элементов данных и множество операций над ними;

C) множество элементов данных и множество способов присваивания им значений;

D) множество элементов данных и множество способов изменения значений;

E) множество элементов данных и множество связей между ними.

I.3.2. Множества

Можно сказать, что встречающиеся в информатике структуры данных строятся на основе множества, так как все они строятся по определенному порядку из множества простых величин. Если так, то мы должны знать определение, обозначение понятия «*множество*», определенных над ними операций и их свойства.

Множество – это совокупность различимых некоторым признаком (свойством) объектов, мыслимое как единое целое.

Объекты, из которых состоит множество, называют *элементами множества.*

Множество обозначается прописными (заглавными) буквами, а его элементы – строчными (маленькими) буквами или арабскими цифрами. Например, множество натуральных чисел обозначается через *N*, а его элементы – арабскими цифрами 1, 2, 3,….

Определение множества можно записать в виде:

<*имя множества*> = {<*определение множества*>},

где в фигурной скобке { и } указываются элементы множества.

Определение множества дается в двух вариантах:

1) определение через указания всех элементов множества;

2) определение через описание всех элементов множества.

Среди всех множество особо выделяется на два множества:

1) *Ø – пустое множество* не содержит ни одного элемента.

2) ***U*** *– универсальное множество* (*универсум*) содержит все элементы в рассматриваемом типе (предметной области).

Например, универсумом может быть:

1) в теории чисел – множество всех целых чисел;

2) в теории языков – множество всех слов в алфавите;

3) в геометрии – множество всех точек в заданном геометрическом пространстве.

Можно определить операции отношения между элементом и множеством. Основной среди них является отношение, показывающее вхождение известного элемента к известному множеству, его обозначают знаком «$\in$», а обратное к нему отношение обозначается через знак «$\notin$». Запись $a \in A$ ($a \notin A$)

показывает, что элемент a принадлежит (не принадлежит) множеству A. Например, эта запись $a \in A$ показывает, что определенный элемент a входит в определенное множество A или элементом множества A является a, а запись $x \notin A$ напротив показывает, что x не содержится в множестве A.

Так же можно определить подобное отношение между множествами: множество A *является подмножеством* множества B, если любой элемент, принадлежащий A, также принадлежит B:

$$A \subseteq B \Leftrightarrow \forall x(x \in A \Rightarrow x \in B) .$$

Мощностью множества A называют количество его элементов, она обозначается через $|\mathbf{A}|$, при этом $|\varnothing| = 0$.

Замечание I.3.2:

1. Пустое множество является подмножеством любого непустого множества A, т.е. $\varnothing \subset A$, $|\varnothing| < |\mathbf{A}|$.
2. Мощность множества натуральных чисел бесконечна, $|N| = \infty$.

Чтобы установить некоторые факты, мы рассмотрим понятие «*счетное множество*». Множество считается счетным, если между его элементами и элементами множества натуральных чисел можно установить взаимнооднозначное соответствие, т.е. элементы счетного множества можно пронумеровать натуральными числами.

3. **Примеры I.3.2.**

1. $D = \{0, 1\}$, здесь элементами множества D являются только величины 0 и 1.
2. Если $A = \{a, b, c, d, e\}$, то $|A| = 5$.
3. $N = \{n \mid n \in Z \ \& \ n > 0\}$ – множество натуральных чисел, здесь Z множество целых чисел.

Задания I.3.2:

Пусть A = $L \cup D \cup S$ – множество всех используемых в компьютере знаков, где L – множество всех латинских букв, D – множество арабских цифр, S – множество всех специальных знаков. Тогда:

1. Определить все элементы множества L;
2. Определить все элементы множества D;
3. Определить все элементы множества S.

Помощь

Применить для определения множества в заданиях 1 и 2 способ перечисления всех элементов, а в задании 3 – способ описания всех элементов и учесть, что специальный знак это не буква и не цифра.

Вопросы I.3.2:

1. Как называется число элементов множества?
2. Как можно определить множество?
3. Чему равна мощность множества пустых цепочек?

Тесты I.3.2.

1.Чему равна $|L|$, если L состоит из латинских строчных букв, т.е. $L = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}$,

A) 1;
B) 26;
C) 28;
D) 0;
E) 30.

2. Если $D = \{d \mid d$ – целое число и выполняется $0 \le d \le 9\}$, то чему равна $|D|$?

A) d;
B) 9;
C) 10;
D) 0;
E) 1.

3. Каким знаком обозначается отношение принадлежности?

A) $\cap$;
B) $\notin$;
C) $\in$;
D) $\varnothing$;
E) &.

I.3.3. Операции над множествами

Над множествами определены несколько операции. Например, если заданы X и Y, то для них можно определить следующие операции:

1. *Операция объединение*: объединением множества X и множества Y называют третье множество Z, элементами которого являются элементы первого множества или второго множества, оно обозначается как:

$$X \cup Y = Z = \{z \mid \ z \in X \vee z \in Y\}.$$

2. *Операция пересечение*: пересечением множества X и множества Y называют третье множество Z, элементами которого являются общие элементы первого и второго множества, оно обозначается как:

$$X \cap Y = Z = \{z \mid \ z \in X \wedge z \in Y\}.$$

3. *Операция разность* (*дополнение*): разностью множества X и множества Y называют третье множество Z, элементами которого являются элементы первого множества, она обозначается как:

$$X \setminus Y = Z = \{z \mid \ z \in X \wedge z \notin Y\}.$$

4. *Операция прямого* (*декартового*) *произведения*: прямым произведением множества X и множества Y называют третье множество Z, элементами которого являются пара, которая состоит из элементов первого и второго множества, оно обозначается как:

$$X \times Y = Z = \{z \mid \ z = (x, y) \wedge x \in X \wedge y \in Y\}.$$

Свойства этих операций над множествами и их описание показаны в таблице I.3.2.

Таблица I.3.2. Свойства операций над множествами

№	Аксиома	Описание
1	$X \cup Y = Y \cup X$	Свойства объединения
2	$X \cup X = X$	

3	$X \cup \varnothing = \varnothing \cup X = X$	
4	$X \cup (Y \cup Z) = (X \cup Y) \cup Z = X \cup Y \cup Z$	
5	$X \cap Y = Y \cap X$	Свойства пересечения
6	$X \cap X = X$	
7	$X \cap \varnothing = \varnothing \cap X = \varnothing$	
8	$X \cup (Y \cup Z) = (X \cup Y) \cup Z = X \cup Y \cup Z$	Закон ассоциативности
9	$X \cap (Y \cap Z) = (X \cap Y) \cap Z = X \cap Y \cap Z$	
10	$X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$	Закон дистрибутивности
11	$X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$	
12	$X \setminus Y \neq Y \setminus X$	Закон разности
13	$X \setminus X = \varnothing$	
14	$X \setminus \varnothing = X$	
15	$X \setminus (Y \setminus Z) = (X \setminus Y) \setminus Z = X \setminus Y \setminus Z$	
16	$X \times Y \neq Y \times X$	Закон прямого произведения
17	$X \times X = X \times X$	
18	$X \times \varnothing = \varnothing \times X = \varnothing$	
19	$X \times (Y \times Z) = (X \times Y) \times Z = X \times Y \times Z$	

Любое *отношение*, определенное между множеством A и множеством B, является подмножеством прямого произведения, определенного между этими двумя множествами. Например, если обозначить такое отношение через знак «~», а подмножество представить знаком «$\subseteq$», то можно записать так:

$$A \sim B \subseteq A \times B.$$

Пусть A и B произвольные множества. Тогда между этими A и B можно определить следующие отношения:

1. A **включено** в B, если каждый элемент множества A принадлежит множеству B:

$$A \subseteq B \Leftrightarrow \forall x (x \in A \Rightarrow x \in B).$$

2. A ***включает*** B, если B включено в A:

$$A \supseteq B \Leftrightarrow B \subseteq A.$$

3. A ***равно*** B, если A и B включены друг в друга:

$$A = B \Leftrightarrow (A \subseteq B) \,\&\, (B \subseteq A).$$

.

4. A ***строго включено*** в B, если A включено в B, но не равно:

$$A \subset B \Leftrightarrow (A \subseteq B) \,\&\, (A \neq B).$$

5. A ***строго включает*** B, если B строго включено в A:

$$A \supset B \Leftrightarrow B \subset A.$$

6. A и B *не пересекаются*, если у них нет общих элементов:

$$A \text{ и } B \textbf{ не пересекаются } \Leftrightarrow \forall a \in A : a \notin B.$$

7. A и B ***находятся в общем положении***, если существует элемент, принадлежащий исключительно множеству A, элемент, принадлежащий исключительно множеству B, а также элемент, принадлежащий обоим множествам:

$$A \text{ и } B \text{ находятся в общем положении } \Leftrightarrow$$
$$\exists a, b, c: (a \in A) \wedge (a \notin B) \wedge (b \in B) \wedge (b \notin A) \wedge (c \in A) \wedge (c \in B).$$

Если поставлено требование, что каждый элемент $y \in Y$ соответствует только одному элементу $x \in X$, то отношение между множествами *X* и *Y* называется *функцией*.

Если обозначить функцию через *f*, т.е. $f \subseteq X \times Y$, то запишем $f: X \to Y$ или выражение элемент $y \in Y$ поставлен в соответствии элементу $x \in X$ запишем как $y = f(x)$.

Элементам множества можно давать *индексы*. Установление элементам индексов называют *индексированием*, а такое множество – *индексным множеством*. Оно задается с помощью функции. Например, обозначить индексирование через *I*, то для индексирования элементов множества *X* можно взять такую функцию $I: X \to \{1,2,3,...,|X|\}$. Тогда можно записать множество *X* так $X = \{x_1, x_2, x_3, ..., x_{|X|}\}$.

Индексирование позволяет сделать ссылку на каждый элемент множества, посредством указания значение его индекса.

Элементы множества могут быть упорядоченными. Упорядоченное множество – множество, на котором задано отношение порядка. Таким отношением может быть известное нам отношение меньше или равно, обозначаемое через знак «≤».

Говорят, что множество *X* *полностью упорядочено*, если выполняются следующие условия:

1) $x_i \le x_j$ или $(x_i, x_j) \in \le$;

2) для любых x_i и x_j справедливо $x_i \le x_j$ или $x_j \le x_i$, $(i \neq j)$;

3) $x_i \le x_j$ и $x_j \le x_k$, то $x_i \le x_k$, $(i \neq j \neq k)$.

Множество *X* *частично упорядочено*, если среди этих условий не выполняется только условие 2).

Примеры I.3.3:

1. Если множество *A* из натуральных чисел, меньших 7, то:

$A = \{x: x$ – натуральное число, $x<7\}$ или $A = \{1, 2, 3, 4, 5, 6\}$;

2. Множество цифр, из которых образовано число 16061952 запишется так {0, 1, 2, 5, 6, 9}.

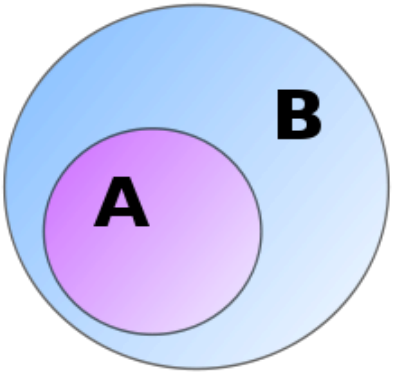
3. Множество букв, из которых образовано слово «информатика» запишется так {*а, и, к, м, н, о, р, т, ф*}.

Задания I.3.3:

1. Найти множество решений уравнения 3x–7=3(x+5).

2. Пусть A – множество всех учеников школы, а B – множество соклассников. Назовите множество B относительно A.

3. Укажите отношение между множествами A и B из рисунка

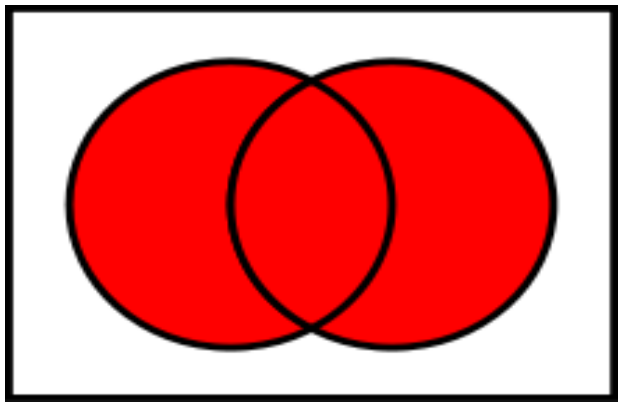


Помощь:

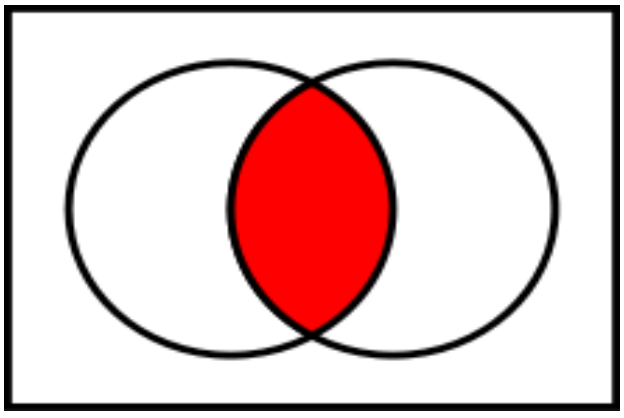
1. Заданное уравнение решается с помощью преобразования.
2. Множество B является частью множества A.
3. Множество A является частью множества B.

Вопросы I.3.3:

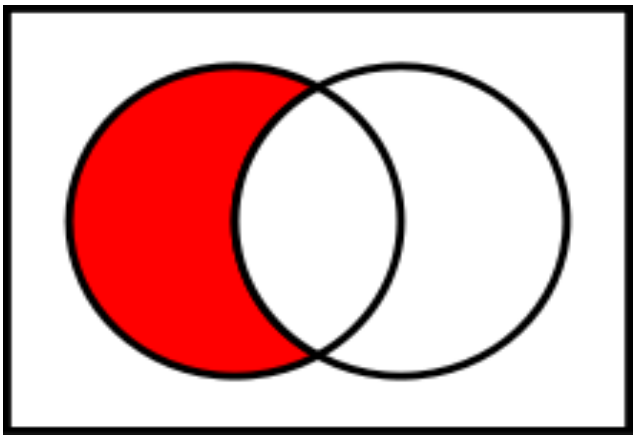
1. Какая операция представлена на рисунке?



2. Какая операция представлена на рисунке?



3. Какая операция представлена на рисунке?



Тесты I.3.3.

1. Какой результат получится после выполнения операции $A \cap B$ для заданных множеств A={0, 2, 4, 6} и B={–2, –1, 0, 1, 2}?

A) {0, 2};

B) {0, 2, 4, 6};

C) {–2, –1, 0, 1, 2};

D) {0, 2, 4, 6,–2, –1, 0, 1, 2};

E) $\varnothing$.

2. Какой результат получится после выполнения операции $A \cup B$ для заданных множеств A={1, 3, 5} и B={2, 4, 6, 8}?

A) {1, 2, 3, 4, 5, 6, 8,} ;

B) {1, 3, 5};

C) {2, 4, 6, 8};

D) $\varnothing$;

E) {1, 3, 8}.

3. Какой результат получится после выполнения операции $A \setminus B$ для заданных множеств A={а, б, в, г, д, е, ж} и B={г, е, ж}?

A) {а, б, в, д};

B) {а, б, в, г, д, е};

C) {г, е, ж};

D) { г, е};

E) $\varnothing$.

I.3.4. Строки, массивы и таблицы

Строки – это статические несвязные линейные структуры данных, образованные из элементов (символов) конечного множества (алфавита). Например, если в качестве алфавита взять множество, состоящее из всех букв, цифр, знаков операций, знаков препинания, различных скобок и других специальных знаков в настоящей книге, то её можно считать одной строкой.

Чтобы различать одну строку от второй строки используется специальный символ-разделитель или изменяется алфавит.

Строки владеют следующими свойствами:

1. Строки должны быть не индексированы и упорядочены, так как доступ к нужным символам в них могут быть начаты с определенных сторон.

2. Если рассматривать строки, как слова определенного языка, то должны быть грамматические правила для построения таких строк, так как в этом случае внутри строк содержатся дополнительные структуры, соответствующие правилам грамматики этого языка.

При обработке строки нужно совершать действия, подобные нахождению определенного символа или определению частоты встречаемости этого символа в строке, или замена известного символа на другой символ.

Любая определенная над строкой сложная операция состоит из композиции самых простых операций *нахождения места символа*, *вставления символа*, *удаления символа* (аналогично операции *сложения* и *вычитания*, определенные над числами). Например, чтобы поменять местами два символа нужно два раза находить место символа, два раза удалить символ и два раза вставить символ.

Массив – статическая структура данных в виде набора однотипных элементов, расположенных в непосредственно друг за другом.

Доступ к отдельным элементам массива является произвольным и осуществляется указанинем его имени и индекса нужного элемента.

Размерность массива – это количество индексов, необходимое для однозначного доступа к элементу массива.

Если массив имеет *один индекс*, то он называется *одномерным массивом*, если массив имеет *два индекса*, то он – *двухмерным массивом*, и т.д. соответственно. При этом общее количество элементов (длина) массива вычисляется произведением максимальных значений индексов каждой размерности.

Одномерный массив соответствует вектору в математике, двумерный – матрице. Чаще всего применяются массивы с одним или двумя индексами, реже — с тремя, ещё большее количество индексов встречается крайне редко.

В массиве однозначное местоположение элемента в памяти компьютера обеспечивается именно однотипностью элементов и определяется произведением значений его индекса (ов) на размер ячейки памяти, занимаемой одним элементом. В одномерном случае местоположение элемента в памяти вычисляется формулой:

$$Адрес(элемент(index)) = index * размер\_ячейки.$$

Форма или структура массива – количество размерностей плюс размер массива для каждой размерности; может быть представлена одномерным массивом.

Таблицы – это статические несвязные структуры данных, которые строятся из однотипных или разнотипных элементов индексированного одного множества или из элементов прямых произведений нескольких индексированных множеств.

Если все элементы таблицы имеют один и тот же тип, то такие таблицы являются *массивами*. Если таблица построена из элементов только одного множества, то её называют *одномерной таблицей*, представляющую собой *линейную структуру данных*. Если таблица построена из элементов прямого произведения нескольких множеств, то её называют *многомерной таблицей*, представляющую собой *нелинейную структуру данных*. Число измерений равно числу индексов, поэтому элементы будут одноиндексными, двухиндексными, трехиндексными и т.д. Например, в одномерной *A*, двумерной *B*, трехмерной *C* таблицах индексы будут такими:

$A = (a_1, a_2, a_3), B = (b_{11}, b_{12}, b_{21}, b_{22}, b_{31}, b_{33}), C = (c_{111}, c_{112}, c_{121,}, c_{122}, c_{211}, c_{212}, c_{221}, c_{222})$.

Указав значения индексов многомерной таблицы, можно осуществить доступ к каждому её элементу. Например, элементы двумерной шестиэлементной таблицы *B* можно представить следующим образом: $b_{11}, b_{12}, b_{21}, b_{22}, b_{31}, b_{33}$.

При хранении табличных данных используются разделители, в двумерных таблицах применяется два типа разделений: вертикальные разделители и горизонтальные разделители. Если нужно сохранить двумерную таблицу в виде символьной строки, то используют один символ-разделитель между элементами, принадлежащими одной строке, и другой разделитель для отделения строк.

Таблицу можно обработать, выполнив числовые, символьные и логические операции, определенные в соответствии с типами элементов. Кроме того, можно упорядочить таблицу и найти или изменить значения элементов, удовлетворяющих определенным условиям.

Замечание I.3.4.

Элементы в таблице могут быть не атомарными, а структурными, т.е. можно получить вложение одной структуры данных в другую структуру данных. Если элементами таблицы являются таблицы, то можно говорить о вложенности таблиц. С помощью таких вложенных структур данных можно реализовать иерархическую и сетевую структуры данных.

Примеры I.3.4.

1. Пусть задано множество *A* = {0, 1}. Тогда цепочка 111101010101 – строка, определенная во множестве *A* и построенная применением операции сцепления над цифрами 0 и 1.

2. В виде вектора *d*(1) *d*(2) *d*(3) *d*(4) *d*(5) можно представить ряд из пяти некоторых однотипных предметов (элементов), где *d* (*i*), *i* = 1,2,3,4,5 указывает на конкретный элемент этого ряда.

3. Таблицы в MS Word, MS Excel, MS Access являются

двумерными, в которых элементы находятся на пересечении индексов строки и индексов столбцов.

4. В двумерных таблицах в качестве *вертикальных разделителей* и *горизонтальных разделителей* используют линии. Ниже показана таблица планет солнечной системы.

Планета	**Расстояние до Солнца, а.е.**	**Относительная масса**	**Количество спутников**
Меркурий	0,39	0,056	0
Венера	0,67	0,88	0
Земля	1,0	1,0	1
Марс	1,51	0,1	2
Юпитер	5,2	318	16

Таблица планет солнечной системы в виде символьной строки выглядит так:

Меркурий*0,39*0,056*0#Венера*0,67*0,88*0#Земля*1,0*1,0*1#Марс*1,51*0,1*2#Юпитер* 5,2*318*16#

Здесь * – символ-разделитель между элементами, # – символ-разделитель между строками, а.е. (астрономическая единица) – среднее расстояние от Земли до Солнца, 1 а.е. = 149 598 000 км.

Задания I.3.4.

1. Представьте название столицы нашей родины в виде строки и укажите алфавит, из элементов которого она образуется.

2. Приведите конкретный пример двумерного массива чисел с помощью математической структуры.

3. Постройте таблицу одногруппников, указав в ней ФИО, пол, национальность.

Помощь:

1. Алфавит будет зависить от используемого языка.
2. В двумерной таблице имеются строки и столбцы.
3. Для этого достаточно применить таблицу MS Word.

Вопросы I.3.4.

1. Как образуется строка?
2. Из чего состоит массив?
3. Чем задается таблица?

Тесты I.3.4.

1. Из каких простых операций может состоять сложная операция над строкой?

A) нахождение, вставление, удаление;
B) соединение, пересечение, дополнение;
C) сложение, вычитание, умножение;
D) конкатенация, итерация, интеграция;
E) проекция, деление, мультипликация;

2. Чему равно число измерений в массиве?

A) количеству индексов;
B) количеству элементов;
C) произведению значений индексов;
D) минимальному значению индексов;
E) максимальному значению индексов.

3. Какие параметры нужно указать для нахождения элементов в таблице?

A) имя таблицы, индексы;
B) вертикальные индексы, горизонтальные индексы;
C) ключи, индексы;
D) индексы, значения;
E) прямые адреса элемента, ключи.

I.3.5. Списки

Список – упорядоченная структура данных, состоящая из переменного числа элементов.

Списки могут быть типизированными или не типизированными. Если список типизирован, то тип его элементов задан, и все его элементы должны иметь типы, совместимые с заданным типом элементов списка.

Список может быть сортированным или несортированным. Если список отражает отношение соседства между элементами, то он называется *линейным списком*. Если в списке не допускаются ограничения на размер (длину), то он представляется в памяти в виде связной структуры.

В зависимости от реализации возможен произвольный доступ к элементам списка.

Линейный односвязный список – это динамическая линейная структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент. Последний элемент списка указывает на ***Nil***. Каждый линейный односвязный список должен иметь особый элемент, называемый головой списка, который обычно по формату отличен от остальных элементов. В графическом представлении связи в списках изображаются с помощью стрелок. На рисунке I.3.5.А показан односвязный список.

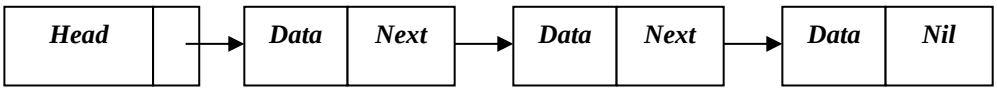


Рисунок I.3.5.А. Структура односвязного списка.

Здесь ***Head*** – голова списка, ***Data*** –данные, ***Next*** – указатель на следующий элемент списка, ***Nil*** – признак конца списка.

Порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними

связями. Обработка линейного односвязного списка не всегда удобна, так как отсутствует возможность продвижения в противоположную сторону. Такую возможность обеспечивает линейный двухсвязный список, в котором предусмотрена возможность двигаться в обе стороны: с головы до конца, и наоборот.

Линейный двухсвязный список – динамическая линейная связная структура данных, каждый элемент которого содержит два указателя: на следующий и на предыдущий элементы списка.

Для удобства обработки списка добавляют еще один особый элемент - указатель конца списка. Наличие двух указателей в каждом элементе усложняет список и приводит к дополнительным затратам памяти, но в то же время обеспечивает более эффективное выполнение некоторых операций над списком. Структура линейного двухсвязного списка показана на рисунке I.3.5.B.

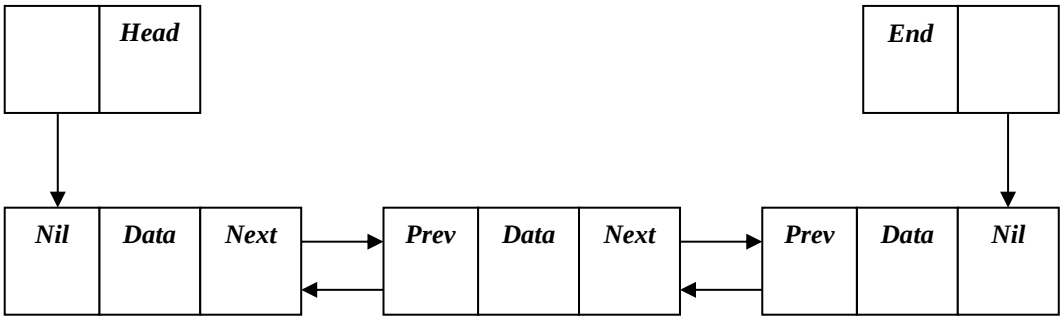


Рисунок I.3.5.B. Структура двухсвязного списка.

Здесь ***Head*** – голова списка, ***Data*** –данные, ***Prev*** - указатель на предыдущий элемент, ***Next*** – указатель на следующий элемент списка, ***Nil*** – признак конца списка.

Кольцевой список – разновидность видов линейных односвязных или двухсвязных списков.

Если кольцевой список организован на основе линейного односвязного списка, то указатель последнего элемента должен указывать на первый элемент.

Если кольцевой список организован на основе линейного

двухсвязного списка, то в первом и последнем элементах соответствующие указатели переопределяются. При работе с такими списками несколько упрощаются некоторые процедуры, выполняемые над списком. Однако при просмотре такого списка следует принять некоторые меры предосторожности, чтобы не попасть в бесконечный цикл.

В памяти список представляет собой совокупность дескриптора и одинаковых по размеру и формату записей, размещенных произвольно в некоторой области памяти и связанных друг с другом в линейно упорядоченную цепочку с помощью указателей. Запись содержит поля данных и поля указателей на соседние элементы списка, причем некоторыми полями данных могут быть указатели на блоки памяти с дополнительной информацией, относящейся к элементу списка. Дескриптор списка реализуется в виде особой записи и содержит такую информацию о списке: *адрес начала списка*, *код структуры*, *имя списка*, *текущее число элементов в списке*, *описание элемента* и т.д. Дескриптор может находиться в той же области памяти, в которой располагаются элементы списка, или для него выделяется другое место.

На рисунке I.3.5.C показана структура кольцевого списка.

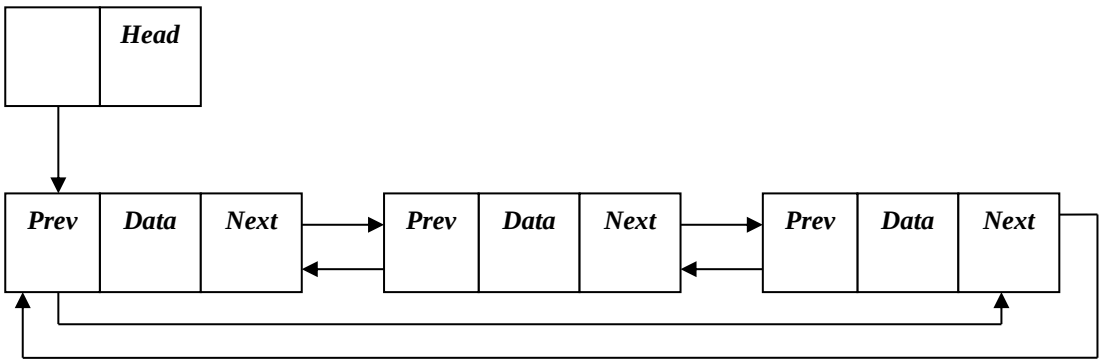


Рисунок I.3.5.C. Структура кольцевого двухсвязного списка.

Иерархический список – симбиоз линейного списка и дерева. Каждый элемент списка может быть также началом списка следующего подуровня иерархии и многосвязным.

На рисунке I.3.5.D показана структура иерархического списка.

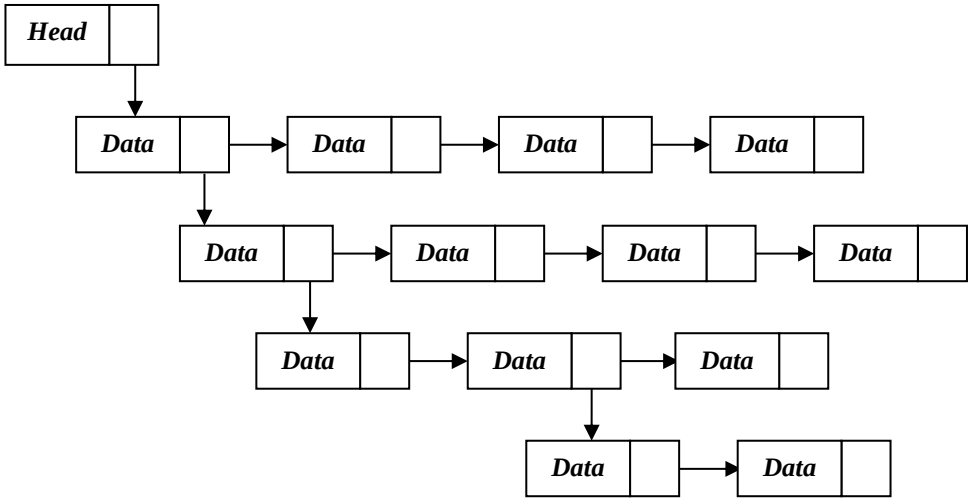


Рисунок I.3.5.D. Структура иерархического списка.

Над списком определены следующие операции:

– операция, вычисляющая головной элемент списка;

– операция доступа к списку, состоящему из всех элементов исходного списка, кроме первого;

– операции добавления элемента список (в начало, конец или вовнутрь после любого элемента списка);

– операция, вычисляющая длину (число элементов) списка;

– операция, проверяющая список на пустоту.

Связные списки имеют свои достоинства и недостатки. К достоинствам относятся:

– лёгкость добавления и удаления элементов;

– размер ограничен только объёмом памяти компьютера и разрядностью указателей;

– динамическое добавление и удаление элементов.

Недостатками связных списков являются:

– сложность определения адреса элемента по ссылке в списке (в массиве по номеру или индексу);

– на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны);

– работа со списком медленнее, чем с массивами, так как к

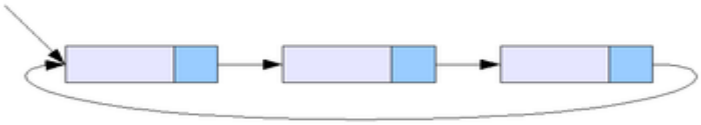
любому элементу списка можно обратиться, только пройдя все предшествующие ему элементы;

– элементы списка могут быть расположены в памяти разреженно, что окажет негативный эффект на кэширование процессора;

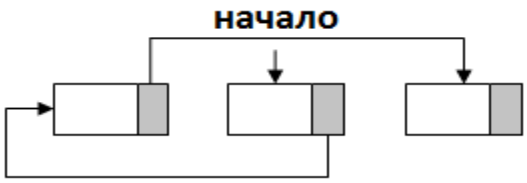
– над связными списками гораздо труднее (хотя и в принципе возможно) производить параллельные векторные операции, такие как вычисление суммы.

Примеры I.3.5.

1. Односвязный кольцевой список: последний элемент содержит указатель на первый:



2. Порядок элементов определяется ссылкой на первый элемент и последовательностью ссылок на остальные элементы списка:



3. Структура интернет форумов является иерархическим списком: последовательность сообщений образует линейный список.

Задания I.3.5.

1. Сформируйте линейный список для целых чисел 1, 9, 5, 2.
2. Постройте иерархический список для представления Food - еды, состоящей из Vegetables - овощей и Fruit - фруктов.
3. Сформируйте двухсвязны список между ченами человеческой семьи.

Помощь:

1.Элемент списка содержит указатель для установки связи с другими: один указатель – линейный, два указателя – двухсвязные.

2. Иерархический список состоит из уровней.

3. Двух связные списки имеют два вида связи (отношения).

Вопросы I.3.5.

1. Что такой линейный односвязный список?
2. Что такой кольцевой список?
3. Что такой иерархический список?

Тесты I.3.5.

1. Какие указатели содержит элемент двухсвязного списка?
A) на следующий элемент и на предыдущий элемент;
B) на первый элемент и на последний элемент;
C) на голову и на конец;
D) на первый элемент и на средний элемент;
E) на средний элемент и на последний элемент.

2. Чему равна длина списка?
A) количеству элементов;
B) количеству указателей;
C) количеству связей;
D) количеству звеньев;
E) количеству ссылок.

3. Какие поля имеются в кольцевом списке?
A) предыдущие, данные, следующие;
B) голова, данные, конец;
C) первые, средние, конец;
D) предыдущие, следующие, конец;
E) первые, следующие, конец.

I.3.6. Хеш-таблицы

Хеш-таблицы – это динамические структуры данных, элементами которых являются пары: *ключи*, *значения*.

Хеш-таблица представляет собой обобщение обычного массива. Однако в то время как ключом массива может быть только целое число, для хеш-таблицы им может быть любой объект, для которого можно вычислить хеш-код.

Над хеш-таблицами определены три операции:

- *добавление новой пары*;
- *поиск значения по ключу*;
- *удаление пары ключ-значение по ключу*.

Идея хеширования основана на распределении ключей в обычном массиве ***H***[0..*m*-1]. Распределение осуществляется вычислением для каждого ключа элемента некоторой хеш-функции *h*. Эта функция на основе ключа вычисляет целое число *n*, которое служит индексом для массива ***H***. Конечно, необходимо придумать такую хеш-функцию, которая бы давала различный хеш-код для различных объектов.

Хеш-таблица оптимизирована для быстрого поиска элементов за счет вычисления адреса (индекса) элемента, как значения хеш-функции. Аргументом хеш-функции является некий ассоциированный с элементом ключ – его порядковый номер. Для каждого типа данных можно разработать свою хеш-функцию. Однако есть основные требования к хеш-функции: она должна распределять ключи по ячейкам хеш-таблицы как можно более равномерно, и должна просто вычисляться.

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Полученное хеш-значение $i = hash\ (key)$ играет роль индекса в массиве ***H*.** Затем, выполняемая операция (добавление, поиск или удаление), перенаправляется к объекту, который хранится в соответствующей ячейке массива ***H***[*i*].

На рисунке 1.3.6.А показано, что индексами ключей в хеш-таблице является результат хеш-функции *h*, применённой к ключу.

Так же этот рисунок иллюстрирует одну из основных проблем. При достаточно маленьком значении размера хеш-таблицы *m* по отношению к количеству ключей *n* или при плохой хеш-функции, может случиться так, что два ключа будут хешированны в одну и ту же ячейку массива ***H***. Такая ситуация называется *коллизией*. Хорошие хеш-функции стремятся минимизировать вероятность коллизий, однако, учитывая то, что пространство всех возможных ключей может быть больше размера нашей хеш-таблицы ***H***, всё же избежать их вряд ли удастся. На этот случай имеются несколько технологий для разрешения коллизий. Основные из них мы и рассмотрим далее.

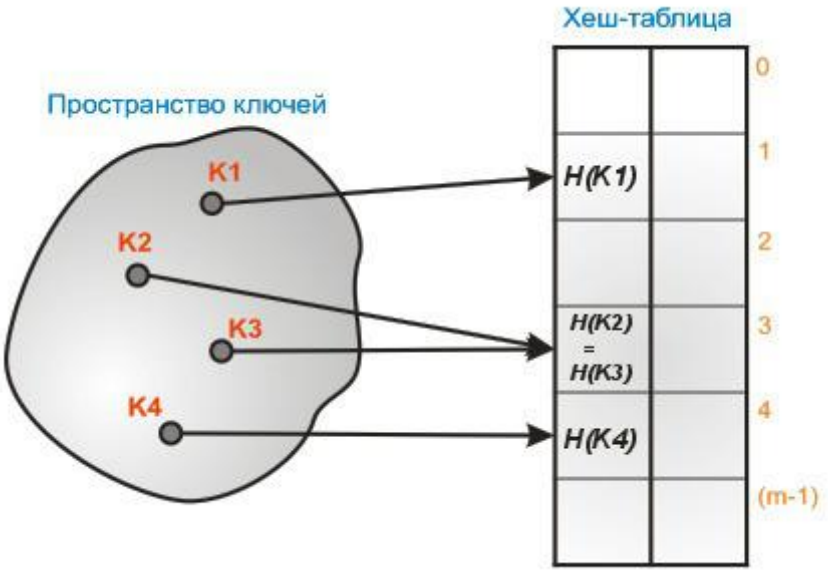


Рисунок 1.3.6.A. Схема индексации ключей в хеш-таблице.

Хеш-таблица содержит некоторый массив ***H***, элементы которого есть списки пар (хеш-таблица, цепочки) или пары (хеш-таблица, открытая адресация).

Существуют два основных варианта хеш-таблиц: *хеширование с цепочками* (*открытое хеширование*) и *хеширование с открытой адресацией* (*закрытое хеширование*).

Хеширование с цепочками. В случае хеширования с цепочками, объединяются элементы, хешированные в одну и ту же ячейку, в связный список. На рисунке 1.3.6.B показано хеширование с цепочками.

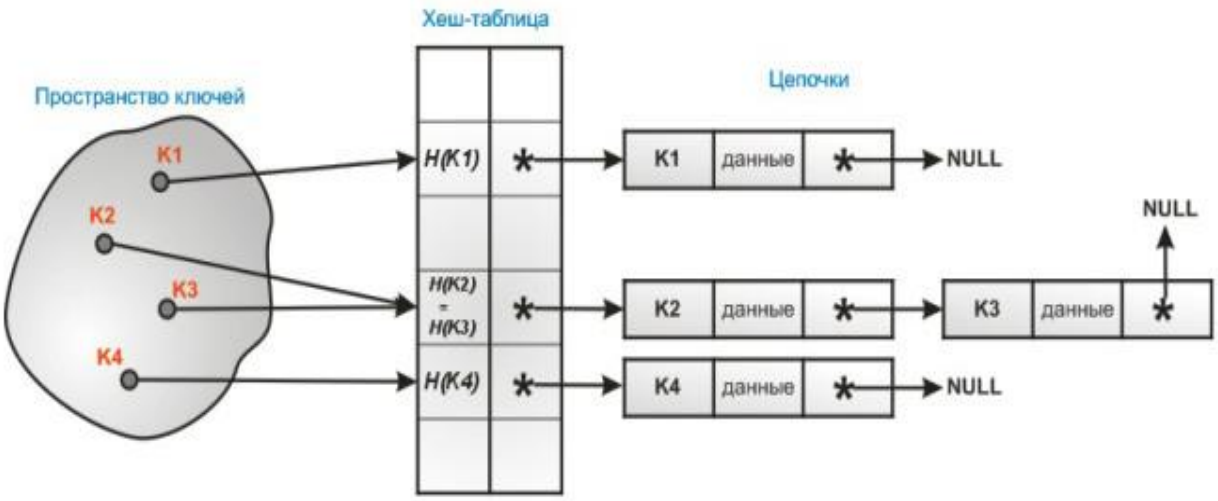


Рисунок 1.3.6.B. Хеширование с цепочками.

Здесь элементы $H(K2)$ и $H(K3)$ хешированы в одну ячейку, поэтому они объеденены в связный список.

Идея обнаружения коллизий достаточно проста. Если при добавлении элементов в хеш-таблицу в заданную ячейку мы встречаем ссылку на элемент связного списка, то случается коллизия. Так, мы просто вставляем наш элемент как узел в список. При поиске мы проходим по цепочкам, сравнивая ключи между собой на эквивалентность, пока не доберёмся до нужного. При удалении ситуация такая же.

Процедура вставки выполняется за $O(1)$ времени, учитывая то, что мы предполагаем отсутствие вставляемого элемента в таблице. Время поиска не больше, чем $O(n)$, и равно $O(n)$, если все элементы хешируются в единственную ячейку.

Хеширование с открытой адресацией. В случае метода открытой адресации все элементы хранятся непосредственно в хеш-таблице, без использования связанных списков. В отличии от хеширования с цепочками, при использовании метода открытой адресации может возникнуть ситуация, когда хеш-таблица окажется полностью заполненной, так что будет невозможно добавлять в неё новые элементы. Так что при возникновении такой ситуации решением может быть динамическое увеличение размера хеш-таблицы, с одновременной её перестройкой.

Для разрешения же коллизий применяются несколько подходов.

Самый простой из них – это метод линейного исследования. В этом случае при возникновении коллизии следующие за текущей ячейки проверяются одна за другой, пока не найдётся пустая ячейка, куда и помещается наш элемент. Так, при достижении последнего индекса таблицы, мы перескакиваем в начало, рассматривая таблицу как «цикличный» массив. На рисунке 1.3.6.C показано хеширование методом линейного исследования.

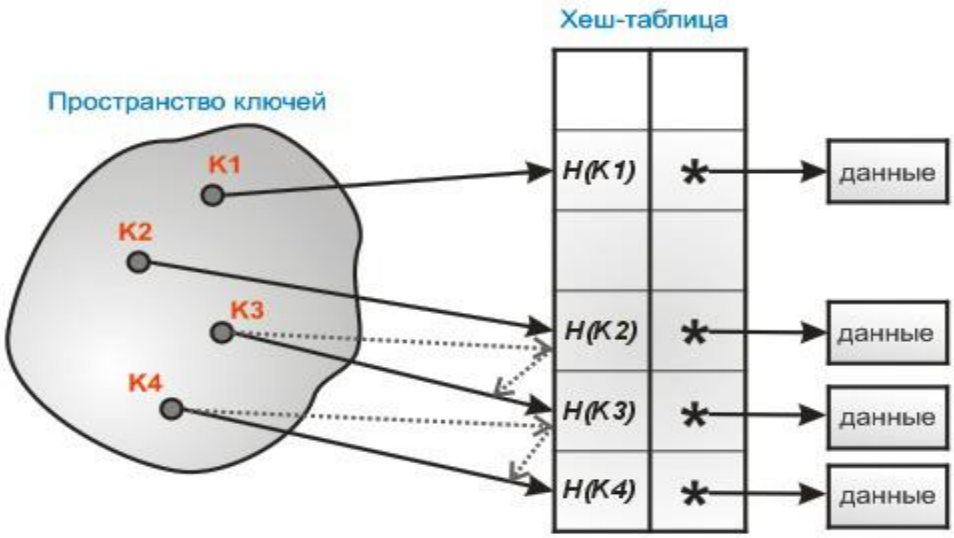


Рисунок 1.3.6.C. Хеширование методом линейного исследования.

Линейное хеширование достаточно просто реализуется, однако с ним связана существенная проблема – кластеризация. Это явление создания длинных последовательностей занятых ячеек, которое увеличивает среднее время поиска в таблице. Для снижения эффекта кластеризации используется другая стратегия разрешения коллизий – двойное хеширование. Основная идея этой стратегии заключается в том, что для определения шага смещения исследований при коллизии в ячейке используется другая хеш-функция, вместо линейного смещения на одну позицию.

Одной из сложных вопросов реализации хеширования с открытой адресацией – это операция удаления элемента. Дело в том, что если мы просто удалим некий элемент из хеш-таблицы, то сделаем невозможным поиск ключа, так как в процессе вставки которого текущая ячейка может быть заполненной. Так, мы можем помечать очищенные ячейки какой-то меткой, чтобы впоследствии это учитывать.

В некоторых специальных случаях удаётся избежать коллизий вообще. Например, если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию, которая распределит их по ячейкам хеш-таблицы без коллизий. Хеш-таблицы, которые используют подобные хеш-функции, не нуждаются в механизме разрешения коллизий и называются хеш-таблицами *с прямой адресацией*.

Если хеш-функция распределяет n ключей по m ячейкам таблицы равномерно, то в каждом списке будет содержаться порядка n/m ключей.

Значение n/m, полученное делением числа хранимых элементов n на размер массива $\boldsymbol{H}$ (число возможных значений хеш-функции) m, называется *коэффициентом заполнения хеш-таблицы – load factor*. Это значение является важным параметром, от которого зависит среднее время выполнения операций.

Теперь можно привести основные требования к хеш-функциям:

- функция должна возвращать 32-битное целое значение;
- функция должна получать на входе не пустую строку без явно заданной длины;
- функция должна давать как можно более равномерное распределение хеш-значения;

Примеры I.3.6.

При вставке в хеш-таблицу размером 365 ячеек всего лишь 23-х элементов вероятность коллизии будет превышать 50%, если каждый элемент может равновероятно попасть в любую ячейку (смотрите парадокс дней рождения).

Задания I.3.6.

1. Постройте хеш-таблицу методом цепочки, в которой ключами будут названия областных центров нашей страны, а значениями – целые числа.

2. Постройте хеш-таблицу методом открытой адресации, в которой ключами будут идентификаторы в программе, а значениями – их адреса.

3. Постройте хеш-таблицу студентов методом линейного хеширования, в которой ключами будут ФИО, а значениями –ИИН.

Помощь:

Используйте определение хеш-таблицы и методы их построения.

Вопросы I.3.6.

1. В каких случаях возможно возникновение коллизий?
2. В каком случае поиск в хеш-таблицах невозможен?
3. Как выбирается метод изменения адреса при повторном хешировании?

Тесты I.3.6.

1. Какие операции опредлены над хеш-таблицами?

A) добавление новой пары, поиск значения по ключу, удаление пары ключ-значение по ключу

B) добавление новой пары;

C) поиск значения по ключу;

D) удаление пары ключ-значение по ключу;

E) добавление новой пары, поиск значения по ключу.

2. Что такое коллизия?

A) один ключ хешируется в одну ячейку;

B) один ключ хешируется в две ячейки;

C) два ключа хешируются в одну ячейку;

D) два ключа хешируются в две ячейки;

E) мног ключей хешируются в много ячеек;

3. Что нужно указать для нахождения элементов хеш-таблицы?

A) ключи, значения;

B) вертикальные индексы, горизонтальные индексы;

C) ключи, индексы;

D) индексы, значения;

E) прямые адреса элемента, ключи.

I.3.7. Графы и деревья

Граф – динамическая сетевая связная структура данных, представленная множеством пар, называемых *вершинами* и *ребрами*. Каждая вершина может быть связана с несколькими другими вершинами или с самой собой при помощи ребер и вершины, не образующую иерархию. Формально граф определяется как множество пар $G = (V, E)$, где V – множество вершин, E – множество рёбер, фактически есть отношение на множестве V, т.е. $E \subseteq V \times V$. Вершины и рёбра графа называются также элементами графа. Число вершин в графе $|V|$ является порядком графа, а число рёбер $|E|$ – размером графа.

Говорят, что в графе ребро $e = (x_i, x_j) \in E$ соединяет вершины $x_i \in V$ и $x_j \in V$, i=1, 2, … ; j=1, 2, … . В свою очередь, вершины $x_i \in V$ и $x_j \in V$ ограничивает ребро $e = (x_i, x_j) \in E$, т.е. они являются концевыми вершинами (концами) ребра.

Касательно рёбер графа можно ввести следующие определения:

1. Два ребра графа являются *смежными*, если они имеют общую концевую вершину;
2. Ребра с одинаковыми концевыми вершинами называются кратными рёбрами;
3. Ребро графа называется *петлёй*, если его концы совпадают, то есть $e = (x_i, x_i)$.

Замечание 1.3.7: При подсчете степени вершин петли, ребро считается дважды.

По отношению вершин графа дополнительно определяются:

1. Две вершины называются *смежными*, если они являются концевыми вершинами одного и того же ребра;
2. Можно сказать, что *вершина инцидентна ребру* (*ребро инциндентно вершине*), если она является концевой вершиной этого ребра;
3. *Степенью вершины* называется количество рёбер, инцидентных этой вершине, при этом петли считают дважды.

4. Вершина называется *висячей* (*листом*), если она является концом ровно одного ребра, т.е. степень висячей вершины равна 1;

5. Вершина называется *изолированной*, если она не инцидентна ни одному ребру, т.е. степень изолированной вершины равна 0;

Имеются несколько типов графа. Относительно типов графов можно ввести следующие определения:

3. Если множества V и E конечны, то граф называется конечным.

4. Конечный граф, имеющий количество вершин n и количество ребер m, называется (n; m)-*графом*.

5. Граф, состоящий только из изолированных вершин, называется пустым или *нуль-графом*.

6. Граф без петель и кратных ребер называется *простым графом*.

7. Простой граф, в котором любые две вершины соединены ребром, называется *полным графом*.

8. Если множество вершин простого графа V допускает такое разбиение на два непересекающиеся подмножества V_1 и V_2, что не существует ребер, соединяющих вершины одного и того же подмножества, то он называется двудольным или биграфом.

9. Граф без петель, но с кратными ребрами называется *мульти-графом*.

10. Граф, содержащий хотя бы одну петлю, называется *псевдографом*. В псевдографе могут быть кратные ребра.

11. Если из каждой вершины графа исходит равное количество ребер и в каждую вершину заходят равное количество ребер, то такой граф называется *регулярным графом*.

12. Если для каждого ребра графа определено направление, то такой граф называется *ориентированным графом*.

13. Если каждое ребро графа имеет вес, то такой граф называется *взвешанным графом*, т.е. можно определить функцию $w : E \to R$, где R – множество действительных чисел, w – вес ребра и $w \geq 0$.

Путем в графе называют последовательность ребер, ведущая от одной вершины к другой вершине, такая, что каждые два соседних ребра имеют общую вершину, и никакое ребро не встречается более

одного раза, т.е. формально путь в графе есть такая последовательность вершин $(x_1, x_2, x_3, ..., x_{m-1}, x_m)$, что пары $\{(x_1, x_2), (x_2, x_3), ..., (x_{m-1}, x_m)\}$ станут ребрами. Две вершины $x_i \in V$ и $x_j \in V$ в графе называются *связными* (*несвязными*), если в нем существует (не существует) путь, ведущий из x_i в x_j. Этот путь может быть в обоих направлениях. Если каждые две вершины в графе связны, то такой граф называется *связным графом.* Если же в графе найдется хотя бы одна пара несвязных вершин, то граф называется *несвязным*. Если все пары вершин графа связны в обоих направлениях, то такой граф будет *сильносвязным графом*.

Если исходящий из одной вершины путь обратно входит в эту же вершину, то такой путь называется *замыканием* (*циклом*), т.е. в замыкании начальная и конечная вершины совпадают. Если замыкание не проходит ни через одну из вершин графа более одного раза, то оно называется *простым замыканием.* Если замыкание исходит из одной вершины и напрямую входит в эту вершину обратно, оно называется *петлей,* т.е. в петле есть единственная вершина.

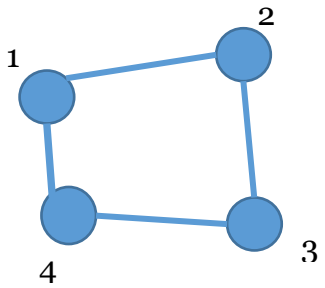
Длиной пути называется число ребер этого пути. Если веса рёбер являются их длинами, то длина пути в графе вычисляется с помощью следующей формулы:

$$w(x_1, x_2, x_3, ..., x_{m-1}, x_m) = \sum_{i=1}^{m-1} w(x_i, x_{i+1}).$$

В графах можно решать следующие задачи: *сравнение двух графов, нахождение наикратчайшего пути из одной вершины в другую, нахождение числа замкнутых путей* и др.

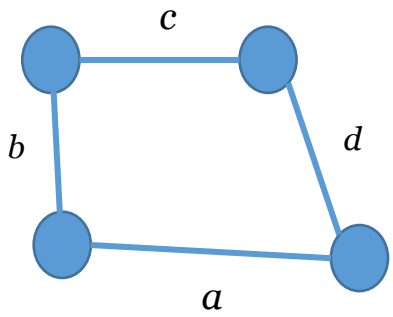
Примеры I.3.7.1:

1. Пусть заданный граф имеет вид:



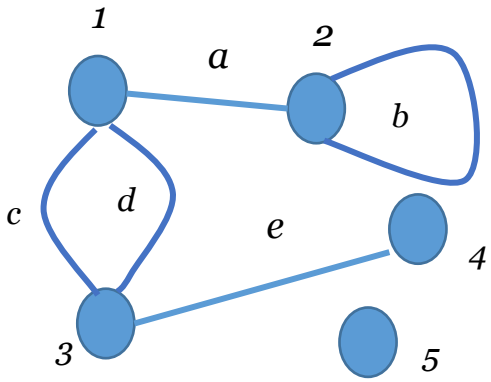
В этом графе вершины 1 и 2 являются смежными, а вершины 1 и 3 не являются смежными.

2. Пусть задан граф:



В этом графе ребра *a* и *b* являются смежными, а ребра *b* и *d* не являются смежными.

3. Пусть задан граф:



В этом графе вершины 1 и 2, 1 и 3, 3 и 4 являются смежными, а вершина 5 – изолированной. При этом вершины 1, 2 и 3 имеют степень три, вершина 4 является висячей вершиной. Кроме того, рёбра *c* и *d* – кратные, ребро *b* – петля.

Дерево – граф, в котором все вершины связаны, а пути незамкнуты, т.е. связный граф без циклов и без петель.

В дереве вершины подразделяются на следующие виды:

1) *корень* – вершина, из которой исходит одно или несколько ребер, но не входит ни одно ребро, т.е. вершина, которая не имеет ни одного предка, но может иметь много потомков;

2) *ветвь* – вершина, в которую входит одно ребро, но от него может исходить много ребер, т.е. вершина, которая имеет единственного предка и может иметь много потомков;

3) *лист* – вершина, в которую входит только одно ребро, но не исходит ни одно ребро, т.е. вершина, которая имеет единственного предка, но не имеет ни одного потомка.

В дереве направление пути проходит от корня через ветви до листьев. Внутри дерева могут быть несколько деревьев, которых будем называть *поддеревьями*.

Теперь можно дать следующее рекурсивное определение (со ссылкой на самого себя):

1. *Рекурсивный базис*: множество $\{v\}$, состоящее только из одной вершины v является деревом, где его единственная вершина есть одновременно корень и лист.

2. *Рекурсивный шаг*: если v – вершина и $A_1, A_2, ..., A_n$ – деревья, то можно построить новое дерево, в котором корнем является вершина v, а ребрами – исходящие из этой вершины и входящие в корни деревьев $A_1, A_2, ..., A_n$.

3. *Рекурсивное заключение*: Деревья получаются только правилами 1 и 2.

Из этого определения явно видно, что дерево является иерархической связной динамической структурой данных, представленной с едиственной корневой вершиной и её потомками. Максимальное количество потомков каждой вершины и определяет размерность дерева.

Это определение можно представить на рисунке I.3.7 следующим образом:

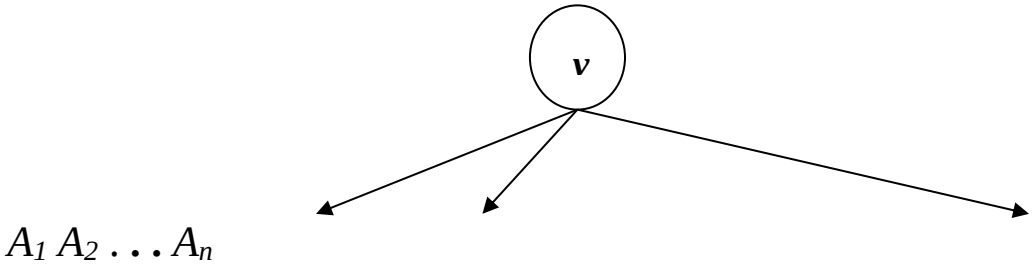


Рисунок I.3.7. Определение дерева

Среди деревьев особо выделяется, так называемые *двоичные* (*бинарные*) *деревья.* Его можно определить следующим образом:

Двоичное дерево – дерево, в котором каждая вершина имеет не более двух потомков. Эта вершина называется *родительской вершиной*, а потомки называются *левым наследником* и *правым наследником*. Дадим рекурсивное определение бинарного дерева. Бинарным деревом называется следуюшее множество вершин:

- либо ничего не содержит (пустое множество);
- либо состоит из корня, который соединяется с двумя бинарными деревьями, называемыми левостороннее поддерево и правостороннее поддерево.

Итак, бинарное дерево является либо пустым, либо состоит из данных и двух поддеревьев, каждое из которых может быть пустым. Если в некоторой вершине оба поддеревья пустые, то она есть лист. Формально бинарное дерево определяется так:

<биндерево > ::= nil | (<данные> < биндерево > < биндерево >), где nil – пусто.

В деревьях решаются следующие задачи: *обход деревьев*, *поиск в дереве*, *добавление новой вершины к дереву*, *уничтожение вершины дерева*, *сравнение деревьев* и др.

Двоичные деревья используются в алгоритмах поиска: каждая вершина двоичного дерева поиска соответствует элементу из

некоторого отсортированного набора, все его левые потомки – меньшим элементам, а все его правые потомки – большим элементам. Каждая вершина в дереве однозначно идентифицируется последовательностью неповторяющихся вершин от корня и до неё – путем. Длина пути и является уровнем вершины в иерархии дерева. Для практических целей обычно используют два подвида бинарных деревьев: *двоичное дерево поиска - binary search tree* (BST) и *двоичная куча*.

Двоичное дерево поиска обладает следующими свойствами:

– левое поддерево и правое поддерево являются двоичными деревьями поиска;

– у всех вершин левого поддерева произвольной вершины *v* значения ключей данных меньше, чем значение ключа данных самой вершины *v*;

– у всех вершин правого поддерева той же вершины *v* значения ключей данных больше, чем значение ключа данных вершины *v*.

Очевидно, что данные в каждой вершине должны обладать ключами, на которых определена операция сравнения.

Двоичная куча или сортирующее дерево обладает следующими свойствами:

– значение в любой вершине не меньше, чем значения в вершинах её потомков;

– глубина листьев (расстояние до корня) отличается не более, чем на один слой;

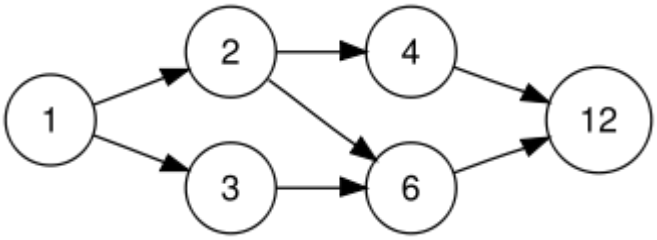
– последний слой заполняется слева направо.

Такая куча называется *max-heap*. Существуют также кучи, где значение в любой вершине, наоборот, не больше, чем значения её потомков. Такие кучи называются *min-heap*.

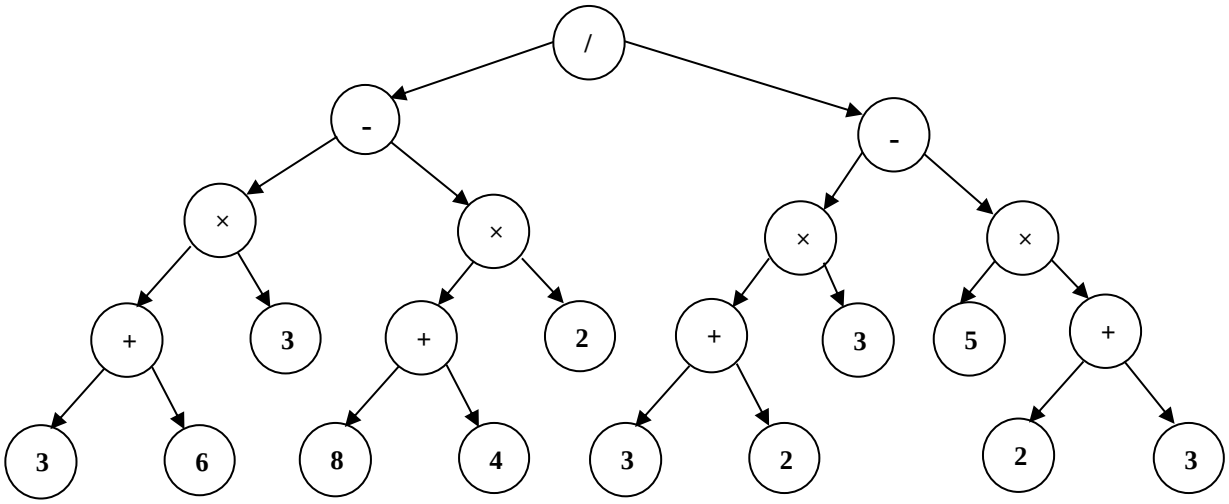
Примеры I.3.7.2:

1. Бинарное отношение над конечными объектами может быть представлено в виде ориентированного графа. Ниже показано

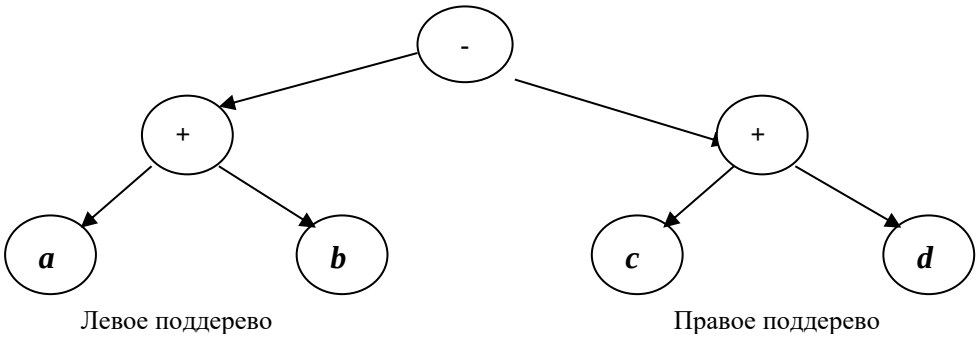
отношение делимости над целыми числами от 1 до 12: 2 и 3 делится на 1; 4 и 6 делится на 2; 6 делится на 2 и 3; 12 делится на 4 и 6.



2. Для вычисления значения арифметического выражения ((3+6)*3 - (8+4)*2) / ((3+2)*3 - 5*(2+3)), представленного бинарным деревом, производится обход дерева сверху вниз и слева направо.



3. Представление выражения (*a*+*b*) - (*c*+*d*) в виде бинарного дерева, где в качестве данных выступают символы: -,+, *a*, *b*, *c*, *d*.



Задания I.3.7:

1. Постройте ориентированный взвешанный граф для описания структуры идентификатора.

2. Постройте дерево для выражения (*a* + *b*) * (*x* -*z*).

Помощь:

1. Весами ребер должны быть буквы и цифры.
2. В соответствующем бинарном дереве листьями служат операнды, а остальными вершинами - операции.

Вопросы I.3.7:

1. Как образуется путь в графе?
2. Что такое дерево?
3. Что такое бинарное дерево?

Тесты I.3.7:

1. На какие виды подразделяются графы?
   A) ориентированный граф, неориентированный граф;
   B) ориентированный граф, определенный граф;
   C) определенный граф, неориентированный граф;
   D) определенный граф, неопределенный граф;
   E) неопределенный граф, неориентированный граф.

2. Что такое дерево?
   A) граф без петель и без циклов;
   B) граф без весов;
   C) граф без сетей и циклов;
   D) граф взвешанный;
   E) граф ориентированны.

3. Что такое бинарное дерево?

A) дерево, в котором каждая вершина имеет не более двух потомков;
B) дерево, в котором имеются две вершины;
C) дерево, в котором нет цикла;
D) дерево, в котором нет петли;
E) дерево, в котором одна вершина не имеет прямых потомков.

I.3.8. Стеки, очереди и деки

Стек (Stack) – динамическая линейная структура данных, в которой элементы, образуя упорядоченную по времени их поступления последовательность, реализуют принцип «*последним пришел и первым ушел – Last In Firsrt Out* (*LIFO*)». Эта последовательность доступна только с одного конца, называемого *вершиной стека*, т.е. все операции над стеком выполняются только на вершине стека. Для работы со стеком необходимо установить указатель вершины стека и определить над ним следующие операции:

– построение стека (создание указателя на вершину);
– добавление (заталкивание - *Push*) элемента в стек;
– удаление (выталкивание - *Pop*) элемента из стека;
– просмотр элемента в вершине стека без удаления;
– проверка состояния (пустоты) стека;
– очистка стека.

Стеки широко применяются в системном программном обеспечении, компиляторах, в различных рекурсивных алгоритмах. Например, в ходе выполнения программного кода, при необходимости вызвать процедуру указатель заносится на место ее вызова в стек, чтобы при завершении выполнения ее кода корректно вернуться к следующей после точки вызова инструкции. Стеки также применяются при организации рекурсивных вызовов: пока не обработан последний вызов, все предыдущие вызовы находятся в стеке.

Стек можно организовать на основе любой структуры данных, где возможно хранение нескольких однотипных элементов и реализация принципа работы стека. Наиболее подходящим для организации стека является однонаправленный список, причём в качестве вершины стека выбирается начало этого списка.

Стек можно изобразить как на рисунке I.3.8.A в виде размеченной на позиции трубки с подпружиненым дном, расположеным вертикально:

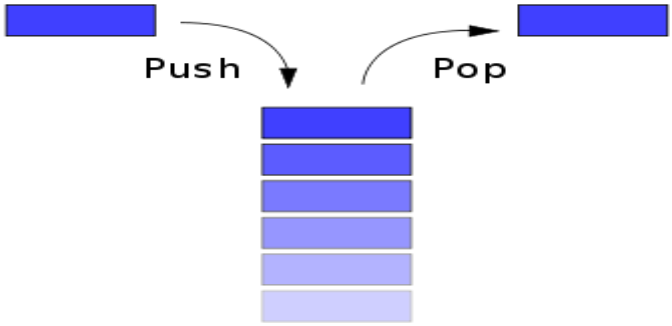


Рисунок I.3.8.A. Изображение стека.

Верхний конец трубки открыт, в него элементы добавляются (заталкиваютя - Push) и из него удаляются (выталкиваются - Pop).

Очередь – динамическая линейная структура данных, в которой элементы, образуя упорядоченную по времени их поступления последовательность, реализуют принцип «*первым пришел и первым ушел – First In First Out* (*FIFO*)». Эта последовательность доступна с двух сторон: с начала - *голова* очереди и с конца - *хвост* очереди. Над очередью определены следующие операции:

– создание очереди (установка указателей на голову и хвост);
– добавление (постановка - *enqueue*) элемента в хвост очереди;
– удаление (убирание - *dequeue*) элемента из головы очереди;
– очистка очереди.

На практике очереди могут реализовываться при помощи *одномерных массивов* или *связных списков*, что хорошо иллюстрирует различие между *логическим* и *физическим* уровнями структур данных (на физическом уровне - массив или список, на логическом уровне - очередь). Однако разумнее всего отобразить очередь на двунаправленный кольцевой список. Тогда она будет называться *кольцевой очередью* и в её заглавном звене будет присутствовать информация, как об указателе на голову, так и на хвост очереди. В этом случае, во-первых, снимается ограничение на размер очереди, и, во-вторых, по-другому используются переменные, указывающие на начало и конец очереди. Фактически устраняются различия между линейной и кольцевой очередями, независимо от типа (линейного или кольцевого) связного списка.

Очередь можно изобразить как на рисунке I.3.8.B в виде разбитые на ячейки ленты с указателями на начало и конец:

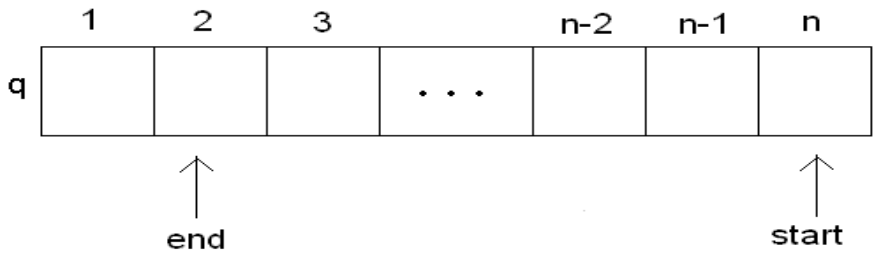


Рисунок I.3.8.B. Изображение очереди.

Области применения очередей могут быть разделены на две группы: *системное применение* и *прикладное применение*.

Системное применение очередей:

- диспетчеризация задач операционной системой;
- буферизация ввода/вывода.

Прикладное применение очередей:

- моделирование процессов (например, систем массового обслуживания);
- использование очередей как вспомогательных структур данных в каких-либо алгоритмах (например, при поиске в графах).

Дек – динамическая линейная структура данных, в которой элементы образуют последовательность и могут добавляться и удаляться с любого из двух концов этой последовательности. Если перемещение в очереди идет в одну сторону от первого элемента к последнему и в стеке движение также происходит в одну сторону, но в обратном порядке от последнего элемента к первому, то в деке движение элементов может осуществляться в обе стороны.

Слово дек происходит от Double Ended Queue (Deq) – *двухвходовая очередь.* Для выполнения основных операций над деком необходимо знать начало и конец дека, которые определяются ссылками на них.

Над деком определены следующие операции:

- добавление элемента с конца дека;
- добавление элемента с начала дека;

– удаление элемента с конца дека;
– удаление элемента с начала дека;
– определение размера дека;
– очистка дека.

Реализовывать дек можно с помощью двусвязных списков. Тогда дек можно представить в виде цепочки звеньев, каждый из которых состоит из трех полей (первое поле – поле ссылки на предыдущий, второе поле – поле элемента, третье поле – поле ссылки на последующий элемент).

Дек можно изобразить как на рисунке I.3.8.C в виде двухсвязного списка с указателями на начало и конец:

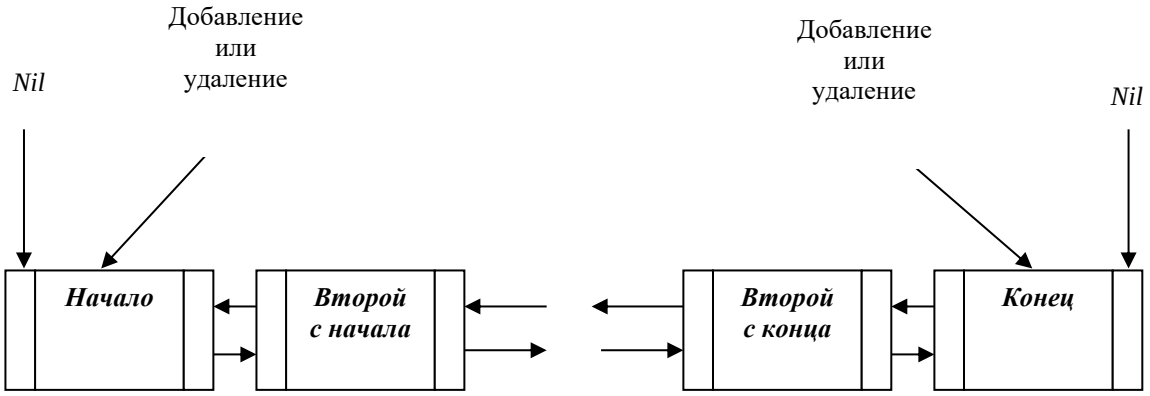


Рисунок I.3.8.C. Изображение дека.

На этом изображении имеются две ссылки *Nil-Пусто*, каждая из которых ограничивает движение по деку на его двух сторонах: в начале и в конце.

Примеры I.3.8:

1. Стеком может служить стопка книг на столе. Книги добавляются и снимаются только с верху стопки.
2. Очередью может служить оказание услуг в порядке поступления заявок на них: самой первой оказываемой услугой станет услуга, сответствующая первой заявке, а самой последней – услуга по последней заявке.

3. Ниже на рисунке I.3.8.D. показана последовательность состояний дека при включении и удалении пяти элементов A, B, C, D, E. На каждом этапе стрелка указывает с какого конца дека (левого или правого) осуществляется включение или исключение элемента.

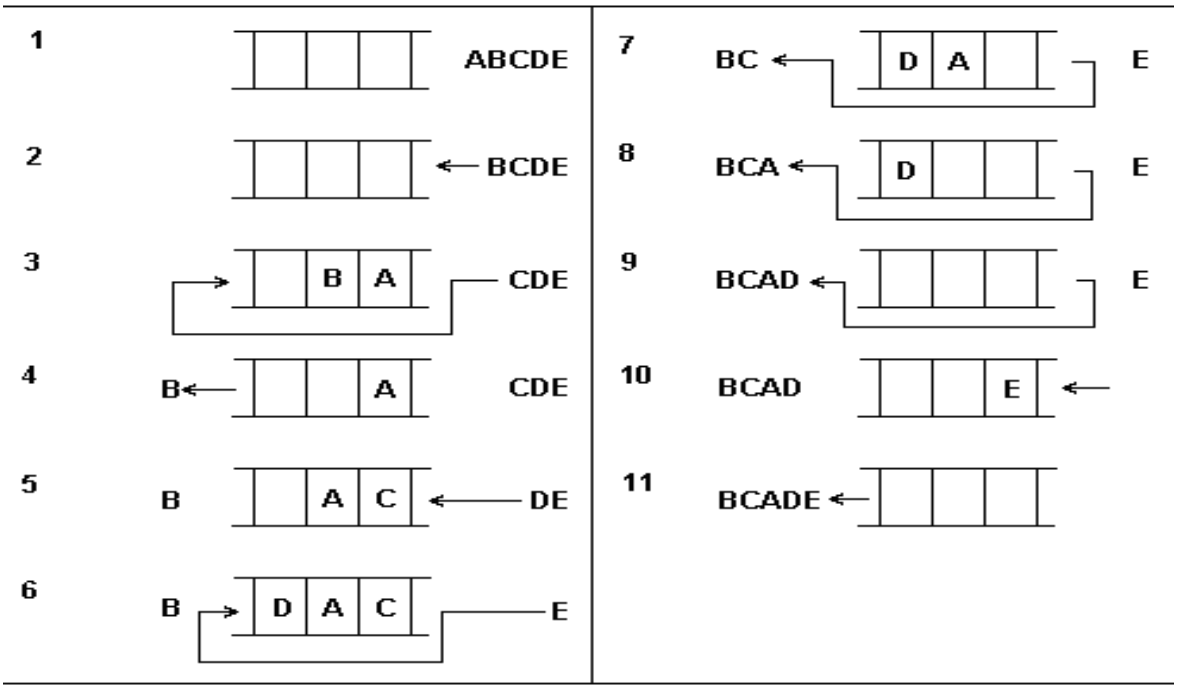


Рисунок. I.3.8.D. Состояния дека в процессе изменения.

Задания I.3.8:

1. Приведите из жизни примеры организации чего-либо по принципу стека.

2. Приведите из жизни примеры организации чего-либо по принципу очереди.

3. Приведите из жизни примеры организации чего-либо по принципу дека.

Помощь

1. Стек реализует принцип «последним пришел, первым ушел».
2. Очередь реализует принцип «первым пришел, первым ушел».
3. В деке движение может осуществляться в обе стороны.

Вопросы I.3.8:

1. Какие операции определены над стеком?
2. Какие операции определены над очередью?
3. Какие операции определены над деком?

Тесты I.3.8:

1**.** Какую структуру данных называют стеком?

A) динамическая линейная структура данных, в которой все операции выполняются только с одного конца;

B) статическая линейная структура данных с одним концом;

C) динамическая линейная структура данных с двумя концами;

D) динамическая иерархическая структура данных;

E) статическая иерархическая структура данных.

2. Какую структуру данных называют очередью?

A) динамическая линейная структура данных, в которой элементы могут добавляться с одного, а удаляться с другого конца;

B) статическая линейная структура данных с одним концом;

C) динамическая линейная структура данных с одним концом;

D) динамическая иерархическая структура данных;

E) статическая иерархическая структура данных.

3. Какую структуру данных называют деком?

A) динамическая линейная структура данных, в которой элементы могут добавляться и удаляться с двух концов;

B) статическая линейная структура данных с одним концом;

C) динамическая линейная структура данных с двумя концами;

D) динамическая линейная структура данных, в которой элементы могут добавляться и удаляться с одного конца;

E) статическая линейная структура данных, в которой элементы могут добавляться и удаляться с одного конца.

II. ТВЕРДОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАТИКИ

II.1. Компьютеры

В этом модуле будут рассмотрены состав, структура, принципы работы, рабочий цикл и поколения компьютеров. Обсуждаются их аппаратные и программные средства. Показывается архитектура и виды персональных компьютеров. Рассматриваются способы представления информации в компьютере.

Ключнвые слова: *автомат, автоматизация, компьютер, аналоговый сигнал, цифровой сигнал, аналоговый компьютер, цифровый компьютер, электронная вычислительная машина, память, вычислительная техника, бит, байт, ячейка, адрес, адресуемая объемная единица, адресное слово, машинное слово, оперативная память, внешняя память, процессор, регистр, персональное устройство, устройство ввода, устройство вывода, принтер, сканер, магнитная диск, пульт управления, принцип работы компьютера, рабочий цикл компьютера, машинный язык, указание, программа, поколение компьютеров, электронная лампа, транзистор, интегральная схема, язык программирования, транслятор, мониторная система, операционная система, многопрограммный режим, режим реального времени, комплекс компьютеров, канал связи, компьютерная сеть, искусственный интеллект, код, кодирование, стандарты кодирования.*

Цель: *рассмотреть историю появления компьютеров, показать состав и структуру компьютера, объяснить принцип работы и рабочий цикл компьютеров, ознакомить с методами методы представления данных в компьтере.*

Структура: Виды компьютеров показаны на рисунке II.

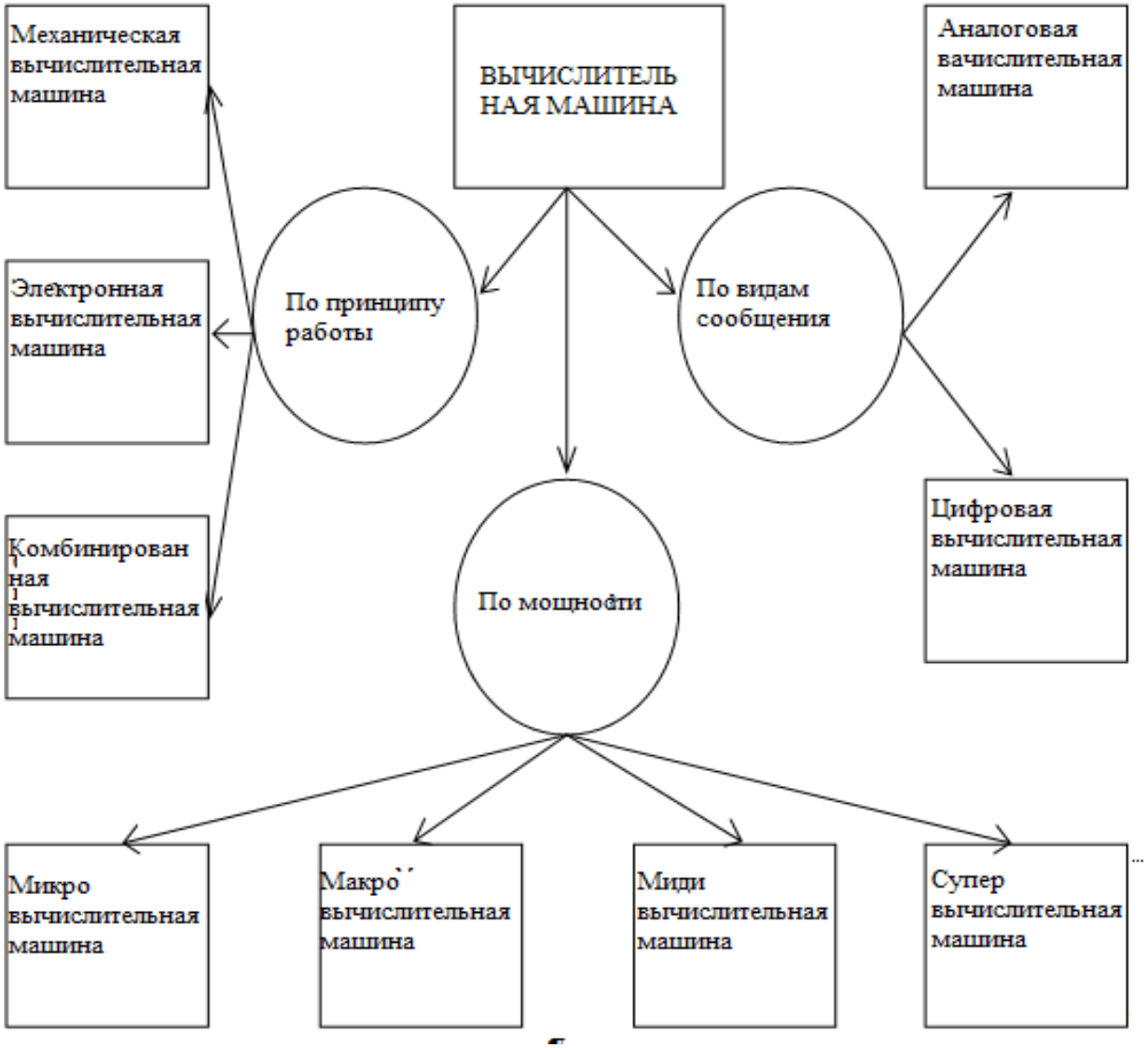


Рисунок II.1. Виды компьютеров.

II.1.1. Состав и структура компьютера

Слово «*Компьютер*» появилось в 40-х годах XX века. Он представляет собой физическое устройство для *автоматизации* хранения и обработки данных. С помощью компьютера можно решать различные задачи, пройдя следующие этапы:

– ввод информации или установка исходных данных;

– обработка или преобразование введенной информации;

– определение результатов и вывод обработанной информации.

Итак, компьютер информацию принимает, хранит, обрабатывает и передает полученный результат пользователю или другому компьютеру в соответствии с некоторым алгоритмом.

Известно, что информация характеризуется своим сообщением и содержанием, а сообщение передается от передатчика к приемнику в материально-энергетическом виде. Прием сообщения прямо зависит от некоторой изменяющейся во времени величины, которая характеризует состояние приемника. Иначе говоря, информационное сообщение можно представить функцией $X(t)$, которая изменяет во времени материально-энергетические параметры физической среды, где происходят информационные процессы. Эта функция может быть как и непрерывной, так и дискретной (например, зависящиеся от времени цепочки символов определенного языка или слова, образованные последовательностями звуков).

Сообщение, представляющееся при помощи непрерывной функции называется *аналоговым сигналом*, а сообщение, представляющееся с помощью дискретной функции – *цифровым сигналом.* Из-за ограниченности возможности человеческие органы чувств воспринимают непрерывную информацию в дискретном виде. В качестве примера можно взять определение с помощью термометра цифрового приближенного значения температуры воздуха. Во многих случаях очень эффективно будет преобразование непрерывного вида информации в дискретный вид. Для этого используют специальные устройства, называемые

аналого–цифровыми преобразователями. Например, в качестве такого устройства можно взять обычные электронные весы, используемые в магазинах: аналоговый сигнал –вес продукта преобразуется в цифровое представление измерений как грамм и килограмм. Поэтому компьютеры в зависимости от вида сообщения информации подразделяются на *аналоговые компьютеры*(АК) или *дискретные компьютеры* (ДК): это похоже на подразделение автомобилей на дизельные и бензиновые.

АК обрабатывает непрерывные данные, представляющие некоторые физические величины. Здесь в качестве физических переменных берутся сила тока, ускорение движения тела и т.п. Обычно с помошью АК можно решать только специальные задачи, т.е. они мало распространены. В дальнейшем АК не рассматриваются.

Самые первые ДК применялись только для обработки численных данных. Поэтому в его названии вместо слова «дискретный» использовали определитель «цифровой» и стали называть *цифровой компьютер* (ЦК).

В состав компьютера входят следующие устройства:

– устройство запоминания данных и команд, необходимые для их обработки, называемое *памятью*;

– устройство выполнения указанных в команде операции, называемое *процессором*;

– устройство управления работами при хранении и обработки данных, называемое *устройством управления*;

– пульт управления для первоначального включения и остановки компьютера, называемыми *пультом управления*.

Имеются три типа памяти: *сверхоперативная память*, *оперативная память*, *внешняя память*. Сверхоперативная память строится на регистрах с очень малой емкостью. Регистры работают очень быстро и используются для временного хранения и преобразования информации. Оперативная память предназначена для запоминания более постоянной по своей природе информации.

Регистры и оперативная память реализуются с помощью последовательности физических устройств, имеющих два устойчивых состояния. Эти состояния обозначаются цифрами 0 и 1 и названы словом **бит** – **bit** (образован от двух от английских слов **bi**nary digi**t**). Последовательность из восьми битов образует ячейку, называемую словом **байт** – **byte**. Поэтому в одном байте можно хранить всего $2^8 = 256$ различимых друг от друга видов информации. Каждый байт в оперативной памяти нумеруется. Номер байта называется *адресом*. В качестве значений адресов берутся положительные целые числа, начинающиеся с числа ноль.

Для выполнения программы ее команды и данные в основном должны размещаться в оперативной памяти. Для организации этого размещения несколько байтов объединяют в одно *слово*. Имеются два вида слова: *адресное слово*, *машинное слово*. В адресном слове размещается представление адреса из 0 и 1. С увеличением емкости адресного слова увеличивается допустимая емкость памяти компьютера. В машинном слове можно разместить необходимое для выполнения одной команды персональное данное или полученный после выполнения операции результат. То есть машинное слово является емкостной единицей, применяемой в рабочей единице компьютера. С увеличением емкости машинного слова увеличивается мошность компьютера.

На рисунке II.1.1.А показана связь между адресным словом и машинным словом в оперативной памяти.

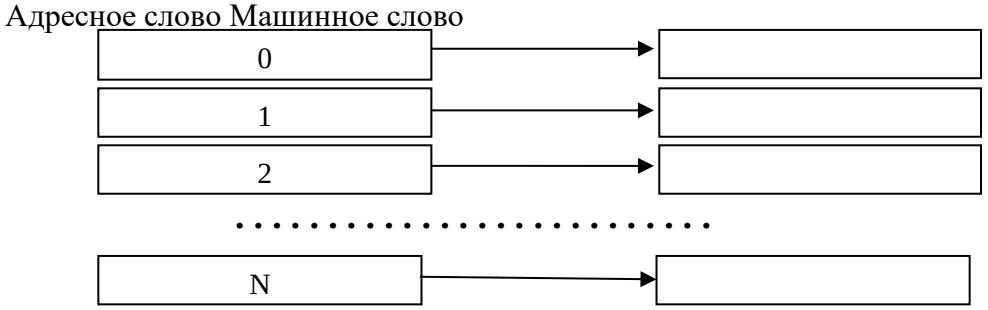


Рисунок II.1.1.А. Структура оперативной памяти.

Внешняя память предназначена для долговременного хранения данных. Данные во внешней памяти могут сохраняться даже после

завершения выполнения создавшей их программы, и впоследствии могут быть многократно использованы той же программой при повторных ее запусках или другими программами. Внешняя память используется также для хранения самих программ, когда они не выполняются, или отдельных частей и данных выполняющейся программы, которые не используются на данном этапе ее выполнения. Поскольку между внешней памятью и оперативной памятью прямой обмен есть, а прямой обмен между внешней памятью и регистрами невозможен, то активные части выполняемой программы и обрабатываемые ею на данном этапе данные должны обязательно быть размещены только в оперативной памяти. В основном внешняя память обладает теми же свойствами, что и оперативная память: свойство адресуемости, структуры данных и единицы измерения емкости.

Соотношения между единицами измерения емкости памяти показываются в таблице II.1.1:

Таблица II.1.1. Соотношения между единицами измерения емкости памяти.

1 килобайт	$= 1024^1$	$= 2^{10}$	= 1024 байт
1 мегабайт	$= 1024^2$	$= 2^{20}$	= 1 048 576 байт
1 гигабайт	$= 1024^3$	$= 2^{30}$	= 1 073 741 824 байт
1 терабайт	$= 1024^4$	$= 2^{40}$	= 1 099 511 627 776 байт
1 петабайт	$= 1024^5$	$= 2^{50}$	= 1 125 899 906 842 624 байт
1 эксабайт	$= 1024^6$	$= 2^{60}$	= 1 152 921 504 606 846 976 байт
1 зеттабайт	$= 1024^7$	$= 2^{70}$	= 1 180 591 620 717 411 303 424 байт
1 йоттабайт	$= 1024^8$	$= 2^{80}$	= 1 208 925 819 614 629 174 706 176 байт

Но внешняя память имеет совершенно иную физическую природу, для нее на физическом уровне применяются иные методы доступа, и этот доступ имеет другие временные характеристики. Это приводит к тому, что структуры данных и алгоритмы, эффективные для оперативной памяти, не оказываются таковыми для внешней

памяти. К внешней памяти относятся *жесткий диск*, *магнитный диск*, *лазерный диск* и др.

В соответствии с типами обрабатываемых данных имеются различные типы выполняемых операций. Поэтому в устройстве выполнения операций имеются соответствующие части для их выполнения. Кроме того, это устройство может иметь свойственные его части собственные регистры, в которых размещаются операнды и результаты операций, а также могут временно хранится команды программы и управляющая информация. Объединение устройства выполнения операций и устройства управления называется *центральным процессором*. Структура центрального процессора показана на рисунке II.1.1.В.

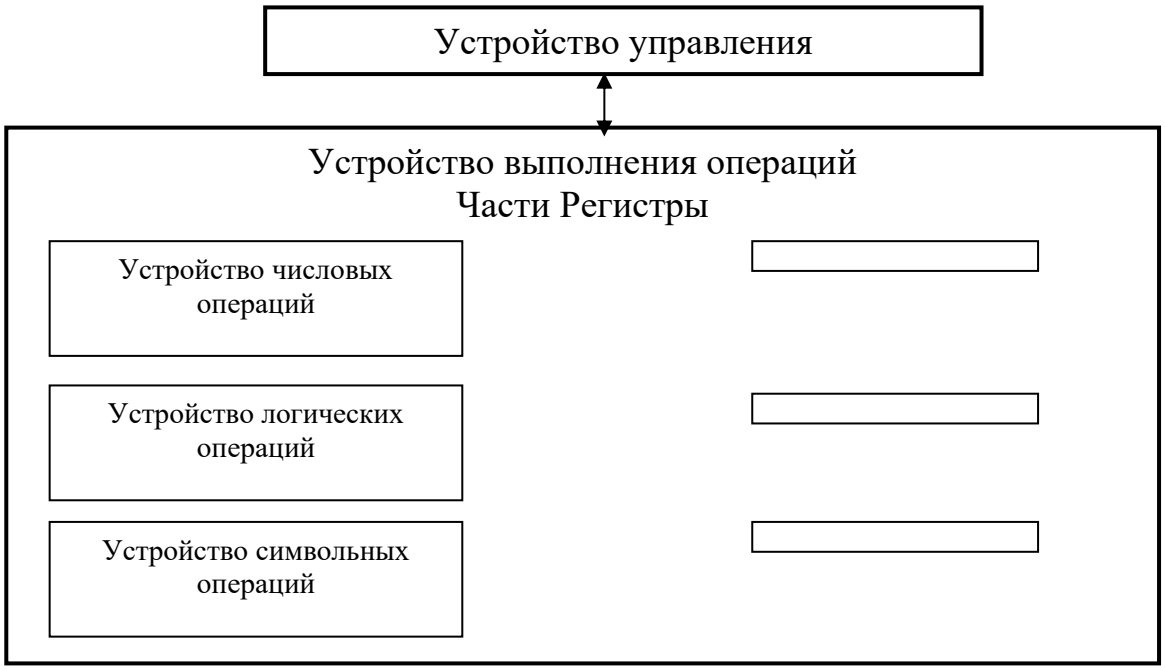


Рисунок II.1.1.В. Структура центрального процессора.

В соответствии с архитектурой компьютера центральный процессор имеет свою *разрядность*, под которой понимается объемное количество информации для выполнения одной операции, она равна емкости соответствующего регистра и измеряется в битах. Разрядность процессора всегда должна быть равна степени числа 2. Например, 2^3 битов = 8 битов, 2^4битов =16 битов, 2^5бтитов = 32 битов, 2^6бтитов = 64 битов и т.д.

В 1946 году впервые ученый США Джон фон Нейман опубликовал основные принципы создания универсальных ЦК, которые объединились в классическую архитектуру компьютера, она показана на рисунке II.1.1.С.

Рисунок II.1.1.С. Классическая архитектура компьютера.

Здесь тонкими стрелками показаны передачи управляющих сигналов, а двойными стрелками – передачи данных и команд.

Устройство ввода выполняет работу по вводу данных и программ с внешней памяти в оперативную память, а устройство вывода – по выводу полученных результатов при исполнении программы из оперативной памяти во внешнюю память.

Существует много внешних устройств не показанных в классической архитектуре компьютера. Эти устройства созданы для удобства работы человека с компьютером. Например, к ним можно отнести *монитор* (*устройство видения*), *принтер* (*устройство печати*), устройство работы с дисками и др. Иногда этих устройств называют *перифирийными устройствами* компьютера.

Примеры II.1.1.

1. Работа на компьютере состоит из этапов ввода информации, обработки введенной информации, вывода полученного результата.

2. Объем памяти всегда кратен восьми: 8 Мб, 16 Мб, 32 Мб,

3. 64 Мб, 128 Мб, 256 Мб, 512 Мб, 1024 Мб (1 Гб), 2048 Мб (2 Гб) и 4096 Мб (4 Гб) и т.д.

4. Обмен данными между процессором и оперативной памятью может производиться непосредственно или через регистр.

Задания II.1.1.

1. Вычислить возможный объем памяти компьютера, если его адресное слово состоит из трех байтов.

2. Определить разрядность процессора, если машинное слово состоит из четырех байтов.

3. Объясните следующие отношения между единицами емкости памяти:

1 килобайт	$= 1024^1$	$= 2^{10}$	= 1024 байт
1 мегабайт	$= 1024^2$	$= 2^{20}$	= 1 048 576 байт
1 гигабайт	$= 1024^3$	$= 2^{30}$	= 1 073 741 824 байт
1 терабайт	$= 1024^4$	$= 2^{40}$	= 1 099 511 627 776 байт
1 петабайт	$= 1024^5$	$= 2^{50}$	= 1 125 899 906 842 624 байт
1 эксабайт	$= 1024^6$	$= 2^{60}$	= 1 152 921 504 606 846 976 байт
1 зеттабайт	$= 1024^7$	$= 2^{70}$	= 1 180 591 620 717 411 303 424 байт
1 йоттабайт	$= 1024^8$	$= 2^{80}$	= 1 208 925 819 614 629 174 706 176 байт

Помощь

1. Надо учесть, что в одном байте можно записать $2^8 = 256$ отличных друг от друга видов информации, а у вас имеются три байта.

2. Разрядность процессора совпадает с числом битов в машинном слове.

3. Указаны в первом столбце - имя единицы, во втором столбце - степень 1024, в третьем столбце - степень 2, а в четвертом столбце - количество байтов в больших единицах.

Вопросы II.1.1.

1. Какие компьютеры имеются в зависимости от вида сообщения?
2. Что означает двойная стрелка в классической архитектуре компьютера?
3. В чем разница между адресным словом и машинным словом?

Тесты II.1.1.

1. Кто предложил принципы построения первого компьютера?
A) Ч. Бэббиж;
B) Джон фон Нейман;
C) В. Лейбниц;
D) А. Тьюринг;
E) В. Ондер.

2. Какие основные устройства имеются в компьютере?
A) оперативная память, процессор, устойство управления;
B) оперативная память, процессор, пульт управления;
C) оперативная память, процессор, принтер;
D) оперативная память, процессор, монитор;
E) процессор, принтер, клавиатура.

3. Что является самой маленькой единицей емкости памяти?
A) Килобит;
B) Бит;
C) Мегабит;
D) Гигабит;
E) Терабайт.

II.1.2. Принцип работы и рабочий цикл компьютера

Для построения компьютеров в соответствии с классической архитектурой фон-Нейман предложил принципы их работы:

1) *принцип адресности памяти* – память компьютера состоит из цепочек байтов и каждая цепочка байтов имеет свой адрес, а запись и чтение данных и команд осуществляются указанием адресов этих цепочек;

2) *принцип единства данных и команд* – команд, после записи их в память, можно обрабатывать как данные, т.е. применив к ним операции можно получить другие команды;

3) *принцип программного управления* – после записи программы в оперативную память вся работа компьютера управляется выполнением этой программы;

4) *принцип дискретности работы компьютера* – работа компьютера состоит из непустой и конечной последовательности команд.

Теперь опишем так называемый регистр, собственную память устройства выполнения операций, которая имеет отношение к спецификации рабочего цикла компьютера. Имеются несколько регистров. Поэтому их нумеруют или им дают имена в зависимости от их функции (услуг):

1) *регистр адреса команд* предназначен для хранения адреса выполняемой команды и имеет емкость, равную адресному слову;

2) *регистр команд* предназначен для хранения самой выполняемой команды и имеет емкость равную машинному слову;

3) *регистр исходных и результирующих данных* предназначен для хранения значений исходных данных или значений результата, имеет емкость равную машинному слову;

4) *регистр признака результата* предназначен для записи 0 или 1 в зависимости от свойства результата, полученного после выполнения операции (например, если результат отрицательный, то 0 иначе 1) имеет емкость равную одному биту;

5) *регистр особого случая* предназначен для хранения знака «особого случая», появляющегося во время выполнения команды, и имеет емкость один бит.

Компьютер выполняет каждый свой шаг в соответствии со своим рабочим циклом.

Рабочим циклом компьютера называется работа, возникающая при выполнении одной команды в заданной программе. Для того, чтобы компьютер выполнил полностью эту программу, он должен несколько раз повторить свой рабочий цикл.

Ниже дается *рабочий цикл компьютера* в классической структуре. Для этого рассматриваются следующие случаи:

1. Если выполняемая программа и ей нужные данные записаны в оперативной памяти (ОП), то рабочий цикл нужно начать с помощью пульта управления, загрузив регистр адреса команд (РАК) адресом самой первой команды программы, и запустить устройство управления (УУ). Для прочтения этой команды УУ отправит сигнал ОП, затем, прочитанную команду отправит в регистр команд (РК). Далее УУ определяет какую операцию или какое указание нужно выполнить по этой команде.

2. Если по команде выполняется операция, то для этой операции определяются адреса необходимых данных. Для чтения по этим адресам данных из ОП и записи их в регистр исходных и результирующих данных (РИРД) отправляется управляющий сигнал. Затем УУ для выполнения команды и для записи, полученного от неё результата. в РИРД отправляет управляющий сигнал в устройство выполнения операции (УВО) и записывает адрес следующей выполняемой команды в РАК. После выполнения команды УУ, в зависимости от полученного результата, запишет значения в регистр признака результатов (РПР) и в регистр особого случая (РОС) и переместит значение результата из РИРД в ОП.

3. Если по команде нужно будет выполнить указание, то УУ само выполняет это указание.

Имеются три вида указаний: *стоп*, *передача безусловного перехода* и *передача условного перехода*.

При выполнении указания стоп вся работа компьютера останавливается.

В указании передачи безусловного перехода указывается адрес следующей выполняемой команды. Поэтому при выполнении указания передачи безусловного перехода УУ записывает в РАР, указанный в этой команде, адрес следующей выполняемой команды.

В указании передачи условного перехода, адрес следующей выполняемой команды будет зависить от верности некоторого условия (например, если условие верно, то следующая выполняемая команда находится по указанному в этой команде адресу). Обычно условие строится в зависимости от результата ранее выполненных команд. Поэтому, при выполнении указания передачи условного перехода, УУ сравнивает определенное значение со значением из РИРД или из РОС. Если они равны, то в РАР записывается адрес команды, которая следует в памяти за этой выполняемой в данный момент командой.

В заключении обсудим свойства основного устойства процессора, который имеет прямое отношение к работе компьютера. Ими являются *частота процессора*, *скорость процессора*, *производительность процессора*.

Частотой процессора называется количество выполняемых тактов (*работа при выполнении одной команды*) за одну секунду. *Частота процессора* измеряется в *герцах* (*гц*), *мегагерцах* (*Мгц*), *гигагерцах* (*Ггц*). Например, в 80-х годах XX века у самого передового 32-х разрядного процессора Intel 80386 частота не превышала 25 Мгц, а затем у процессора Intel Pentium IV частота достигла 1,70 Ггц, т.е. этот показатель вырос почти в тысячу раз.

Скоростью процессора называется количество выполняемых операций за одну секунду, она измеряется в *операция/секунд* (*о/с*). Например, созданные в 50-х годах XX века компьютеры за одну секунду выполняли несколько тысяч операций (Стрела – 3000 *о/с*), в 70-х этого века компьютеры семейства IBM 360 и ЕС ЭВМ за одну секунду выполняли несколько сот тысяч операций, первый суперкомпьютер Cray–1 имел скорость 100 млн *о/с*, а сейчас

некоторые компьютеры за одну секунду могут выполнять несколько сотен миллиардов операций, т.е. за полвека скорость компьютеров выросла в несколько миллионов раз. Скорости будущих нанокомпьютеров и нейрокомпьютеров еще будут расти.

Производительность процессора будет зависеть от класса (научно-технические, экономические и др.) решаемых задач. Для разных задач веса операций будут разными. Веса операций для научно-технических задач показаны в таблице II.1.1.

Таблица II.1.1. Веса научно–технических задач.

№	Название операции	Вес
1	Сложение и вычитание для действительных с фиксированной точкой	33
2	Умножение для действительных с фиксированной точкой	0,6
3	Деление для действительных с фиксированной точкой	0,2
4	Сложение и вычитание для действительных с плавающей точкой	7,3
5	Умножение для действительных с плавающей точкой	4,0
6	Деление для действительных с плавающей точкой	1,6
7	Логические операции инверсия	1,7
8	Логические операции конъюнкция	1,7
9	Логические операции, дизъюнкция	1,7
10	Без условный переход	17,5
11	Сравнение	4,0
12	Условный переход	6,5
13	Индексирование	19,0
14	Сдвиг на 8 разрядов	4,6

Производительность процессора вычисляется с помощью метода Гибсон:

$$P = \frac{\sum_{i=1}^{n} p_i}{\sum_{i=1}^{n} p_i t_i},$$

где p_i – частота появления (вес) i-ой операции в динамике вычисления, t_i – время выполнения i-ой операции.

Для оценки производительности компьютеров используются единицы, произошедшие от слово флопс - FLOPS (Floating-point Operations Per Second), являющейся внесистемной единицей, используемой для измерения производительности компьютеров, показывающая, сколько операций с плавающей точкой в секунду выполняет данный компьютер. Их список приведен в таблице II.1.2.

Таблица II.1.2. Единицы измерения производительности.

№	Название измерения	Год использования	Степень 10	Числовое название
1.	Флопс	1941	10^0	Один
2.	Килофлопс	1949	10^3	Тысяча
3.	Мегафлопс	1964	10^6	Миллион
4.	Гигафлопс	1987	10^9	Биллион
5.	Терафлопс	1997	10^{12}	Триллион
6.	Петафлопс	2008	10^{15}	Квадриллион
7.	Эксафлопс	После 2019	10^{18}	Квинтиллион

Примеры II.1.2.

1. В конце 60-х годов XX века самым быстродействующим в мире компьютером считали советскую электронную вычислительную машину БЭСМ-6, её быстродействие достигало до 1 миллиона операций в секунду, т.е. 1 Мегафлопс.

2. В начале 2000-х годов производительность персонального компьютера с процессором Intel Pentium доходила до 50 миллионов операций в секунду, т.е. 50 Мегафлопс.

3. В конце 2000-х годов самым быстрым в мире компьютером являлся китайский суперкомпьютер Tianhe-2 с производительностью 30.7 петафлопс.

Задания II.1.2.

1. Перечислите принципы работы компьютера.
2. Назовите виды регистров и опишите их назначения.
3. Опишите рабочий цикл компьютера.

Помощь

1. Эти принципы касаются основных устройств, данных и команд компьютера.
2. Регистры имеют большую скорость работы и малую емкость, обеспечивая связь между основными устройствами компьютера.
3. Рабочий цикл состоит из тактов.

Вопросы II.1.2.

1. Что такое частота процессора?
2. Что такое скорость процессора?
3. Что такое производительность компьютера?

Тесты II.1.2.

1. В чем измеряется частота процессора?
A) герц;
B) байт;
C) флопс;
D) секунда;
E) команда.

2. В чем измеряется скорость процессора?
A) операция/секунд;
B) указание/секунд;
C) командо/секунд;
D) программо/секунд;
E) байт/секунд.

3. В чем измеряется производительность компьютера?
A) флопс;
B) герц;
C) байт;
D) секунда;
E) команда.

II.1.4. Поколение компьютеров

Одним из способов классификации компьютеров – это разделение компьютеров на поколение. Вообще, условие разделения компьютеров на поколение зависит в основном от изменения их элементной базы, от изменения видов и свойств входящих в его состав устройств и от изменения программного обеспечения в соответствии с появлением новых групп задач, решаемых с помощью компьютеров. Сейчас можно говорить, что известно шесть поколений компьютеров.

Первое поколение компьютеров (1948 – 1958 гг.). В этом поколении конструктивными элементами устройства управления компьютеров были электронные лампы, их скорость десятки тысяч *о/с*., разрядность 31–43 бит, объем оперативной памяти 1–4 Кб., рабочий цикл операций строго последовательный, т.е. следующая выполняемая операция начиналась только после завершения выполнения текущей операции во время выполнения операции «ввод/вывод» центральный процессор простаивает. Программы записывались на машинном языке и в основном выполнялись в ручную. Рабочий режим был открытым: каждый программист, сидя за пультом управления, сам вводил свою программу и заставлял работать. В основном решались научно-технические задачи, связанные с числовыми величинами, но использование символьных величин не было. Начали создаваться стандартные программы. Веса этих компьютеров были 5–30 тонн и из-за того, что объемы были очень большими, для их размещения требовались отдельные большие помещения и здания. Самым первым компьютером, который полностью реализовал принципы классической архитектуры компьютеров был «EDSAC», созданный в 1949 году в университете Кембридж (Англия). Через год в 1950 году в США появился универсальный компьютер «EDVAC». В бывшем Советском Союзе самым первым компьютером был «МЭСМ», который создавался 1947-1951 годы, а в 1952–1953 годы создался следующий компьютер «БЭСМ–1». Самым первым вышедшим на

свободный рынок компьютером был UNIVAC, который появился в 1951 году в США.

Второе поколение компьютеров (1955 – 1967 гг.). Компьютеры этого поколения были транзисторными, имели скорость сотни тысячи о/с., разрядность 31–48 бит, объем оперативной памяти 8–128 Кб. Появилась система прерывания и обработки работы процессора (она в основном включалась во время выполнения операции ввода/вывода). Появились языки ассемблера, которые превратили двоичные коды машинного языка в мнемонические коды. Также появились ассемблеры, трансляторы –программы автоматического перевода программ с алгоритмических языков на машинный язык, т.е. для их составления стали использовать языки программирования высокого уровня:

Fortran (1957) США – для решения численных задач;

Cobol (1958) США – для решения экономических задач;

Lisp (1958) США – для решения задач символьной обработки и задач принятия решений;

Algol (1958–1960) IFIP – для решения научно–технических задач;

PL/1 (1960) США – универсальный язык программирования.

Кроме того, увеличился фонд стандартных программ, применялись закрытые режимы работы, т.е. программисты не стали непосредственно работать с машиной, а стали передавать свои программы на языке высого уровня группе оказания услуг по проведению программы через машины. Для контроля и управления работой программы появились первые мониторные системы, они имели собственные языки управления заданиями. Появление индексной арифметики, непрямая адресация, использование динамической памяти и возможность работы с символьными величинами стали подчеркивать конструктивные особенности этого поколения. К этим компьютером относятся в США - UNIVAC, LARC, CDC 6600, IBM-1401, -7030, а бывшем СССР - Урал-11, -14, -16, Минск-12, -14, -22, М-220, -222, БЭСМ-2, -4,-6, Мир-1, их

оперативные памяти имели объемы от 32 Кб до 64 Кб, а скорости достигали 20–30 тыс *о/с*. Среди них самым быстрым компьютером был БЭСМ-6, его скорость доходила до 1 млн *о/с*.

Третье поколение компьютеров (1968 – 1973 гг.). К этому поколению относятся компьютеры и компьютерные комплексы, созданные на интегральных микросхемах. Их скорость миллионы *о/с*., разрядность 32–64 бит, объем оперативной памяти 64–1024 Кб, скорости достигали до 10 млн *о/с*. Были развитые системы прерывания для параллельного выполнения операций ввода/вывода и операций центрального процессора применялись дополнительные процессоры (каналы). Многие работы, ранее выполняемые программой, в том числе организация и обработка прерывания, стали реализовываться аппаратно. Появились сенсорные установки для того, чтобы компьютеры могли воспринимать внешнюю среду и влиять на неё. Все это превратило компьютер из детерминированно (однозначно) обрабатывающего устройства заранее введенных данных в интеллектуальное устройство, способное работать в зависимости от возникновения состояний внешней среды. Реализованы защита и динамическое распределение пар памяти. К многим задачам стали применяться проблемно-ориентированные языки программирования высокого уровня, в том числе символьным задачам языки Snobol, Lisp, Refal и логическим задачам языки Prolog, Miranda. Увеличились доли символьных задач и логических задач. Начали функционировать развитые операционные системы, управляющие всеми работами (умеющие целенаправленно отвечать на внутренние и внешние ситуации) программ.

Основной особеностью этого поколения является появление компьютерных комплексов, состоящих из нескольких однотипных с возрастающими возможностями моделей, программы которых снизу вверх совместимые. Например, едийные системы компьютеров в СССР ЕС ЭВМ 1020–1050, а в США IBM/360–370. С помощью этих компьютеров появилась возможность по созданию вычислительной системы, превращающей поля оперативной памяти и внешних

устройств общедоступными. Реализован режим мультипрограммирования, разбивающего время центрального процессора так, чтобы одновременно могли работать несколько программ. Кроме того, появились программы, работающие в реальном масштабе времени. Они позволяют управлять технологическими процессами, полетом летательных аппаратов и работой других сложных устройств. В 1969 году появилась первая глобальная компьютерная сеть – она стала зачатком глобальной компьютерной сети, называемой сейчас Интернет. В то время одновременно появились операционная система Unix и язык программирования С («Си»). Они сыграли огромную роль в развитии программного мира и до сих пор сохраняют свои лидерские позиции.

Четвертое поколение компьютеров (1974 – 1982 гг.) связывают с появлением микропроцессора, созданного в 1971 году фирмой Intel (США) на основе больших интегральных электронных схем. Компьютеры этого поколения спроектированы для эффективного использования языков программирования высокого уровня и облегчения процесса программирования для проблемных программистов. Они предназначены в основном для быстрого повышения производительности труда в науке, производстве, управлении, здравохранении, сфере услуг и быту.

С аппаратной стороны они созданы с помощью больших или сверх больших интегрированных микросхем. Большая степень интеграции привела к уплотнению размешения электронных аппаратур, а это, в свою очередь, привело к уменьшению объема, увеличению быстродействия и удешевлению стоимости компьютеров. Все это значительно повлияло на логическую структуру и программное обеспечение компьютеров. Началось утеснение связи между логической структурой и программным обеспечением компьютера, программы операционной системы без участия человека организуют непрерывную работу компьютера. К этим компьютерам относятся многопроцессорные

суперкомпьютеры и *микрокомпьютеры* (позже они стали называться *персональным компьютером*). Скорости суперкомпьютеров достигают до миллионов *о/с.* Например, скорость суперкомпьютера Cray-1 достигла 100 млн *о/с*. Вообще с помощью компьютеров этого поколения методы связи получили дальнейшее развитие, соединив с линией телефона, телеграфа построены *компьютерные локальные, корпоративные и глобальные сети* (например, Интернет). Собраны очень большие архивы данных, развилось задание и обработка данных в визуальном виде, широко реализованы системы реального времени. К этому поколению в СССР относятся следующие компьютеры: ЕС-1015, -1025, -1035, -1045, -1055, -1065, -1036, -1046, -1066, СМ-1420, -1600, -1700, все персональные компьютеры (“Электроника МС 0501”, “Электроника-85”, “Искра-226”, ЕС-1840, -1841, -1842 и др.). Кроме того, к этому поколению относятся очень мощные многопроцессорные вычислительные комплексы "Эльбрус-1, -2, 3" и суперкомпьютеры Amdahl 470V16, работающие на основе конвейерной обработки команд эффективные принципы параллельности. Позже к группе суперкомпьютеров стали включать модели с быстродействием 20 мегафлопс (1 мегафлопс = 1 млн операции с плавающей точкой /секунд). Первой такой моделью считается созданная в США в 1975 году ILLIAC-IV, её быстодействие достигало до 50 мегафлопс. В истории суперкомпьютеров особое место занимает созданная в США в 1976 году с быстродействием 130 мегафлопс модель Cray-1. Архитектура модели основана на сверхбольших интегральных схемах, обработка данных основана на принципе векторной и скалярной обработки. Следующие модели Cray-2, Cray X-MP, Cray-3 и Cray-4 имели быстродействие до 10 тыс мегафлопс, а модель Cray MP на новой кремниевой микросхеме содержала 64 процессоров, быстродействие доходило до 50 гигафлопс.

В нашей стране к четвертому поколению относятся персональные компьютеры, которые будут рассмотрены в II.1.5.

Пятое поколение компьютеров (1982 – 1992 гг.) созданы на основе сверхбольших, ультрабольших, гигабольших интегральных схем. Это поколение началось с заявленного Японией в 1980 году десятилетнего проекта, где язык логического программирования Prolog в качестве машинного языка был реализован аппаратно и была создана система искусственного интеллекта. Для решения этих пробем предлагались следующие направления:

1. Разработать простой интерфейс, позволяющий пользователю проводить диалог с компьютером для решения своей задачи. Такой интерфейс может быть организован двумя способами: на естественном языке и графическом языке. Однако доступный интерфейс решил только половину проблемы. Потому что он не участвовал при составлении программного обеспечения, а позволял общение только с заранее готовыми.
2. Привлечь конечного пользователя в проектировании программного продукта, в результате это позволит сократить время на разработку и повысить его качество. Такая технология касательно автоформализации профессионального знания конечного пользователя предлагает два этапа проектирования программного продукта: программист готовит универсальную «пустую» оболочку, пользователь наполняет его реальным знанием и пользуясь этим решает свои практические задачи.

В разработке искусственного интеллекта в качестве одного основного составителя можно взять базу знаний в различных направлениях науки и техники. Для создания и использования базы знаний нужно высокое быстродействие вычислительной системы и большой объем памяти. Для создания программы заполнения, обработки и обеспечения обновлений базы знаний были разработаны специальные объектно-ориентированные и логические языки программирования. Структуры этих языков требуют, чтобы с традиционной фон-неймановской архитектуры компьютеров перейти на новую архитектуру. Этот проект закончился безрезультатно. Однако, сейчас есть компьютеры со своим искусственным интеллектом, т.е. самостоятельно умеющие по

заданной постановке задачи находить ее решение, строить утверждения и доказывать их, сочинять на определенную тему стихи или музыку и выполнять другие интеллектуальные работы. Но они широко не распространены, так как их стоимость очень велика и для работы с ними требуется высокая квалификация в области информационных технологий. Тем не менее, к пятому поколению относится мультимедийный компьютер, построенный на основе процессора Pentium, позволяющий создать виртуальную среду. Он имеет бустродействие 100 млн *о/с* и суперкомпьютер со скоростью 120 млн *о/с*, такие как Cray, Эльбрус.

Шестое поколение компьютеров – началось в середине 90-х годов XX века. Эти компьютеры создаются на основе *нейронных сетей, мнозначной логики* и *нечеткой логики*, теории *квантового вычисленияи и генетических алгоритмов*. У этих компьютеров будет развитый искусственный интеллект: способность самообучаться и самостоятельно понять некоторые задачи (распознавать образы), их проектировать, решить или реализовать нужную программу или же конструрировать устройство.

Примеры II.1.4:

1. Самый первый компьютер ЭНИАК (ENIAC) создан в 1945 году в США, имеет память 20 число–слово. Вычислительная мощность: за 1 секунду 357 операции умножения или 5000 операции сложения, вес 27 тонн.

2. PDP-8 – первый успешный коммерческий миникомпьютер второго поколения, производившийся в 1960-х. Было продано более 50 тысяч штук, самое большое количество для того времени.

3. IBM PC-совместимые компьютеры наиболее распространены в Казахстане, их мощность постоянно увеличивается, а область применения расширяется. Они могут объединяться в сети, что позволяет пользователям обмениваться информацией и одновременно получать доступ к общим базам данных.

4. Компьютер между вторым и третьем поколнении БЭСМ-6 создан в 1965 году в СССР, память 128 Кб, скорость1 млн о/сек.


Компьютер БЭСМ-6

5. Компьютер четвертого поколения Эльбрус-2 создан в 1985 году в СССР, скорость 125 млн о/с (10 процессор), память 144 МБайт, машинное слово 72 бит, канал ввода/вывода 120 Мб/с.

1. Компьютер третьего поколения IBM 360 создан в 1964 году в США, процессор 32-разрядный, тактовая частота процессора доходила до 5MHz, память 16Mb.

2. Компьютер пятого поколения PIM/m-1 один из немногих, увидевших свет, создан в Японии.

Задания II.1.4:

1. Назовите элементную базу компьютеров первого поколения.
2. Назовите программное обеспечение компьютеров второго поколения.
3. Назовите особенности компьютеров пятого поколения.

Вопросы II.1.4:

1. Чем отличаются элементные базы в разных поколениях компьютеров?
2. В насточщее время сколько поколений компьютеров известно?
3. Какие технические характеристики имеются у шестого поколения компьютеров?

Тесты II.1.4.:

1. К какому поколению относятся персональные компьютеры типа IBM PC?

A) Четвертое;
B) Пятое;
C) Второе;
D) Первое;
E) Третье.

2. Сколько поколнений компьютеров имеются в настоящее время?

A) Шесть;
B) Восемь;
C) Четыре;
D) Девять;
E) Десять.

3. К какому поколению относится компьютер iMac, созданный в США?

A) Четвертое;
B) Первое;
C) Второе;
D) Третье;
E) Пятое.

II.1.5. Персональные компьютеры

Выше представлена классическая архитектура, свойственная первым трем поколениям компьютеров. Естественно, что в эпоху, когда компьютерная техника развивается бурными темпами, эта архитектура не могла не измениться.

Как сказано выше, третье поколение компьютеров связано с переходом физической основы с транзисторов на интегральные электронные микросхемы. Сильное уменьшение размеров электронных схем привело к существенному уменьшению объемов основных устройств компьютеров и повлияло на появление процессоров с очень высоким быстродействием. Возникло большое противоречие между очень быстродействующими основными устройствами и медленно работающими устройствами ввода/вывода. Процессор, управляющий работами внешних устройств, много времени простаивает, ожидая прихода внешних данных или вывода результата наружу. Для решения этой проблемы принято освободить процессор от управления общения с внешним миром, в результате эта работа была поручена специально изготовленным устройствам, называемыми процессорами внешних устройств. У этих устройств несколько названий: канал связи, процессор ввода/вывода, контроллер.

Контроллер имеет свою систему команд и управляет работой переданной ему внешних устройств с помощью программ, составленных из этих команд. Результат такой программы записывается в регистр контроллера так, чтобы его мог прочитать процессор. Когда выполняется программа, решающая поставленную задачу, потребуется общение с внешним миром. Центральный процессор об этом дает задание контроллеру и продолжает свою работу, связанную с выполнением программы, а работу по обмену с внешним миром управляет сам контроллер. Вся связь в компьютере осуществляется через общую шину. Шина имеет три части: *шина передачи данных*, *шина передачи адресов*, *шина передачи управления*.

На рисунке II.1.5 показана архитектура персонального компьютера.

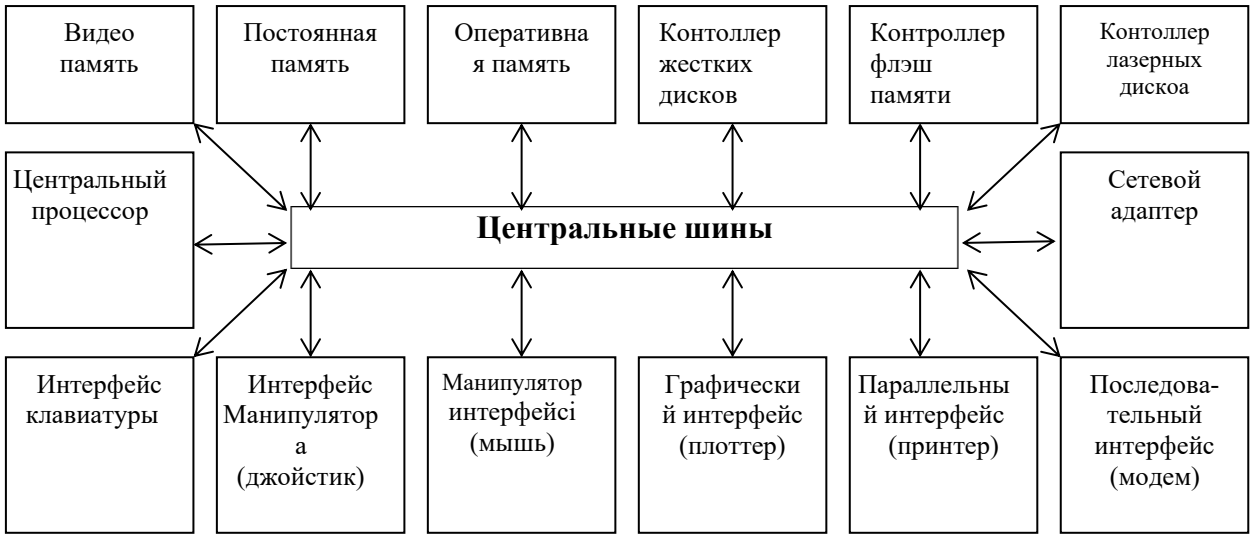


Рисунок II.1.5. Архитектура персонального компьютера.

Архитектура, представленная выше называется открытой. Поэтому, добавляя к ней только нужное вам устройство можно оптимизировать или добавляя новые устройства можно усиливать её возможности.

В связи с увеличением потоков данных, потребовалось несколько шин. Один из них внедрен для очень быстродействующих устройств, а остальные – для медленнодействующих устройств.

Для вывода данных создано специальное устройство – монитор. Его принцип работы, как у обычного телевизора. Поскольку монитор работает очень быстро, то для него нужна отдельная память, называемая видео памятью.

Итак открытая архитектура персональных компьютеров повлияла на постоянный рост количества и качества внешних устройств и создание новых с высокими возможностями основных устройств. В таких компьютерах кроме центрального процессора начали работать несколько специальных процессоров (математические, видео, символьные и др.), стали развиваться связи между компьютерами и появились различные компьютерные сети (локальные, корпоративные, глобальные).

Персональные компьютеры строятся на основе универсальных

микропроцессоров. В 1974 году разработан первый универсальный 4500 элементный микропроцессор Intel–8080 и стандарт микрокомпьютерной технологии. Они стали основами создания первого персонального компьютера. На основе Intel–8080 в 1974 году Эдвард Робертс создал самый первый переснальный компьютер (ПК) и назвал его Altair–808. Для этого компьютера Пол Аллен и Бил Гейтс разработали транслятор с широко распространенного простого языка программирования Basic, это значительно расширило интеллектуальность компьютера. После этого ПК Altair–8800 стали создавать более 20-ти различные компании и фирмы, появилась ПК-индустрия (производство ПК, их продажа, сервисные центры, периодические издания, выставки, конференции и т.д.). В 1977 году поставлены на серийное производство 3 модели ПК: Apple–2 (фирма Apple Computers), TRS–80 (фирмаTandy Radio Shark) и PET (фирма Commodore). Среди них в конкурентной борьбе в производстве ПК лидером стала фирма Apple Computers, в 1980 году её годовой доход дошел до $117 млн.

Начиная с 1981 года, фирма IBM стала выпускать широко распространенные ПК IBM PC/XT/AT и PS/2 и открыли главную эру ПК. В связи с этим очень важные для пользователей проблемы как стандартизация, унификация, развитое программное обеспечение и другие стали оптимально решаться.

Поскольку интерфейсы ПК очень развиты, то с ними работать удобно и эффективно. А производство ПК большими объемами влияет на постоянное снижение их стоимости. Поэтому их применение с экономической точки зрения очень выгодно.

К основным устройствам ПК относятся системный блок, монитор, клавиатура, мышка, а к дополнительным устройствам – принтер, сканер, микрофон и др.

Системный блок состоит из системной (материнской) платы, процессора, оперативной памяти, жесткого диска, блока питания, видеокарты, дисковода и других важных устройств, в его задней

части расположены порты для соединения монитора, клавиатуры, мышки, принтера, модема, сканера, микрофона и др.

Монитор является устойством с экраном для вывода данных, его внешний вид похож на обычный телевизор.

Клавиатура является устройством, которое предоставляет возможность управлять работой компьютера и вводить необходимую информацию, т.е. она позволяет с помощью функциональных кнопок управлять, а клавишей букв, цифр и других знаков вводить в компьютер любую информацию.

Мышка является устройством, которое предоставляет возможность поиска, выбора, отметки и управлять через них работой компьютера.

Принтер является устройством, которое предоставляет возможность распечатки информации на бумаге, есть матричные, струйные, лазерные, сетодиодные и други виды принтеров.

Сканер является устройством, которое предоставляет возможность автоматически ввести в компьютер изображение на бумаге текста и рисунка, т.е. он преобразует изображение в двоичный вид и записывает в память компьютера.

Примеры II.1.5:

1. Системный блок ПК Altair–8080, имеет оперативную память 256 байт, возможность соединения внешних устройств

2. Вид персонального компьютера IBM PC

3. Вид ПК с плоским экраном, клавиатурой, динамиком и мышкой

4. Вид переносимого ПК (Ноутбук – Notebook)

5. Вид планшетного ПК

Задания II.1.5.

1. Нарисуйте логическую структуру персонального компьютера.

Помощь. Нужно учесть, что главным элементом является центральная шина, а остальные элементы будут связаны с ней.

Вопросы II.1.5:

Что относится к внешним устройствам персонального компьютера?

1. Как называется электронная схема, которая выполняет все вычисления?

2. В каком году был создан первый персональный компьютер в мире?

Тесты II.1.5.

1. Какие различные по функции части есть у шины персонального компьютера?

A) Шина данных, шина адресов, шина управления;

B) Шина команд, шина адресов, шина управления;

C) Шина данных, шина алгоритмов, шина управления;

D) Шина данных, шина адресов, шина указаний;

E) Шина управления, шина методов, шина команд.

2. Какой из них самый первый созданный персональный компьютер?

A) Altair–808;

B) Apple;

C) IBM PC XT;

D) Macintosh;

E) Yamaha.

3. Что относится к основным устройствам персонального компьютера?

A) системный блок;

B) принтер;

C) сканер;

D) модем;

E) плоттер.

II.2. Представление информации в компьютере

II.2.1. Представления символов в компьютере

Информацию перед обработкой на компьютере нужно сохранить в её памяти, где она запишется в виде последовательности, состоящей только из 0 и 1.

Представление информации в виде последовательности из 0 и 1 называют *кодом*. Прямое соответствие между информацией и кодом называют *кодированием*, а ему обратное соответствие – *декодированием* (они одинаковы с понятиями прямая функция и обратная функция в математике). Следовательно, любая вводимая в компьютер информация должна *кодироваться*, а извлекаемая из компьютера информация для использования человеком – *декодироваться* (обратное кодирование), т.е. преобразовываться в символьный или графический вид. Получение прямого и обратного соответствий называют *правилом кодирования*.

Представления любых составных данных в памяти компьютера образуются из представлений их составляющих простых данных. Из блока I.2 известно, что простые данные подразделяются на *логические*, *символьные* и *числовые*. Поэтому достаточно рассмотреть способы представления этих простых данных.

Логические данные представляются в памяти компьютера с помощью одного бита. Из урока II.1.1 известно, что его значением может быть только 0 или 1, здесь 0 соответствует логической ложности, а 1 – логической истинности.

Символьные данные представляются в памяти компьютера с помощью последовательности образованных из 0 и 1 кодов, входящих в их состав символов. Код каждого символа задается значениями индексов так называемой *кодовой таблицей* двумерного массива, содержащего этот символ. В кодовой таблице порядок кодирования называют *стандартом кодирования*. То есть каждый стандарт определяет свою кодовую таблицу. К таким стандартам относятся широко распространенные ASCII, ANSI, Unicode и

другие. В стандарте ASCII каждый символ занимает 8 бит, т.е. 1 байт. Из-за того, что ASCII был предназначен для западных языков, его использование было ограничено в странах и регионах, чьи языки содержали символы, не включенные в $2^8 = 256$ символов, поддерживаемых ASCII.

Чтобы обойти это ограничение, Международная организация по стандартизации (ISO - International Standards Organization) создала новый стандарт кодировки символов, названный Latin-1, который содержал европейские символы, не вошедшие в набор ASCII. Microsoft расширила Latin-1 и назвала этот стандарт ANSI. Но ANSI по-прежнему осталась 8-битной кодировкой, которая может представлять только 256 уникальных символов. Многие языки имеют тысячи символов, особенно языки, такие как китайский, корейский и японский.

Для преодоления ограничений стандарта 8-битовой кодировки символов, Microsoft в сотрудничестве с такими компаниями, как Apple Computer, Inc., и IBM, создала некоммерческий консорциум Unicode целью которого стало определение нового стандарта на кодировку символов для международных наборов символов. Работа, проделанная в Unicode, была объединена с работой, которая велась в ISO, и результатом стал стандарт Unicode для кодировки символов.

Unicode является 16-битным стандартом, что дает возможность кодировки $2^{16} = 65\,536$ разных друг от друга символов. Поэтому с помощью этого стандарта можно кодировать символы алфавита любого национального язык. Он поддерживает даже архаические языки, такие как древнетюркские руники, санскрит и египетские иероглифы, и включает знаки препинания, математические и графические символы. В его таблице кодирования имеются коды применяющихся в различных отраслях науки знаков и различных декоративных знаков.

Первая версия Unicode (1991 г.) представляла собой 16-битную кодировку с фиксированной шириной символа. Во второй версии Unicode (1996 г.) было решено значительно расширить кодовую

область; для сохранения совместимости с теми системами, где уже был реализован 16-битный Unicode, и был создан UTF (Unicode Transformation Format)-16, являющийся одним из способов кодирования символов из Unicode в виде последовательности 16-битных слов. Данная кодировка позволяет записывать символы Unicode. Один символ кодировки UTF-16 представлен последовательностью двух байтов. Который из двух идёт впереди, старший или младший, зависит от порядка байтов. Для определения порядка байтов используется метка порядка байтов.

На основе этих стандартов можно разработать национальные стандарты, касающиеся знаков алфавита каждого национального языка, в том числе и казахского языка. Эти стандарты должны обеспечить создание, прием, обработку, печать и передачу казахских текстов в офисных программах, издательских системах и графических пакетах, а также и в среде Internet при создании Web страниц и использовании электронной почты и т.д.

Для букв казахского алфавита 8-битовая кодировка для операционной системы MS DOS показана в таблице II.2.1.А.

Таблица II.2.1.А. 8-битовая кодировка казахских букв для MS DOS

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0		У		П		Р		А		В		Л		Я		Ю		Щ		И		Е			С	И	М	В	О	Л	Ы		1
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	3
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_	5
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	⌂	7
8	А	Б	В	Г	Д	Е	Ж	З	И	Й	К	Л	М	Н	О	П	Р	С	Т	У	Ф	Х	Ц	Ч	Ш	Щ	Ъ	Ы	Ь	Э	Ю	Я	9
A	а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п	▒	▒	▓	│	┤	╡	╢	╖	╕	╣	║	╗	╝	╜	╛	┐	B
C	└	┴	┬	├	─	┼	╞	╟	╚	╔	╩	╦	╠	═	╬	╧	╨	╤	╥	╙	╘	╒	╓	╫	╪	┘	┌	▒	▒	I	i	h	D
E	р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я	Ө	Ғ	Қ	Ң	Ө	Ұ	Ү	h	ә	ғ	қ	ң	ө	ұ	ү		F

В стандарте ANSI коды для букв казахского алфавита должны размещаться только во второй части 8-битовой таблицы кодировки, начиная со строки 8 до строки 15 ($15_{10} = F_{16}$), так как в первой части

этой таблицы до 8-й строки размещены коды букв английского алфавита, арабских цифр, управляющих и других обязательных символов. Кодировка казахских букв для Windows показана в таблице II.2.1.B.

Таблица II.2.1.B. 8-битовая кодировка казахских букв для Windows

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
8	128	129	‚ 130	131	„ 132	… 133	† 134	135	136	‰ 137	138	‹ 139	140	Қ 141	Һ 142	143
9	144	‘ 145	’ 146	“ 147	” 148	• 149	– 150	— 151	152	™ 153	154	› 155	156	қ 157	һ 158	159
A	160	Ұ 161	ұ 162	Ә 163	¤ 164	Ө 165	166	§ 167	168	© 169	Ғ 170	171	172	173	® 174	Ү 175
B	176	± 177	І 178	і 179	ө 180	µ 181	¶ 182	183	184	№ 185	ғ 186	187	ә 188	Ң 189	ң 190	ү 191
C	А 192	Б 193	В 194	Г 195	Д 196	Е 197	Ж 198	З 199	И 200	Й 201	К 202	Л 203	М 204	Н 205	О 206	П 207
D	Р 208	С 209	Т 210	У 211	Ф 212	Х 213	Ц 214	Ч 215	Ш 216	Щ 217	Ъ 218	Ы 219	Ь 220	Э 221	Ю 222	Я 223
E	а 224	б 225	в 226	г 227	д 228	е 229	ж 230	з 231	и 232	й 233	к 234	л 235	м 236	н 237	о 238	п 239
F	р 240	с 241	т 242	у 243	ф 244	х 245	ц 246	ч 247	ш 248	щ 249	ъ 250	ы 251	ь 252	э 253	ю 254	я 255

16-битовая кодировка казахских букв для Windows показана в таблице II.2.1.C.

Таблица II.2.1.C. 16-битовая кодировка казахских букв для Windows

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
040	**Ё**						**І**									
041	**А**	**Б**	**В**	**Г**	**Д**	**Е**	**Ж**	**З**	**И**	**Й**	**К**	**Л**	**М**	**Н**	**О**	**П**
042	**Р**	**С**	**Т**	**У**	**Ф**	**Х**	**Ц**	**Ч**	**Ш**	**Щ**	**Ъ**	**Ы**	**Ь**	**Э**	**Ю**	**Я**
043	**а**	**б**	**в**	**г**	**д**	**е**	**ж**	**з**	**и**	**й**	**к**	**л**	**м**	**н**	**о**	**п**
044	**р**	**с**	**т**	**у**	**ф**	**х**	**ц**	**ч**	**ш**	**щ**	**ъ**	**ы**	**ь**	**э**	**ю**	**я**
045		**ё**					**і**									
046																
047																
048																
049			**Ғ**	**ғ**							**Қ**	**қ**				
04A			**Ң**	**ң**											**Ү**	**ү**
04B	**Ұ**	**ұ**									**Һ**	**һ**				
04C																
04D									**Ә**	**ә**						
04E									**Ө**	**ө**						
04F																

Примеры II.2.1.

В нижней таблице дается 8-битовые коды некоторых символов.

Символы	**Коды**
4	1 1 1 1 0 1 0 0
A	1 1 0 0 0 0 0 0
B	1 1 0 0 0 0 0 1
+	0 1 0 0 1 1 1 0
=	0 1 1 1 1 1 1 0

В памяти компьютера цепочка символов A + B = 4 будет представляться.

II.2.1. Задания

1. 11000000110011011000000 расшифруйте значение цифр.
2. Закодируйте цепочку символов 2015.
3. Закодируйте цепочку DEF

Помощь

1. Нужно разбить последовательность на части по 8 цифр.
2. Нужно подряд писать коды каждой цифры.
3. Нужно подряд писать коды каждой буквы.

II.2.1. Вопросы

1. Как представляется логические данные в памяти?
2. Как представляется в памяти компьютера символы?
3. В чем разница между стандартами ANSI и Unicode ?

Тесты II.2.1.

1. В какой системе кодируется данные в ANSI?

A) в двоичной системе;
B) в троичной системе;
C) в восьмеричной системе;
D) в десятичной системе;
E) в шестнадцатеричной системе.

2. В какой системе кодируется данные в Unicode?

A) в шестнадцатеричной системе;
B) в троичной системе;
C) в восьмеричной системе;
D) в десятичной системе;
E) в двоичной системе.

3. Сколько байтов используется для кодировки в Unicode?

A) 2;
B) 1;
C) 3;
D) 5;
E) 4.

II.2.2. Представления чисел в компьютере

В компьютере обрабатываются только целые числа и действительные числа. Поэтому должны быть способы их представления в памяти компьютера. Любое целое число или действительное число представляется в двоичной системе счисления и для него выделяется одно *машинное слово*, в том числе для знака числа выделяется только один бит: отрицательный знак изображается цифрой 1, а положительный знак – цифрой 0. Одно машинное слово состоит из нескольких (1, 2, 4 или 8) байтов, их количество зависит от модели компьютеров. Во многих моделях компьютеров слово состоит из 4-х байтов, т.е. из 32 битов.

Образец преставления целого числа в памяти компьютера показана на рисунке II.2.2.A.

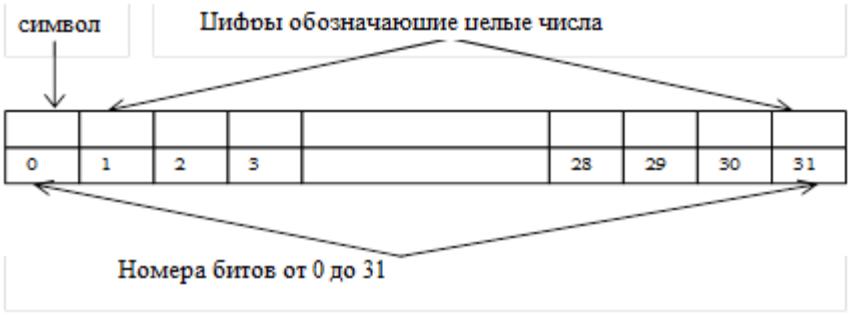


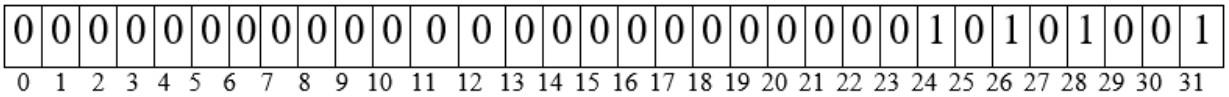
Рисунок II.2.2.A. Представление целого числа в памяти компьютера

В памяти компьютера действительное число представляется в форме плавающей точкой, его образец показан на рисунке II.2.2.B.

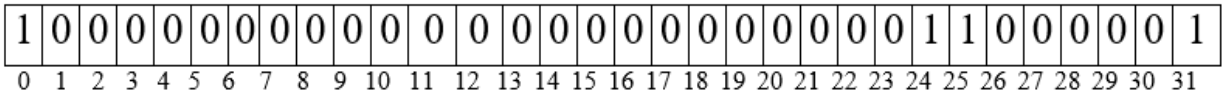
Рисунок II.2.2.B. Представление действительного числа в памяти компьютера.

Примеры II.2.2.

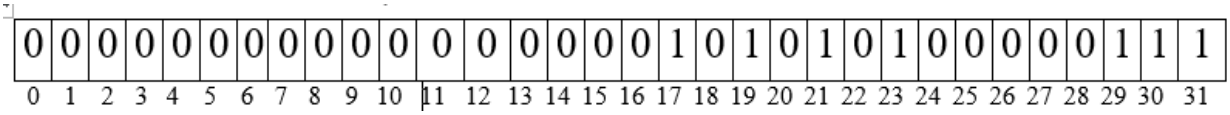
1. Представление положительного целого числа +10101001:



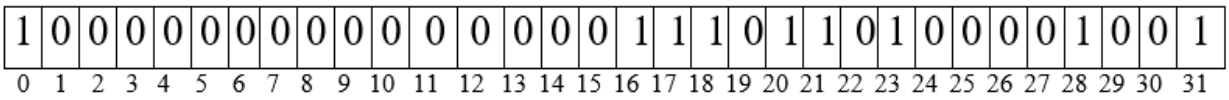
2. Представление положительного отрицательного числа – 11000001:



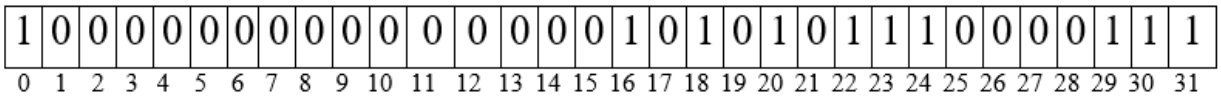
3. Представление положительного действительного числа с плавающей точкой $+0{,}1010101 * 2^3$:



4. Представление отрицательного действительного числа с плавающей точкой $-0{,}11101101 * 2^5$:



5. Представление отрицательного действительного числа с плавающей точкой $-0{,}10101011 * 2^{-7}$:



Задания II.2.2.

1. Представьте в памяти положительное целое число 63.
2. Представьте в памяти отрицательное действительное число с плавающей точкой $-0{,}11101101 * 2^5$.

Помощь

1. Нужно число 63 перевести из десятичной системы счисления в двоичную систему счисления и воспользоваться образцом представления целого числа, учесть знак числа.

2. Нужно воспользоваться образцом представления действительного числа с плавающей точкой, учесть знаки его мантиссы и порядка.

Вопросы II.2.2.

1. Как представляется в памяти компьютера целое число?
2. Как представляется в памяти компьютера действительное число с плавающей точкой?
3. Как представляется в памяти компьютера действительное число с фиксированной точкой?

Тесты II.2.2.

1. Из скольких байтов состоит машинное слово?
   A) из 4 байтов;
   B) из 1 байта;
   C) из 5 байтов;
   D) из 3 байтов;
   E) из 6 байтов.

2. Что отводится для кодирования знака число?
   A) 1 бит;
   B) 2 бита;
   C) 4 битов;
   D) 6 битов;
   E) 8 битов.

3. Сколько байтов отводится для кодирования порядка?
   A) 1 байт;
   B) 2 байта;
   C) 3 байта;
   D) 4 байтов;
   E) 5 байтов.

III. МЯГКОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАТИКИ

III.1. Алгоритмы

Ключевые слова: *алгоритм, обработка, указание, операция, данные, результат, действие, сложное действие, простое действие, присваивание, составитель, исполнитель, язык общения, протокол исполнения, автоматизация, вычислительная техника, понятность, точность, дискретность, конечность, массовость.*

Цель: *дать определение понятия алгоритма, ввести понятие обработки, рассмотреть свойства алгоритма, построить, исполнить, проверить реальные алгоритмы.*

Структура: Структура понятия алгоритма представлена на рисунке III.1

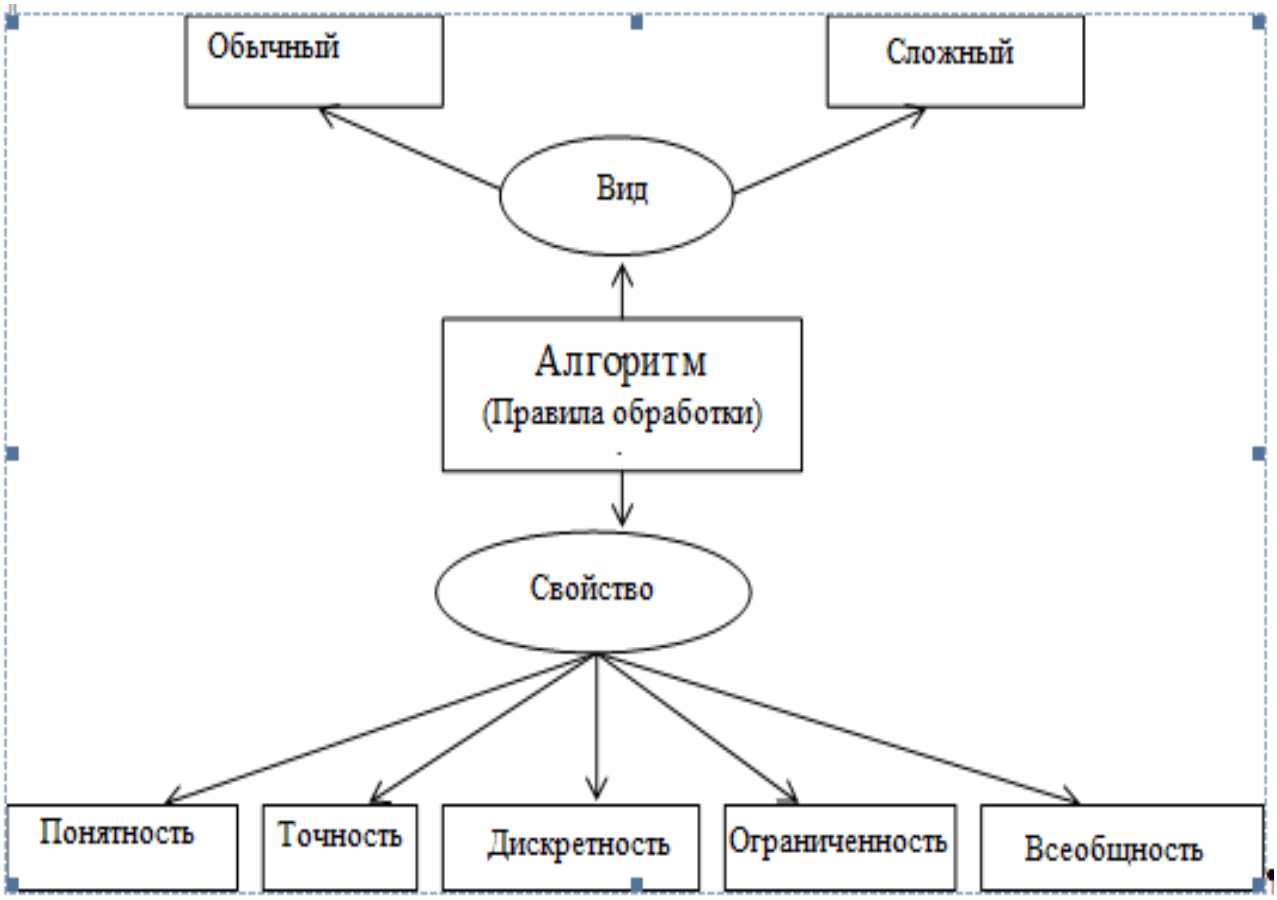


Рисунок III.1. Структура понятия алгоритма

III.1.1. Понятие обработки

Понятие «*обработка*» является одним из основных понятий в информатике. Он имеет несколько смыслов. Но нам нужен такой его смысл: обработкой называется выполнение операций над заданными величинами по заранее определенным указаниям, собранным в одно правило.

Для удобства в дальнейшем вместо понятия величины будем использовать понятие «данные».

Данные называются перед началом обработки *исходными данными*, во время обработки *промежуточными данными*, а после завершения обработки *результирующими данными*. Например, если операцию умножения двух заданных чисел представить с помощью последовательности их сложения, то значения заданных чисел – *исходные данные,* значения промежуточного сложения – *промежуточные данные,* значение конечной суммы – *результирующие данные*.

Выполнение операций по определенному указанию называют *действием*. Действия могут быть простым и сложным.

В информатике одним из простых действий является *присваивание*. В общем виде его можно представить как:

<Имя> ← ***<Выражение>***,

где в угловой скобке написаны понятия, в качестве имени берется произвольный идентификатор, «←» – знак присваивания, в качестве выражения – числовое выражение, символьное выражение или логическое выражение. В качестве знака присваивания можно брать и другие знаки, например « := ».

Семантика (*Смысел*) *действия присваивания*: сначала вычисляется значение выражения, затем оно присваивается имени, после завершения действия значение имени и значение будут равны между собой.

Например, в действии *X←2+4*8* и значение выражения и значение имени после завершения действия равны *34*.

Любое действие можно обозначить, обычно прописной латинской буквой, и представить в графическом виде. Например, если ***A*** простое действие, то его можно представить в графическом виде как на рисунке III.1.1.A.

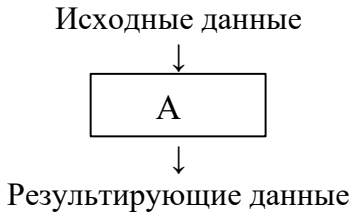


Рисунок III.1.1.A. Представление простого действия

Сложное действие состоит из последовательности простых действий. Его можно представить в графическом виде как на рисунке III.1.1.B.

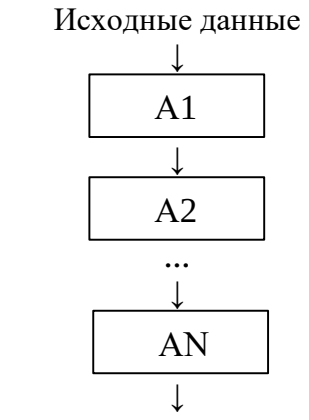


Рисунок III.1.1.B. Представление сложного действия

Здесь A1, A2,.., AN – простые действия, составляющие сложное действие. Порядок их выполнения будет сверху вниз. Результат каждого верхнего действия служит в качестве исходных данных следующего нижнего действия.

Итак, для определения обработки нужно построить правило. В этом правиле будут ответы на такие вопросы:

«Какие типы исходных данных?»;

«Какие действия и для чего нужно выполнить?»;

«Какой порядок выполнения действий?»;

«Что считается в качестве результата?».

Вообще, когда заводится речь об обработке, не надо забывать о двух субъектах. Один из них *составитель* правила, другой *исполнитель* составленного правила.

Исполнитель должен обладать способностью выполнить действия, указанные в правиле. Для этого он, с одной стороны, должен понимать указания в правиле, с другой строны, знать выполнения заданных операций. Поэтому при составлении таких правил всегда надо учитывать способность исполнителя. Потому что, некоторый исполнитель может выполнить правила, а у другого исполнителя такая способность отсутствует.

Правило пишется на одном языке общения. Составитель и исполнитель правила должны понимать одинаково смыслы всех предложений этого языка. Язык общения должен быть однозначным: при интерпретации его выражения должен быть получен только один смысл.

Примеры III.1.1.

1. Исходными данными обработки «Целочисленное деление» являются значение целых чисел.
2. Промежуточным данным обработки «Решение квадратного уравнения» является значение его дискриминанта.
3. Результатом обработки «Перевод с казахского языка на русский» является текст в русском языке.

Задания III.1.1.

1. Разложить на простые обработки сложную обработку «Найти корень квадратного уравнения».
2. Определить исходные данные и результат обработки «Приготовить чай».
3. Назвать промежуточные результаты обработки «Получить высшее образование».

Помощь:

1. Для нахождения корня квадратного уравнения установить значения его коэффициентов и вычислить значение дискриминанта.
2. Для приготовления чая нужны сухой чай, вода, молоко, сахар и другие продукты.
3. Для получения диплома нужно без задолженностей заканчивать каждый курс обучения.

III.1.1. Вопросы

1. Какой смысл имеет слово «обработка» в информатике?
2. Что подается на вход обработки?
3. Что такое результат?

Тесты III.1.1.

1. Какая среди них простая операция?
   A) сравнение двух заданных чисел;
   B) найти число букв в заданном слове;
   C) найти наибольший среди трех чисел;
   D) расположить буквы в слове в порядке возрастания;
   E) найти дискриминант квадратного уравнения.

2. Что выходит в самом конце обработки?
   A) исходные данные;
   B) промежуточные данные;
   C) простые данные;
   D) результирующие данные;
   E) структурные данные.

1. Какая из них сложная операция?
   A) A*B +C;
   B) X&Y;
   C) A-B;
   D) 3,14*R;
   E) 2,174/2.

III.1.2. Понятие алгоритма

В информатике правило обработки рассматривается как алгоритм. Слово «*алгоритм*» происходит от латинского написания имени великого ученого *аль-Хорезми – Algorithm*, родившегося в IX веке в Средней Азии. Он впервые составил правила выполнения арифметических операций (*сложение*, *вычитание*, *умножение*, *деление*) для многоразрядных целых чисел. Например, для сложения двух многоразрядных целых чисел нужно выполнить следующее правило:

1) определить значения двух многоразрядных целых чисел;

2) написать эти числа друг под другом, чтобы соответствовали их разряды;

3) если у одного из этих чисел не достают старшие разряды, то их надо заполнить нулями;

4) выполнить операцию сложения в самом младшем разряде и перейти к рассмотрению следующего разряда: если появилась единица перехода на следующий разряд, то её надо запомнить;

5) выполнить операцию сложения в текущем разряде, продвигаясь справа налево пока не закончатся все разряды, при этом нужно учитывать ранее запомненную единицу и запомнить вновь появившуюся единицу перехода в старший разряд, если они имеются;

6) взять в качестве результата число, полученное после выполнения операции сложения во всех разрядах: если имеется ранее сохраненная единица перехода на старший разряд, то её надо считать значением самого старшего разряда результата.

Для сложения двух многоразрядных целых чисел согласно этому правилу, исполнитель должен знать русский язык (так как правило написано на русском языке) и иметь способность сложения одноразрядных чисел.

В таблице III.1.2.А показан протокол сложения двух многоразрядных целых чисел.

Таблица III.1.2.А. Протокол сложения двух целых чисел.

Номер выполняемого указания	След выполненного указания	Пояснение
1	997 и 76	Определены значения чисел
2	997 76	Написаны друг под другом
3	997 076	Недостающие разряды заполнены нулями
4	997 + 076 13	При сложении в самом младшем разряде появилась единица перехода на старший разряд
5	997 + 076 1 73	При сложении в следующем разряде появилась единица перехода на старший разряд
6	997 + 076 1 073	При сложении в последнем разряде появилась еще раз единица перехода на старший разряд
7	1073	Получен результат с учетом единицы перехода на самый старший разряд

Теперь считая, что наш *исполнитель* знает десятичное число, умножение двух многоразрядных целых чисел и русский язык, построим правило умножения двух десятичных чисел:

1) определить значения двух десятичных чисел;

2) написать эти числа друг под другом как целые множители, не учитывая в них запятую, показывающую десятичный знак;

3) найти произведение, умножив множители, применяя предыдущее правило;

4) в обоих заданных числах сложить количество знаков, расположенных справа от запятой;

5) взять в качестве результата число, полученное после выполнения шага 3) так, чтобы в нем количество цифр после запятой было равно числу, полученному после выполнения шага 4). При этом, если количество цифр в произведении меньше числа, полученного после выполнения шага 4), то их разница определяет количество нулей, вставляемых сразу после запятой и перед значением полученного произведения.

В свое время такие правила выполнения арифметических операций понимались как алгоритм, позже слово «алгоритм» стал применяться для обозначения правил решения различных задач. Например, правило о нахождении наибольшего общего делителя двух натуральных чисел, называется *алгоритмом Евклида*:

1) найти значения двух натуральных чисел;
2) нужно определить большое число среди них: если два числа равны между собой, то в качестве результата берется любое из них;
3) нужно заменить большое число на разницу между большим числом и меньшим числом;
4) повторять действие, начиная с шага 2).

Этот алгоритм показывает свойство, что, если заданы два натуральных числа M и N, $M > N$, то наибольший общий делитель чисел M и N равно наибольшему общему делителю чисел $M–N$ и N.

Из вышеприведенного примера можно заметить, что исполнитель согласно указанию в алгоритме выполняет многочисленные очень простые и похожие друг на друга операции. Выполнение таких операций можно поручить другому исполнителю, предварительно обучив его. Отсюда возникает идея *автоматизации действия исполнителя*.

Для того чтобы реализовать эту идею сначала нужно дать точное определение понятию алгоритм. Это проблема породила новую отрасль в математике, называемой «*Теория алгоритмов*». Она направлена на точное определение понятия алгоритм и исследование возникающих в связи с этим теоретических проблем.

Точное математическое определение понятия алгоритм впервые было дано в 1937 году английским ученым А.Тьюрингом с помощью абстрактной вычислительной машины, которая умеет преобразовать дискретные данные. Точно в том же году американский логик Э.Л.Пост придумал еще одну абстрактную машину. Позже они названы «Машина Тьюринг» и «Машина Поста». Эти абстрактные машины принципиально показали, что любую проблему могут решать автоматы, если она будет алгоритмизирована с учетом исполняемых операций и её алгоритм будет записан на языке абстрактной машины.

Успехи в отрасли «*Теория алгоритмов*» доказали возможность построения *компьютеров*, позволяющих автоматизировать выполнения алгоритмов.

Появление компьютеров было обусловлено распространением понятия алгоритм. Под алгоритмом стали понимать не только правила решения математических задач, но и правила решения других задач. К ним относятся *экономические задачи*, *задачи управления*, *задачи перевода с одного языка на другой язык* и др.

Сейчас тяжело найти отрасль интеллектуальной деятельности человека, где не применяется компьютеры. Это поставило перед человечеством сложную проблему: уметь составлять алгоритмы и представить их на понятном компьютеру (исполнителю) языке.

Однако для того, чтобы научиться составлять алгоритма необязательно знать теорию алгоритмов, но достаточно знать интуитивное определение алгоритма, его свойства и способы представления.

Можно дать следующее интуитивное определение алгоритма:

Алгоритм – это конечная последовательность понятных и точных указаний о том, что какие операции и в каком порядке нужно выполнять для решения любой задачи заданного класса за конечное число шагов.

Из этого определения вытекают следующие свойства алгоритма:

1. *Понятность* – требует, чтобы указания должны быть написаны на понятном исполнителю языке;

2. *Точность* – требует, чтобы указания должны быть однозначными;

3. *Дискретность* – требует, чтобы указания должны образовать цепочку и они выполнялись отдельными шагами;

4. *Конечность* – требует, чтобы выполнение указаний осуществлялось совершением конечного числа шагов и завершилось получением конкретного результата;

5. *Массовость* – требует, чтобы алгоритм должен быть применимым для класса задач.

Примеры III.1.2.

Обозначив алгоритм нахождения наибольшего общего делителя двух натуральных чисел M и N через $НБОД(M, N)$, напишем алгоритм Евклида следующим образом:

1) Определить значения натуральных чисел M, N;
2) Если $M=N$, то $НБОД(M, N) \leftarrow M$;
3) Если $M>N$, то $M \leftarrow M–N$ и повторять с шага 2;
4) $N \leftarrow N–M$ и повторять с шага 2.

Посмотрим на свойства этого алгоритма.

1) Если исполнитель понимает русский язык и знает смыслы приведенных операций, то ему этот алгоритм будет *понятен*;

2) Каждый исполнитель, который понял алгоритм получит одинаковый результат, так как все его указания однозначны;

3) Указания, разделяясь друг от друга образовали последовательность, и их выполнение требуют совершения отдельных шагов;

4) Выполнение этих указаний будут не бесконечными. Так как при каждом повторении уменьшается значение числа M или числа N. В конце они будут равны и получится конкретный результат;

5) Придавая числам M и N различные значения можно применить этот алгоритм для любых натуральных чисел.

То, что приведено в верхней таблице называется *протоколом выполнения алгоритма*, список исходных данных и результатов каждого шага при выполнении алгоритма.

Номер примера	Номер выполняемых указаний	Ход выполнения указаний
1	1	M←8, N←4
	3	8>4, M←8–4
	2	4=4, НБОД (8,4)←4
2	1	M←7, N←3
	3	7>3, M←7–3
	3	4>3, M←4–3
	4	1<3, N←3–1
	4	1<2, N←2–1
	2	1=1, НБОД (7,3) ←1

Задания III.1.2.

1. Построить алгоритм нахождения наибольшего из чисел A, B.
2. Построить алгоритм нахождения наименьшего из чисел A, B.
3. Найти арифметическое среднее заданных чисел A, B.

Помощь:

1. После установления значения этих чисел, нужно предусмотреть применение отношения сравнения.
2. После установления значения этих чисел, нужно предусмотреть применение отношения сравнения.
3. Арифметическое среднее заданных чисел равно результату от деления суммы этих чисел на их количество.

Вопросы III.1.2.

1.Что такое интуитивное определение алгоритма?

2. Что такое точное определение алгоритма?

3. Что такое протокол выполнения алгоритма?

Тесты III.1.2.

1. Перечислите все свойства, которыми обладает любой алгоритм?

A) Понятность, точность, дискретность, конечность, массовость;

B) Понятность, точность, ясность, указанность, массовость;

C) Понятность, точность, дискретность, конечность, общность;

D) Понятность, точность, натуральность, конечность, массовость;

E) Бесконечность, точность, дискретность, ясность, массовость.

2. Перечислите все требования, которые необходимы для свойства массовости алгоритма?

A) Требует, чтобы алгоритм мог применяться не только для конкретной задачи, но и для класса задач, подобных этой задаче;

B) Требует, чтобы был написан на понятном исполнителю языке;

C) Требует, чтобы текст алгоритма был понятным, однозначным и точным;

D) Требует, чтобы образовал последовательность точных действий;

E) Требует, чтобы указания должны были давать результаты для любых исходных данных.

3. Перечислите все требования, которые необходимы для свойства конечности алгоритма?

A) Выполнение указания должно быть точным;

B) Выполнение указания должно быть понятными;

C) Выполнение указания должно быть однозначными;

D) Выполнение указания должно быть последовательно и конечно;

E) Выполнение указания должно закончиться за конечное число шагов и получить конкретный результат.

III.2. Алгоритмический язык

В этом разделе рассматриваются виды алгоритмических языков и их свойства.

Ключевые слова: *алгоритмический язык, естесственный язык, искусственный язык, графический язык, блок-схема, древовидная графика, алфавит, структуры управления, последовательность, ветвление, повторение.*

Цель: *ознакомиться со способами представления алгоритмов, анализ графических языков, определение искусственного алгоритмического языка, научиться построению алгоритма в искусственном языке.*

Структура: Виды алгоритмических языков показаны на рисунке III.2.

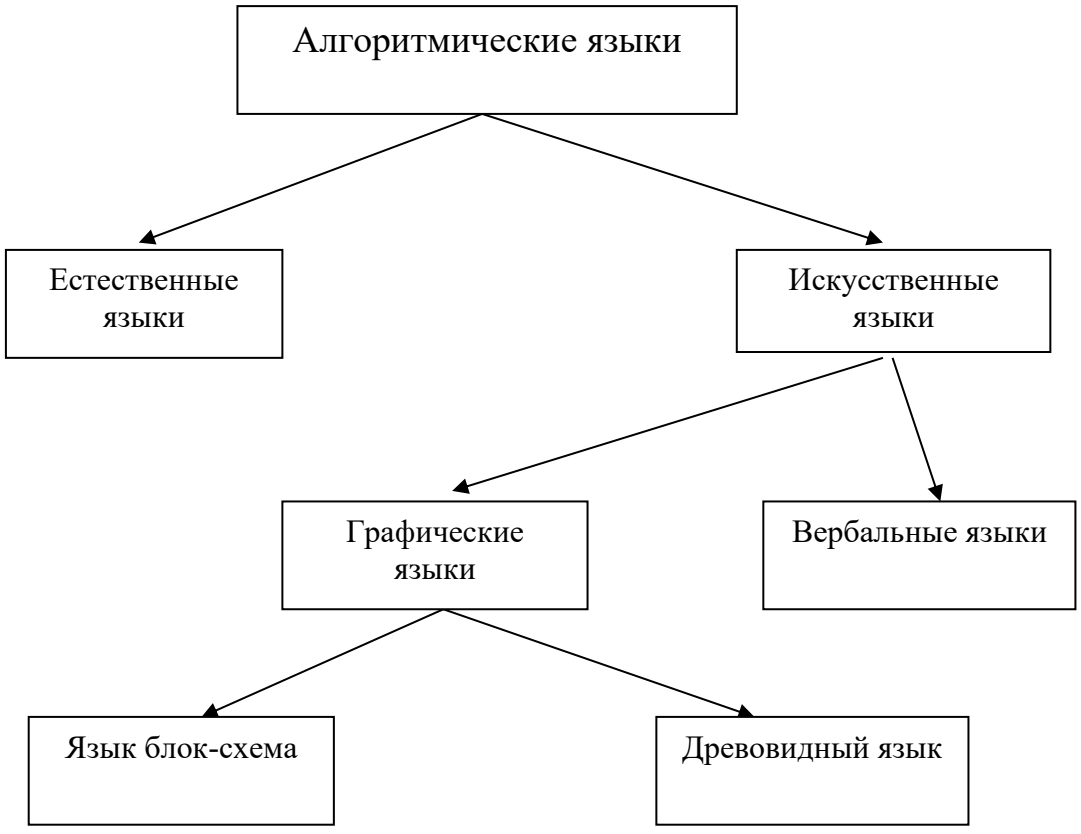


Рисунок III.2. Виды алгоритмических языков.

III.2.1. Вербальный алгоритмический язык

В предыдущем разделе мы ознакомились с представлением алгоритма на естественном языке, на котором общаются люди. К естественным языкам относятся, например, казахский язык, русский язык, английский язык и другие языки. Иногда для уточнений указаний и операций в алгоритмах к этим языкам добавляются математические выражения.

Запись алгоритма на любом естественном языке может иметь несколько значений из-за неоднозначности естественных языков. Это приведет к тому, что выполнение алгоритма на естественном языке будет затруднительным или становится невозможным. Поэтому в информатике, несмотря на богатство естественных языков, для представления применяются однозначные искусственные языки. К ним относятся *искусственные вербальные* (словесные) и *графические языки*.

Теперь определим искусственный вербальный алгоритмический язык, основанный на русском языке и математических выражений. В этом языке используются несколько «фиксированных» слов русского языка. У каждого из них будет только одно заранее определенное значение. Поэтому их можно ввести в алфавит искусственного вербального алгоритмического языка. Кроме того, в этот алфавит включаются знаки латинского и русского алфавитов, арабские и римские цифры, знаки препинания, скобки, знак операции присваивания, знак пробела и специальные математические знаки. Этот алфавит приведен в таблице III.2.1.

Таблица III.2.1. Алфавит искусственного алгоритмического языка.

Виды знаков	**Представление в алфавите**
Буквы	
Латинские буквы	A, B, C,.., X, У, Z, a, в, c,.., x, y, z
Русские буквы	А, Б, В,..,Э, Ю, Я, а, б, в, …, э, ю, я
Цифры	
Арабские цифры	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Римские цифры	I, V, X
Специальные знаки	
Скобки	
Структурные скобки	(,)
Символьные скобки	‘, ‘
Измерительные скобки	[,]
Знаки операций	
Числовые операции	+, –, *, /, ↑, %
Символьные операции	·, |
Операции отношения	=, <, >, ≤, ≥
Логические операции	⌉, ∧, ∨
Операция присваивания	←
Знаки препинания	., ,, :, ;, !
Знак пробела	␣
Фиксированные слова	АЛГОРИТМ, НАЧАЛО, КОНЕЦ, ВВОД, ВЫВОД, ЕСЛИ, ТО, ИНАЧЕ, ВЫБОР, ВЫПОЛНИТЬ, ПОКА, ДО, ДЛЯ, ШАГ, ПОВТОРЕНИЕ, ЗАКОНЧИЛСЯ

Примеры III.2.1:

1. Нужно просчитать дневную заработную плату служащего, в зависимости от его должности и рабочего времени. Имеются три категории должности М1, М2, М3 с рабочими временами Т1, Т2, Т3. Цена одного часа будет 100, 200, 300 рублей соответственно. Если служащий работает больше 8 часов, то к его заработной плате добавляется надбавка в размере 10% от заработной платы. Решение этой задачи представлено следующим алгоритмом:

```
АЛГОРИТМ зарплата
  ВВОД (М1, М2, М3, Т1, Т2, Т3)
     НАЧАЛО
          ЕСЛИ Т1 > 8 ТО М1 ←100 * Т1 + 100 * Т1 * 10 %
```

ИНАЧЕ М1 ← 100 * Т1
ЕСЛИ Т2 > 8 ТО М2 ←200 * Т2 + 200 * Т2 * 10 %
ИНАЧЕ М2 ← 200 * Т2
ЕСЛИ Т3 ≤ 8 ТО М3 ← 300 * Т3
ИНАЧЕ М3 ← 300 * Т + 300 * Т3 * 10 %
КОНЕЦ
ВЫВОД (М1, М2, М3)
ЗАКОНЧИЛСЯ

2. Построить алгоритм для решения квадратного уравнения вида A*X↑2 + B*X + C = 0. Для этого сначала вычисляется значение дискриминанта этого уравнения, затем в зависимости от его значения ищутся решения. Следовательно, для решения таких квадратных уравнений можно составить следующий алгоритм:

АЛГОРИТМ квадратное уравнение
ВВОД (A, B, C)
НАЧАЛО
D = B↑B – 4A*C
ЕСЛИ D>0 ТО ВЫВОД(
‘Первое решение уравнения X1= ’(–B+Sqrt(D) /2*A,
‘Второе решение уравнения X2= ’ (–B–Sqrt(D)) /2*A)
ЕСЛИ D=0 ТО ВЫВОД ‘Единственное решение X = ’ –B/2*A
ЕСЛИ D<0 ТО ВЫВОД ‘Уравнение не имеет решения’
КОНЕЦ
ЗАКОНЧИЛСЯ

Задание III.2.1.

1. Составить алгоритм для нахождения произведения двух целых чисел путем последовательного их сложения.

2. Составить алгоритм для возведения в степень одного числа на другое путем последовательного умножения.

3. Составить алгоритм для нахождения половины квадрата числа.

Помощь

Надо уметь представлять сложные операции с помощью простых операций.

Вопросы III.2.1.

1. На какие виды подразделяются алгоритмические языки?
2. Что такой алфавит вербального алгоритмического языка?
3. Какие слова есть в вербальном алгоритмическом языке?

Тесты III.2.1.

1. От имени какого великого ученого произошел термин алгоритм?

A) Джон фон Нейман;
B) Чарльз Беббидж;
C) Мухаммед аль-Хорезми;
D) Вильгельм Лейбниц;
E) Ада Лавлейс.

2. Как называется каждая команда в алгоритме?

A) Условие;
B) Система команд;
C) Алгоритмический процесс;
D) Простая команда;
E) Шаг.

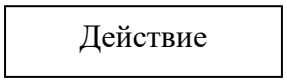
3. Какой из них ближе к определению алгоритма?

A) Реальные правила, определяющие порядок выполнения действий в целях нахождения результата;
B) Порядок выполнения действий;
C) Получение результата по некоторым шагам;
D) Выбор однородных задач среди многих значений;
E) Правила получения правильного результата.

III.2.2. Графический алгоритмический язык

Способ графического представления алгоритма называют графическим алгоритмическим языком. Имеются два вида: язык *блок-схем*, *древовидный язык*. *Язык блок-схем* может показать хронологические связи различных действий в алгоритме. В этом языке для указания различных действий и порядка их выпонения используются разные геометрические фигуры и стрелки соответственно. Они составляют алфавит языка блок-схем, а их смыслы задаются по предварительным соглашениям. Алфавит языка блок-схем показан в таблице III.2.2.

Таблица III.2.2. Алфавит языка блок-схем.

Название	**Представление**	**Назначение**
Овал		Представление начала и конца алгоритма
Параллелограмм		Представление операции ввода и вывода
Прямоугольник		Представление действия и его номера
Ромб		Представление условного ветвления
Окружность		Представление объединения нескольких путей
Стрелки		Представление направления путей

Смыслы используемых действий «ввод данных» и «вывод результата» мы определим при изучении структуры компьютера. Пока вместо них будем использовать операцию присваивание под

названием «Определение значений исходных данных» и «Подготовка значений результата» соответственно.

Язык блок-схем является графическим способом, который предназначен для представления простых алгоритмов. Однако несмотря на то, что он понятен и удобен для начинающих составителей, его нельзя использовать для представления сложных алгоритмов. Так как блок-схемы частей сложного алгоритма будут изображаться на нескольких страницах, поэтому тяжело проследить связи между ними и понять общую логику алгоритма. Следовательно, мы познакомимся еще с одним графическим способом представления алгоритма – *древовидным языком*. Общий вид алгоритма на древовидном языке показан на рисунке III.2.2

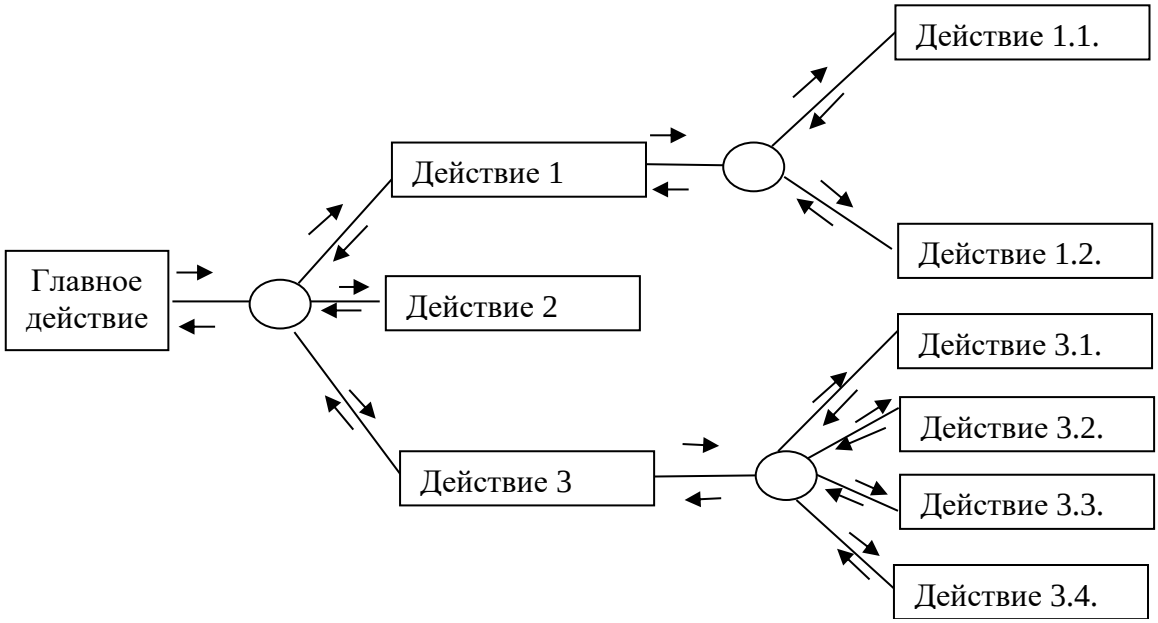


Рисунок III.2.2. Общий вид алгоритма в древовидном языке.

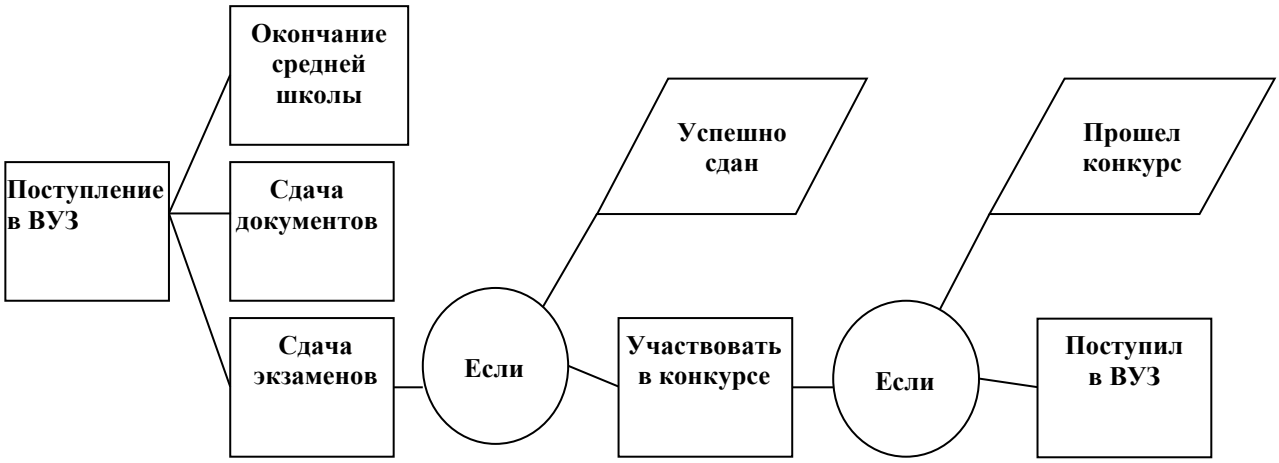
Представление алгоритма на рисунке III.2.2 похож на дерево. Поэтому его называет древовидным языком. Самое крайнее слева действие есть корень дерева, а самые крайние справа будут его листьями. Сначала представляется главное действие как корень, затем оно разлагается на несколько составляющих себя действий, эти действия могут быть еще разложены на их составляющие меньшие действия, а в конце представляются не разлагаемые дальше простые действия. Для представления действий применяются прямоугольники, их составляющие действия начинаются с окружности, хронологический порядок действий в алгоритме

указываются цифрами, а обход дерева можно показывать стрелками. Путь обхода дерева показывает порядок разложения действий в алгоритме и порядок их выполнения. Обычно, *порядок разложения* осуществляется *слева направо*, а *порядок выполнения – сверху вниз*. Поэтому при представлении алгоритма на этом языке можно не указывать стрелки.

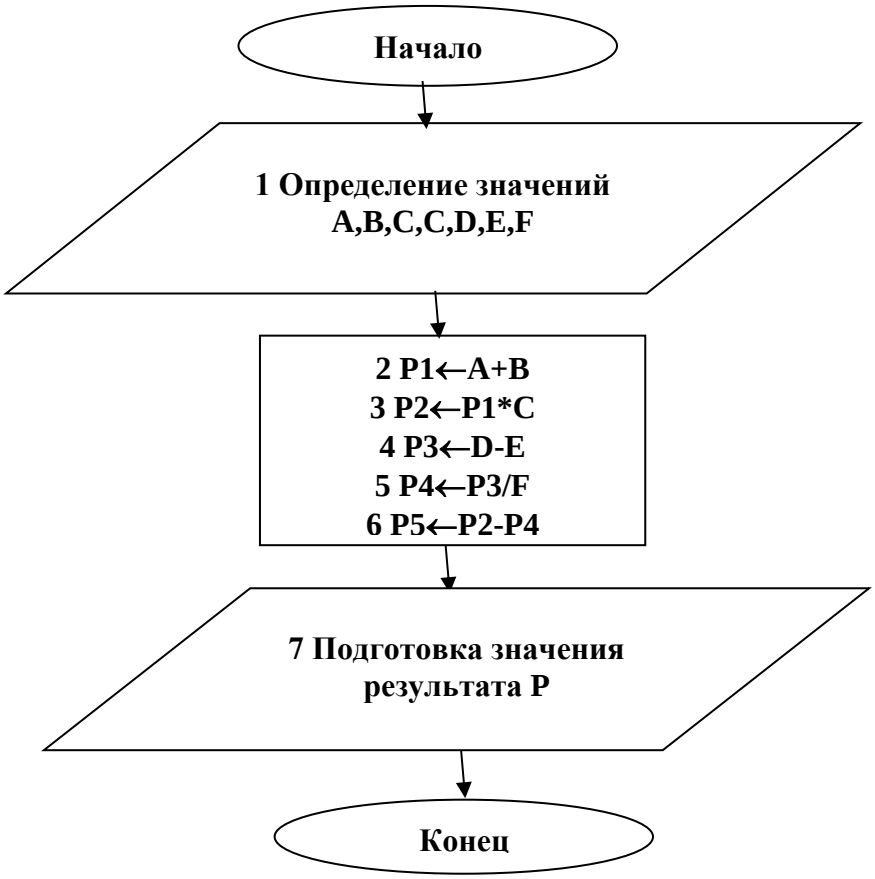
Древовидный язык, позволяет представлять иерархическую структуру алгоритма, показывая наряду с хронологическими связями указаний, их логического разложения – декомпозицию. На этом языке можно представлять и условные ветвления и повторения. Для этого в окружность пишется ключевое слово «Если», «Пока» или «До», а само условие ветвления или повторения записывается внутри ромба.

Примеры III.2.2.

1. Абитуриент для поступления в высшее учебное заведение совершает несколько действий и среди них некоторые реализуются только при выполнении определенного условия. Самые важные из них ниже представлены с помощью древовидного языка:



2. Представление алгоритма вычисления значении выражения (A+C)*C–(D–E)/F, придавая именам данных A, B, C, D, E и F различные значения, на языке болок-схем показано ниже.



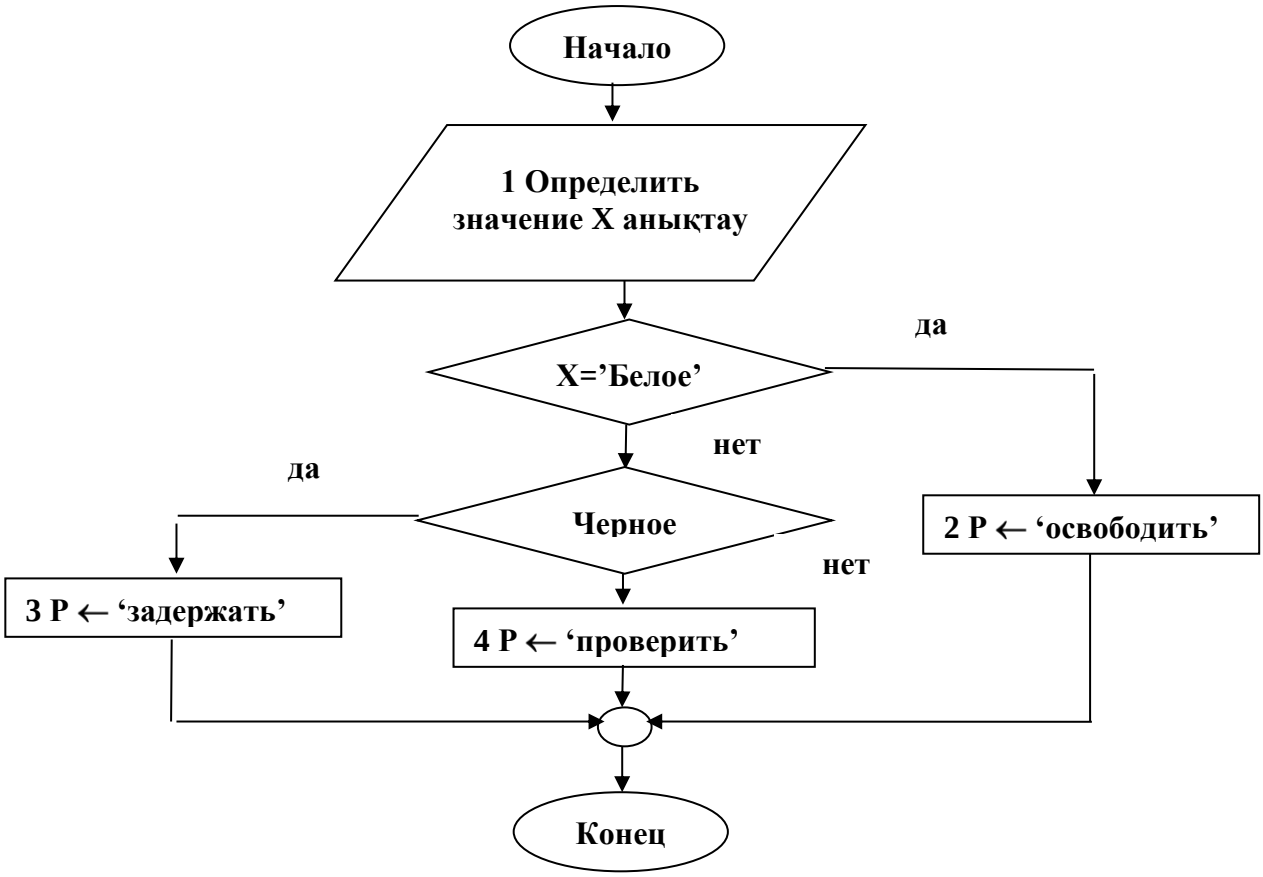
Здесь **P1, P2, P3, P4, P5** – имена промежуточных результатов. Протокол выполнения этого алгоритма показан в нижней таблице.

Номер примера	Номер указания	Ход выполнения указания
I	1	A←2, B ←4, C ←2, D ←10, E ←2, F ←2
	2	P1 ← 2 + 4
	3	P2 ← 6 * 2
	4	P3 ← 10 – 2
	5	P4 ← 8/2
	6	P5 ← 12 – 4
	7	P ← 8
II	1	A ←3, B ←5, C ←4, D ←15, E ←15, F ←2
	2	P1 ← 3+5
	3	P2 ← 8 * 4
	4	P3 ← 15 – 5
	5	P4 ← 10/2
	6	P5 ← 32 – 5
	7	P ← 27

В этой таблице вместо операции ввода и вывода в блок-схеме применены операции присваивания в указаниях 1 и 7. Так как, операции ввода и вывода являются одними из видов операции присваивания

3. Пусть величина **X** имеет несколько значений. Из них первое – ‘Белое’, второе – ‘Черное’, а другие не известны. Нужно совершить различные действия в зависимости от значения **X**. Из первое – придать символьной величине **P** значение ‘освободить’, если **X**=’Белое’, второе – придать символьной величине **P** значение ‘задержать’, если **X**=’Черное’, а третье – придать символьной величине **P** значение ‘проверить’, если **X** принимает произвольное значение.

Для вышесказанного ниже построим следующую блок-схему:



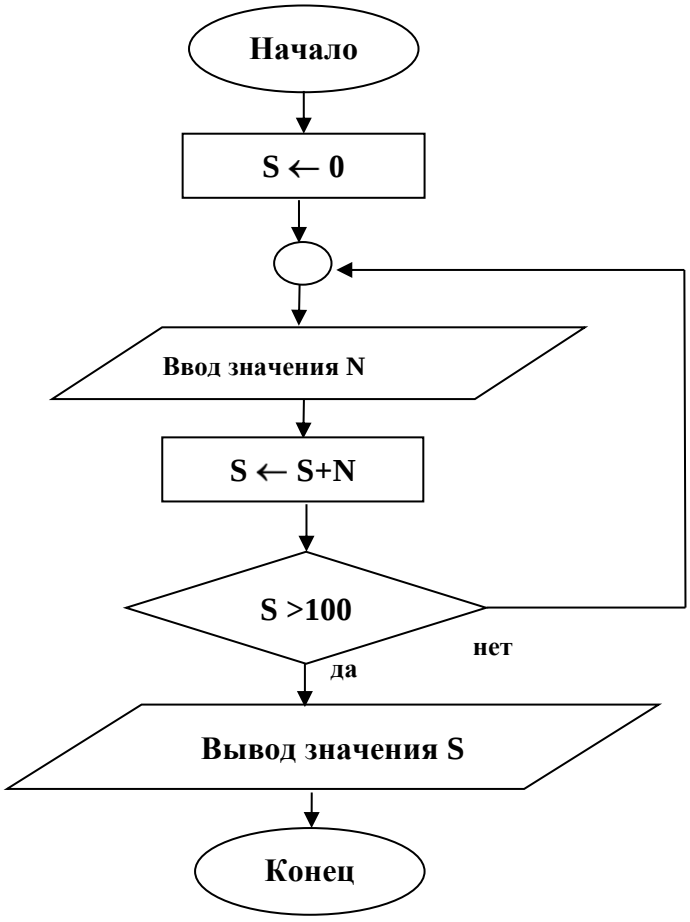
В этой блок-схеме можно заметить, что из обозначающего условного ветвления ромбы исходит две стрелки в любом, кроме верхнего, направлении. Вообще говоря, составитель блок-схемы может разворачивать её по своему усмотрению в любую сторону развертки плоскости.

Протокол выполнения этого алгоритма показан ниже.

Номер примера	Номер указания	Ход выполнения указания
I	1	X ← ‘Белое’
	2	P ← ‘освободить’
II	1	X ← ‘Черное’
	3	P ← ‘задержать’
	1	X ← ‘Не знаю’
III	4	P ← ‘проверить’

Задания III.2.2.

1. Постройте блок-схему алгоритма рабочего цикла компьютера.
2. Перепишите следующий алгоритм на вербальном языке.



3. Представить в виде дерева порядок выполнения операции в этом выражении X← (A+B) ∗ (C - D).

Помощь:

1. В этом языке будут использованы несколько ключевых слов на русском языке с фиксированными значениями.

2. Построение алгоритма должно быть в соответствии с правилами рабочего цикла компьютера, приведенными в II.1.2.

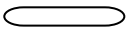
3. В представляемом алгоритме на древовидном языке нет необходимости велючать ветвление по какому-то условию.

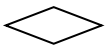
Вопросы III.2.2.

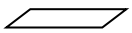
1. Какие виды есть графических языков?
2. Что входит в алфавит языка блок-схем?
3. Что показывает обход дерева?

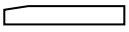
Тесты III.2.2.

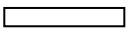
1. Какая фигура в блок-схеме представляет начало и конец?

A) ; 

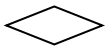
B) ; 

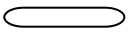
C) ; 

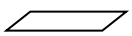
D) ; 

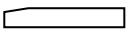
E) . 

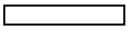
2. Какая фигура в блок-схеме представляет условие?

A) ; 

B) ; 

C) ; 

D) ; 

E) . 

3. Какая фигура в блок-схеме представляет операцию ввод или вывод?

A) Параллелограмм;

B) Ромб;

C) Окружность;

D) Прямоугольник;

E) Овал.

III.3. Структуры управления

Ключнвые слова: *последовательность, ветвление, простое ветвление, альтернативное ветвление, многозначное ветвление, повторение, повторение с предусловием, повторение с постусловием, параметрическое повторение.*

Цель: *знакомство со структурами управления и различать их свойства, научить способам построения алгоритмов на искусственном языке* с *помощью структур управления.*

Структура: Структура изучаемого материала «Структуры управления» показана на рисунке III.3.

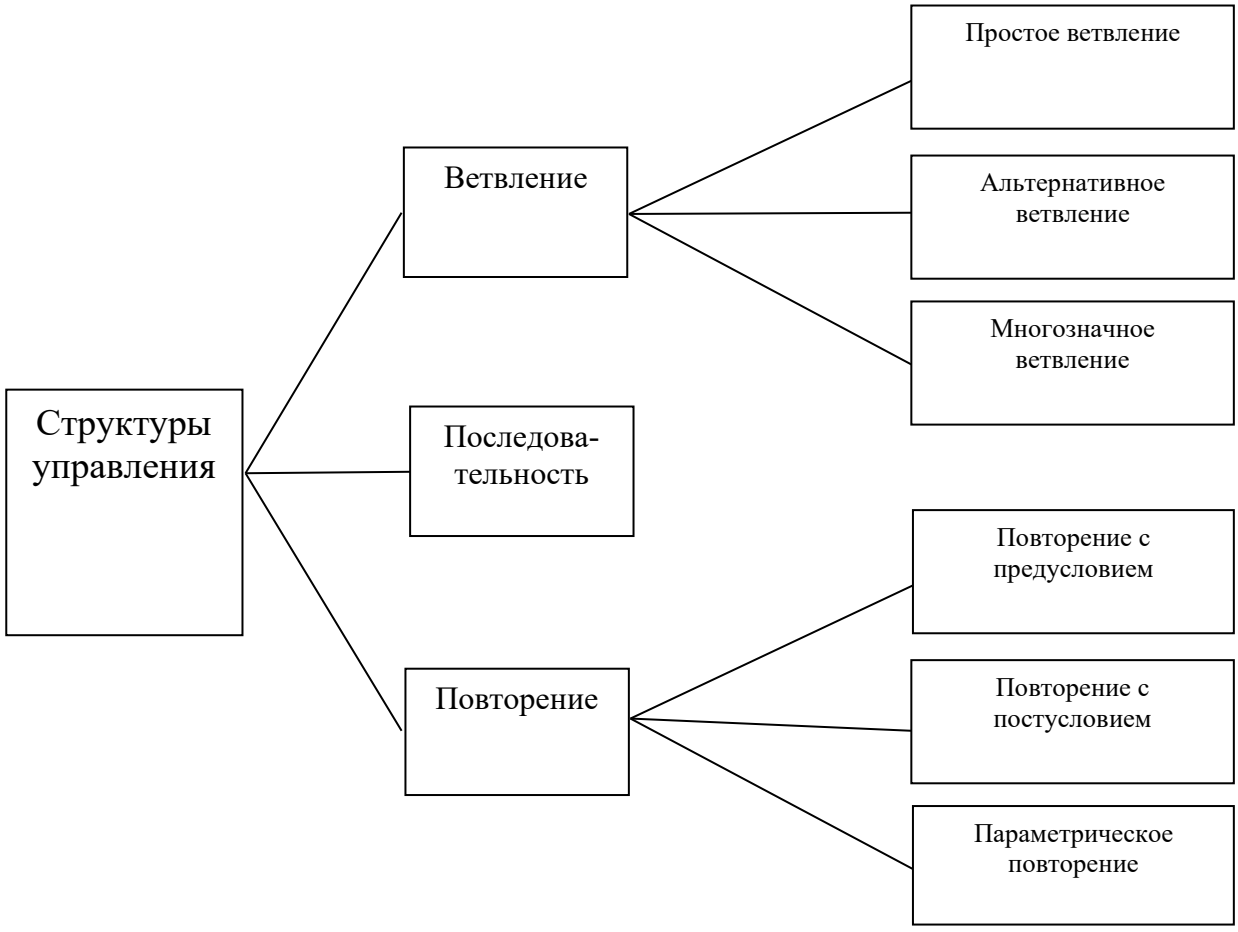


Рисунок III.3. Виды структуры управления.

III.3.1. Структуры управления и их значения

После рассмотрения способов записи алгоритмов, вполне закономерным представляется вопрос о методологии их построения, т.е. возникла необходимость создания технологии их построения. В связи с этим появилась идея *структурного программирования* в 1970 гг в компании IBM, предложенная учеными Э. Дейкстра, Х. Милс, Э. Кнут, С. Хоор.

Поскольку алгоритм определяет порядок обработки информации, он должен содержать, с одной стороны, *действия по обработке*, а с другой стороны, порядок их следования, называемым *потоком управления*, *имеющие* следующиее *свойства*:

(1) Каждый блок выполняется не более одного раза;

(2) Выполняется каждый блок.

Здесь блоки, связанные с обработкой данных, делятся на простые и условные. Простое действие имеет один вход и один выход, а условное обладает двумя выходами в зависимости от выполнимости некоторого того, истинным ли окажется условие.

Простое действие не означает, что оно единственное, оно может быть последовательностью действий. Часть алгоритма, организованная как простое действие, т.е. имеющая один вход (выполнение начинается всегда с одного и того же действия) и один выход (после окончания выполнения данного блока всегда выполняется одно и то же действие), называется *функциональным блоком.* Согласно положениям структурного программирования можно выделить всего три различных варианта организации потока управления действиями алгоритма. Для пояснения этих вариантов возьмем два функциональных блоков S_1 и S_2.

Первый тип потока управления называется *сцеплением*, он организует выполнение обоих функциональных блоков S_1 и S_2: при этом, если выполняются оба свойства потока управления, то поток

управления будет *линейным.* В нем несколько блоков могут быть объединены в один функциональных блок. Графическое изображение объединения последовательно выполняющихся блоков S_1 и S_2 в один функциональный блок S_1S_2 на рис. III.3.1.

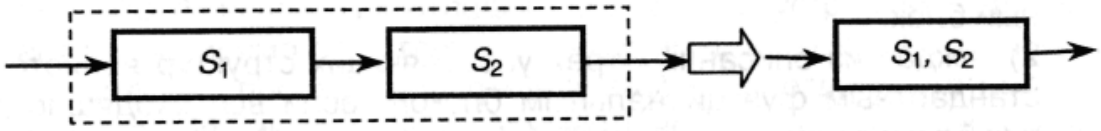


Рисунок III.3.1. Объединение двух блоков в один блок

Второй тип потока управления называется *ветвлением*, он организует выполнение одного из двух функциональных блоков S_1 и S_2 в зависимости от значения логического условия. Здесь свойство (1) выполняется, а свойство (2) – не выполняется. Если поток управления содержит оба функциональных блоков S_1 и S_2, то ветвление называется *альтернативным*. Если поток управления не содержит одного из блоков S_1 и S_2, то ветвление называется *простым*. Вообще, ветвления подразделяются на три вида: *простое ветвление*, *альтернативное ветвление*, *многозначное ветвление*.

Третий тип потока управления называется *повторением*, он организует многократное повторение выполнения функционального блока, называемого *телом повторения*. Число повторения задается заранее или определяется в зависимости от некоторого условия. Для циклического потока выполняется свойство (2), но не выполняется (1). Повторение в зависимости от выполнения некоторого условия подразделяется на два вида: *повторение с предусловием*, *повторение с постусловием*. В повторении с предусловием условие проверяется раньше, чем выполняется тело поторения, а в *повторении с постусловием* – после тела повторения. Если число повторения известно заранее, то такое повторение называется *параметрическим повторением*, он относится к группе повторения с постусловием.

Поскольку ветвление и повторение имеют один вход и один выход, они в целом также подходят под определение функционального блока. Теперь определим рекурсивноо понятие *стандартного функционального блока*:

1) каждое простое действие есть стандартный функциональный блок (СФБ);

2) каждый из описанных трех потоков управления является СФБ, если все входящие в них блоки являются СФБ;

3) других СФБ не существуют.

В литературе СФБ иногда называют *базовыми структурами управления*, поэтому в дальнейшем мы будем пользоваться им.

Можно говорить *структурный алгоритм*, если он представлен только базовыми структурами управления. Другими словами, структурный алгоритм представляет собой комбинацию только трех рассмотренных выше *структур управления*.

Структурные алгоритмы обладают рядом преимуществ по сравнению с неструктурными:

– *понятность* и *простота* восприятия алгоритма (поскольку невелико число исходных структур, которыми он образован);

– *проверяемость* (для проверки любой из основных структур достаточно убедиться в правильности входящих в нее функциональных блоков);

– *модифицируемость* (она состоит в простоте изменения структуры алгоритма, поскольку составляющие функциональные блоки относительно независимы).

Значения структуры управления в составлении алгоритма очень велики. Поэтому ниже рассматриваются представление структуры управления в различных алгоритмических языках и их свойства.

Для представления структуры управления применяются *язык блок-схем*, *древовидный язык* и *искусственный вербальный язык*.

Замечание III.3.1:

В искусственном вербальном языке предложение, которое начинается со знака «:» является пояснением касательно алгоритму и действия, указанные в нем не выполняются. Фиксированные слова применяются как собственные знаки (из его алфавита), угловые скобки «< » и « > » не являются собственными знаками языка и выражение внутри неё является не представлением действия в этом языке, а обозначением. Иначе говоря, если представление действия пишется явно, то угловая скобка не применяется. Также знаки прямой скобки « [» и «] » не являются собственными знаками этого языка. Внутри неё записываются не обязательные части, т.е. эти части можно и не показывать.

Примеры III.3.1.

1. Последовательные действия записывается с помощью структуры управления последовательность.
2. Разветвляющиеся действия записываются с помощью структуры управления ветвления.
3. Повторяющиеся действия записываются с помощью структуры управления повторения.

Задания III.3.1.

1. Покажите пример объединения двух стандартного функционального блока в один.
2. Перечислите виды ветвления.
3. Перечислите свойства структурного алгоритма.

Помощь

1. Нужно выбрать такие ФСБ, чтобы они удовлетворяли оба свойства потока управления.
2. В засимости от выполнения логического условия будет организовано выполнение одного или нескольких ФСБ.

3. Эти свойства повышают удобства, облегчает составление и сопровождение и др

Вопросы III.3.1.

1. Что такое последовательность?
2. Что такое ветвление?
3. Что такое повторение?

Тесты III.3.1.

1. Какие виды есть у алгоритмической структуры управления ветвления?

A) простое, альтернативное, многозначное.

B) простое, альтернативное,

C) альтернативное, многозначное.

D) простое, сложное

E) с предусловием, с постусловием, параметрическое

2. Какие виды есть у алгоритмической структуры управления повторения?

A) простое, альтернативное, многозначное;

B) с предусловием, с постусловием, параметрическое;

C) условное, безусловное;

D) простое, сложное;

E) среднее, сложное.

3. С помощью чего можно представить алгоритмические структуры управления?

A) просто, альтернативно, многозначно;

B) на языке блок-схем, на древовидном языке, на вербальном языке;

C) в алгоритме;

D) в программе;

E) в естественном языке.

III.3.2. Следование

Структура управления следование показывает возможность построения одного сложного действия из нескольких простых действий.

В следовании сложное действие ***S*** образуется из последовательности простых действий ***S1***, ***S2,..***, ***SN.*** Порядок выполнения простых действий соответствует порядку возрастания их номеров.

В следовании в качестве исходных данных сложного действия ***S*** берутся исходные данные первого простого действия ***S1***, а в качестве результата действия ***S*** считается результат последнего простого действия ***SN***.

На рисунке III.3.2.A показано представление следования на языке блок- схемы:

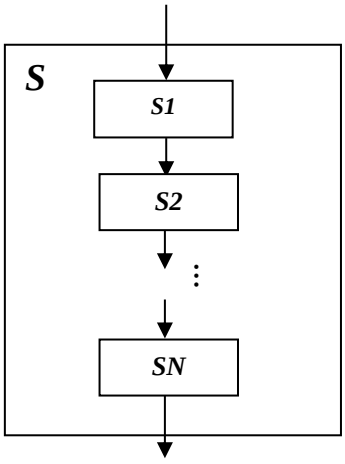


Рисунок III.3.2.A. Последовательность на языке блок-схемы.

На рисунке III.3.2.B показано представление следования на древовидном языке:

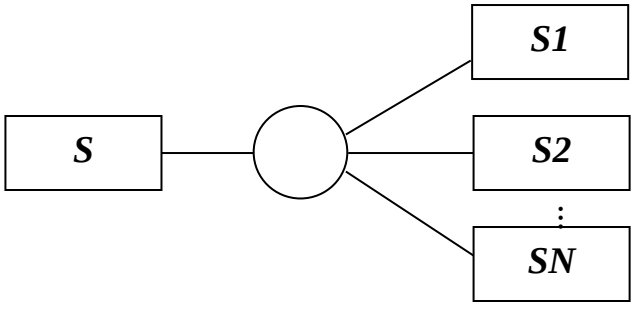


Рисунок III.3.2.B. Последовательность на древовидном языке.

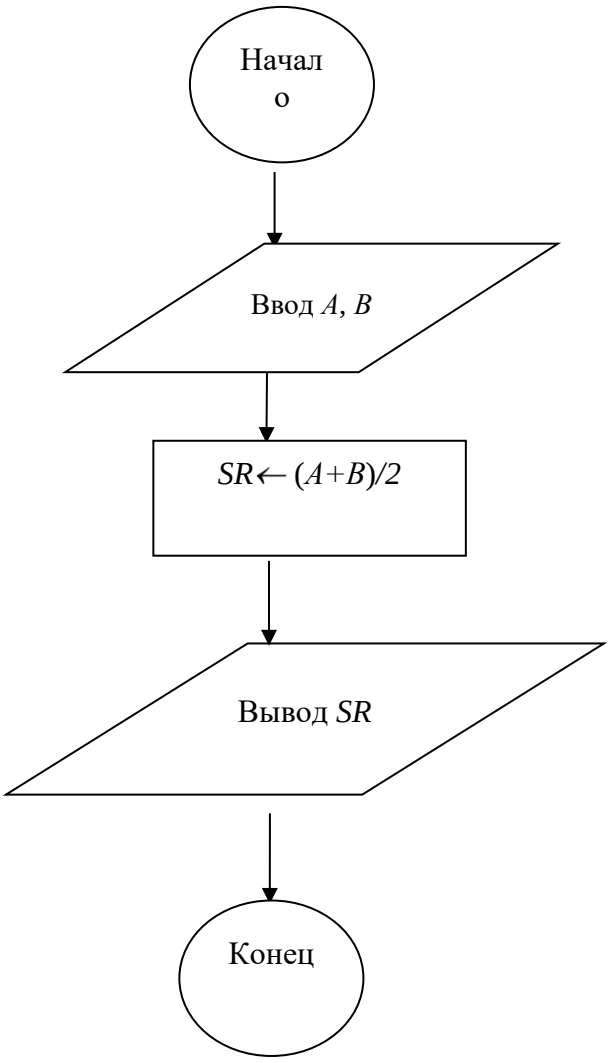
На рисунке III.3.2.C показано представление следования на искусственном вербальном языке:

: **S** тело сложного действия
НАЧАЛО
< ***S1*** >
< ***S2*** >
..
< ***SN***>
КОНЕЦ

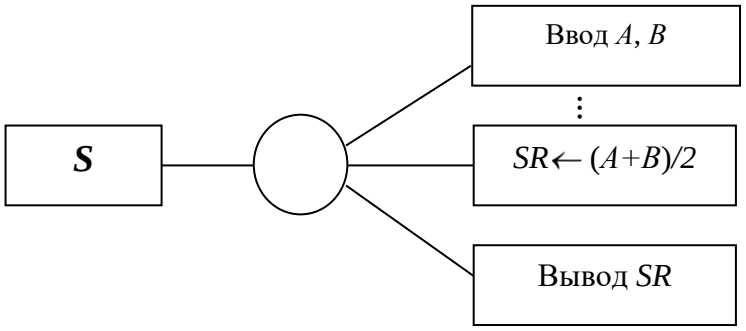
Рисунок III.3.2.C. Следование на вербальном языке.

Примеры III.3.2. Представление алгоритма нахождения среднеарифметического заданных двух чисел:

1. на языке блок-схемы:



2. на древовидном языке:



3. на вербальном языке:

Алг среднеарифметическое
Начало
Вещественное *A*, *B*, *SR*
Ввод *A*, *B*
SR:=(*A*+*B*)/ 2
Вывод *SR*
Конец

Задания III.3.2.

1. Представьте на древовидном языке алгоритм вычисления дискриминанта квадратного уравнения.

2. Представьте на вербальном языке алгоритм вычисления площади прямого треугольника с основанием *B* и с высотой *H*.

3. Представьте на языке блок-схем алгоритм вычисления площади окружности.

Помощь:

1. Дискриминант квадратного уравнения *D* вычисляется по формуле: $D = B^2-4*A*C$, где *A*, *B*, *C* – коэффициенты.

2. Площадь прямого треугольника *S* вычисляется по формуле:

$$S = B*H/2.$$

3. Если радиус окружности обозначить через *R*, то её площадь *S* вычисляется по формуле:

$$S = \pi*R^2, \text{ где } \pi = 3{,}14.$$

Вопросы III.3.2.

1. Как будет представляться на языке блок-схем сложное действие *S* из последовательности простых действий S1, S2,..., SN?
2. Как будет представляться на вербальном языке сложное действие *S* из последовательности простых действий S1, S2,..., SN?
3. Что считаются исходными данными и результатом для сложного действия ***S***, образованного из последовательности простых действий S1, S2,..., SN?

Тесты III.3.2.

1. Какие виды имеются у алгоритмических структур управления?

A) Следование, ветвление, простое;
B) Простое, ветвление, повторение;
C) Следование, сложное, повторение;
D) Следование, ветвление, повторение;
E) Простое, сложное, следование.

2. Для образования следовании минимально сколько действий берется?

A) 5;
B) 3;
C) 2;
D) 4;
E) 1.

3. Какой вид структуры управления нужно применить для вычисления суммы двух чисел?

A) Следование;
B) Простое;
C) Повторение;
D) Ветвление;
E) Альтернативное.

III.3.3. Простое ветвление

Сложное действие ***S*** простого ветвления показывает, что в зависимости от условия ***B*** выполняется действие ***S1*** или ничего не выполняется. Иначе говоря, в случае истинности условия ***B*** выполняется действие ***S1***, а в противном случае ничего не выполняется.

На рисунке III.3.3.A показана блок-схема простого ветвления:

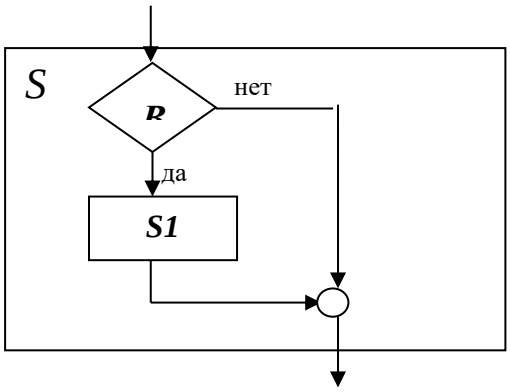


Рисунок III.3.3.A. Блок-схема простого ветвления.

На рисунке III.3.3.B показано простое ветвление на древовидном языке:

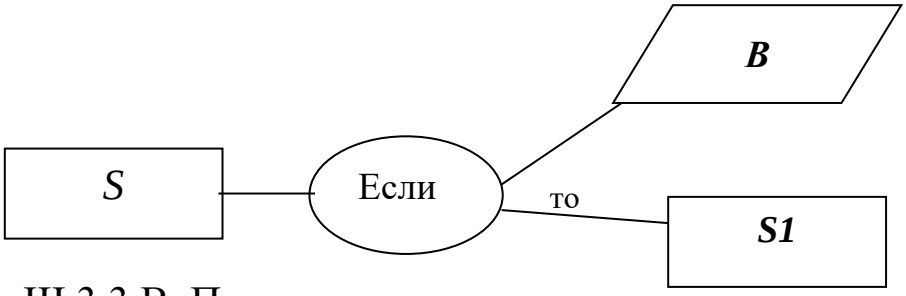


Рисунок III.3.3.B. Простое ветвление на древовидном языке.

На рисунке III.3.3.C показано простое ветвление на искусственном вербальном языке:

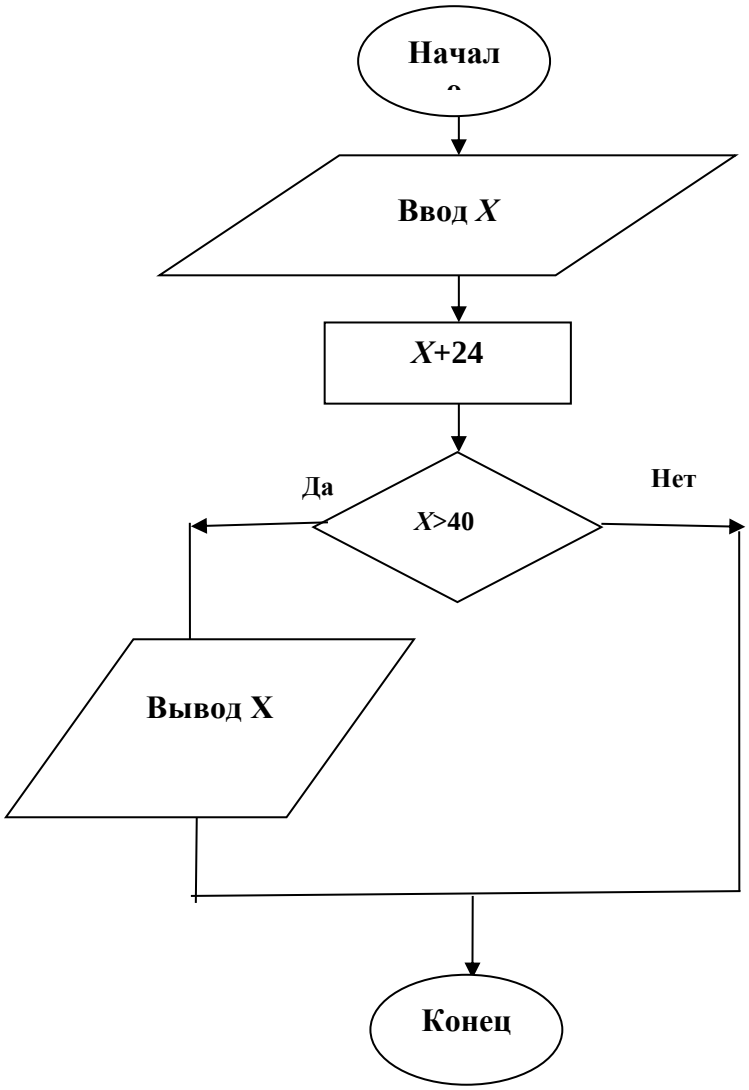
: ***S*** тело сложного действия

ЕСЛИ <***B*** > ТО < ***S1*** >

Рисунок II.3.3.C. Простое ветвление на вербальном языке.

Примеры III.3.3.

Блок-схема нахождения значения *X* при выполнении условия *X*+24>40 показана ниже:



Задания III.3.3.

1. Постройте блок-схему для размещения X и Y в порядке возрастания.
2. Постройте алгоритм размещения X и Y в порядке возрастания на древовидном алгоритмическом языке.
3. Напишите алгоритм размещения X и Y в порядке возрастания на искусственном алгоритмическом языке.

Помощь:

Если переменные удовлетворяют условию X < Y , то оставляете без измения; если X > Y, то нужно поменять их местами.

Вопросы III.3.3.

1. Как представляется простое ветвление на вербальном алгоритмическом языке?

2. Как изображается простое ветвление на древовидном алгоритмическом языке?

3. Как изображается простое ветвление на алгоритмическом языке блок-схем?

Тесты III.3.3

1. Какой алгоритм применяется для нахождения большего среди двух чисел?

A) линейный;

B) циклический;

C) ветвление;

D) иерархический;

E) рекурсивный.

2. Какие виды имеются у алгоритмических структур управления ветвления?

A) повторение пока, повторение до, повторение для;

B) ветвление, следование, повторение;

C) простое ветвление, альтернативное ветвление, многозначное ветвление;

D) простое ветвление, повторение, повторение до;

E) повторение, следование.

3. Как выполняется алгоритмическая структура управления ветвление?

A) Несколько раз выполнение некоторых команд;

B) Выполнение некоторых команд в зависимости от условия;

C) Вычисление табличной функции;

D) Составление различных случаев для игры;

E) Дать задания в двух алгоритмах.

III.3.4. Альтернативное ветвление

Сложное действие ***S*** альтернативного ветвления показывает, что в зависимости от условия ***B*** выполняется действие ***S1*** или выполняется ***S2***. Иначе говоря, в случае истинности условия ***B*** выполняется действие ***S1***, иначе выполняется действие ***S2***.

На рисунке III.3.4.A блок-схема альтернативного ветвления:

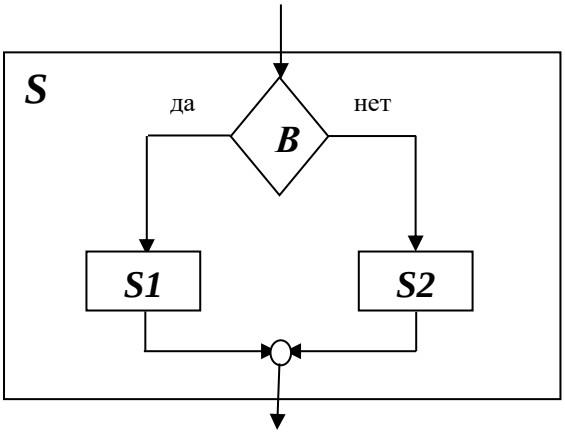


Рисунок III.3.4.A. Блок-схема альтернативного ветвления

На рисунке III.3.4.B показано альтернативное ветвление на древовидном языке:

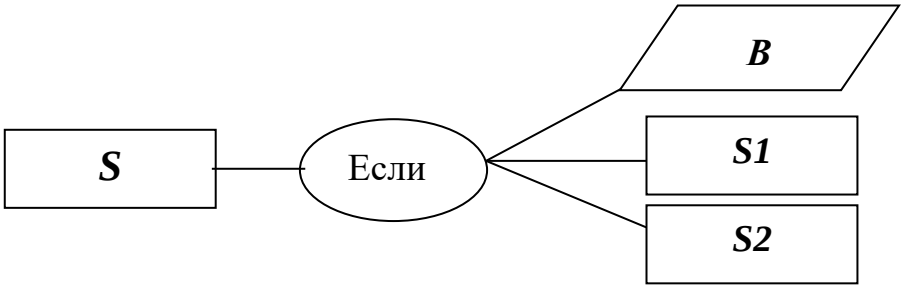


Рисунок III.3.4.B. Альтернативное ветвление на древовидном языке

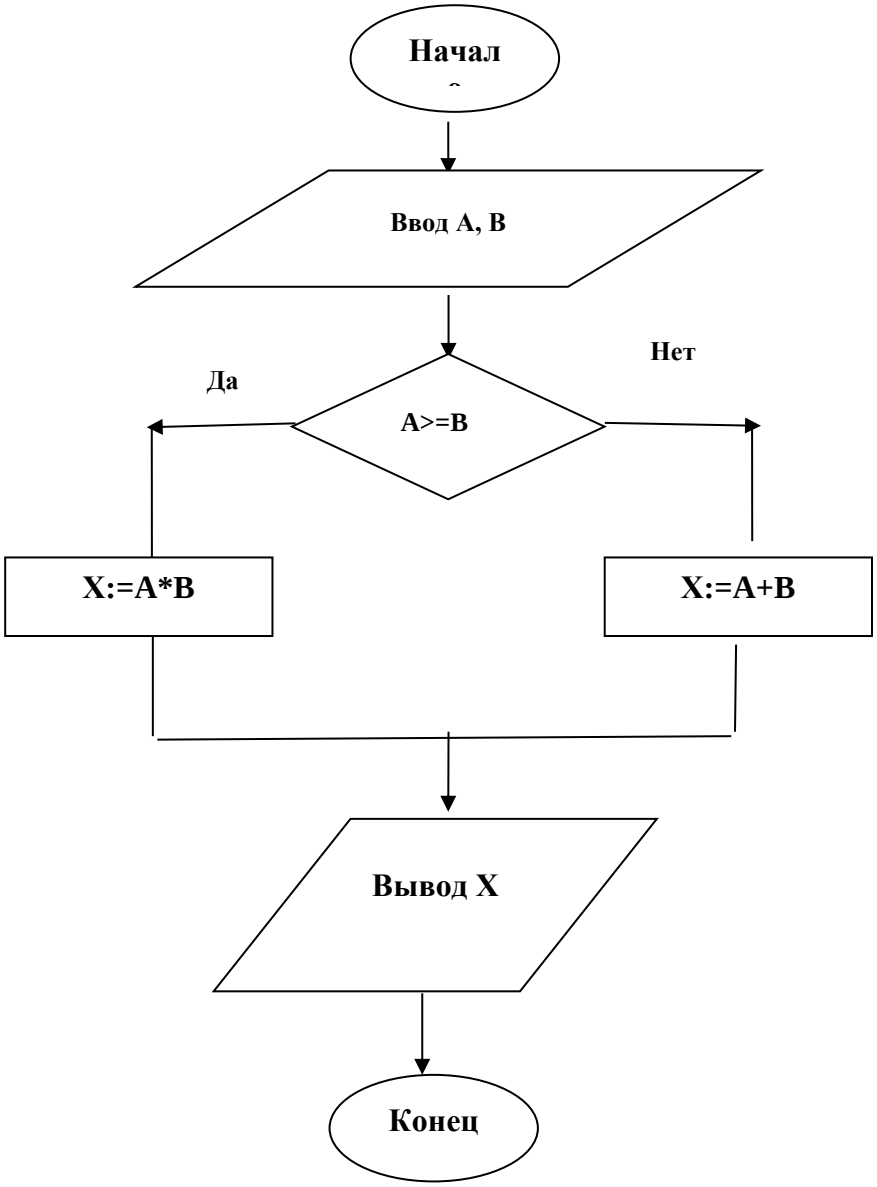
На рисунке III.3.4.C показано альтернативное ветвление на искусственном вербальном языке:

: **S** тело сложного действия
ЕСЛИ <***B*** > ТО < ***S1*** >
ИНАЧЕ < ***S2*** >

Рисунок III.3.4.C. Альтернативное ветвление на вербальном языке.

Примеры III.3.4.

В алгоритме *A* и *B* – исходные данные, *X* – результат. Если *A*>=*B* истина, то выполняется команда *X*:=*A***B*, иначе выполняется команда *X*:=*A*+*B*. В результате выводится значение *X*.

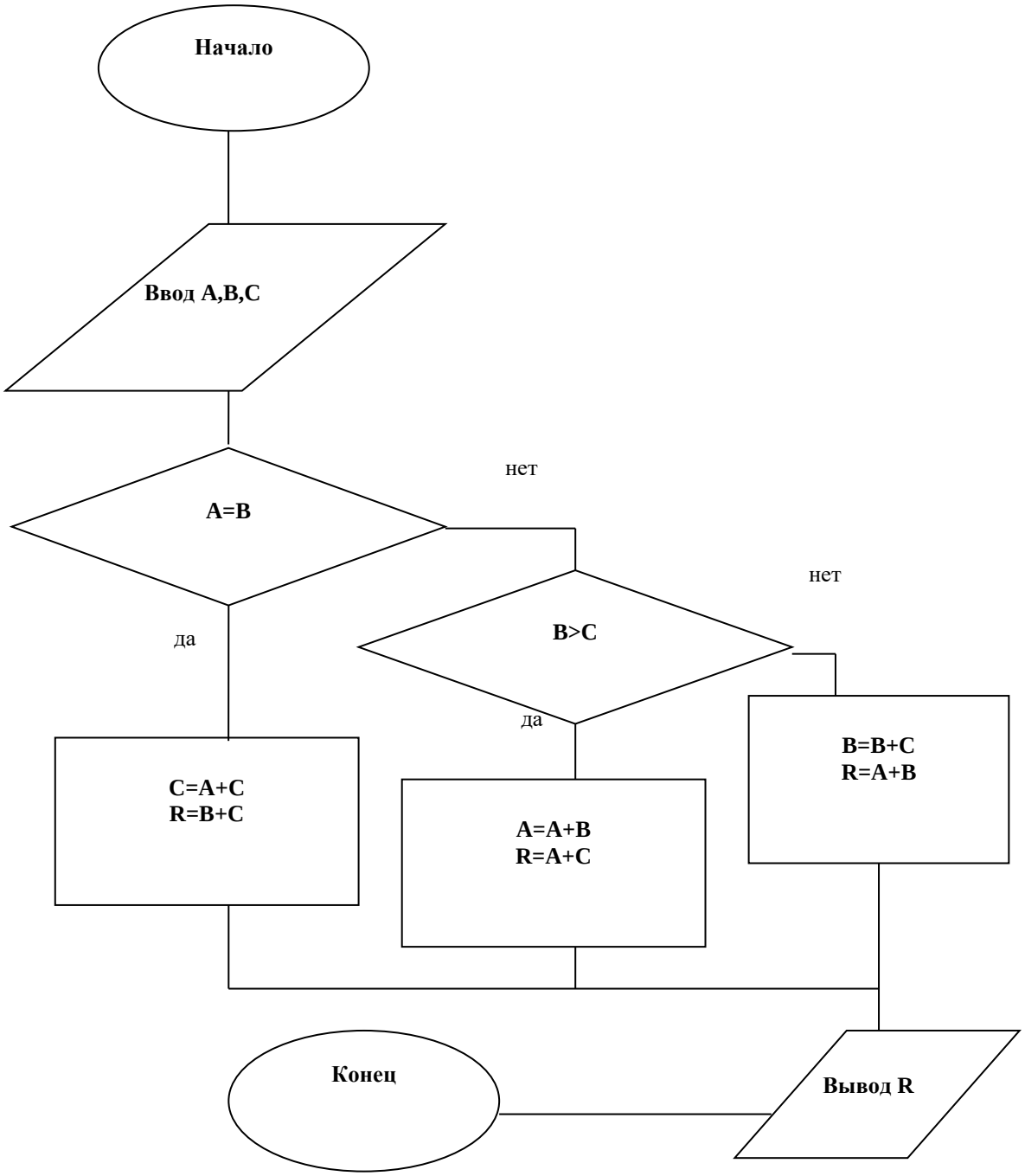


Задания III.3.4.

1. На вербальном языке написать ветвление, которое имени *A* присваивает большее число, а имени *B* - меньшее число.
2. Какому виду ветвления относится это:

: **S** тело сложного действия
ЕСЛИ <***B*** > ТО < ***S1*** >
ИНАЧЕ < ***S2*** >

3. Определить значение R, если в нижнем алгоритме значения A,B,C являются 1, 2, 6 соответственно.



Помощь:

1. По условию два действия должны выполняться альтернативно.
2. Альтернатиное ветвление.
3. Нужно выполнять действия в альтернативных ветвлениях.

Вопросы III.3.4.

1. Как изображается альтернативное ветвление на языке блок-схем?

2. Как изображается альтернативное ветвление на древовидном языке?

3. Как представляется альтернативное ветвление на вербальном языке?

Тесты III.3.4.

1. Сколько ветвей в структуре управления альтернативное ветвление?

A) 2;
B) 4;
C) 1;
D) 3;
E) 0.

2. Когда выполняется альтернативное действие в структуре управления ветвление?

B) если условие справедливо;
C) если условие истино;
D) если условие ложно;
E) если условие не правильно.

3. Сколько простых ветвлений нужно для моделирования альтернативного ветвления?

A) Два;
B) Один;
C) Неизвестно;
D) Три;
E) Более двух.

III.3.5. Многозначное ветвление

Сложное действие ***S*** многозначного ветвления (выбора) показывает, что переменная величина ***V*** обязательно примет одно из возможных значений ***1***, ***2,..***, ***N*** и в соответствии с этим значением обязательно выполняется одно из действий ***S1***, ***S2,..***, ***SN***. Иначе говоря, если ***V***=1, то выполняется ***S1***, если ***V=2***, то выполняется ***S2***,..., если ***V=N–1***, то выполняется ***SN–1***, иначе выполняется ***SN***.

На рисунке III.3.5.А показано многозначное ветвление на языке блок-схем:

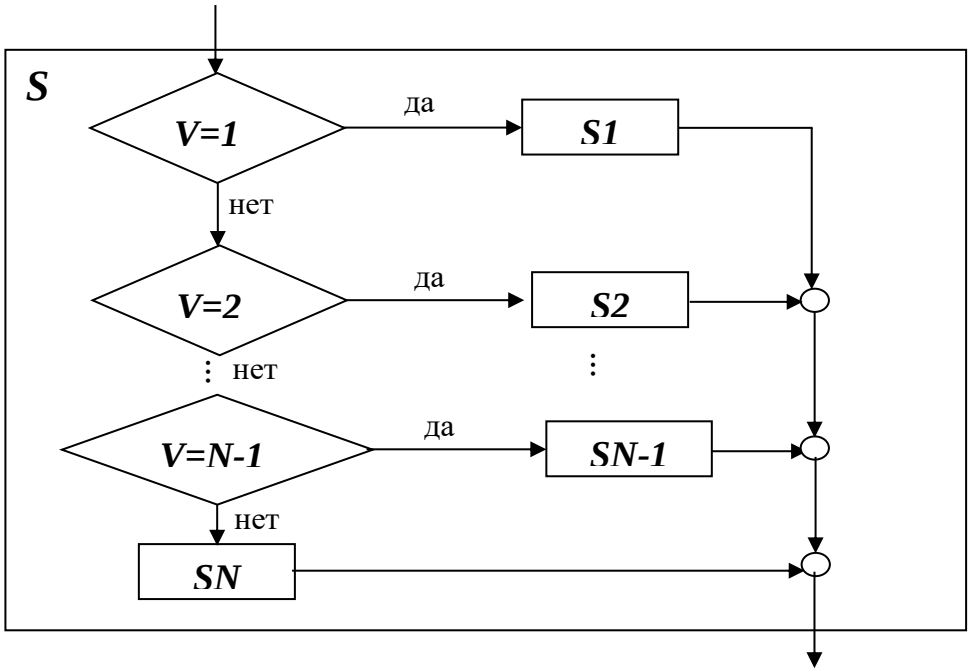


Рисунок III.3.5.А. Многозначное ветвление на языке блок-схем.

На рисунке III.3.5.В показано многозначное ветвление на вербальном языке:

: ***S*** тело сложного действия
ВЫБОР
V= 1 ТОГДА ***S1***
V = 2 ТОГДА ***S2***
. . . . . .
V = N ТОГДА ***SN***

Рисунок III.3.5.В. Многозначное ветвление на вербальном языке.

На рисунке III.3.5.C показано многозначное ветвление на древовидном языке:

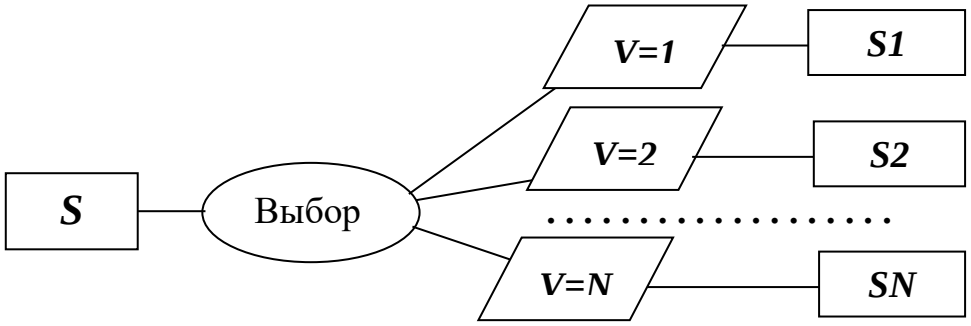


Рисунок III.3.5.C. Многозначное ветвление на древовидном языке

Если в вербальном языке не предусмотрена специальная конструкция многозначного ветвления (например, Выбор), то его можно выразить с помощью альтернативного ветвления. Такая запись многозначного ветвления показана на рисунке III.3.5.D:

: ***S*** тело сложного действия
ЕСЛИ ***V = 1*** ТО <***S1*** >
ИНАЧЕ ЕСЛИ ***V = 2*** ТО <***S2*** >
ИНАЧЕ ЕСЛИ ***V = 3*** ТО <***S3*** >
. . . . . .
ИНАЧЕ ЕСЛИ ***V = N-1*** ТО <***SN-1*** >
< ***SN*** >

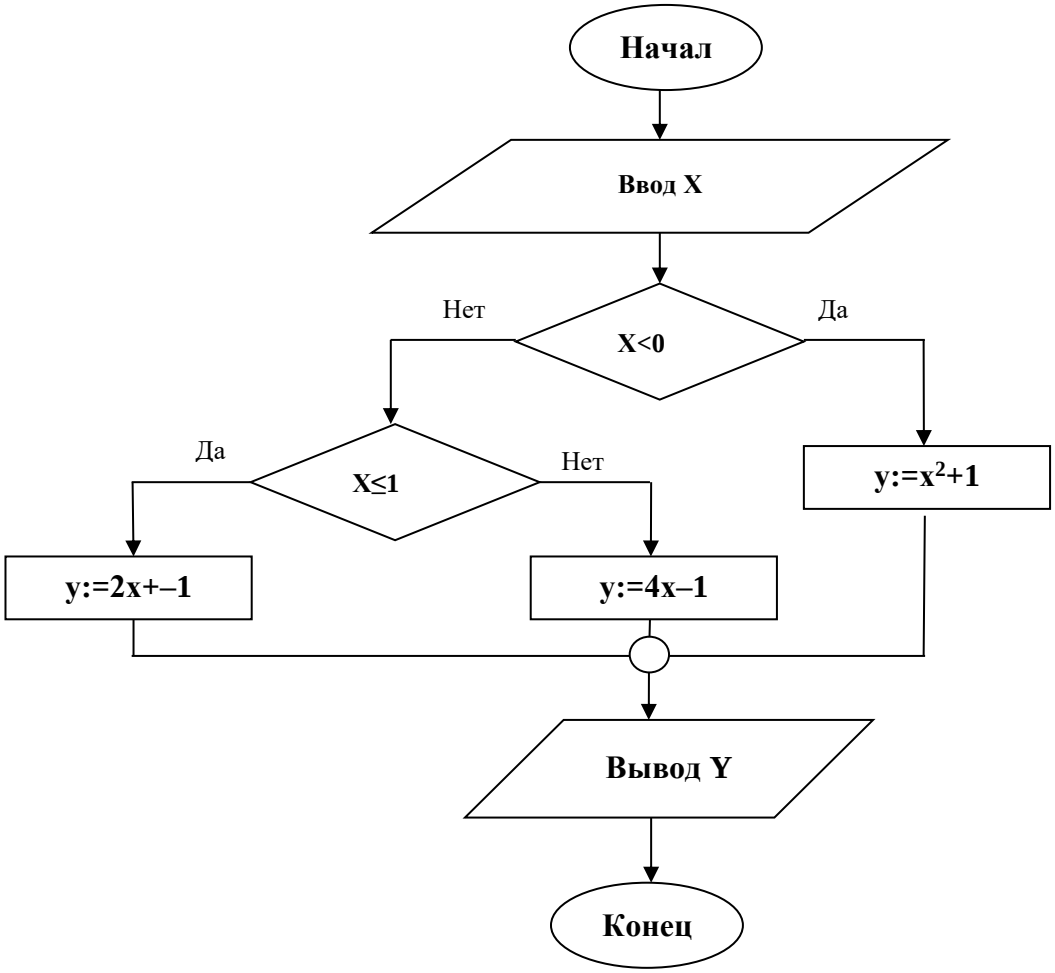
Рисунок III.3.5.D. Выбор с помощью альтернативного ветвления.

Примеры III.3.5.

Рассмотрим пример построения блок-схемы для определения значения заданной функции:

$$y = \begin{cases} x^2 + 1, & x < 0 \\ 2x + 1, & 0 \le x \le 1 \\ 4x - 1, & x > 1 \end{cases}$$

Этот алгоритм на языке-блок схем изображается так:



Задания III.3.5.

1. Построить блок-схему для определения большего из заданных трех чисел.

2. Постройте блок-схему алгоритма, выводящий название введенных чисел от 1 до 3.

3. Покажите какое ветвление представлено следующей записью:

ЕСЛИ <условие 1>
 ТО <команда 1>
  ЕСЛИ <условие 2>
   ТО <команда 2>
 .....
  ЕСЛИ <условие N>
  ТО <команда N>

Помощь:

1. Действие ***S*** будет одним из действий ***S1***, ***S2,..***, ***SN*** в зависимости от значения переменной ***V***.

2. Нужно применить самое сложное из трех видов ветвления.

3. Нужно применить многозначное ветвление для сравнения чисел от 1 до 3.

Вопросы III.3.5.

1. Как представляется выбор на языке блок- схем?
2. Как представляется выбор на вербальном языке?
3. Как представляется выбор на древовидном языке?

Тесты III.3.5.

1. Какие виды ветвления имеются?

A) пока повторение, повторение до, повторение для;

B) ветвление, следование, повторение;

C) простое ветвление, альтернативное ветвление, многозначное ветвление;

D) простое ветвление, повторение, повторение для;

E) повторение, следование.

2. В каком ветвлении проверяется только одно условие ?

A) Простое ветвление;

B) Альтернативное ветвление;

C) Многозначное ветвление;

D) Выбор;

E) Вложенное ветвление.

3. Пусть F(x) = $\begin{cases} x, & \textit{если} \quad x > 0 \\ 0, & \textit{если} \quad x = 0 \\ -x^2, & \textit{если} \quad x < 0 \end{cases}$

С помощью какой структуры управления находится значение функции?

A) ветвление;

B) параметрическое повторение;

C) выбор;

D) пока повторение;

E) следование.

III.3.6. Повторение с предусловием

Сложное действие ***S*** *повторения с предусловием* показывает, что в зависимости от условия ***B*** повторяется выполнение действия ***S1*** или ничего не выполняется. Иначе говоря, в случае истинности условия ***B***, повторяется выполнение действия ***S1***, а в противном случае ничего не выполняется.

На рисунке III.3.6.A показано представление повторение с предусловием на языке блок-схем:

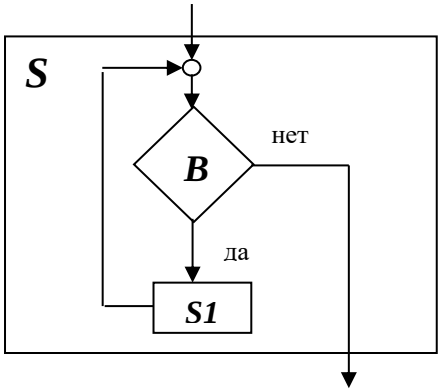


Рисунок III.3.6.A. Повторение с предусловием на языке блок-схем.

На рисунке III.3.6.B показано представление повторение с предусловием на древовидном языке:

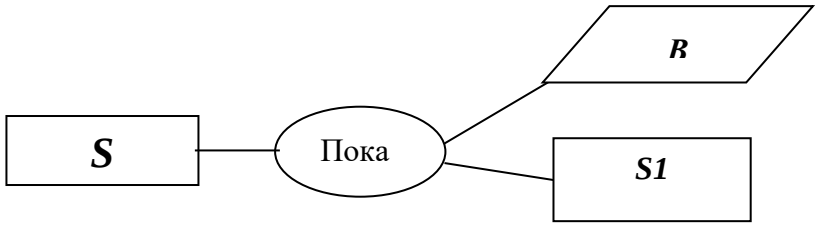


Рис. III.3.6.B. Повторение с предусловием на древовидном языке.

На рисунке III.3.6.C показано представление повторение с предусловием на вербальном языке:

: ***S*** тело сложного действия
ПОКА <***B***>
ПОВТОРЯТЬ <***S1***>

Рис. III.3.6.C. Повторение с предусловием на вербальном языке.

Итак по этой структуре управления возможны следующие:

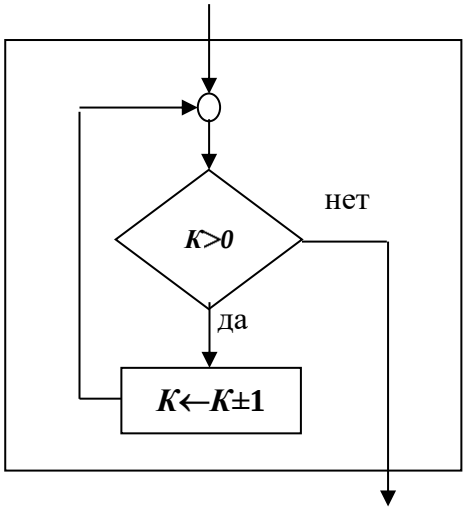
1. Если условие ***B*** перед началом выпонения сложного действия ***S*** не удовлетворяется, то действие ***S1*** не выполняется вообще;

2. Если условие ***B*** перед началом выполнения сложного действия ***S*** удовлетворяется и выполнение действия ***S1*** не влияет на значение условия ***B***, то выполнение действия ***S1*** требует бесконечного повторения;

3. Если условие ***B*** перед началом выполнения сложного действия ***S*** удовлетворяется и выполнение действия ***S1*** влияет на значение условия ***B***, то действие ***S1*** выполняется несколько раз (повторяется выполнение действия ***S1***) пока не изменится значение условия ***B***;

Примеры III.3.6.

Пусть задана конструкция повторения с предусловием вида:



1. Если вместо знака двойной операции “±” взять знак операции плюс “+”, то тело повторения ***K←K+1*** не влияет на условие ***K>0***:

Случай 1: если значение переменной величины ***K*** заранее не будет положительным числом, то есть ***K<0***, то тело повторение никогда не выполняется;

Случай 2: если значение переменной величины К заранее будет положительным числом, то есть ***K>0***, то выполнение тела повторения выполняется бесконечно.

2. Если вместо знака двойной операции “±” взять знак операции минус “-”, то тело повторения ***K←K+1*** влияет на заданное условие ***K >0***. Поэтому, если значение переменной величины ***K*** будет заранее положительным числом, то выполнение тела повторения будет повторяться пока значение переменной ***K*** не достигнет нуля или не станет отрицательным, то есть пока не станет ***K≤0*** (Замечание 3);

Задания III.3.6.

Построить алгоритм вывода на экран чисел от 1 до 10.

Помощь:

Примените структуру управления пока повторение.

Вопросы III.3.6.

1. В чем смысл повторения с предусловием?

2. Какие случаи имеются в повторении с предусловием?

3. Что произойдет, если в повторении с предусловием проверяемое условие выполняется не зависимо от тела повторения?

Тесты III.3.6.

1. Если перед началом выполнения сложного действия ***S*** условие ***B*** не удовлетворяется, то что будет с действием ***S1***?

A) выполняться;
B) разветвляться;
C) повторяться;
D) изменяться;
E) не выполняться

2. Как называется алгоритм, который повторяет выполнение отдельных команд или группы команд.?

A) Ветвящий;
B) Линейный;
C) Циклический;
D) Последовательный;
E) Рекурентный.

3. К какому виду относится алгоритм Евклида?

A) Ветвящий;
B) Линейный;
C) Циклический;
D) Последовательный;
E) Рекурсивный.

III.3.7. Повторение с постусловием

Сложное действие ***S*** *повторения с постусловием* показывает, что в зависимости от условия ***B*** повторяется выполнение или прекращается выполнение действия ***S1***. Иначе говоря, в случае ложности условия ***B***, повторяется выполнение действия ***S1***, а в противном случае повторение действия ***S1*** прекращается.

На рисунке III.3.7.A показано представление повторение с постусловием на языке блок-схем:

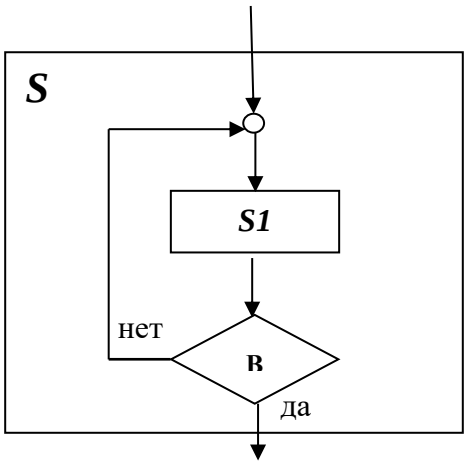


Рисунок III.3.7.A. Блок-схема повторение с постусловием.

На рисунке III.3.7.B показано представление повторение с постусловием на древовидном языке:

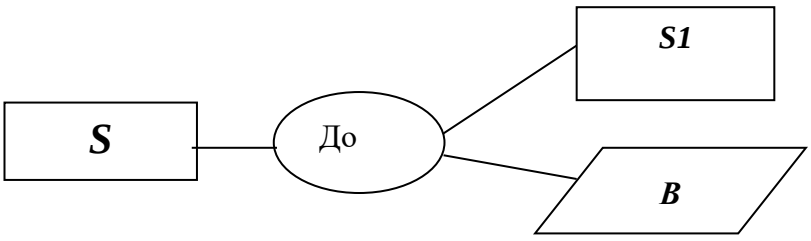


Рис. III.3.7.B. Повторение с постусловием на древовидном языке.

На рисунке III.3.7.C показано представление повторение с предусловием на вербальном языке:

: ***S*** тело сложного действия
ПОВТОРЯТЬ <***S1***>
ДО <***B***>

Рисунок III.3.7.C. Повторение с постусловием на вербальном языке.

Итак по этой структуре управления возможны следующие:

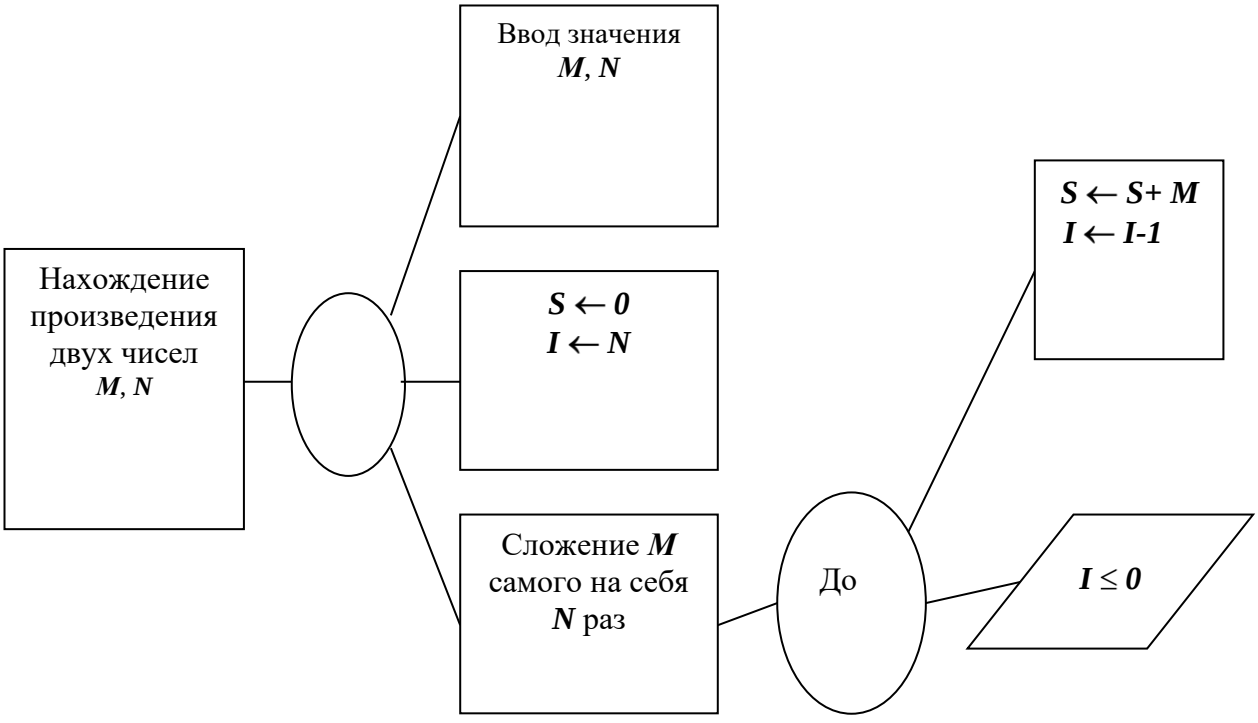
1. Если перед началом выполнения сложного действия ***S*** выполнение действия ***S1*** не влияет на значение условия ***B*** и условие ***B*** удовлетворяется, то действие ***S1*** выполняется только один раз;

2. Если перед началом выполнения сложного действия ***S*** выполнение действия ***S1*** не влияет на значение условия ***B*** и условие ***B*** не удовлетворяется, то выполнение действия ***S1*** повторяется бесконечно;

3. Если перед началом выполнения сложного действия ***S*** выполнение действия ***S1*** влияет на значение условия ***B*** и условие ***B*** не удовлетворяется, то выполнение действия ***S1*** повторяется несколько раз до изменения значения условия ***B***.

Примеры III.3.7.

Построить алгоритм умножения двух целых чисел путем сложения одного из них самого на себя столько раз, пока оно не станет равным второму числу. Заведем переменную ***S*** для сбора суммы, а переменную ***I*** для накопления количества выполняемых операции сложения. Тогда алгоритм нахождения произведения двух заданных ***M*** и ***N*** на древовидном языке будет следующим:



Здесь выполнение действия ***S*** ← ***S+ M*** повторится ***N*** раз, потому что исходное значение ***I*** равно ***N***, т.е ***M*** слагается сам на себя ***N*** раз.

Задания III.3.7.

1. Определить значение переменной ***S*** после выполнения следующего алгоритма:

```
АЛГ СУММА
    НАЧАЛО
        N=3, I=1, S=0
        M: S=S+(3*I+2)
        I=I+1
      ЕСЛИ I<=N ТО ПЕРЕЙТИ M
    КОНЕЦ
```

2. Определить значение переменной ***S*** после выполнения следующего алгоритма:

```
  АЛГ ДИСКР
     A=2, B=5, C=3
     НАЧАЛО
    D ← B*B - 4*A*C
   ВЫВОД D
   КОНЕЦ
```

Помощь:

1. Если перед началом выполнения сложного действия ***S*** условие ***B*** не удовлетворяется, то ***S1*** не выполняется;

2. Если перед началом выполнения сложного действия ***S*** условие ***B*** удовлетворяется и выполнение ***S1*** не влияет на ***B***, то ***S1*** требует бесконечного повторения выполнения;

3. Если перед началом выполнения сложного действия ***S*** условие ***B*** удовлетворяется и выполнение ***S1*** влияет на ***B***, то выполнение ***S1*** повторяется до изменения значения условия ***B***.

Вопросы III.3.7.

1. Как выполняется повторение с постусловием?

2. Как выглядит блок-схема повторения с постусловием?

3. Что произойдет при выполнении, если условие не поменяет свое значение?

Тесты III.3.7.

1. Какие из них являются базовыми видами циклов в алгоритме?

A) «До цикл», «Пока цикл», «Для цикл»;

B) «Внешний цикл», «Внутренний цикл»;

C) « Рекурсивный цикл», «Конечный цикл»;

D) «Повторный цикл», «Вложенный цикл»;

E) «Замкнутый цикл», «Процедурный цикл».

2. Что называется повторением (циклом) в записи произвольного алгоритма?

A) Выполнение команд в определенном порядке много раз;

B) Часто повторяющиеся некоторые команды;

C) Вычисление табличной функции;

D) Построение различных случаев в игре;

E) Выдача заданий двум алгоритмам.

1. В следующем алгоритме, если Q=2, то какой будет результат вычисления?

```
ВВОД Q
        S=0, I=1
        M:   S=S+I; I=I+1
        ЕСЛИ S<=Q ТО ПЕРЕЙТИ M
ВЫВОД I-2
```

A) 0;

B) 1;

C) 2;

D) 3;

E) 4.

III.3.8. Параметрическое повторение

В структуре управления параметрическое повторение число выполнений тела повторения определяется начальным и конечным значениями переменной величины, называемой параметром, и постоянной величиной, называемой шагом: при каждом выполнении тела повторения к начальному значению параметра добавляется значение шага (если не показано значение шага, то добавляется 1) до достижения конечного значения параметра.

Обычно, начальное и конечное значения параметра, значение шага являются целыми числами. Поэтому количество повторения равно числу, полученному добавлением единицы целой части от деления разницы конечного значения и начального значения на значение шага. Например, если 5 – это начальное значение параметра, 20 – конечное значение параметра, 2 – значение шага, то число повторения вычисляется по формуле [(20–5) : 2] + 1= 8, где в квадратной скобке находится целая часть от деления 15 на 2.

Вообще, параметрическое повторение является частным случаем повторения с предусловием и повторения с постусловием. Поэтому параметрическое повторение не представляется в графических языках. Однако во многих вербальных языках для него имеются специальные представления. Параметрическое повторение на вербальном языке представлено на рисунке III.3.8:

: ***S*** тело сложного действия

ДЛЯ <***V=I***, ***N***> ШАГ <***K***>
ПОВТОРЕНИЕ <***S1***>

Рисунок III.3.8. Параметрическое повторение на вербальном языке.

Здесь ***V*** – параметр, ***I*** – начальное значение параметра, ***N*** – конечное значение параметра, ***K*** – шаг, ***SI*** – тело повторения. Все время должно быть ***I*** < ***N***. Шаг можно не показывать. Если шаг не показывается, то считается, что его значение равно 1.

Примеры III.3.8.

1. Алгоритм вычисления суммы нечетных от 1 до 100 будет так:

: Сумма нечетных чисел
S ← ***0***
ДЛЯ ***V=1,100*** ШАГ ***2***
ПОВТОРЕНИЕ ***S*** ←***S+V***

2. Рассмотрение повторения могут быть вложены в друг друга, как обычные скобки:

: Вложенные повторения
S ← ***0***
: Внешнее повторение
ДЛЯ ***L=1,4***
ПОВТОРЕНИЕ
    : Внутреннее повторение
    ДЛЯ ***I*** = ***1,5***
    ПОВТОРЕНИЕ
    S ← ***S + L * I***

Здесь выражение ***S*** ← ***S + L * I*** выполняется по внутреннему повторению 5 раз, а по внешнему повторению – 4 раза. Иначе говоря, всего выполняется 20 раз и оставляет следующий след:

I. ***L = 1***

1. ***I*** = 1, S ← 0 + 1;
2. ***I*** = 2, S ← 1 + 2;
3. ***I*** = 3, S ← 2 + 3;
4. ***I*** = 4, S ← 5 + 4;
5. ***I*** = 5, S ← 9 + 5.

II. ***L = 2***

6. ***I*** = 1, S ← 14 + 2;
7. ***I*** = 2, S ← 16 + 4;
8. ***I*** = 3, S ← 18 + 6;
9. ***I*** = 4, S ← 26 + 8;

10. ***I*** = 5, S ← 34 + 10.

III. ***L = 3***

11. ***I*** = 1, S ← 44 + 3;
12. ***I*** = 3, S ← 47 + 6;
13. ***I*** = 3, S ← 53 + 9;
14. ***I*** = 4, S ← 62 + 12;
15. ***I*** = 5, S ← 74 + 15.

IV. ***L = 4***

16. ***I*** = 1, S ← 89 + 3;
17. ***I*** = 3, S ← 92 + 6;
18. ***I*** = 3, S ← 98 + 9;
19. ***I*** = 4, S ← 107 + 12;
20. ***I*** = 5, S ← 119 + 15.

III.3.8. Задания.

1. Построить алгоритм для нахождения наименьшего числа среди введенных 15 чисел.
2. Построить алгоритм для нахождения суммы нечетных чисел от 1 до вашего года рождения, задаваемого переменной Y.
3. Построить алгоритм для нахождения факториала числа 100.

Помощь:

1. Использовать параметрическое повторение до 15 и проводить попарное сравнение.
2. После ввода значения переменной Y, повторять выполнение сложения до значения Y с шагом 1.
3. Повторять выполнение умножения чисел с использованием предела 100 и шага 1.

III.3.8. Вопросы

1. В каких случаях в параметрическом цикле повторение не выполняется?

2. Каким образом работает в алгоритме параметрическое повторение?

3. В каком случае для вычисления применяют параметрическое повторение?

Тесты III.3.8.

1. Какой из этих действий является не повторяющим (циклическим)?

A) решение квадратного уравнения;

B) найти наибольшее среди N чисел;

C) расположить в порядке возрастания N чисел;

D) уничтожить по одному все шары, вынутых из коробки;

E) Игра Баше.

2. Каким алгоритмом вычисляется сумма заданных чисел от 1 до 100?

A) ветвление;

B) параметрическое повторение;

C) выбор;

D) пока повторение;

E) линейный.

3.Какому виду алгоритмической структуры управления относится следующая запись?

ДЛЯ <***V=I,N***> ШАГ <***K***>
ПОВТОРЕНИЕ <***S1***>

A) ветвление;

B) параметрическое повторение;

C) выбор;

D) пока повторение;

E) линейный.

III.4. Языки программирования

Ключевые слова: *программа, машинный язык, язык программирования, транслятор, мнемокод, автокод, язык ассемблера, подпрограмма, макрос, операционная система, управляющая программа, служебная программа, рабочая программа, язык управления заданиями, язык запроса, машинно-зависимый язык, проблемно-ориентированный язык, универсальный язык, процедурный язык, функциональный язык, продукционный язык, логический язык, объектно-ориентированный язык, численные задачи, символьные задачи, логические задачи, экономические задачи, задачи моделирования.*

Цель: *рассматриваются классификация и свойства языков программирования.*

Структура: на рисунке III.4 показаны виды языков программирования

Рисунок III.4. Виды языков программирования.

III.4.1. Понятие программы и языка программирования

Известно, что в современных компьютерах необходимые для обработки данные и относящиеся к ним алгоритмы должны быть представлены с помощью цепочек из 0 и 1.

Представление *указания* и *операции* в алгоритме с помощью цепочек из 0 и 1 называют *командой*.

Структура команды показана на рисунке III.4.1:

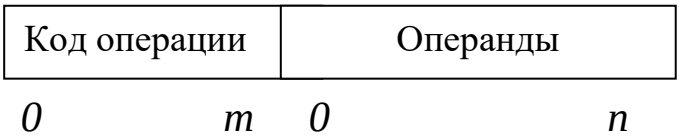


Рисунок III.4.1. Структура команды.

Здесь *код операции* представляет собой двоичный код той или иной операции (например, операцию сложения можно закодировать как $C1_{16} = 11000001_2$); *операнды* – представляют собой адреса исходных данных и результата операции; m, n – целые числа, указывающие количество битов, необходимых для размещения в памяти компьютера кода операции и операндов соответственно.

Каждая команда должна допустить свое выполнение (свою реализацию) аппаратными средствами компьютера. Множество команд, которых в принципе может выполнить компьютер называют *машинным языком* с алфавитом, состоящим только из 0 и 1. Представление алгоритма на машинном языке называется *программой*. Отсюда следует, что каждый компьютер может выполнить произвольную программу, написанную только на своем машинном языке.

Формальный (однозначный) язык, на котором может быть написана произвольная программа, называется *языком программирования.* Поэтому машинный язык считается самым первым языком программирования.

Все команды на машинном языке выполняются с помощью входящих в состав компьютера функциональных средств. Возможности машинного языка определяют первоначальный интеллектуальный уровень компьютера. Затем этот уровень можно повышать, запоминая в памяти компьютера программы на машинном языке, решающие различные интеллектуальные задачи.

Следует отметить, что для человека программирование на машинном языке очень сложный из-за трудностей. Во-первых, запоминания двоичных кодов операций и значений абсолютных (физических) адресов памяти для размещения операндов и самих команд, во-вторых, перевода указаний и операций в алгоритме на машинный язык, в-третьих, представления исходных данных и результатов в двоичном виде, в-четвертых, просчитывания абсолютного адреса, куда передается управление при выполнении указания ветвления и повторения. Кроме того, требуется много раз выполнять заново ранее сделанные работы. Все это приводит к появлению многочисленных невидимых ошибок. Написанные программы становятся нечитабельными и непонятными. Следовательно, их отладка (нахождение и исправление ошибок) потребует неимоверного усилия и много времени, которые существенно повлияют на стоимости программных продуктов. Поэтому люди для облегчения своих трудов начали придумывать искусственные языки, которые позволяют легко и удобно записывать алгоритмы.

Однако, чтобы превратить в программу алгоритм, написанный на произвольном искусственном алгоритмическом языке, нужно его перевести на машинный язык. В ходе реализации такого перевода в науке информатики появилась новая отрасль. Она занимается разработкой новых формальных языков, методами формализации синтаксиса и семантики искусственных языков, разработкой способов трансляции и др.

Перевод осуществляется программой, называемой «транслятор». Это привело к тому, что при использовании для

записи алгоритмов на искусственных алгоритмических языках работа компьютера свелась к двум этапам:

1. Трансляция алгоритма с искусственного языка на машинный язык программирования;
2. Выполнение программы на машинном языке.

Есть два метода реализации работ, указанных в этих этапах:

1. *Компиляция* – сначала текст программы переводится (компилируется) с искусственного алгоритмического языка на машинный язык, затем эта программа выполняется сначала.
2. *Интерпретация* – каждая команда (указание или операция) в искусственном языке выполняется (интерпретируется) на машинном языке без предварительного перевода.

Итак, в методе компиляции алгоритм на искусственном языке рассматривается только один раз для перевода его в программу. При этом переведенную программу можно несколько раз выполнить. В методе интерпретации перед каждым выполнением программы нужно рассматривать заново алгоритм на искусственном языке. Поэтому полученная программа по методу компиляции требует много памяти для размещения всей переведенной программы, но она работает быстро. Наоборот, метод интерпретации требует мало места в памяти, но медленно работает.

Таким образом, такие искусственные языки становятся понятными компьютеру, поэтому их тоже можно называть языками программирования.

В настоящее время имеются многочисленные языки программирования, позволяющие с помощью компьютера решать самые разные задачи. У каждого языка программирования имеются свои достоинства и недостатки. Например, в некоторых языках, несмотря на то что легко и быстро удается составление программы, её выполнение на компьютере требует много времени или много места в памяти. Поэтому, прежде чем приступить к процессу программирования, нужно тщательно выбрать соответствующий

вашей задаче язык, который удовлетворял бы все требования, предъявляемые нормативными и проектными документами, по всему жизненному циклу составляемой программы.

Возможности программирования всегда были ограничены либо возможностями компьютера, либо возможностями человека. В прошлом веке главным ограничением были низкие производительные способности компьютера. В настоящее время физические ограничения отошли на второй план. С глубоким проникновением компьютеров во все сферы человеческой деятельности, программные системы становятся всё более простыми для пользователя и сложными по внутренней архитектуре. Программирование стало делом команды и на смену алгоритмическим идеологиям программирования пришли эвристические, позволяющие достичь положительного результата различными путями.

Примеры III.4.1.

1. В компьютерах IBM/360 и ЕС ЭВМ для кода операции отводились восемь битов, т.е. он записывался на одном байте.
2. Самый первый транслятор в мире создан в 1957 году в США, он автоматически переводил появивший в 1954 году язык Fortran на машинный язык.
3. Количество операндов в команде определяет число адресов компьютера. Например, если в командах имеются два операндов, то компьютер будет двухадресным.

Задания III.4.1.

1. Определить количество всех возможных команд в компьютерах IBM/360 и ЕС ЭВМ.
2. Перечислите достоинства и недостатки компилятора и интерпретатора.
3. Укажите этапы работы компьютера при использовании алгоритмических языков для записи алгоритма.

Помощь

1. Нужно просчитать, сколько различной, друг от друга, информации можно разместить в одном байте.
2. Нужно сравнить по количеству времени и по объему памяти.
3. Нужно перевести на машинный язык и выполнить.

Вопросы III.4.1.

1. Что такой транслятор?
2. У какого языка программирования самый низкий уровень?
3. Какой язык программирования основан на логическом исчислении?

Тесты III.4.1.

1. Какой язык имеет самый низкий уровень?

A) машинный;
B) процедурный;
C) функциональный;
D) логический;
E) продукционный.

2. Какой из них не идентификатор?

A) 1NOM;
B) GRUPP;
C) A5;
D) NOM1;
E) ABC.

3. Какой язык программирования предполагает представление алгоритма в виде вложенных абстрактных типов данных?

A) объектно-ориентированный;
B) логический;
C) продукционный;
D) процедурный;
E) функциональный.

III.4.2. Классификация языков программирования

Языки программирования классифицируются по следующим признакам и свойствам:

1) В зависимости от уровня языков, т.е. от степени близости языков к машинным языкам;

2) В зависимости от ориентированности языков на проблемы, т.е. от наличия возможности легко и удобно решать задачи определенной отрасли;

3) В зависимости от модели языков, т.е. от принципов, определяющих стиль составления программы, организацию вычислений и структурирование работы.

Принято, что самый низкий уровень – у машинных языков, а самый высокий уровень – у естествненных языков. Поэтому под уровнем другого языка называют степень его близости к машинным языкам или к естественным языкам. С одной стороны, чем ближе язык программирования находится к машинным языкам, то тем ниже его уровень, и наоборот. С другой стороны, чем ближе язык программирования находится к естественным языкам, то тем выше его уровень, и наоборот. Итак, в зависимости от уровня, языки программирования подразделяются на два класса:

- низкий уровень машинно-зависимых языков;
- высокий уровень машинно-независимых языков.

Машинно-зависимые языки с низким уровнем учитывают архитектурные особенности отдельных марок компьютеров или классов компьютеров. Поэтому они подразделяются на две группы:

- машинно-ориентированные специальные языки;
- машинно-ориентированные универсальные языки.

К машинно-ориентированным специальным языкам относятся *мнемокоды* и *автокоды*.

Переписанные двоичные коды всех команд, данных и их адресов в машинном языке на цепочки из букв и цифр в мнемокоде, называют *мнемоникой*. В качестве мнемоники можно взять первую букву или начальные буквы от имени. Мнемоника представляется

прописными буквами. Например, в качестве мнемоники операции сложения (вычитания), берется соответствующее имя операции латинская буква ***A*** (***S***), так как её английское имя *Adition* (*Subtract*). Соотношение мнемокода к машинному языку 1:1, т.е. имени каждой команды в машинном языке дается только одна мнемоника.

Автокод есть мнемокод, но в отличие от него, в автокоде, помимо использования мнемоник, имеются возможности организации *подпрограммы* и *макрооперациии*, последовательность нескольких простых операций. Например, при решении задачи, если встретится некая логически завершенная часть программы, то эту часть организовав как подпрограмму или макроопределение, можно применить её имя там, где это необходимо, и показать значение параметра в зависимости от контекста. Автокоды являются развитьем мнемокодов путем добавления возможностей организации и применения подпрограмм и макросов.

Объединение мнемокода и автокода называют *языком ассемблера*. Программа, которая автоматически переводит язык ассемблера на машинный называется *ассемблером*. Исходными данными ассемблера являются программы на мнемокоде или автокоде, а его результатом – программы на машинном языке.

В машинно-ориентированных универсальных языках имеются операции и указания, учитывающие особенности нескольких компьютеров одного класса. Они являются развитыми машинно-ориентированными языками. К ним относятся, язык *Almo* или *Uncol*, позже язык *C* и его развитие *C*++, а в конце язык Java.

Для планирования, организации и управления работы компьютера необходимо общение с ним. Все это вышесказанное реализуется с помощью выполнения некоторых программ. Объединив их называют *операционной системой* (ОС). ОС, во время выполнения решающей поставленной задачи рабочей программы, оказывает услуги по вводу, хранению и выводу необходимых ей данных. Кроме того ОС для этой рабочей программы выделяет необходимые оборудования, определяет очередь её исполнения и совершает другие работы. Среди программ ОС имеются

управляющие программы и служебные программы. Иначе говоря, компьютерные программы образуют «программное общество». В этом обществе для реализации общения между программами и запуска/отмены их работы применяется язык общения, который называется *языком операционной системы* или *языком управления заданиями.* С одной стороны, когда операционная система настраивается на определенную марку компьютера, эти языки охватывают особенности и свойства этой марки. Поэтому их можно отнести к классу языка общения с машиной. С другой стороны, языки некоторых операционных систем независимы от марки машин (например, в операционной системе *Unix*).

Машинно-независимые языки с высоким уровнем близки к естественным языкам. В основном они созданы для того, чтобы легко и понятно составлять алгоритмы. Можно сказать, что в них нет указаний и операций, учитывающих особенности компьютеров. Но программы на этих языках, по сравнению с программами на машинно-зависимых языках, требуют значительно много компьютерных ресурсов. Например, сохранение программы занимает много места в памяти, а выполнение - много процессорного времени.

Эти языки в зависимости от удобства решения задач подразделяются на следующие группы:

- языки для решения численных задач;
- языки для решения символьных задач;
- языки для решения логических задач;
- языки для решения экономических задач;
- языки для решения задач моделирования и т.д.

К численным задачам в основном относятся научно-технические задачи. Обычно, для решения этих задач используются *Fotran*, *Алгол*, *Бейсик*, *Pascal*, *Ада* и др.

К символьным задачам относятся задачи перевода с одного языка на другой, аналитические преобразования и т.д. Для решения таких задач применяются *Snobol*, *Lisp*, *Рефал* и др.

К логическим задачам относятся доказательства теорем, верификация и синтез программ, построение системы принятия решений и др. Для решения этих задач применяют такие языки, как *Пролог* и *Лого*.

К экономическим задачам в основном относятся задачи бухгалтерского учета, финансового анализа и статистического анализа и др., связанные с вычислениями над табличными данными и построениями различных диаграмм. Для этих были созданы специальные языки *Cobol, RPG*, а также *электронные таблицы*, как *Lotus* и *Excel* со своими встроенными языками общения. Для решения этих задач в середине 70-х годов XX века начали создавать *системы управления базами данных* (СУБД) со своими встроенными *языками запроса* и *языками манипулирования данными*. К СУБД относятся *dBase, Foxpro, Access, My SQL, Informix, Oracle, Сасне* и др.

К задачам моделирования в основном относятся задачи имитационного моделирования, связанные с нелинейными динамическими системами. Эти задачи требуют очень много компьютерных ресурсов. Поэтому в соответствующих языках дожны быть возможности распараллеливания решения задач. Для решения таких задач применялись *Simula, Dynam*.

Все рассмотренные языки относятся к *проблемно-ориентированным языкам*.

Были созданы универсальные языки программирования, которые пригодны для решения всех перечисленных выше классов задач. К таким языкам относятся *PL/I, Algol-68, Ada*. В этих языках определены все типы данных и операции над ними. Однако в этих языках объемы программ становились очень большими. Программы занимали много места в памяти и работали очень медленно.

Языки программирования различаются, как и естественные языки, в зависимости от своих моделей. Например, естественные языки подразделяются на тюркские, славянские, романские, арабские, персидские и др., а языки программирования подразделяются на *процедурные языки, функциональные языки,*

логические языки, *продукционные языки*, *объектно-ориентированные языки*.

Среди языков программирования самыми первыми созданными и самыми распространенными являются процедурные языки. К ним относятся *Fotran, Algol, Basic*, *Pascal, Ada*.

В процедурном языке алгоритм записывается представлением действий и необходимых для их выполнения указаний. В процедурных языках программными единицами считаются *операторы*. Самой маленькой единицей является *оператор присваивания*:

<переменная> ← ***<выражение>***,

где "←" – знак операции присваивания, он в разных языках представляется разным знаком, например, "=" – в языке *Бейсик*, ":=" – в языке *Pascal*. Выполнение оператора присваивания происходит в два этапа: сначала вычисляется значение выражения, затем это вычисленное значение присваивается переменной.

В каждом языке программирования *оператор ввода данных* и оператор *вывода данных* являются частными случаями оператора присваивания: при вводе данных в качестве переменной берется адрес оперативной памяти, а в качестве выражения – имя внешнего устройства или адрес внешней памяти, где набираются или находятся вводимые данные. При выводе данных наоборот: переменная – внешнее устройство или адрес внешней памяти, а выражение образуется с помощью адресов оперативной памяти.

Другими программными единицами являются операторы *последовательность*, *ветвление* и *повторение*.

Функциональные языки программирования базируются на лямбда исчислении, где алгоритм решаемой задачи записывается с помощью построения функций. Функции могут быть простыми или сложными. Сложная функция состоит из композиции (суперпозиции) простых функций. В функциональных языках программной единицей считается функция. Примерами языков функционального программировния являются *Lisp, Planer*, *Conniver*, *KRL*, *FRL* и *FP*.

Логические языки программирования базируются на логические исчисления, где алгоритм решаемой задачи записывается с помощью представления утверждений о свойствах и отношений её исходных данных и необходимых для их обработки операций. Каждое утверждение можно рассматривать как логическое правило и каждое правило дает одно условие. Результат решения задачи определяется с помощью запроса. Если по этому запросу докажем выполнимость логических правил, то задача имеет результат, иначе надо поставить новый запрос, либо поставленный запрос имеет отрицательный ответ. Для реализации этого применяют известную формальную дедукционную систему. К языкам логического программирования относятся *Prolog* и *Logo*.

Продукционные языки программирования базируются на нормальные алгорифмы Маркова, где алгоритм решаемой задачи представляется в виде схем продукционных правил (подстановок). Каждая подстановка состоит из двух частей - «левая часть подстановки» и «правая часть подстановки». В левой части подстановки размещают имя подстановки и образцы исходных данных, а в правой части указывается то что надо делать с исходными данными, которые соответствуют образцам из левой части. Для этого можно вызвать и другие подстановки, и самого себя. Допускается прямая рекурсия (вызов самого себя без посредников) и косвенная рекурсия (вызов самого себя через другие правила). Построение схемы подстановки осуществляется «сверху вниз». Сначала строится подстановка, соответствующая общему содержанию задачи. Затем строятся подстановки, соответствующие новым понятиям, которые появились при решении этой задачи. Иначе говоря, программы в этом языке имеют иерархическую (древовидную) структуру. Можно сделать проверку и вносить изменения на любом уровне иерархии. К этим языкам относятся *Snobol* и *Refal*.

Объектно-ориентированные языки программирования (ООПЯ) основаны на теории абстрактных типов данных, где алгоритм решаемой задачи представляется построением типов данных

(классов), состоящих из набора данных и операций над ними. Сначала строится глобальный тип (супер класс), соответствующий общей сущности задачи. Все остальные сущности, возникающие в ходе построения решения поставленной задачи являются подтипами (подклассами) или объектами глобального типа (экземпляр конкретного класса).

ООПЯ поддерживают принципы *абстрагирования данных*, *инкапсуляции*, *наследования*, *полиморфизма*, «позднего связывания».

Абстрагирование (*Abstracting*) – это принцип, позволяющий выделить набор значимых характеристик объекта, исключая из рассмотрения незначимые. Соответственно, абстракция — это набор всех таких характеристик.

Инкапсуляция (*Encapsulation*) – это принцип, позволяющий объединить данные и методы (код), работающие с ними в классе, и скрыть детали реализации от пользователя (защитить данные от прямого внешнего доступа и неправильного использования), что обеспечить доступ к данным класса только посредством методов этого же класса.

Инкапсуляция позволяет вносить изменения в части программы безболезненно для других её частей, что существенно упрощает сопровождение и модификацию программного обеспечения.

Наследование (*Inheritance*) – это принцип, позволяющий описать новый класс (объект) на основе уже существующего с частично или полностью заимствующейся функциональностью. Точнее, класс (объект) может наследовать основные свойства другого класса (объекта) и добавлять к ним свойства и методы, характерные только для него.

Класс, от которого производится наследование, называется базовым, родительским или суперклассом. Новый класс – потомком, наследником или производным классом. В одном классе можно объединить возможности нескольких других классов.

Наследование бывает двух видов:

Одиночное – класс (он же подкласс) имеет один и только один суперкласс (предок);

Множественное – класс может иметь любое количество предков (в Java запрещено).

Полиморфизм – это принцип, позволяющий использовать объекты с одинаковым интерфейсом без информации о типе и внутренней структуре объекта для решения двух или более похожих, но несколько отличающихся задач, т.е. реализуется идея «один интерфейс, множество методов».

При полиморфизме некоторые части (методы) родительского класса заменяются новыми, реализующими специфические для данного потомка действия. Таким образом, интерфейс классов остаётся прежним, а реализация методов с одинаковым названием и набором параметров различается. С полиморфизмом тесно связано *позднее связывание*.

Позднее связывание означает, что объект связывается с вызовом функции только во время исполнения программы, а не раньше, достигается в с помощью использования виртуальных функций и производных классов. Его достоинством является высокая гибкость. Оно может использоваться для поддержки общего интерфейса, позволяя при этом различным объектам иметь свою собственную реализацию этого интерфейса. Более того, оно помогает создавать библиотеки классов, допускающие повторное использование и расширение. Использование позднего связывания оправдано только тогда, когда оно улучшает структурированность и управляемость программы.

Объектно-ориенntированные языки программирования (ООЯП) содержит следующий набор элементов:

- объявление классов с полями (данными – членами класса) и методами (функциями – членами класса).
- механизм расширения класса (наследования) – порождение нового класса от существующего с автоматическим включением всех особенностей реализации класса-предка в состав класса-потомка.

– полиморфные переменные и параметры функций (методов), позволяющие присваивать одной и той же переменной экземпляры различных классов.

– полиморфное поведение экземпляров классов за счёт использования виртуальных методов.

Некоторые языки добавляют к указанному минимальному набору те или иные дополнительные средства. В их числе:

– конструкторы, деструкторы, финализаторы;

– свойства (аксессоры);

– индексаторы;

– средства управления видимостью компонентов классов (интерфейсы или модификаторы доступа, такие как ***public***, ***private***, protected, feature и др.).

В применении к объектно-ориентированным языкам программирования понятие класса и объекта конкретизируется:

Класс содержит описание данных и операций над ними. В классе дается обобщенное описание некоторого набора родственных, реально существующих объектов.

Объект – обладающий именем набор данных (полей объекта), физически находящихся в памяти компьютера, и методов, имеющих доступ к ним. Имя используется для доступа к полям и методам, составляющим объект. В предельных случаях объект может не содержать полей или методов, а также не иметь имени.

Любой объект относится к определенному классу. Объект является конкретным экземпляром класса.

Краеугольным камнем наследования и полиморфизма предстает следующая парадигма: «*объект подкласса может использоваться всюду, где используется объект суперкласса*».

При вызове метода класса он ищется в самом классе. Если метод существует, то он вызывается. Если же метод в текущем классе отсутствует, то обращение происходит к родительскому классу и вызываемый метод ищется у него. Если поиск неудачен, то он продолжается вверх по иерархическому дереву вплоть до корня (верхнего класса) иерархии.

Одни языки отвечают принципам объектно-ориентированного программирования в полной мере – в них все основные элементы являются объектами, имеющими состояние и связанные методы. Подобными языками являются *Smalltalk, Eiffel, Ada.* Существуют гибридные языки, совмещающие объектную подсистему в целостном виде с подсистемами других парадигм как «два и более языка в одном языке», позволяющие совмещать в одной программе объектные модели с иными, и размывающие грань между объектно-ориентированного программирования и другими парадигмами. Такими языками являются *CLOS, Dylan, Python, Ruby, Objective-C*. Однако распространены языки, включающие средства эмуляции объектной модели поверх императивной семантики. Это называется «склеиванием» в противовес «чистоте стиля» языков, воплощающих некую парадигму непосредственно. К ним относятся *Visual Prolog, Visual Basic, Object Pascal, Modula,* C++, C#, *Java.*

Объектно-ориентированные языки объединяют и расширяют возможности, свойственные к процедурным и функциональным языкам. Они позволяют использовать преимущества объектно-ориентированных способов не только при проектировании и конструировании программных систем, но и при их реализации, тестированиии и сопровождении. Такими языками являются прежде всего *Simula, Smoltok* и *Ada.* Позднее концепция объектно-ориентированности начала реализовываться и в других языках. К ним относятся *Visual Prolog, Visual Basic, Object Pascal*, C++, C#.

Замечание III.4.2:

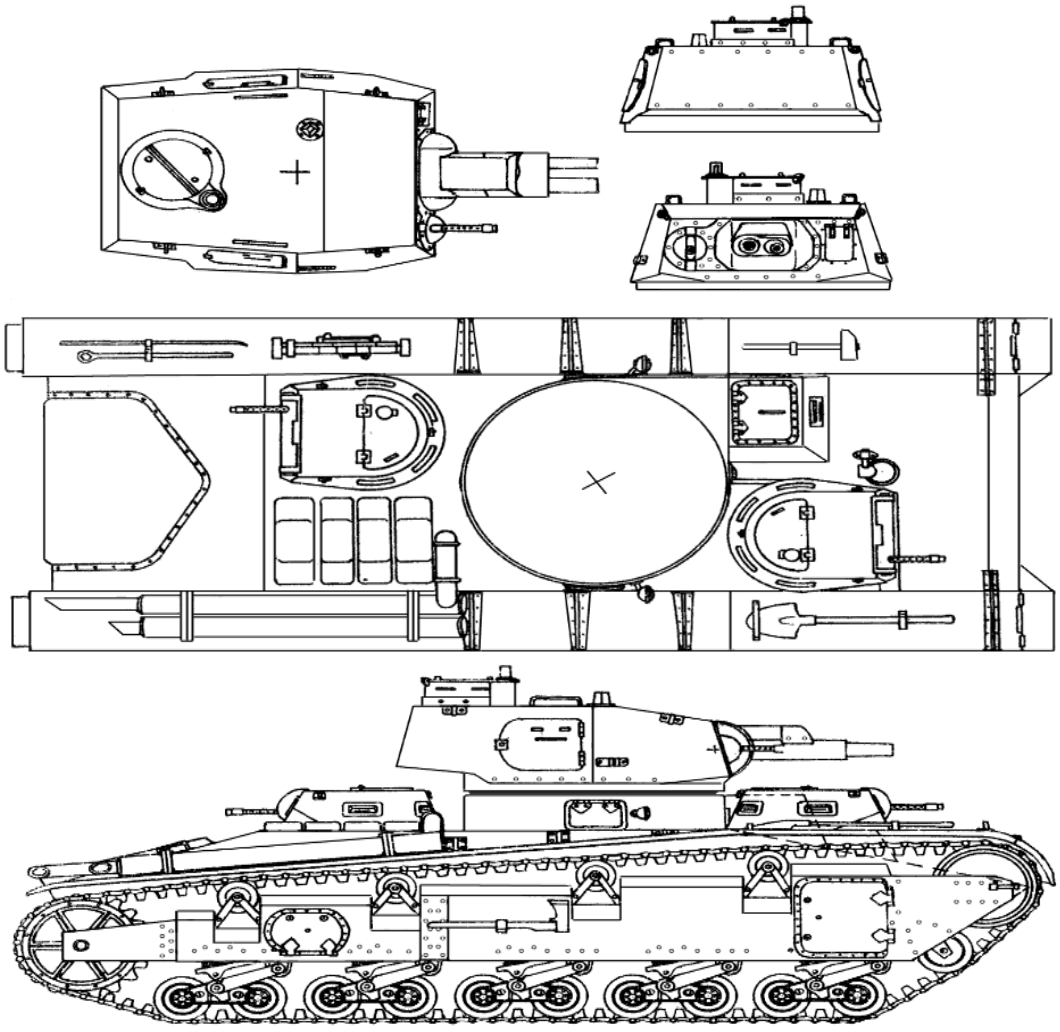
1. Во всех языках программирования для именования переменных, функций и др. применяется фундаментальное понятие информатики «*идентификатор*», который определяется как последовательность букв и цифр, начинающейся с буквы.

2. При выборе языка программирования для решения вашей задачи необходимо обратить внимание на то, что существуют два подхода (парадигмы), которые управляют построением программ. Первый подход называется процессо-ориентированной моделью,

которую можно представлять как кодовое воздействие на данные. Процессо-ориентированное программирование предполагает, чтобы запись программы происходила вокруг того «на что этот процесс влияет». В этом подходе возникают проблемы, когда возрастает размер и сложность программы. Второй подход называется объектно-ориентированной моделью, которую можно характеризовать как управляемый данными доступ к коду. ООП организует программу вокруг своих данных (объектов) и набора хорошо определенных интерфейсов с этими данными.

Примеры III.4.2.

В качестве примера можно привести чертеж танка или его описание (класс) и реальный танк (экземпляр класса, или объект).



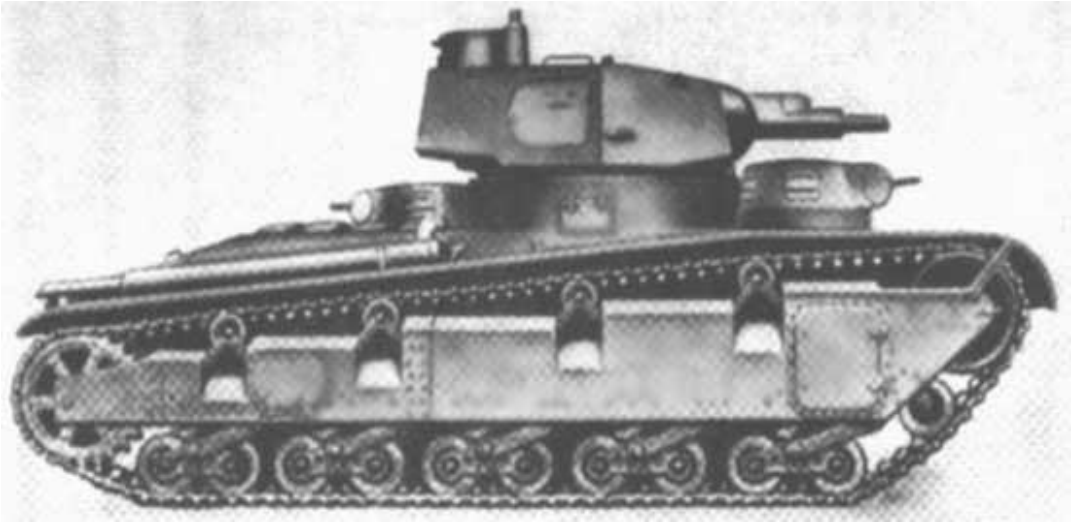


Рисунок III.4.2.1. Описание класса и реальный объект.

Класс принято обозначать в виде прямоугольника, разделенного на три части:

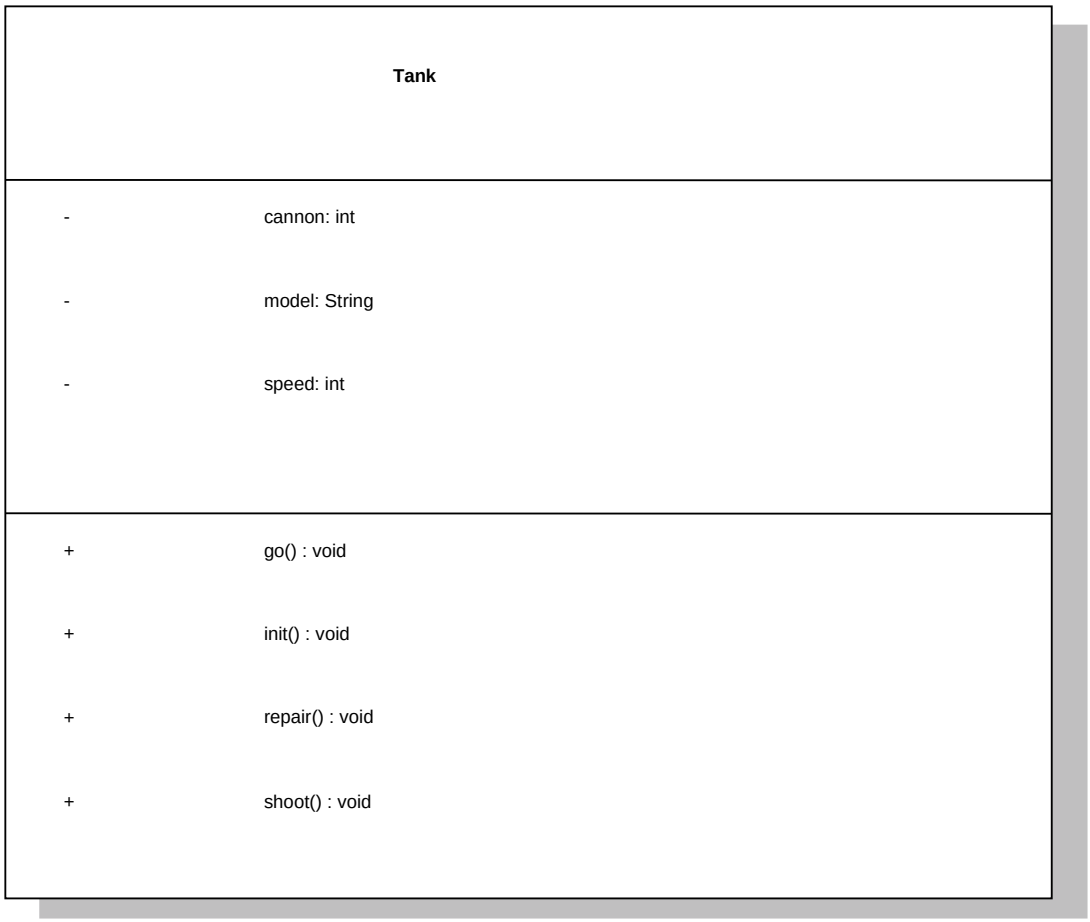


Рисунок III.4.2.2. Графическое изображение класса.

Графически наследование изображается в виде диаграмм UML, класс «Auto» называется суперклассом, а «Tank» – подклассом:

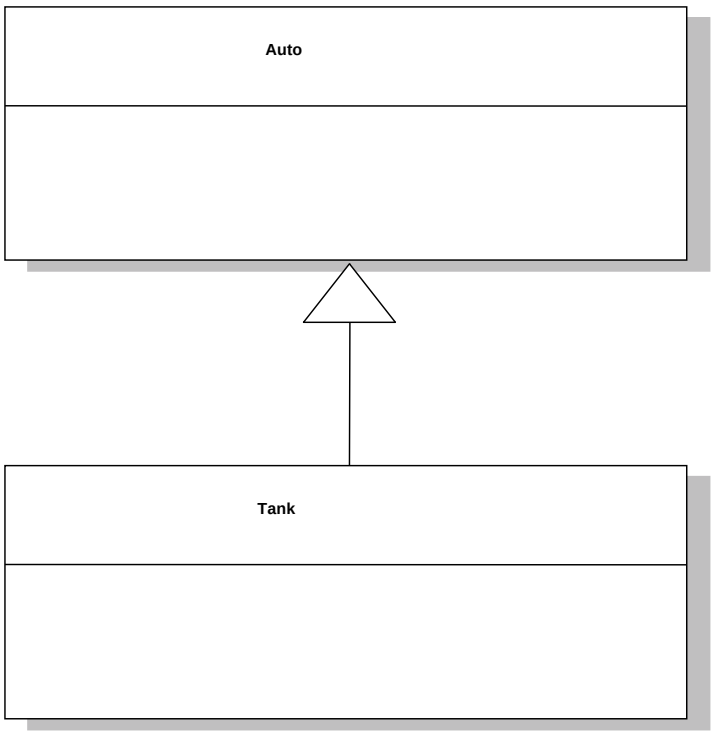


Рисунок III.4.2.3. Графическое изображение наследования.

Задания III.4.2.

1. Опишите семантику оператора присваивания в процедурных языках ***<переменная>*** ← ***<выражение>***.
2. Перечислите современные парадигмы в программировании и опишите на чем они освовываются.
3. Напишите на любом языке алгоритм, определите значения D при выполнении D ← B*B -4*A*C.

Помощь

1. Надо помнить сначала вычисление, затем присвоение.
2. Надо обратить внимание на модели вычисления, процессы и объекты в этих парадигмах.
3. Для вычисления значения заданного выражения нужно ввести значения переменных и построить оператор присваивания.

Вопросы III.4.2.

1. Какие виды трансляции реализованы в существующих трансляторах?

2. В связи с чем появились языки программирования высокого уровня и какие их виды есть?

3. На какие принципы основано объектно-ориентированное программирование?

Тесты III.4.2.

1. Что является программной единицей в логическом языке программирования Prolog?

A) утверждение;
B) функция;
C) оператор;
D) операция;
E) отношение.

2. Что является программной единицей в функциональном языке программирования Lisp?

A) функция;
B) отношение;
C) оператор;
D) утверждение;
E) подстановка.

3. Что является программной единицей в продукционном языке программирования Refal?

A) подстановка;
B) функция;
C) оператор;
D) утверждение;
E) отношение.

III.4.3. Процедурные языки программирования

В настоящее время существуют очень много процедурных языков программирования. Однако нас будет интересовать только наиболее распространенные классические процедурные языки программирования. Информация (наименовании, время создания и взаимовлияние) о классических процедурных языках программирования показана на рисунке III.4.3.

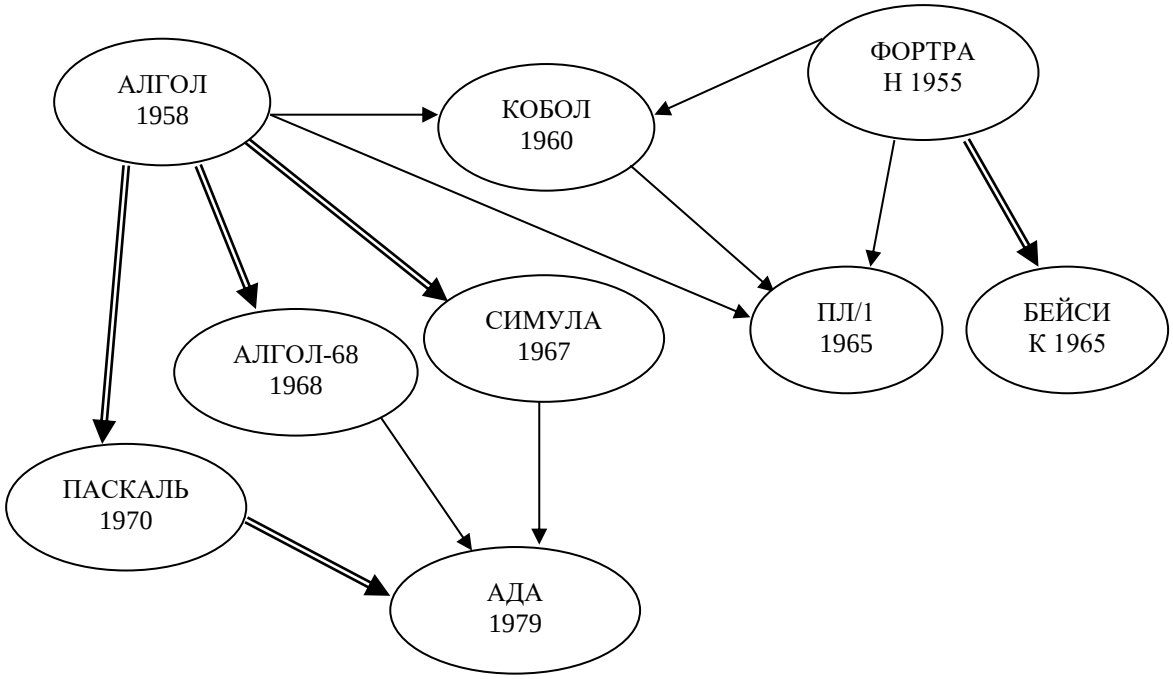


Рисунок III.4.3. Информация о классических процедурных языках программирования.

Здесь двойной стрелкой показано сильное влияние, простой стрелкой – слабое влияние.

Про процедурных языков в своей лекции американский ученый, основоположник функционального программирования Дж.Бэкус во время вручения ему премии Тьюринга (самая высокая премия в области информатики) утвердил, что язык *Ada* есть предел развития процедурных языков, дальше процедурные языки программирования не смогут развиваться, так как в языке *Ada* реализованы решения всех теоретических вопросов.

Среди процедурных языков самого распространенного и удобного для изучения представителя языка *Pascal* в целях обучения методам программирования специально разработал в 1970 году

Швейцарский ученый Н. Вирт. Он назвал этот язык именем великого французского ученого В. Pascal.

Ниже описывается абстрактный (укороченный) вид языка *Pascal*. Правильно составленная программа на укороченном языке *Pascal*, будет правильным и в полном варианте этого языка. Поэтому в этом учебнике примеры программы, в том числе программы сортировки, написаны на укороченном языке *Pascal*.

Pascal-программа записывается в виде цепочек символов (латинских букв, русских букв, арабских цифр, знаков препинания, знаков операций, знака пробела и скобок) и каждое предложение заканчивается знаком «;».

Для обозначения исходных данных и результатов применяются постоянные и переменные.

Постоянная числовая величина задается в виде целых, действительных чисел в десятичной системе счисления, а постоянная символьная величина – в виде цепочки символов, заключенной в апострофы (при выводе апострофы не видны и не печатаются).

В программе для объявления переменных применяют специальное слово *var*, а для представления их типов – специальные слова:

числовой тип: *integer* – бүтін число, *real* – действительное число;

символьный тип: *char* –цепочка символов.

Например, пусть переменные *i* – целое число; *a, b* – действительное число; *x, y* –цепочка символов. Тогда они в языке *Pascal* запишутся так:

var *i, j* : *integer;*

var (*x, y*) : *real;*

var (*a, b*) : *char;*

В программировании под типом нужно понимать множество, где определены операции, а под его элементами – значения этих типов. С этой точки зрения, например, типы *integer* и *real* – числовые множества, где определены арифметические операции и операции сравнения, а *char* – множество символов с оределенными

операциями сцепления и операциями сравнения. Этих типов называют *стандартными типами*, потому что они определены в большинстве языков программирования.

В языке Pascal из имеющихся типов можно получить новые не *стандартные типы*. Рассмотрим одну из таких возможностей. В математике наряду с некоторым множеством часто рассматривают множество упорядоченных пар этих множеств, множество упорядоченных троек этих множеств и т.д. В языке Pascal такие упорядоченные пары, тройки и т.д. задаются в виде множества однотипных индексироваггых элементов и называются *массивами*. Типы элементов и способы задания индексов даются в определении типа, где будет находиться массив. Например, выражение вида

t = **array**[1.20] **of** *real*

определяет новый тип с именем *t,* элементами которого являются 20 пар (интервал изменения значения индекса от 1 до 20) типа *real*. В программе перед таким определением записывается специальное слово **type**. Например, если в программе переменная *a* описывается как переменная типа *t*:

var *a* : *t*;

то при выполнении программы значение переменной *a* будет массивом, состоящим из 20 элемнентов типа *real*, т.е., их можно записывать как *a*[1], *a*[2], …, *a*[20]. Здесь *t* – тип переменной *a*, *real* – тип переменных *a*[1], *a*[2], …, *a*[20] и определенная над объектом *t* операция – доступ к каждому элементу с помощью индексов, а определенная над каждым элементом операция – определенная в типе *real* числовая операция. Итак все сказанное можно в программе можно полностью записать в виде:

type *t*= **array**[1.20] **of** *real*;

var *a* : *t*;

В общем виде массив определяется в виде:

u= **array**[n_1.n_2] **of** *r*;

где *u* **–** имя нового типа, целое число n_1 – нижняя граница значения индекса, целое число n_2 – верхняя граница значения индекса и $n_1 \leq n_2$**,**

а r – стандартный тип (один из типов *integer, real, char*) или имя другого ранее определенного типа.

Множество всех определений типов записывается так:

type $T_1, T_2, \ldots, T_m$

где $T_1, T_2, \ldots, T_m$ – определения отдельных типов. Это множество пишется в программе перед объявлением переменных.

В большинстве случаев для составления программы решения заданной задачи нужны смещанные типы: один элемент символьный, второй элемент числовой и т.д. Например, определив тип *ученик* и можно сказать, что его элементами могут быть такими: *имя* – символьное, *возраст* – действительное число, *оценка* – целое число и т.д.

В языке Pascal смешанный тип задается с помощью *записи*. Элементы записи называются *полем* (столбик). В записи каждое поля имеет отдельное имя. Например, если в качестве имени поля выбраны *имя, возраст, оценка* и если через *x* обозначим запись, то поля этой записи обозначим через *x.имя, x.возраст, x.оценка*. Количество полей записи, их типы и имена записываются в определении типа, касающего записи. Например, можно писать:

ученик = **record** *имя* : *char*; *возраст* : *real*; *оценка*: *integer* **end**

это определение трехэлементной записи *ученик* и с названием полей *имя, возраст, оценка*, а их типы *char, real, integer* соответственно.

Если в программе смешанный тип *v* определен структурой **record.. end**, то он в общем виде в следующем виде:

v = **record** $l_1 : r_1; l_2 : r_2; \ldots; l_k : r_k$ **end**

где $l_1, l_2, \ldots, l_k$ – имена полей, $r_1, r_2, \ldots, r_k$ – имена стандартных типов *integer, real, char* или других ранее определенных типов.

Операция присваивания обозначается знаком «**:=**», например, *i* := 0; *x* := 2.71; *a*:= ‘предусловие’.

Для оператора ввода и *оператора вывода* применяются специальные слова *read* и *write* соответственно. Например, если нужно ввести значения переменных *u, v* и *w* и вывести значения переменных *r, s t*, то это можно записать так:

read(*u,v, w*)*;*

write(*r*, *s*, *t*)*;*

Последовательность записывается межу специальными словами **begin** и **end**, например, **begin** *j*:=1; *y*:=.14; *b*:= ‘постусловие’ **end**

Если обозначить через *B* условие, необходимое для ветвления, а действие каждой ветви обозначить через *S1* и *S2* соответственно, то ветвление записывается в виде:

if *B* **then** *S1* **else** *S2*

Например, если переменной *max* нужно присвоить наибольшее значение переменных *x1* и *x2*, то это можно написать в виде:

if *x1>x2* **then** *max* := *x1* **else** *max* := *x2*

Предусловное повторения записывается в виде:

while *B* **do** *S*

где *B* – условие повторения, *S* – тело повторения.

Относящееся к постусловному повторению параметрическое повторение заисывается в виде:

for *i* := *K* **to** *N* **do** *S*

где *i* – параметр повторения (переменная целого типа), *K* и *N* – начальное значение и конечное значение параметра повторения (переменные целого типа), *K*≤*N*, *S* – тело повторения.

Параметрическое повторение имеет еще один вид:

for *i* := *K* **downto** *N* **do** *S*

где *K*≥*N*, поэтому i постепенно принимает значения *K*, *K–1*,.., *N* и для каждого из них выполняется тело повторения *S*, ал если *K*<*N*, то тело повторения *S* ни разу не выполняется.

В языке Pascal для обозначения начало программы используется специальное слово program. Если для программы нужен *параметр*, то после этого слово записываетмя имя программы и следм за ним в скобке записывае(ю)тся параметр(ы), при этом, если имеются несколько параметров, то они друг от друга разделяются запятыми. Например,

program *аудан* (*input*, *output*) ;

где *input* – показывает, что в программе имеется оператор ввода, а *output* – показывает, что в программе имеется оператор вывода.

При программировании иногда приходится несколько раз

писать одну и ту же последовательность операторов. Если каким-то образом можно именовать такие пословательности и изменяемые в зависимости от контекста программы значения её величин обозначить через параметры, то её определив только один раз и вызвав много раз там, где требуется её услуга, можно компактно составлять программу.

Это реализуется с помощью двух программных средств:

1) *Процедура* – именованная последовательность операторов, совершающая некоторое действие;

2) *Функция* – именованная последовательность операторов, вычисляющая некоторое значение.

Для именования последовательносьти операторов в программу нужно добавить описание процедуры. Например, если для последовательности операторов S нужно ввести имя процедуры P, то описание будет иметь вид:

procedure *P; S*

Процедуру с параметром можно определить так:

procedure < имя> (<список параметров>);

где <список параметров> – один или несколько параметров, разделенные запятыми.

Вместе с параметрами можно показывать и их типы: если показан тип параметра, то между ними пишется двоеточие, например, n : *integer*, u : *real*.

Описание функции начинается со слово **function**, затем пишется имя функции и в круглой скобке её параметры, а в конце указывается тип значения функции.

Например, можно предложить следующие описания функций:

function *f1* (*k*: *integer; l* : *integer; m* : *integer*;) : *integer*;
function *f2* (*a*: *real*; *b* : *real*; *c* : *real*;) : *real*;

Вызов функции осуществляется в правой части оператора присваивания. Они встречаются и в операции отношения.

Во время определения процедуры и функции параметры

называются *формальными параметрами*, а при их вызове указанные параметры называются *фактическими параметрами*.

В языке Pascal имеются много пригодных к использованию стандартные функции. В таблице III.4.3 приводится такие функции.

Таблица III.4.3. Стандартные функции

№	В языке Pascal	Имя
1	*abs*(*E*)	Абсолютная величина
2	*arctan*(*E*)	Арктангенс
3	*Cos*(*E*)	Косинус
4	*Exp*(*E*)	Показательная функция
5	*ln*(*E*)	Натуральный логарифм
6	*Sin*(*E*)	Синус
7	*Sqr*(*E*)	Возведение в квадрат
8	*sqrt*(*E*)	Квадратный корень
9	*mod* (*A*,*B*)	Остаток от целочисленного деления
10	*trunc* (*E*)	Возвращает целочисленную часть числа с плавающей запятой.

Примеры III.4.3.

1. Составить программу вычисления площади тругольника по известным значениям его сторон по следующим формулам:

$$p = \frac{a+b+c}{2};\ s = \sqrt{p(p-a)(p-b)(p-c)}\,.$$

Здесь *a, b, c* – стороны, *p* – половина периметра, *s* – площадь треугольника.

Требуемая программа на языке Pascal будет иметь вид:

```
program площадь (input, output) ;
    var a, b, c, p, s : real;
        begin read (a, b, c);
            p:=(a+ b+ c)/2; s=sqrt(p*(p–a) *(p–b) *(p–a));
            write (s)
        end.
```

2. Построить функцию для вычисления значения факториала заданного целого положительного числа по формуле:

$$0! = 1$$
$$n! = n*(n–1)!$$

Эта формула реализуется на языке Pascal так:

```
function fact (n:integer); integer;
    begin
        if n = o then fact: =1
                    elce fact: = n * fact (n–1)
    end.
```

Задания III.4.3.

1.Составить программу нахождения высоты треугольника по заданным его сторонам a, b, c.

2. Составить программу вычисления биномиального коэффициента C_n^k.

Помощь

1. Нужно использовать формулу $H = \frac{2}{a}\sqrt{p(p-a)(p-b)(p-c)}$.

2. Нужно использовать формулу $C_n^k = \frac{n!}{k!(n-k)!}$.

Вопросы III.4.3.

1. Как описываются переменные в языке программирования Pascal?
2. Как описываются процедуры в языке программирования Pascal?
3. Как описываются функции в языке программирования Pascal?

Тесты III.4.3.

1. Как записывается выражение $\frac{e^x + a*b}{Cos^2 x + Sinx}$ на языке программирования Pascal?

A) (EXP(X)+A*B)/ (COS(X)*COS(X)+SIN(X))
B) EXP^X+A*B)/ (COS(X)*COS(X)+SIN(X))
C) (EXP(1)+A*B)/ (SQR(COS(X))+SIN(X))
Г) EXP(X)+A*B)/ (SQRT(COS(X))+SIN(X))
E) (EXP(1)+A*B)/ (SQR(COS(X))+SIN(X)

2. Чему будет равен результат выполнения арифметического выражения 6 MOD 4?

A) 4
B) 2
C) 1
Г) 0
E) 6

3. Чему будет равен результат выполнения выражения TRUNC(8.915) ?

A) 8.9
B) 9
C) 8
D) 8.91
E) 8.92

III.4.4. Функциональные языки программирования

Самым первым и самым распространенным функциональным языком программирования является Lisp, который был создан в 1960 году американским ученым Д.Маккарти. Этот язык предназначени для представления и обработки специальной структуры данных, называемой *списком*.

В языке Lisp самой минимальной неделимой обрабатываемой единицей является *атом*. В качестве атома берется постоянная величина и переменная величина. Большинство Lisp–системы позволяет использовать в качестве имен переменной некоторое специальное слово, но об этом предупреждает. Например, слова F, T, NIL не могут быть использованы в качестве имен переменных, так как, они являются значениями логических величин: F – ложно, T – истина, NIL – нет или ложно.

В языке Lisp: каждая программная структура является только функцией, составление программы означает построение сложной функции с помощью простых функций. Отношения между функциями реализуются их вызовами только во время выполнения программы. Любая функция заключается внутри двух скобок «(» и «)» и их вид будет таким:

(<имя функции> <список аргументов>)

Имена функции и аргументов задаются *идентификатором*. В качестве аргумента берется и постоянная величина, в этом случае записывается реальное представление этой величины. Кроме того, в качестве аргумента могут использованы другие функции. В этом случае, говорят, что определена операция композиция (суперпозиция), т.е., ределена сложная функция: она в качестве своего аргумента возьмет результат примененной функции–аргументу операции *аппликация*.

Например, для решения квадратного уравнонения нужно вычислить значение дискриминанта и он находится по формуле $D = B^2 - 4AC$. Эту формулу на языке Lisp можно представить в виде:

(**SETQ** D (**SUB** (**MUL** B B) (**MUL** 4 A C))),

где **MUL** – имя функции умножения, **SUB** – имя функци вычитания, **SETQ** – имя функции присваивания значение переменной. **MUL** и **SUB** числовые функции, т.е. аргументты и результаты являются числовыми величинами, а **SETQ** – определена над любым типом.

Список представляет собой упорядоченный множество элементов, которыми являются *атомы* или списки, т.е., список является *рекурсивной структурой*.

Каждый элемент состоит из двух частей: в первой части находится ссылка на значение элемента, а во второй – ссылка на следующий элемент. В связи с этим рассмотрим два случая:

1. Если элементом списка являяется атом, то его значением может быть постоянное число, символ или логическое значение (F – ложно, T – истина, NIL – нет). Например, в качестве списка возьмем выражение из семи атомарных элементов (1 2 3 A B C D), графический вид которого показан ниже на рисунке III.4.4.1.

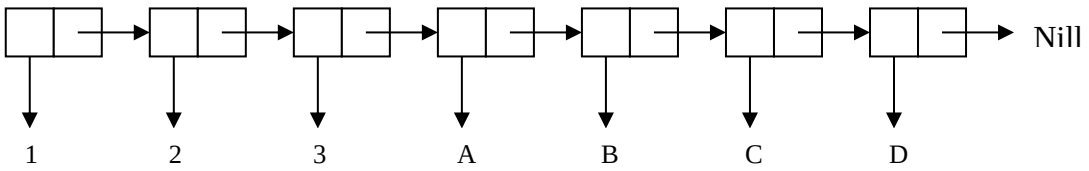


Рисунок III.4.4.1. Простой список (1 2 3 A B C D).

Если к этому списку применить функцию SETQ, т.е., напишем выражение (**SETQ** X (1 2 3 A B C D)), то будет затруднение при присваивании переменной X список. Так как первый атом в списке (1 2 3 A B C D) должен быть именем функции, представленным идентификатором, а у нас он 1. Поэтому нам нужна функция, которая превращает свой аргумент в свой результат. Имя такой функции QUOTE. Теперь, если применить эту функцию к нашему выражению (**SETQ** X (**QUOTE** (1 2 3 A B C D))), то значением будет (1 2 3 A B C D). Есть другой способ для получения списка в качестве значения функции. Для этого нужно применить функцию LIST.

Например, значением выражения (LIST 1 2 3 A B C D) будет список (1 2 3 A B C D).

2. Если элементом списка является список, то для его обработки можно применить различные функции. Например, пусть задано выражение (A (B C) D). В этом сложном списке есть три элемента и один из них является простым списком (B C), состоящим из двух атомов. Графический вид этого сложного списка показан на рисунке.

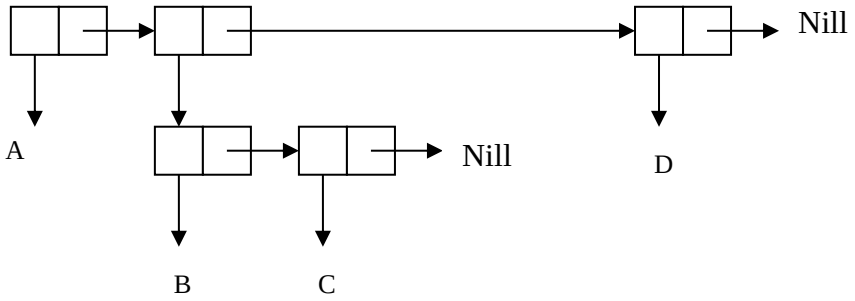


Рисунок III.4.4.2. Сложный список (A (B C) D).

В языке Lisp основную структуру данных называют *S-выражением*, где *S* является первой буквой английского слово *Symbolic*. *S-выражение* может быть либо атомом, либо списком.

В языке Lisp определены следующие стандартные функции **CAR, CDR, CONS, EQUAL, GREAT, COND, DEFINE, LAMBDA.**

1. Аргументом функции **CAR** является список, а результатом будет первый элемент этого списка. Например, при разном значении переменной X функция (**CAR** X) дает разные значения:

a) если значение X есть (1 2 3 4), то значение (CAR X) будет 1;

b) если значение X будет (1 (2 (3 (4)))), то значение (CAR X) будет 1;

c) если значение X будет ((1 2) (3 4)), то значение (CAR X) будет (1 2);

d) если значение X будет (((1 2) 3) 4), то значение (CAR X) будет ((1 2)3);

e) если значение X будет ((((1 2) 3) 4)), то значение (CAR X) будет (((1 2) 3) 4);

2. Аргумент функции **CDR** есть список, а результатом будет остаток от уничтожения первого элемента этого списка. Например, можно определить следующие значения функции (**CDR** X):

a) если значение X есть (1 2 3 4), то значение (CDR X) будет (2 3 4);

b) если значение X есть (1 (2 (3 (4)))), то значение (CDR X) будет (2 (3 (4)));

c) если значение X есть ((1 2) (3 4)), то значение (CDR X) будет (3 4);

d) если значение X есть (((1 2) 3) 4), то значение (CDR X) будет (4);

e) если значение X есть ((((1 2) 3) 4)), то значение (CDR X) будет Nil ;

3. Функция **CONS** обратно объединяет результатов функций **CAR** и **CDR**. Например, при вышеопределенных значений X и Y функция (**CONS** X Y) выдаст следующие результаты:

a) если значение X есть 1 и Y значение есть (2 3 4),
то значение (CONS X Y) будет (1 2 3 4);

b) если значение X есть 1 и Y значение есть (2 (3 (4))),
то значение (CONS X Y) будет (1 (2 (3 (4)))) ;

c) если значение X есть (1 2) и Y значение есть (3 4),
то значение (CONS X Y) будет ((1 2) (3 4));

d) если значение X есть ((1 2)3) и Y значение есть (4),
то значение (CONS X Y) будет (((1 2)3)4) ;

e) если значение X есть (((1 2) 3) 4) и Y значение Nil,
то значение (CONS X Y) будет (((1 2) 3) 4);

4. Функция **EQUAL** имеет двух аргументов. Результатом этой функции будет T – истина, если значения её аргументов будут равны, в противном случае Nil – ложно.

5. Функция **GREAT** имеет двух аргументов. Результатом этой функции будет T – истина, если значение первого аргумента будет больше, чем значение второго аргумента, в противном случае Nil – ложно.

6. Функция **COND** реализует ветвление. У этой функции число аргументов неопределено. Но каждый ее аргумент список, состоящий из двух элементов: представляет условие ветвления, а второй показывает либо величину, либо действие. При выолнении функции сначала проверяется условие ветвления: если это условие выполняется, то значением функции второй элемент, а если это условие не выполняется, то проеверяется условие следующего аргумента и так далее будет продолжаться. Если не выполняется условие ни одного аргумента, то значение функции будет NIL.

7. Функция **DEFINE** своей встречи в программе показывает, что после неё стоит определение функции, т.е. она стоит всегда перед любой функции. Эта функция имеет только один аргумент и он является списком определяемых функций. А каждая определяемая функция сама является двух элементным списком: первый элемент имя функции, а второй элемент выражение, определяющее этой функции. Например, функцию DEFINE можно представить в виде (**DEFINE** ((**F1** X1) (**F2** X2) (**F3** X3))), где аргумент функции ((**F1** X1) (**F2** X2) (**F3** X3)) – список определяемых функций; **F1, F2, F3** – имена определяемых функций; X1, X2, X3 – спецификация определяемых функций. Если в качестве спецификаций определяемых новых функций записаны имена известных функций, то новые функции дадут точные копии тех старых функций. Например, в этом выражении (**DEFINE** ((**F1** MUL) (**F2** SUB) (**F3** SETQ))) определяемые функции **F1, F2, F3** будут соответствовать известным функциям **MUL, SUB, SETQ** и будут работать точно как они.

8. Функция **LAMBDA** неоходима для определения других функции. У этой функции всегда будут два аргументов: первый аргумент выполняет роль списка аргументов определяемых функций, а второй аргумент – выражения, которые будут использованы определяемые функций. Например, соответственно выражению $X^2 - Y^2$ на языке Lisp можно наисать такую функцию:

(**LAMBDA** (X Y) (**SUB** (**MUL** X X) (**MUL** Y Y))),

где X и Y переменные выполняют роль параметров.

Значением функции **LAMBDA** не будут число, имя, список и т.д., его значением считается своя спецификация и это значение применяется везде, где встречается имя этой функци. То есть, можно сказать, что **LAMBDA** не функция емес, а выражение. Это основано на «λ – исчисление», которое в 1941 году было внедрено в математику английским ученым Чёрчом: вышерассмотренный пример можно записать в виде выражения $\lambda xy(x^2 - y^2)$ и надо учесть, что его значение будет другим, чем значение выражение $\lambda yx(x^2 - y^2)$.

В функции **LAMBDA** можно связать параметры с реальными значениями. Для этого в конце определения функции нужно писать список значений параметров. Например, значени такого выражения

((**LAMBDA** (X Y) (**SUB** (**MUL** X X) (**MUL** Y Y))) 5,4)

будет 9, потому что $5^2 - 4^2 = 25 - 16 = 9$.

Теперь используя вышесказанные можно определить простые функции и составлять программу на языке Lisp.

Примеры III.4.4.

1. Составить программу вычисления площади тругольника по известным значениям его сторон по следующей формуле:

$$p = \frac{a+b+c}{2};\ s = \sqrt{p(p-a)(p-b)(p-c)}$$

Здесь a, b, c – стороны, p – половина периметра, s – площадь треугольника.

Учитывая сказанные программу на языке Lisp можно написать так:

DEFINE(((**AUDAN** (**LAMBDA** (A B C)
(**SETQ** P (**DIV** (**ADD** A B C) 2))
(**SETQ** S (**SQRT** (**MUL** P (SUB P A)
(**SUB** P B) (**SUB** P C))))))))

3. Построить функцию для вычисления значения факториала заданного целого положительного числа по формуле:

0! =1

N! = N*(N–1)!

Соответствующая этой формуле программа на языке Lisp будет иметь следующий вид:

DEFINE(((**FACT** (**LAMBDA** (N)
(**COND** ((**EQUAL** 0 N) 1) (T (**MUL** N
(**FACT** (**SUB** N 1))))))))

3. Написать програму на языке Lisp с именем **NOD** для вычисления по алгоритму Евклида наибольшего общего делителя двух заданных натуральных чисел N1 и N2.

Соответствующая этому алгоритму программа на языке Lisp будет иметь следующий вид:

DEFINE(((**NOD** (**LAMBDA** (N1 N2)
(**COND** ((**EQUAL** 0 N2) N1) (T (**NOD** (**DIV** N 1 N2)))))))

Задания III.4.4.

1) Если значение X есть (6 7 8 9), то что будет значением (CAR X)?

2) Если значение X есть ((1 2) (7 8)), то что будет значением (CDR X)?

3) Если значение X есть ((3 4)5) и Y значение (6), то будет значением (CONS X Y)?

Помощь

1) Значением функции **CAR** будет первый элемент своего аргумента.

2) Значением функции **CDR** будет остаток от удаления первого элемента своего аргумента.

3) Функция **CONS** объединяет результатов функций **CAR** и **CDR**

Вопросы III.4.4.

1. Что является самой минимальной единицей в функциональных языках программирования?

2. Какие стандартные функции имеются в функциональном языке программирования Lisp?

3. Какие структуры данных используются в функциональном языке программирования Lisp?

Тесты III.4.4.

1. Если значение X есть (1 2 3 4), то чему равно значение функции (CAR X)?

A) 0
B) 2
C) 3
D) 4
E) 1

2. Если значение X есть ((1 2)3) и Y значение (4), то чему равно значение функции (**CONS** X Y)?

A) (((1 2)3)4)
B) (1 2 3 4)
C) ((1 2) (3 4))
D) (1 2)3)4)
E) (((1 2 3 4))

3. Что является программной елиницей в функциональном языке программирования Lisp?

A) оператор
B) список
C) условие
D) функция
E) процедура

III.4.5. Логические языки программирования

Логические языки программирования появились в результате реализации идеи, выдвинутой Грином в 1960-е годы, о возможности применения языка математической логики предикатов в качестве языков программирования.

Суть этой идеи в следующем: *программист не указывает компьютеру последовательность шагов действий, ведущих к решению задачи, как это делается в процедурных языках программирования, а опишет на логическом языке только постановку нужной задачи* (*свойства касающихся к задаче объектов и отношений между ними*), *а решение задачи компьютер сам находит в качестве логического заключения*.

Среди логических языков программирования самым распространенным является язык *Prolog*, который был создан в 1972 году под научным руководством французского ученого Колмераур на основе частного случая логики предикатов 1-го порядка, *ограниченной хорновскими дизъюнктами и снабженной методом резолюции* как единственным правилом вывода.

В языке Prolog программной единицей считается логическое утверждение.

Самыми минимальными программными единицами считаются *факты*, а следующими единицами – *правила* и *запросы*.

Факты это атомарные логические формулы вида $\boldsymbol{P(t_1, t_2, \ldots, t_k)}$, где $\boldsymbol{P}$ – предикатные символы, $\boldsymbol{t_1, t_2, \ldots, t_k}$ – *термы*, образованные из постоянных величин, переменных величин и функциональных символов. Наличие в программе факта $\boldsymbol{P(t_1, t_2, \ldots, t_k)}$, содержащего переменных $X_1, X_2, \ldots, X_r$ означает, что для любых объектов $X_1, X_2, \ldots, X_r$ имеет место $\boldsymbol{P(t_1, t_2, \ldots, t_k)}$ истинен. Например, наличие в программе факта $\boldsymbol{меньше(x, x+1)}$ означает, что объект $\boldsymbol{x}$ меньше чем объект $\boldsymbol{x+1}$.

Правило есть выражение, записанное в виде $\boldsymbol{P_1, P_2, \ldots, P_n :- P_0}$, здесь $\boldsymbol{P_0, P_1, P_2, \ldots, P_n}$ – факты, сцепленные операцией конъюнкция, «:–» – знак заключения.

Если в правиле встречаются переменные X_1, X_2, ..., X_r, то суть правила в следующем: то для всех объектов X_1, X_2, ..., X_r из утверждений $\boldsymbol{P_1}$, $\boldsymbol{P_2}$, **…**,$\boldsymbol{P_n}$ следует утверждение $\boldsymbol{P_0}$ или для всех X_1, X_2, ..., X_r, если истинны утверждения $\boldsymbol{P_1}$, $\boldsymbol{P_2}$, **…**, $\boldsymbol{P_n}$, то истинно утверждение $\boldsymbol{P_0}$.

Запрос есть выражение, записанное в виде ?– $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$, здесь $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$ – факты, сцепленные операцией конъюнкция, «?–» – знак запроса.

Запрос, не содержащий переменных, читается так, верно ли, что $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$?

Если в запросе встречаются переменные X_1, X_2, ..., X_r, то суть запроса в следующем: спрашивается, что для каких объектов X_1, X_2, ..., X_r верно $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$?

В языке Prolog считается, что компьютеру известны только данные в программе, в том числе свойства стандартных предикатов (типа =, <, >) и стандартных функций (типа +, –, *, /) и ответ на запрос заключается из этих данных.

Таким образом, логическая программа имеет простую, понятную и естественную семантику: запрос ?– $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$ с переменными X_1, X_2, ..., X_r к программе $\boldsymbol{P}$ понимается как требование вычислить все значения переменных X_1, X_2, ..., X_r при которых $\boldsymbol{Q_1}$, $\boldsymbol{Q_2}$, **…**, $\boldsymbol{Q_m}$ логически следует из утверждений, содержащихся в этой программе $\boldsymbol{P}$.

Программой на языке Prolog называют последовательность из фактов и правил. В программе все факты записываются перед правилами. В каждой программе имеются свои запросы. Имена фактов, правил и запросов задаются с помощью *идентификатора* (последовательности из букв и цифр, начинающеяся с буквы).

В языке Prolog данные задаются как *атом* или *список*. Атомами называются постоянные и переменные с числовыми или символьными типами. Список образуется из атомов, разделенных друг от друга запятыми и заключенных в прямые скобки. Например, [A, B, C, D] – список из четырех атомов. Запись вида [A| B] показывает, голова списка A, а хвост B.

В языке Prolog применяются следующие операции: *арифметические операции* (+ – сложение, - – вычитание, * – умножение, / – деление); *қатынас операции* (= – равно, <= – меньше или равно, < – меньше, > = – больше или равно, > – больше); *логические операции*(not – отрицание, and – и, or – или);

В языке Prolog имеются многочисленные готовые к применению, т.е. сохраненные с определениями, стандартные предикаты: **sin**(X) – синус, **cos**(X) – косинус, **tan**(X) – тангенс, **arctan**(X) – арктангенс, **abs**(X) – абсолютная величина, **exp**(X) – экспонента, **ln**(X) – натуральный логарифм, **log**(X) – десятичный логарифм, **sqrt**(X) – квадратный корень, ***random*** (X) – случайное действительное число от нуля до единицы. Кроме того, имеются предикаты, работающие на уровне отдельных битов (самые маленькие единицы памяти): **bitnot**(A,B) – нет, **bitand** (A,B,E) – и, **bitor** (A,B,E) – или, **bitleft** (A,N,E) – сдвиг на N бит налево, **bitright** (A,N,E) – сдвиг на N бит направо.

На основе вышесказанных можно составлять программу на языке Prolog.

Примеры III.4.5.

1. В этой логической программе

наука(*математика*)**.**

наука(*информатика*)**.**

абстрактная(*математика*)**.**

прикладная(*информатика*)**.**

трудная(*X*) *:–* ***наука***(*X*), ***абстрактная***(*X*)**.**

полезная(*X*) *:–* ***наука***(*X*), ***прикладная***(*X*)**.**

запрос

?– ***трудная***(*X*)

получает такой ответ:

X= *математика*,

а запрос

?– ***полезная***(*X*)

получает такой ответ:

X= информатика,

То есть, утверждения ***трудная***(*математика*) и ***полезная***(*информатика*) логически выведены в этой программе.

2. На языке Prolog составив факты ***отец*** и ***мать*** построить правила для утверждений ***дедушка***, ***бабушка*** и ***прабабушка***:

отец(*Болат*, *Дина*)**.**

отец(*Дулат*, *Болат*)**.**

отец(*Абай*, *Айжан*)**.**

мать(*Сауле*, *Айжан*)**.**

мать(*Айжан*, *Жанна*)**.**

мать(*Айжан*, *Адия*)**.**

мать(*Жанна*, *Шолпан*)**.**

мать(*Адия*, *Болат*)**.**

дедушка(*X*, *Z*) :– ***отец***(*X*, *Y*), ***отец***(*Y*, *Z*)**.**

дедушка(*X*, *Z*) :– ***отец***(*X*, *Y*), ***мать***(*Y*, *Z*)**.**

бабушка(*X*, *Z*) :– ***мать***(*X*, *Y*), ***отец***(*Y*, *Z*)**.**

бабушка(*X*, *Z*) :– ***мать***(*X*, *Y*), ***мать***(*Y*, *Z*)**.**

прабабушка(*X*, *Z*) :– ***мать***(*X*, *Y*), ***дедушка***(*Y*, *Z*)**.**

прабабушка(*X*, *Z*) :– ***мать***(*X*, *Y*), ***бабушка***(*Y*, *Z*)**.**

прабабушка(*X*, *Z*) :– ***бабушка***(*X*, *Y*), ***отец***(*Y*, *Z*).

прабабушка(*X*, *Z*) :– ***бабушка***(*X*, *Y*), ***мать***(*Y*, *Z*).

В этой программе некоторые запросы имеют такие ответы:

?– ***дедушка***(*X*, *ДинА*), то ответ ***истина***, *X=Дулат*

?– ***прабабушка***(*X*, *ДинА*), то ответ ***ложно***

?– ***бабушка***(*X*, *Жанна*), то ответ ***истина***, *X=Даметкен*

?– ***бабушка***(*X*, *Шолпан*), то ответ ***истина***, *X=Айжан*

?– ***прабабушка***(*X, Шолпан*), то ответ ***истина***, *X=Даметкен*

?– ***дедушка***(*Болат*,*Y*), то ответ ***ложно***

?– ***бабушка***(*Адия*,*Y*), то ответ ***истина***, *Y= Дина*

?– ***прабабушка***(*Айжан*,*Y*), то ответ ***истина***, *Y=Дина*

?– ***прабабушка***(*Даметкен*,*Y*), то ответ ***истина***, *Y=Шолпан*

3.Написать программу для вычисления факториала целого неотрицательного числа ***N***. Для этого используем рекурсию.

fact (0, 1).

fact (N, R) : – ***sub*** (N, 1, N1), ***fact*** (N1, P), ***mul*** (P, N, R).

или

fact (0,1)**.**

fact (N, R) : – N1=N–1, ***fact*** (N1, P), R=P*N.

Здесь N – исходное число, R – результат, P – промежуточный результат.

Задания III.4.5.

На языке Prolog составив факты ***отец***, ***мать*** и правила для утверждений ***дедушка*** и ***бабушка***, постройте правила для утверждений ***сын***, ***дочь***, ***внук*** и ***внучка***

Помощь:

Определение самого старшего поколения должен содержать определение среднего поколония, которое в свою очередь будет содержать определение младшего поколения.

III.4.5. Вопросы

1. Что является программной единицей в логическом языке?
2. Правильно ли работает эта программа дифференцирования алгебраического выражения?

% ***dif*** (+Выражение,+Переменная,–Результат).

dif(*X*,*X*,*1*) :– !.

dif(*C*,*X*,*0*) :– atomic(C).

dif(*–U*,*X*,*–A*) :– ***dif***(*U*,*X*,*A*). % (*–u*)'=*–u*'

dif(*U+V,X,A+B*) :– ***dif***(*U,X,A*), ***dif***(*V,X,B*). % (*u+v*)'=*u'+v'*.
dif(*U–V,X,A–B*) :– ***dif***(*U,X,A*), ***dif***(*V,X,B*).
dif(*C*U,X,C*A*) :– atomic(C),C\=X, ***dif***(*U,X,A*),!.
dif(*U*V,X,U*B+A*V*) :– ***dif***(*U,X,A*), ***dif***(*V,X,B*). % (*uv*)'=*uv'+u'v*
dif(*U/V,X,(A*V–U*B)/V?2*) :– ***dif***(*U,X,A*), ***dif***(*V,X,B*).
dif(*U?C,X,C*A*U?*(*C–1*)) :– ***dif***(*U,X,A*).
dif(*log*(*U*),*X,A/U*) :– ***dif***(*U,X,A*).
*?–**dif***(*x*x–2,x,R*).
*R=x*1+1*x–0*

III.4.5. Тесты

1. Какой из них является языком логического программирования?
   A) ML
   B) Prolog
   C) Lisp
   D) Pascal
   E) Parlog

2. Какой ответ Prolog-системы для следующего запроса?
Goal: ***random*** (X).
   A) Случайное действительное число от 0 до 1
   B) Целая часть числа X.
   C) Случайное целое число от 0 до 1
   D) Yes
   E) Округление X до целого числа

3. Как записывается утверждение на языке Prolog «Кто имеет детей, тот счастливый»?
   A) счастливый(X) :– отец(X, _)
   B) счастливый(X,Y) :– отец(X, Z)
   C) счастливый(X)
   D) отец(X, Z):–счастливый(X, Z)
   E) счастливый(X):–ребенок(X)

III.4.6. Продукционные языки программирования

Продукционные языки программирования появились при попытке программной реализации идеи *нормальных алгорифмов*, которые были разработаны в 1954 году А.Марковым для математического определения (формализации) понятия алгоритма.

В продукционных языках программирования основной единицей программы является *правило подстановки* (на английском – *rewriting*, на русском – *подстановка*), называемое *продукцией.*

Составление программы на продукционном языке сводится к тому, что используя простых правил подстановок, нужно строить сложные правила подстановки. Правила подстановки, группируясь друг с другом могут образовать *схемы правил подстановок*. Каждая схема правил подстановки именуется одним общим именем и строится ориентированно к логически целостного действия или отношения.

Среди продукционных языков самым распространенным и удобным является язык *Refal*, которого в 1968 году создал В. Турчин. Этот язык очень удобный для составления программ решения задач символьной обработки, в том числе перевода с одного языка на другой. В свое время на этом языке были разработаны трансляторы многих языков программирования Algol, Fortran, Simula, Dynamo, ТРИКС и т.б.

В языке Refal основной программной единицей является схема правил подстановки, названной *функцией*. Программа на этом языке состоит из именованных последовательностей функций, образованных правилми подстановками. Каждое правило имеет две части, разделенные между собой знаком «=». Левая часть определяет условие преминимости (образец исходных данных) правила подстановки, а правая часть – указывает на следующее выполняемое действие или проверяемое отношение. Кроме того, в правой части записываются вызовы функций, в том числе и рекурсивные вызовы (саму себя).

Применяется два вида рекурсивного вызова: прямой вызов и косвенный вызов. При прямом вызове указывается имя функции, которая содержит этого правила, а при косвенном вызове через другие функции осуществляется вызов саму себя этой функции.

В каждом рабочем шаге программы на языке Refal однозначно опрделяется какая функция вызывается и что является её входом и выходом. Входные и выходные данные любого правила подстановки, функции и программы задаются в виде специального символьного выражения, построенного с помощью некоторого правила. Имя функции задается *идентификатором* (цепочкой из букв и цифр, начинающейся с буквы).

Как и в других языках программирования в языке Refal имеются стандартные функции. Они объединившись образуют несколько групп. Спецификации некоторых из них даются ниже:

1. *Арифметические функции* определены только для *целых чисел*: **ADD** – *сложение*, **SUB** – *вычитание*, **MUL** – *умножение*, **DIV**– *деление*. Если через S1 и S2 обозначить целые числа, то вызовы и результаты функции будут следуюшими:

Вызов	**Результат**
<**ADD** (S1) S2>	S1+ S2
<**SUB** (S1) S2>	S1– S2
<**MUL** (S1) S2>	S1* S2
<**DIV** (S1) S2>	S1\ S2

Здесь результатыі функций *сложение*, *вычитание* и *умножение* как обычно, вопросов не вызают, а результат функции деления задается в виде N1(N2), где N1– частное, N2 – остаток и они удовлетворяет условию S1 = S2*N1+N2 & N2≤S2. Например, если:

<**DIV** (/5/) /3/>, то результат /1/(/2/);

<**DIV** (–/5/) /3/>, то результат –/1/(–/2/);

<**DIV** (–/5/) –/3/>, то результат /1/(–/2/);

<**DIV** (/4/) /2/>, то результат /2/(/0/),

где целые числа заключаются между двумя знаками «/».

2. *Символьные функции* определены над символьными выражениями. К ним относятся **FIRST, LAST** и **TYPE**.

Функция **FIRST (LAST)** имеет два аргумента и разбивает представленное вторым аргументом символьное выражение на две части, при этом в круглые скобки заносит термы сначала – с левой стороны (с конца – с правой стороны), количество которых равно числу, представленному первым аргументом. Обращение к этим функциям имеет вид <**FIRST** N E> и <**LAST** N E> соответственно, где N – целое число, E –произвольное символьное выражение. Их результаты зависят от двух случаев:

1) Если длина символьного выражения E больше или равно числа N, то (E1) E2 – для **FIRST** и E1(E2) – для **LAST**;

2) Если длина символьного выражения E меньше числа N, то *E – для **FIRST** и E* – для **LAST**;

Функция **TYPE** вызывается в виде < **TYPE** E > и она выбирает первую часть слевой стороны выражения E и в зависимости от её значения порождает различные символы спецификации, затем выдает результат в виде ξE, где ξ – символы спецификации. Значения символов спецификации показывается ниже:

№	**Первая часть левой стороны выражения E**	**Значение ξ**
1	Составной символ–метка	F
2	Составной символ–число	N
3	Объектный знак – буква	L
4	Объектный знак – цифра	D
5	Объектный знак – не буква и не цифра	O
6	Левая структурная (круглая) скобка	B
7	Пустая цепочка	*

Примеры III.4.6.

1. Составить программу проверки структуру входных данных на предмет совпадения со структурой «*идентификатор*». Нужная нам программа имеет следующий вид:

** Проверка первого символа*

ID E1 **= < ID1 <TYPE** E1>>

** Значение первого символа будет буквой?*

ID1 'L' SA E1 **= < ID2** (SA) **<TYPE** E1>>

** Значение следующего символа будет буквой или цифрой?*

ID2 (EX)'L' SA E1 **= < ID2** (EX SA) **<TYPE** E1>>

(EX)'D' SA E1 **= < ID2** (EX SA) **<TYPE** E1>>

(EX)'*****' = (EX)

(EX) SA E1 **= <PRINT '***** Ошибка**:** SA – не буква, не цифра'>

Вид вызова этой программы имеет вид <**ID** E>, а вид результата (E), и он выходит из третьего правила функции **ID2**, здесь E – любое символьное выражение, в том числе и пустая цепочка.

3. Если алгоритм Евклида для нахождения наибольшего общего делителя двух чисел N1 и N2 обозначить с помощью **NOD**, то можно на
4. писать следующее правило:

NOD (E1) E2 (/0/) = E1

(E1) E2 (E3) = < **NOD** (E3) <**DIV** (E1) E3>>

где E1, E2, E3 – символьное выражение, принимающее любое значение, в том числе и пустую цепочку.

Эта функция вызывается в виде: < **NOD** (N1) (N2) >.

3. Программа вычисления факториал числа N будет иметь вид:

FACT /0/ = /1/

S1 = <**MUL** (S1) <**FACT** <**SUB** (S1) /1/>>>

Задания III.4.6.

Какие результаты будут после вызова следующих функции?

a. <**ADD** (/5/) /7/>
b. <**SUB** (/13/) /4/>

c. <**MUL** (/3/) /4/>

d. <**DIV** (/8/) /2/>

Помощь

Здесь результаты функций *сложение*, *вычитание* и *умножение* будут обычными, для функции деление результат будет в виде двух N1(N2), удовлетворяющий условие S1 = S2*N1+N2 & N2≤S2.

Вопросы III.4.6.

1. Что является программной единицей в продукционных языках?
2. Какие стандартные функции имеются в языке Refal?
3. Для построения программы синтаксического анализа на языке Refal какие стандартные функции нужно применить?

Тесты III.4.6.

1. Какие функции используются для целых чисел?

A) ADD, SUB, MUL, DIV

B) DIV, SIN, EXP, LOG

C) ADD, SUM, MOD

D) MUL, COS, ADD, SUB

E) NOD, ID, MUL, FACT.

2. Какие виды есть у рекурсивного вызова?

A) Символьный вызов, косвенный вызов.

B) Косвенный вызов, встроенный вызов.

C) Прямой вызов, косвенный вызов.

D) Прямой вызов, встроенный вызов.

E) Косвенный вызов, символьный вызов.

3. Какой самый распространенный продукционный язык?

A) Dynamo

B) Algol

C) Fortran

D) Simula

E) Refal

III.4.7. Объектно-ориенттированные языки программирования

Одним из ярких представителей ООЯП является язык Java, разработанный компанией Sun Microsystems в 1995 году. Приложения Java обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной Java-машине вне зависимости от компьютерной архитектуры. Ниже будет описан язык Java.

Алфавит языка Java состоит из букв, десятичных цифр и специальных символов. Буквами считаются латинские буквы (кодируются в стандарте ASCII), буквы национальных алфавитов (кодируются в стандарте Unicode, кодировка UTF-16), а также соответствующие им символы, кодируемые управляющими последовательностями (о них будет рассказано чуть позже).

Имена переменных, подпрограмм-функций и других элементов языка программирования обозначаются с помощью идентификаторов, в которых можно применять только буквы и цифры, причём первой всегда должна быть буква (в том числе символы подчёркивания и доллара), а далее может идти произвольная комбинация букв и цифр с произвольной длиной. Некоторые символы национальных алфавитов рассматриваются как буквы, и их можно применять в идентификаторах. Но некоторые используются в качестве символов разделителей, и в идентификаторах их использовать нельзя.

Язык Java является *регистро-чувствительным*. Это значит, что идентификаторы чувствительны к тому, в каком регистре (верхнем или нижнем) набираются символы. Например, имена *i*1 и I1 соответствуют разным идентификаторам.

Переменная в языке Java именует ячейку памяти, содержимое которой может изменяться. Перед тем, как использовать какую-либо переменную, она должна быть задана в области программы, предшествующей месту, где эта переменная используется. При объявлении переменной сначала указывается тип переменной, а

затем идентификатор задаваемой переменной. В Java существует ряд предопределённых типов: ***int*** – целое число, ***float*** – вещественное число, ***boolean*** – логическое значение, Object – самый простой объектный тип (класс) Java, и т.д. Также имеется возможность задавать собственные объектные типы (классы), о чём будет рассказано позже.

Объявление переменных *a1* и *b1*, имеющих некий тип ***MyType***1, осуществляется так:

MyType*1 a1*,*b1*;

При этом ***MyType***1 – имя типа этих переменных.

Другой пример – объявление переменной *j* типа ***int*** :

int *j*;

Типы бывают предопределённые и пользовательские. Например, ***int*** – предопределённый тип, а ***MyType***1 – пользовательский. Для объявления переменной не требуется никакого зарезервированного слова, а имя типа пишется перед именами задаваемых переменных.

Объявление переменных может сопровождаться их инициализацией – присваиванием начальных значений. Приведём пример такого объявления целочисленных переменных *i*1 и *i*2 :

int *i*1=5;

int *i*2=-78;

либо

int *i*1=5, *i*2=-78;

Присваивания вида ***int*** *i*1=*i*2=5;, характерные для C/C++, запрещены. Для начинающих программистов отметим, что символ “=” используется в Java и многих других языках в качестве символа *присваивания*, а не символа равенства, как это принято в математике. Он означает, что значение, стоящее с правой стороны от этого символа, *копируется* в переменную, стоящую в левой части. То есть, например, присваивание b=a означает, что в переменную (ячейку) с именем b надо скопировать значение из переменной (ячейки) с

именем a. Поэтому неправильное с точки зрения математики выражение $x=x+1$ в программировании вполне корректно. Оно означает, что надо взять значение, хранящееся в ячейке с именем *x*, прибавить к нему 1 (это будет происходить где-то вне ячейки *x*), после чего получившийся результат записать в ячейку *x*, заменив им прежнее значение.

После объявления переменных они могут быть использованы в выражениях и присваиваниях:

переменная=значение;

переменная=выражение;
переменная1= переменная2;
и так далее. Например,
*i*1=*i*2+5**i*1;

Примитивными типами называются такие, для которых данные содержатся в одной ячейке памяти, и эта ячейка не имеет подъячеек. *Ссылочными типами* называются такие, для которых в ячейке памяти (*ссылочной переменной*) содержатся не сами данные, а только адреса этих данных, то есть *ссылки* на данные. При присваивании в ссылочную переменную заносится новый адрес, а не сами данные. Но непосредственного доступа к адресу, хранящемуся в ссылочных переменных, нет. Это сделано для обеспечения безопасности работы с данными – как с точки зрения устранения непреднамеренных ошибок, так и для устранения возможности намеренного взлома информации.

Если ссылочной переменной не присвоено ссылки, в ней хранится нулевой адрес, которому дано символическое имя null. Ссылки можно присваивать друг другу, если они совместимы по типам, а также присваивать значение null. При этом из одной ссылочной переменной в другую копируется адрес. Ссылочные переменные можно сравнивать на равенство, в том числе на равенство null. При этом сравниваются не данные, а их адреса, хранящиеся в ссылочных переменных.

В Java все типы делятся на примитивные и ссылочные. К примитивным типам относятся следующие предопределённые типы: целочисленные типы ***byte***, ***short***, ***int***, ***long***, ***char***, типы данных в формате с плавающей точкой ***float***, ***double***, а также булевский (логический) тип ***boolean*** и типы-перечисления, объявляемые с помощью зарезервированного слова ***enum*** (сокращение от enumeration – “перечисление”). Все остальные типы Java являются ссылочными.

Во многих конструкциях Java используется один оператор, но часто можно использовать последовательность из нескольких операторов, оформив их как составной оператор.

Составной оператор это блок кода между фигурными скобками {}: Имеется два способа форматирования текста с использованием фигурных скобок.

В первом из них скобки пишут друг под другом, а текст, находящийся между ними, сдвигают на 1-2 символа вправо (изредка – больше). Например:

```
Оператор
{
 последовательность простых или составных операторов
}
```

Во втором, открывающую фигурную скобку пишут на той же строке, где должен начинаться составной оператор, без переноса его на следующую строку, а закрывающую скобку - под первым словом. Например:

```
Оператор
{
последовательность простых или составных операторов
 }
```

После фигурных скобок по правилам Java, как и в /C++, ставить символ “;” не надо. Но его можно ставить после фигурных скобок в тех местах программы, где разрешается ставить ничего не делающий пустой оператор “;” .

Условный оператор ***if*** . У условного оператора ***if*** имеется две

формы: ***if*** и ***if- else***.

Первая форма:

if(условие)

оператор 1;

Если условие равно true, выполняется оператор 1. Если же условие==false, в операторе не выполняется никаких действий. Вторая форма:

if(условие)

оператор 1;

else

оператор 2;

В этом варианте оператора ***if*** если условие==false, то выполняется оператор 2. Обратите особое внимание на форматирование текста. Например:

```
if(a<b)
a=a+1;
else if(a==b)
a=a+1;
else{
a=a+1;
b=b+1;
};
```

Из этого правила имеется исключение: если подряд идёт большое количество операторов ***if***, умещающихся в одну строку, для повышения читаемости программ бывает целесообразно не переносить другие части операторов на отдельные строки. Надо отметить, что в операторе ***if*** в области выполнения, которая следует после условия, а также в области ***else***, должен стоять только один оператор, а не последовательность операторов. Поэтому запись оператора в виде

```
if(условие)
оператор 1;
оператор 2;
else
```

оператор 3;

недопустима.

В таких случаях применяют составной оператор, ограниченный фигурными скобками. Между ними, как мы знаем, может стоять произвольное число операторов:

if(условие){

оператор 1;

оператор 2;

}

else

оператор 3;

Если же мы напишем:

if(условие)

оператор 1;

else

оператор 2;

оператор 3;

Никакой диагностики ошибки компилятор не выдаст! Оператор 3 в этом случае никакого отношения к условию ***else*** иметь не будет – подобное форматирование текста будет подталкивать к логической ошибке. При следующем форматировании текста программы, эквивалентному предыдущему при компиляции, уже более очевидно, что оператор 3 не относится к части ***else***: ***if***(условие)

оператор 1;

else

оператор 2;

оператор 3;

Для того, чтобы оператор 3 относился к части ***else***, следует использовать составной оператор:

if(условие)

оператор 1;

else{

оператор 2;

оператор 3;

};

В случае последовательности операторов типа: ***if***(условие1) ***if***(условие 2) оператор 1 ***else*** оператор 2; имеющийся ***else*** относится к последнему ***if***, поэтому лучше отформатировать текст так:

if(условие 1)
if(условие 2)
оператор 1;
else
оператор 2;

Таким образом, если писать соответствующие ***if*** и ***else*** друг под другом, логика работы программы становится очевидной.

Оператор выбора ***switch*** является аналогом ***if*** для нескольких условий выбора. Синтаксис оператора следующий:

switch(выражение){
case значение 1: операторы 1;
……
case значение N: операторы N;
default: операторы;
}

Правда, крайне неудобно, что нельзя ни указывать диапазон значений, ни перечислять через запятую значения, которым соответствуют одинаковые операторы. Тип выражения должен быть каким-нибудь из целых типов. В частности, недопустимы вещественные типы. Работает оператор следующим образом: сначала вычисляется выражение. Затем вычисленное значение сравнивается со значениями вариантов, которые должны быть определены ещё на этапе компиляции программы. Если найден вариант, которому удовлетворяет значение выражения, то выполняется соответствующий этому варианту последовательность операторов, после чего НЕ ПРОИСХОДИТ выхода из оператора ***case***, что было бы естественно. Для такого выхода надо поставить оператор ***break***. Эта неприятная особенность Java унаследована от языка C. Часть с ***default*** является необязательной и выполняется, если ни один вариант не найден. Например:

switch(*i*/*j*){
case 1:
i=0;
break;
case 2:
i=2;
break;
case 10:
i=3;
j=*j*/10;
break;
default:
i=4;
};

У оператора ***switch*** имеется две особенности:

1. Можно писать произвольное число операторов для каждого варианта ***case***, что весьма удобно, но полностью выпадает из логики операторов языка Java;

2. Выход из выполнения последовательности операторов осуществляется с помощью оператора ***break***. Если он отсутствует, происходит “проваливание” в блок операторов, соответствующих следующему варианту за тем, с которым совпало значение выражения. При этом никакой проверки соответствия очередному значению не производится. И так продолжается до тех пор, пока не встретится оператор ***break*** или не кончатся все операторы в вариантах выбора. Такие правила проверки порождают типичную ошибку, называемую “забытый ***break***”.

Оператор цикла ***for***(блок инициализации; условие выполнения тела цикла; блок изменения счётчиков) оператор;

В блоке инициализации через запятую перечисляются операторы задания локальных переменных, область существования которых ограничивается оператором ***for***. Также могут быть присвоены значения переменным, заданным вне цикла. Но

инициализация может происходить только для переменных одного типа.

В блоке условия продолжения цикла проверяется выполнение условия, и если оно выполняется, идёт выполнение тела цикла, в качестве которого выступает *оператор*. Если же не выполняется – цикл прекращается, и идёт переход к оператору программы, следующему за оператором ***for***. После каждого выполнения тела цикла (очередного шага цикла) выполняются операторы блока изменения счётчиков. Они должны разделяться запятыми. Например:

```
for(int i=1,j=5; i+j<100; i++,j=i+2*j){
...
};
```

Каждый из блоков оператора ***for*** является необязательным, но при этом разделительные “;” требуется писать. Наиболее употребительное использование оператора ***for*** – для перебора значений некоторой переменной, увеличивающихся или уменьшающихся на 1, и выполнения последовательности операторов, использующих эти значения.

Переменная называется счетчиком цикла, а последовательности операторов – телом цикла. Пример 1: вычисление суммы последовательно идущих чисел.

Напишем цикл, в котором производится суммирование всех чисел от 1 до 100. Результат будем хранить в переменной result.

```
int result=0;
for(int i=1; i<=100; i++){
result=result+i;
};
```

Замечание III.4.7: В Java отсутствует специальная форма оператора ***for*** для перебора в цикле элементов массивов и наборов. Тем не менее оператор ***for*** позволяет последовательно обработать

все элементы массива или набора.

Пример поочерёдного вывода диалогов со значениями свойств компонентов, являющихся элементами массива компонентов главной формы приложения:

```
java.util.List components= java.util.Arrays.asList(this.getComponents());
for (Iterator iter = components.iterator();iter.hasNext();) {
Object elem = (Object) iter.next();
javax.swing.JOptionPane.showMessageDialog(null,"Компонент: "+
elem.toString()); }
```

Оператор цикла с предусловием ***while***

```
while(условие)
оператор;
```

Пока условие сохраняет значение true — в цикле выполняется оператор, иначе действие цикла прекращается. Если условие с самого начала false, цикл сразу прекращается, и тело цикла не выполнится ни разу.

Цикл ***while*** обычно применяют вместо цикла ***for*** в том случае, если условия продолжения достаточно сложные. В отличие от цикла ***for*** в этом случае нет формально заданного счётчика цикла, и не производится его автоматического изменения. За это отвечает программист. Хотя вполне возможно использование как цикла ***for*** вместо ***while***, так и наоборот. Многие программисты предпочитают пользоваться только циклом ***for***. Например:

```
i=1;
x=0;
while(i<=n){
x+=i;//эквивалентно x=x+i;
i*=2;//эквивалентно i=2*i;
};
```

В операторе ***while*** очень часто совершаются ошибки, приводящие к неустойчивости алгоритмов из-за сравнения чисел с плавающей точкой для проверки предела цикла. Это происходит из-

за ошибок представления таких чисел в компьютере. Но большинство программистов почему-то считает, что при сравнении на неравенство проблем не возникает, хотя это не так.

Например, если организовать с помощью оператора ***while*** цикл с вещественным счётчиком, аналогичный разобранному в разделе, посвящённому циклу ***for***. Пример: типичной ошибки в организации такого цикла приведён ниже:

```
double a=…;
double b=…;
double dx=…;
double x=a;
while(x<=b){
…
x=x+dx;
};
```

Как мы уже знаем, данный цикл будет обладать неустойчивостью в случае, когда на интервале от *a* до *b* укладывается целое число шагов. Например, при *a*=0, *b*=10, *dx*=0.1 тело цикла будет выполняться при *x*=0, *x*=0.1, …, *x*=9.9. А вот при *x*=10 тело цикла может либо выполниться, либо не выполниться – как повезёт! Причина связана с конечной точностью выполнения операций с числами в формате с плавающей точкой. Величина шага *dx* в двоичном представлении чуть-чуть отличается от значения 0.1, и при каждом цикле систематическая погрешность в значении *x* накапливается. Поэтому точное значение *x*=10 достигнуто не будет, величина *x* будет либо чуть-чуть меньше, либо чуть-чуть больше. В первом случае тело цикла выполнится, во втором – нет. Пройдёт либо 100, либо 101 итерация (число выполнений тела цикла).

Оператор цикла с постусловием ***do...while***

```
do
оператор;
while(условие);
```

Если условие принимает значение false, цикл прекращается.

Тело цикла выполняется до проверки условия, поэтому оно всегда выполнится хотя бы один раз. Например:

```
int i=0;
double x=1;
do{
i++; // i=i+1;
x*=i; // x=x*i;
}
while(i<n);
```

Если с помощью оператора ***do…while*** организуется цикл с вещественным счётчиком или другой проверкой на равенство или неравенство чисел типа ***float*** или ***double***, у него возникают точно такие же проблемы, как описанные для циклов ***for*** и ***while***. При необходимости организовать бесконечный цикл (с выходом изнутри тела цикла с помощью оператора прерывания) часто используют следующий вариант:

```
do{
…
}
while(false);
```

Операторы ***break*** и ***continue***

В некоторых случаях требуется изменить ход выполнения программы. В традиционных языках программирования для этих целей применяется оператор ***goto***, однако в Java он не поддерживается. Для этих целей применяются операторы ***break*** и ***continue***.

Оператор ***continue.*** Оператор ***continue*** может использоваться только в циклах ***while***, ***do***, ***for***.

Если в потоке вычислений встречается оператор ***continue***, то выполнение текущей последовательности операторов (выражений) должно быть прекращено и управление будет передано на начало блока, содержащего эту последовательность.

```
...
int x = (int)(Math.random()*10);
```

int *arr*[10] = {....}***for***(***int*** *cnt*=0;
cnt<10;***cnt***++) {
if(***arr***[***cnt***] == x) ***continue***;
...}

В данном случае, если в массиве *arr* встретится значение, равное *x*, то выполнится оператор ***continue*** и все операторы до конца блока будут пропущены, а управление будет передано на начало цикла.

Если оператор ***continue*** будет применен вне контекста оператора цикла, то будет выдана ошибка времени компиляции. В случае использования вложенных циклов оператору ***continue***, в качестве адреса перехода, может быть указана метка, относящаяся к одному из этих операторов.

Рассмотрим пример:

public class Test {
public Test() {}
public static void main(String[] args) {
Test t = new Test();
for(***int*** *i*=0; *i* < 10; *i*++){
if(*i* % 2 == 0) ***continue***;
System.out.print(" *i*=" + *i*);
} } }

В результате работы на консоль будет выведено:

i=1 *i*=3 *i*=5 *i*=7 *i*=9

При выполнении условия в строке 7 нормальная последовательность выполнения операторов будет прервана и управление будет передано на начало цикла. Таким образом, на консоль будут выводиться только нечетные значения.

Оператор ***break.*** Этот оператор, как и оператор ***continue***, изменяет последовательность выполнения, но не возвращает исполнение к началу цикла, а прерывает его.

public class Test {
public Test() {}
public static void main(String[] args) {

```
Test t = new Test();
int [] x = {1,2,4,0,8};
int y = 8;
for(int cnt=0;cnt < x.length;cnt++) {
if(0 == x[cnt]) break;
System.out.println("y/x = " + y/x[cnt]);
} } }
```

На консоль будет выведено:

y/x = 8;

y/x = 4;

y/x = 2.

При этом ошибки, связанной с делением на ноль, не произойдет, т.к. если значение элемента массива будет равно 0, то будет выполнено условие в строке 9 и выполнение цикла ***for*** будет прервано.

В качестве аргумента ***break*** может быть указана метка. Как и в случае с ***continue***, нельзя указывать в качестве аргумента метки блоков, в которых оператор ***break*** не содержится.

Примеры III.4.7:

Составить программу вычисления площади тругольника по известным значениям его сторон по следующей формуле:

$$p = \frac{a+b+c}{2};$$

$$s = \sqrt{p(p-a)(p-b)(p-c)}.$$

Здесь *a*, *b*, *c* – стороны, *p* – половина периметра, *s* – площадь треугольника.

Требуемая программа на языке Java будет иметь вид:

```
package kz.enu.inf;
import static java.lang.Math.sqrt;
import static java.lang.Math.pow;
public class Calculate {
  private int a = 20;
  private int b = 40;
```

```
 private int c = 50;
public void Find() {
        double p, s;
        p = (a+b+c)/2;
        s=sqrt(p*(p-a)*(p-b)*(p-c));
        System.out.println("s=" + s);
 }
public static void main(String[] args) {
Calculate obj = new Calculate();
        obj.Find();
}
}
```

Задания III.4.7:

1 Составить программу вычисления площадь полной поверхности и объем усеченного конуса по формулам:

$$S = \pi(R^2 + (R + r)\lambda + r^2);$$

$$V = \frac{1}{3}\pi h\left(R^2 + r^2 + Rr\right),$$

где $R = 20$ см, $r = 15$ см, $\lambda = 13$ см, $h = 12$ см.

2 Составить программу определения координаты точки, делящий отрезок AB, соединяющий две точки $A(x_1, y_1)$ и $B(x_2, y_2)$ в отношении $m : n$ по формулам:

$$x = \frac{x_1 + kx_2}{1 + k};$$

$$y = \frac{y_1 + ky_2}{1 + k},$$

где $k = m / n.$

Вопросы III.4.7:

1 Когда и какой компанией был разработан язык Java?

2 С помощью каких средств операционной системы запускается приложение Java?

3 Что такое идентификаторы?

Тесты III.4.7:

1. Какие варианты оператора условного перехода корректны?

A) if (i<j) { System.out.print("-1-"); }

B) if (i<j) then System.out.print("-2-");

C) if i<j { System.out.print("-3-"); }

D) if [i<j] System.out.print("-4-");

E) if {i<j} then System.out.print("-6-");.

2. Какие из идентификаторов являются корректными?

A) 2int;

B) int_#;

C) @_int;

D) +_2_;

E) $int;

3. Что будет выведено в результате запуска и компиляции следующего программного кода?

```
public class Quest6 {
    public static void main (String [] args) {
        System.out.print (“A”);
        main (“java 7”);
    }
    private static void main (String args) {
        System.out.print (“B”);
    }
}
```

A) ошибка компиляции

B) BA

C) AB

D) AA

E) компиляция пройдет успешно, при выполнении программа зациклится

IV. УМНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАТИКИ

IV.1. Методы сортировки

Ключевые слова: *запись*, *поле*, *ключ*, *массив*, *сортировка*, *методы сортировки*, *алгоритмы сортировки*, *трудность*, *структура данных*, *порядок*, *убывание*, *возрастание*, *поиск*, *мера эффективности*, *сортировка включением*, *сортировка выбором*, *сортировка обменом*, *ключ элемента*, *индекс элемента*.

Цель: *описание методы сортировки*, *строить алгоритмов упорядочивания*, *оценка трудности алгоритма сортировки*, *показать их достоинства и недостатки*.

Структура: Виды сортировки показаны на рисунке IV.1.

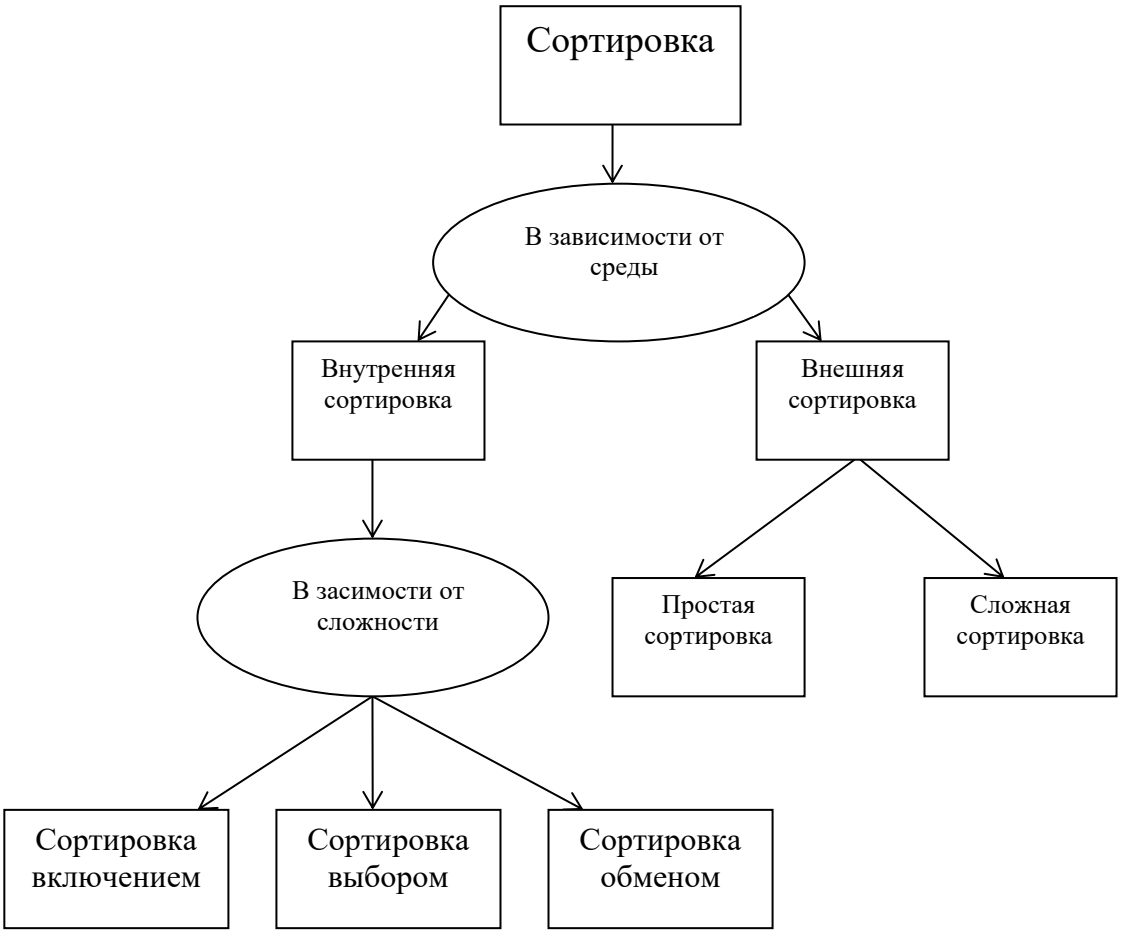


Рисунок IV.1. Виды сортировки.

IV.1.1. Понятие сортировки

Сортировка – перестановка мест элементов некоторой структуры данных для их размещения в порядке возрастания или убывания. Сортировка необходима для поиска нужных данных в любом месте, например, в справочниках, словарях, энциклопедиях, архивах и т.д. Итак основной целью сортировки структуры данных является облегчение работы по поиску нужного элемента.

Задачу сортировки в общем виде можно определить так: задаются однотипные записи – элементы массива; их одно поле – значение ключа, тип ключа должен обеспечить операцию сравнения ("=", ">", "<", "≥", "≤"); задача сортировки преобразование исходной последовательности в последовательность, содержащей эти записи, но значения их ключей должны располагаться в порядке возрастания (или убывания).

В информатике методы сортировки являются очень важными. Кроме того, сортировка связана со многими методами построения алгоритмов в будущих информационных технологиях. Поэтому методы и алгоритмы сортировки являются основными разделами изучения информатики и информационных технологий.

Существуют несколько методов и алгоритмов сортировки. Каждая сортировка является программой. Поэтому разница между двумя алгоритмами сортировки в зависимости от метода программирования может быть в несколько раз меньше, чем разница между двумя программами, реализуюшими один и тот же алгоритм сортировки и написанными на одном языке программирования.

Это также касается и производительности разных версий одного и того же алгоритма на языке программирования низкого уровня и высокого уровня.

В сортировке нет единственного эффективного алгоритма, так как для оценки его эффективности есть много параметров: *время,*

память, устойчивость.

Методы сортировки подразделяются на две большие группы: *внутрение сортировки* и *внешние сортировки*.

Внутренние сортировки – это методы, применяемые только к полностью размещенным в оперативной памяти структурам данных (например, массиву) и имеющим свободный доступ к элементам.

Внешние сортировки – это методы, применяемые к невмещающим в оперативной памяти структурам данных (например, файлам), которые размещаются на внешней памяти и имеют не свободный, а только последовательный доступ к элементам.

Во внешнем методе сортировки определенная часть файла считывается в оперативной памяти, там упорядочивается, а затем переносится во внешнюю память. Этот процесс будет повторяться пока файл полностью не упорядочится. У внешней сортировки технические проблемы, связанные с применяемой системы больше, чем алгоритмическая проблема. Поэтому мы внешнюю сортировку не будем рассматривать.

Основным требованием, предъявляемым к методам внутренней сортировки является экономия объема памяти. Это означает, что перестановку элементов надо осуществлять на том же месте, без привлечения и размещения вспомогательной структуры данных. Поэтому мы не будем рассматривать методы, требующих вспомогательные структуры данных.

Классификация методов сортировки по критерию экономии объема памяти соответствует с эффективностью времени и скорости. *Мера эффективности* определяется с помощью числа необходимых *сравнений ключей C* и *пересылок элементов M*, которые задаются некоторыми функциями от числа сортируемых элементов ***n***.

Обычно простые методы сортировки требуют много сравнений, чем сложные методы сортировки. Простые методы сортировки имеют следующие свойства:

– простые методы удобны для анализа свойств многих принципов сортировки;

– простые методы реализуются с помощью легких операций и легки для понимания;

– в простых методах программы состоят из небольшого количества операций и для размещения текста программы потребуется малый объем памяти;

– при малом количестве сортируемых элементов n простые методы работают быстрее, но их нельзя применить при большом n.

Имеются три принципа сортировки: *сортировка включением*, *сортировка выбором*, *сортировка обменом*. Для их обсуждения введем следующие обозначения:

1) В качестве сортируемой структуры данных возьмем одномерный массив чисел.

2) Сортируемый массив обозначим через a, а его элементы представляются в виде $a[1]$, $a[2]$,..., $a[n]$, здесь в квадратных скобках показаны *индексы элементов*.

Тогда сортировка массива a означает такую перестановку $a[k_1]$, $a[k_2]$,..., $a[k_n]$, что при заданной функции упорядочения F будет справедливо n-местное отношение:

$$F(a[k_1]) \le F(a[k_2]) \le ... \le F(a[k_n]).$$

Здесь для заданных индексов $1 \le i \le n$ и $1 \le k_i \le n$ может быть $a[i] \ne a[k_i]$ и вместо знака “$\le$” можно использовать и знак “$\ge$”.

Обычно функция упорядочения не вычисляется по какому-то специальному правилу, а содержится в каждом элементе в виде явной компоненты, значение которой называется *ключом элемента*, т.е. функцию упорядочения F можно считать как функцию присваивания значений *индексным переменным* $a[k_1]$,$a[k_2]$,..., $a[k_n]$.

3) Алгоритмы будут записаны на абстрактном языке Pascal, в которых будут использоваться необходимые для представления

элементов и индексов типы данных *item* и *index*, заданные как:

```
type item = record key : integer;
        type index = 0..n;
        var a : array[1. n] of item
end
```

Примеры IV.1.1:

1. Рассматриваются два элемента массива, которые в пузырьковой сортировке стоят рядом: если значение элемента с меньшим индексом больше, чем значение элемента с большим индексом, то их меняем местами. При наличии таких пар это действие будет повторяться. В результате элементы массива будут отсортированы. На каждом шаге алгоритма хотя бы один элемент займет свое место:

```
Program BubbleSort;
    var a : array[1.1000] of integer;
    n, i, j, p : integer;
    begin
    for i:=1 to n do
        for j:=1 to n–i do
        if a[j]>a[j+1] then
            begin {Обмен элементов}
                p:=a[j]; a[j]:=a[j+1]; a[j+1]:=p;
            end;
    end.
```

2. При быстрой сортировке элементы массива разделяются на две части, так чтобы все элементы первой части будут меньше, чем любой элемент второй части. Затем каждая часть сортируется отдельно. Разделение на две части касается некоторому элементу, т.е. этот элемент не меньше всех элементов первой части и не больше всех элементов второй части.

```
Program QuickSort;
    var a : array[1.1000] of integer;
```

```
n, t : integer;
procedure Sort(p, q : integer);
    {p, q – начало и конец сортируемых частей}
        var i, j, r : integer;
      begin
            if p<q then
            begin
                r:=a[p]; i:=p–1; j:=q+1;
                while i<j do
                begin
                        repeat
                            i:=i+1;
                        until a[i]>=r;
                        repeat
                            j:=j–1;
                        until a[j]<=r;
                        if i<j then
                        begin
                            t:=a[i]; a[i]:=a[j]; a[j]:=t;
                        end;
                  end;
                    Sort(p, j); Sort(j+1, q);
            end;
      end;
```

Два индекса с двух сторон ищут не попавшего в свою часть элемента. Если находятся такие элементы, то их меняют местами. На каком элементе эти индексы пересекутся, этот элемент будет разделителем частей:

Задания IV.1.1:

1. Покажите начальные условия, выполняемые действия и результаты задачи сортировки.
2. Покажите разницу между внутренней сортировкой и внешней сортировкой.
3. Какую работу облегчает сортировка и для чего она нужна?

Помощь

1. Надо рассматривать структуру данных с однотипными элементами и между ними имеются отношения упорядочивания.
2. Надо знать причину почему внутренняя сортировка работает быстрее, чем внешняя сортировка.
3. Надо вспомнить в чем заключается работа в Интернете.

Вопросы IV.1.1:

1. Что такая сортировка?
2. Как определяется функция упорядочения?
2. Какие виды сортировки имеются?

Тесты IV.1.1:

1. Какие принципы сортировки есть?

A) сортировка включением, выбором, обменом;
B) сортировка выводом, замены, делением;
C) сортировка анализом, выбором, сжатием;
D) сортировка внедрением, обменом, сбором;
E) сортировка сравнением, объединением, выбором.

2. Какие виды сортировки есть?

A) внутренняя сортировка, внешняя сортировка;
B) раздельная сортировка, совместная сортировка;
C) медленная сортировка, быстрая сортировка;
D) последовательная сортировка, параллельная сортировка;
E) прямая сортировка, поперечная сортировка.

3. Чем определяется измерение эффективности сортировки?

A) числом сравнений, числом обмена;
B) числом ввода, числом вывода;
C) числом повторения, числом последовательности;
D) количеством элементов, количеством времени;
E) числом просмотра, числом смещения.

IV.1.2. Сортировка включением

Идея сортировки включением проста: выбирается некоторый элемент, остальные элементы сортируются, выбранный элемент «включается», т.е. его ставят на место среди других элементов.

Суть *метода сортировки вкючением элемента* состоит в:

1) Сортируемый массив *a* условно разделяется на две последовательности:

входную последовательность a[1], a[2], ..., a[n]

готовую последовательность a[1], a[2], ..., a[i-1].

2) Сортировка происходит по шагам и на каждом шаге, начиная с *i=2* и увеличивая *i* на единицу, берут *i*-й элемент входной последовательности и передают в готовую последовательность, вставляя его на подходящее место.

При поиске подходящего места в *a[1], a[2],...,a[i]* удобно чередовать сравнения и пересылки, т. е. как бы «просеивать» *x*, сравнивая его с очередным элементом *a[i]* и либо вставляя *x*, либо пересылая *a[i]* направо и продвигаясь налево. Заметим, что «просеивание» может закончиться при двух различных условиях:

1. Найден элемент *a[i]* с ключом меньшим, чем ключ *x.*

2. Достигнут левый конец готовой последовательности.

Алгоритм сортировки простым включением имеет вид

```
for t:=2 to n do
    begin x:=a[i];
       «включить x в последовательность a[1], a[2], ..., a[i]»
    end
```

Этот типичный пример цикла с двумя условиями окончания дает нам возможность рассмотреть хорошо известный *прием фиктивного элемента*, используемого в качестве барьера. Его можно легко применить в этом случае, установив барьер *a[0] = x.* При этом индекс для *a* начинается с 0, т.е. нужно расширить диапазон индексов в описании *a* до *0, ..., n.*

Алгоритм сортировки включением представлен в виде

следующей программы IV.1.2.1.

```
program jear;
var i,j : index; x : item;
begin
      for i:=2 to n do
            begin x:=a[i]; a[0]:=x; j:=i–1;
                  while x.key < a[j].key do
                        begin a[j+1]:=a[i]; j:=j–1;
                        end ;
                  a[j+1]:= x
            end
end.
```

Программа IV.1.2.1. Алгоритм сортировки включением

Число сравнений ключей C_i при 1-м просеивании составляет самое большее i-1, самое меньшее 1 и, если предположить, что все перестановки n ключей равновероятны, в среднем равно $i/2$. Число пересылок (присваиваний) M_i с учетом барьера равно $C_i + 2$. Общее число сравнений и пересылок показано в таблице IV.1.2.2.

Таблица IV.1.2.2. Число сравнений и перестановок.

Число сравнения	**Число перестановки**
$C_{\min} = n - 1$	$M_{\min} = 2(n-1)$
$C_{mid} = (n^2 + n - 2)/4$	$M_{mid} = (n^2 + 9n - 10)/4$
$C_{\max} = (n^2 + n)/2 - 1$	$M_{\max} = (n^2 + 3n - 4)/2$

Здесь *min* – минимальный, *mid* – средний, *max* – максимальный, *min* появляются, если элементы с самого начала упорядочены, а *max* встречается, если элементы расположены в обратном порядке.

Алгоритм описывает *устойчивую* сортировку: не изменяет порядок элементов с одинаковыми ключами.

Алгоритм сортировки включением легко можно улучшить тем, что готовая последовательность *a[1], a[2], ..., a[i-1]* уже упорядочена. Место включения находится значительно быстрее. Здесь можно применить бинарный поиск, который исследует средний элемент готовой последовательности и продолжает деление пополам, пока не будет найдено место включения.

Анализ сортировки бинарными включениями. Место включения найдено, если $a[l].key \le x.key < a[r].key$. Иинтервал поиска в конце должен быть равен 1; это означает, что интервал из i ключей делится пополам $[\log_2 i]$ раз. Итак, $C = \sum_{i=1}^{n} [\log_2 i]$. Мы аппроксимируем эту сумму с помощью интеграла

$$\int_1^n \log x dx = x(\log x - c)\Big|_1^n = n(\log n - c) + c$$

где $c = \log e = 1/\ln 2 = 1.45269\,\text{K}$. Количество сравнений не зависит от исходного порядка элементов. Но из-за округления при делении интервала поиска пополам действительное число сравнений для элементов может быть на 1 больше ожидаемого. Природа «перекоса» такова, что в результате места включения в нижней части находятся в среднем быстрее, чем в верхней части. Это дает преимущество тогда, когда элементы изначально не упорядочены. На самом деле минимальное число сравнений требуется, если элементы вначале расположены в обратном порядке, а максимальное – если они уже упорядочены. Поэтому это случай *неестественного поведения* алгоритма сортировки:

$$\boldsymbol{C = n(} \log \boldsymbol{n} - \log \boldsymbol{e} \pm 0.5)$$

Улучшение, получаемое использованием метода бинарного поиска, касается только числа сравнений, а не числа пересылок. В действительности, поскольку пересылка элементов, т. е. ключей и

сопутствующей информации, обычно требует значительно больше времени, чем сравнение двух ключей, то это улучшение ни в коей мере не является решающим: важный показатель ***M*** по-прежнему остается порядка n^2. В самом деле, пересортировка уже рассортированного массива занимает больше времени, чем при сортировке включением с последовательным поиском.

Примеры IV.1.2.

Для случайно взятых восьми целых чисел процесс сортировки включением по возрастанию ключей показан в таблице IV.1.2.1. На каждом шаге подчеркиваются числа, нарушающие порядок.

Таблица IV.1.2.17 Пример сортировки включением

Начальные значения	**45**	**57**	**13**	**33**	**96**	**21**	**07**	**74**
i=2	***45***	**57**	**13**	**33**	**96**	**21**	**07**	**74**
i=3	**13**	**45**	**57**	**33**	**96**	**21**	**07**	**74**
i=4	**13**	**33**	**45**	**57**	**96**	**21**	**07**	**74**
i=5	**13**	**33**	**45**	**57**	**96**	**21**	**07**	**74**
i=6	**13**	**21**	**33**	**45**	**57**	**96**	**07**	**74**
i=7	**07**	**13**	**21**	**33**	**45**	**57**	**96**	**74**
i=8	**07**	**13**	**21**	**33**	**45**	**57**	**74**	**96**

Задания IV.1.2.

1. Покажите случаи окончания фильтрации элемента.
2. Откройте смысл сортировки внедрением.
3. Покажите условия применения бинарной сортировки.

Помощь

1. Надо учесть, что фильтрация останавливается в двух случаях.
2. Рассматриваются начальная и готовая последоваетельности.
3. Начальная последовательность подразделяется на два.

Вопросы IV.1.2.

1. Какие последовательности нужны в сортировке включением?

2. В каком случае число сравнений ключей составляет самое большее *i*-1, самое меньшее 1?

3. Зависимо ли число сравнений от начальной упорядоченности элементов?

Тесты IV.1.2.

1. К чему относится эффективность, полученная применением бинарной сортировки?

A) числу сравнений;
B) числу обмена;
C) числу элементов;
D) числу операций;
E) числу внедрения.

2. Какие значения имеют число сравнений в алгоритме сортировки?

A) самое маленькое, среднее, самое большое;
B) начальное, вычисленное, результирующее;
C) первое, среднее, конечное;
D) заданное, вычисленное, случайное;
E) возрастающее, постоянное, убывающее.

3. Какие значения имеет число обмена в алгоритме сортировки?

A) самое маленькое, среднее, самое большое;
B) первое, среднее, конечное;
C) начальное, вычисленное, результирующее;
D) заданное, вычисленное, случайное;
E) возрастающее, постоянное, убывающее.

IV.1.3. Сортировка выбором

Метод сортировки выбором основан на правиле:

1. Выбирается элемент с наименьшим ключом.

2. Он меняется местами с первым элементом *a*[*1*].

Эти операции затем повторяются с оставшимися *n*-1 элментами, затем *n-2* элементами, пока не останется только один элемент– наибольший. Реализацию этого метода можно написать в виде следующего алгоритма:

for *i*:=*1* **to** *n–1* **do**

begin

«Присвоить k индекс самого меньшего элемента среди элементов a[*1*],…,*a*[*n*]*»*;

«перестановка мест a[*i*] *и a*[*k*]*»*

end

Метод сортировки выбором, в некотором смысле противоположен сортировке включением; при сортировке включением на каждом шаге рассматривается только один очередной элемент входной последовательности и все элементы готового массива для нахождения места включения; при сортировке выбором рассматриваются все элементы входного массива для нахождения элемента с наименьшим ключом, и этот один очередной элемент отправляется в готовую последовательность.

Анализ сортировки выбором. Число сравнений ключей ***C*** не зависит от начального порядка ключей. С этой точки зрения этот метод проще, чем метод простой сортировки включением. Число сравнений ключей:

$$C = (n^2 - n)/2$$

В случае изначально прямо упорядоченных ключей число перестановок будет минимальным:

$$M_{\min} = 2(n-1)$$

А в случае изначально обратно упорядоченности ключей, число перестановок принимает наибольшее значение:

$$M_{\max} = trunc(n^2 / 4) + 3(n-1)$$

Среднее значение M_{mid} трудно определить, несмотря на простоту алгоритма. Оно зависит от того, сколько раз определяется, что k_i меньше всех предшествующих величин k_1,.., k_{i-1} при просмотре последовательности чисел k_1,.., k_n. Среднее значение перестановки

$$M_{mid} = n(\gamma + 1) + \sum_{i=1}^{n} \ln i,$$

где γ == *0.577216.. – константа Эйлера.*

Апроксимировав эту сумму с помощью следующего интеграла

$$\int_1^n \ln x dx = x(\ln x - 1)\Big|_1^n = n\ln n - n + 1$$

можно получить в среднем для всех перестановок n ключей, число которых равно

$$M_{mid} = n(\ln n + \gamma)$$

IV.1.3. Примеры. Метод сортировки выбором показан в таблице IV.1.3.

Таблица IV.1.3. Метод сортировки выбором

Начальные ключи	**45**	**57**	**13**	**33**	**96**	**21**	**07**	**74**
i=2	**07**	**57**	**13**	**33**	**96**	**21**	**45**	**74**
i=3	**07**	**13**	**57**	**33**	**96**	**21**	**45**	**74**
i=4	**07**	**13**	**21**	**33**	**96**	**57**	**45**	**74**
i=5	**07**	**13**	**21**	**33**	**96**	**57**	**45**	**74**
i=6	**07**	**13**	**21**	**33**	**45**	**57**	**96**	**74**
i=7	**07**	**13**	**21**	**33**	**45**	**57**	**96**	**74**
i=8	**07**	**13**	**21**	**33**	**45**	**57**	**74**	**96**

Задания IV.1.3.

1. Объясните смысл элемента с самым маленьким ключом.
2. Укажите сколько элементов рассматривается на каждом шаге.
3. Укажите от чего не зависит число сравнений ключей.

Помощь

1. Следует помнить о готовой последовательности.
2. Это зависит от вида последовательности.
3. Учесть начальный порядок ключа.

IV.1.3. Вопросы

1. На каком правиле основана сортировка выбором?
2. Как находится элемент с наименьшим ключом при сортировке выбором?
3. В сортировке простым выбором зависит ли число сравнений ключей от их начальной упорядоченностей?

Тесты IV.1.3.

1. В каком случае число сравнений будет минимальным?
A) первоначально ключи прямо упорядочены;
B) первоначально ключи обратно упорядочены;
C) первоначально ключи отсортированы;
D) первоначально ключи не сортированы;
E) первоначально ключи прямо не упорядочены.

2. Что будет когда ключи обратно упорядочены?
A) число сравнений не принимает значение;
B) число сравнений принимает минимальное значение;
C) число сравнений принимает максимальное значение;
D) число сравнений не принимает максимальное значение;
E) число сравнений не будет минимальным.

3. В каком случае число сравнений будет максимальным?
A) первоначально ключи обратно упорядочены;
B) первоначально ключи не упорядочены;
C) первоначально ключи частично упорядочены;
D) первоначально ключи прямо упорядочены;
E) первоначально ключи прямо не упорядочены.

IV.1.4. Сортировка обменом

Сортировка обменом основан на принципе сравнения и обмена пары соседних элементов массива (последовательности) до тех пор, пока не будут рассортированы все элементы. Она предполагает, что элементы сортируемого массива имели свои веса (ключи) и расположилась вертикально.

Метод сортировки обменом выполняет просмотр сортируемого массива несколько раз, уменьшая с каждым разом противоречия упорядочивания. При этом, если в каждом просмотре оставшуюся часть массива от предыдущего просмотра выбирается самый легкий (маленький) элемент, то он поднимается на уровень в соответствии со своим весом. Иначе говоря, в каждом просмотре появляется пузырек - самый легкий элемент. Поэтому этот метод назвали *методом пузырька*. *Алгоритм сортировки по методу пузырька* показан в программе IV.1.4.A.

```
procedure jaar;
    var i, j: index; x: item;
    begin for i:=2 to n do
        begin for j:= n downto i do
            if a[j-1].key > a[j].key then
            begin x:=a[j-l]; a[j-1]:=a[j]; a[j]:= x
            end
        end
    end
```

Программа IV.1.4.A. Сортировка методом пузырька

Для облегчения работы алгоритма нужно запоминать какой обмен производится. Если обмена не будет, то алгоритм заканчивает свою работу. При выполнении обмена надо контролировать место (индекс) последнего обмена *k*. Так как очевидно, что элементы, у которых индексы меньше, чем *k*, уже отсортированы. Поэтому в следующем просмотре, не достигая заранее установленной нижней границы *i*, при достижении этого индекса *k* можно остановить работу.

При анализе алгоритма сортировки методом пузырька нужно

учесть наличие следующей *ассиметрии*: в отсортированном массиве один «легкий» элемент (пузырек), расположенный в конце «тяжелой» стороны, найдет свое место только за один просмотр, а один «тяжелый» элемент, расположенный в начале «легкой» стороны найдет свое место только за один шаг в каждом просмотре. Например, при сортировке методом пузырька первая последовательность 21 33 45 57 74 96 07 отсортируется только за одни просмотр, а для сортировки второй последовательности 96 07 13 21 33 45 57 74 потребуются семь просмотров.

Эта ассиметрия показывает, что можно усовершенствовать сортировку методом пузырька. Для этого в каждом следующем просмотре, чтобы найти пузырек, нужно менять направление (вверх, вниз). Полученный таким образом алгоритм называется *шейкер-сортировкой*, он показан в программе IV.1.4.B.

```
procedure shakersort;
    var j,k,l,r: index; x: item;
    begin l:=2; r:=n; k:=n;
        repeat
            for j:=r downto l do
                if a[j-1].key>a[j].key then
                begin x := a[j-1]; a[j-1]:=a[j]; a[j]:=x;
                k:=j
                end ; l:= k+1;
            for j :=l to r do
                if a[j-1].key > a[j].key then
                begin x:=a[j-1]; a[j-1]:=a[j];a[j] :=x; k:=j
                end ;
            r:=k-1;
        until l > r
    end
```

Программа IV.1.4.B. Шейкер-сортировка.

Анализируем метода пузырька и метода шейкер-сортировка.

В методе обмена (пузырька) число сравнений будет

$$C = (n^2 - n)/2,$$

а минимальное, среднее и максимальное числа обмена такие:

$$M_{\min} = 0,\ M_{mid} = 3(n^2 - n)/4,\ M_{\max} = 3(n^2 - n)/2.$$

Для шейкер-сортировки минимальное число сравнений будет

$$C_{\min} = n - 1$$

Среднее число просмотра последовательности пропорционально $n - k_1\sqrt{n}$, а среднее число сравнений пропорционально $[n^2 - nk_2 + \ln n]/2$.

Во время сортировки среди *n* элементов среднее растояние каждого из них равно *n/3*. Это послужит основанием для разработки эффективных методов сортировки. В основном все простые методы на каждом шаге сдвигают каждый элемент на одну позицию. Поэтому они требуют, чтобы количество шагов было $n.^2$ Любое усовершенствование должно основываться на принципе отправления на дальнее растояние на одном цикле.

Теперь сравним рассмотренные выше все методы простой сортировки. Для этого покажем в таблице IV.1.4.A все формулы, дающие возможность вычислить минимальное, среднее и максимальное значения числа сравнений и числа обмена.

Таблица IV.1.4.A. Сравнение методов простой сортировки

Название метода	**Минимальное значение**	**Среднее значение**	**Максимальное значение**
Простое включение	$C_{\min} = n - 1$ $M_{\min} = 2(n-1)$	$C_{mid} = (n^2 + n - 2)/4$ $M_{mid} = (n^2 + 9n - 10)/4$	$C_{\max} = (n^2 + n)/2 - 1$ $M_{\max} = (n^2 + 3n - 4)/2$
Простой выбор	$C_{\min} = (n^2 - n)/2$ $M_{\min} = 3(n-1)$	$C_{mid} = (n^2 - n)/2$ $M_{mid} = n\ (\ln n + 0.57)$	$C_{\max} = (n^2 - n)/2$ $M_{\max} = n^2/4 + 3(n-1)$
Ростой обмен	$C_{\min} = (n^2 - n)/2$ $M_{\min} = 0$	$C_{mid} = (n^2 - n)/2$ $M_{mid} = (n^2 - n)*0.75$	$C_{\max} = (n^2 - n)/2$ $M_{\max} = (n^2 - n)*1.5$

Из указанных методов можно сделать следующие заключения:

1. Метод обмена (пузырька) самый плохой среди всех сравненных методов. Его усовершенствованный вид шейкер-

сортировка тоже не эффективен по сравнению с методами сортировки включения и выбора.

2. Метод сортировки выбора является самым хорошим среди сравненных методов.

Итак, из этого анализа можно заметить, что сортировка обменом намного сложнее, чем сортировка включением и сортировка выбором.

Примеры IV.1.4.

1. Исходная последовательность 45 57 13 33 96 21 07 74 сортируется методом пузырька и получился следующий результат 07 13 21 33 45 57 74 96, который потребовал восемь просмотров показанных в таблице IV.1.4.B.

Таблица IV.1.4.B. Пример сортировки методом пузырька.

Исходные ключи	*i=2*	*i=3*	*i=4*	*i=5*	*i=6*	*i=7*	*i=8*
45	**07**	**07**	**07**	**07**	**07**	**07**	**07**
57	**45**	**13**	**13**	**13**	**13**	**13**	**13**
13	**57**	**45**	**21**	**21**	**21**	**21**	**21**
33	**13**	**57**	**45**	**33**	**33**	**33**	**33**
96	**33**	**21**	**57**	**45**	**45**	**45**	**45**
21	**96**	**33**	**33**	**57**	**57**	**57**	**57**
07	**21**	**96**	**74**	**74**	**74**	**74**	**74**
74	**74**	**74**	**96**	**96**	**96**	**96**	**96**

Усовершенствовать метод пузырька легко. В таблице IV.1.4.B последние три просмотра не оказывают никакого влияние на порядок элементов. Пусть 45 57 13 33 96 21 07 74 есть исходная последовательность, к которой применяется шейкер-сортировка для расположения её элементов в порядке возрастания, чтобы получить

07 13 21 33 45 57 74 96. Работа алгоритма шейкер-сортировки, представленного в программе IV.1.4.B, для решения этой конкретной задачи, показана в таблице IV.1.4.C.

Таблица IV.1.4.C. Пример шейкер-сортировке.

i=2 ***r=8*** ↑	***i=3*** ***r=8*** ↓	***i=3*** ***r=7*** ↑	***i=4*** ***r=7*** ↓	***i=4*** ***r=4*** ↑
45	**07**	**07**	**07**	**07**
57	**45**	**45**	**13**	**13**
13	**57**	**13**	**45**	**21**
33	**13**	**33**	**21**	**33**
96	**33**	**57**	**33**	**45**
21	**96**	**21**	**57**	**57**
07	**21**	**74**	**74**	**74**
74	**74**	**96**	**96**	**96**

Здесь видно, что поочередно три раза вверх и два раза вниз меняется направление просмотра.

Задания IV.1.4.

1. Указать на каком принципе основан и что предпочитает сортировка обменом.
2. Определить количество просмотров при сортировке заданной последовательности 25 37 42 54 75 91 06 методом пузырька.
3. Определить количество просмотров при сортировке заданной последовательности 19 31 43 52 69 87 05 шейкер-сортировкой.

Помощь

1. Сортировка обменом основана на принципе сравнения и обмена пары соседних элементов массива.
2. Составляемая программа должна основываться по алгоритму

сортировки обменом, представленному в программе IV.1.4.A и её протокол работы должен быть как протокол из таблицы IV.1.4.B.

3. Составляемая программа должна основываться по алгоритму *шейкер-сортировка*, представленному в программе IV.1.4.B и её протокол работы должен быть как протокол из таблицы IV.1.4.C.

Вопросы IV.1.4.

1. На каком принципе основана сортировка обменом?
2. Какая ассиметрия имеется в методе пузырька?
3. Когда останавливается алгоритм по методу пузырька?

Тесты IV.1.4.

1. Какой из них является минимальное число сравнений для шейкер-сортировки?

A) $C_{\min} = n - 1$;

B) $C_{\min} = (n^2 - n)/2$;

C) $M_{\min} = 2(n-1)$;

D) $M_{\min} = 3(n-1)$;

E) $M_{\min} = 0$.

2. Какой из методов сортировки самый эффективный?

A) Сортировка выбором;

B) Сортировка обменом;

C) Сортировка включением;

D) Шейкер-сортировка;

E) Все методы эффективны.

3. Какой из методов сортировки самый плохой?

A) Сортировка выбором;

B) Сортировка включением;

C) Сортировка обменом;

D) Шейкер-сортировка;

E) Все методы плохи.

ЛИТЕРАТУРА

1. Colmeraurer J., Kanoui V., Prolog bases et developpements actuels, T.A.1,vol.2, №4, 1983, pp. 112–120.
2. McCarty J. A LISP programmer’s manual. –Cambridge, Mass.: M.T.T. Press, 1963.
3. Бауэр Ф. Гооз Г. Информатика –М.: Мир, том 1,2, 1990. – 742 с.
4. Вирт Н. Алгоритмы+структуры данных = программы. –М.: Мир, 1985.
5. Грэхем Иан. Объектно-ориентированные методы. Принципы и практика. –3-е изд. –М.: Вильямс, 2004. – 880 с.
6. Кнут Д. Искусство программирования. Сортировка и поиск. - М.: Мир, 2000, т.3. – 822 с.
7. Лорин Г. Сортировка и системы сортировки. - М.: Наука, 1983, –384 с.
8. Пратт Т., Зелковиц М. Языки программирования. Разработка и реализация. 4-е издание. –М.:, С.П.: Питер, 2002. –688 с.
9. Турчин В. Программирование на языке Рефал. - М.: Препринт ИПМ АН СССР, 1971, № 41, № 43, № 44, №48.
10. Шәріпбаев А. Информатика. - Алматы: Қазақ университеті, 1992. –72 б.
11. Шәріпбаев А., Ыскакова А., Рахлетова А. Информатика пәнінен қазақша, орысша, ағылшынша түсіндірме сөздігі. – Алматы: Сөздік-Словарь, 2002. –156 б.
12. Шәріпбаев А. Информатика. –Астана: Нұржол, 2003. –168 б.
13. Шарипбаев А.А. и др. СТ РК 1048 – 2002. Информационные технологии. 8-битовая кодовая таблица казахского алфавита.
14. Шарипбаев А.А. и др. Информатика и вычислительная техника. Казахско-русский, русско-казахский словарь. Одобрено Республиканской терминологической комиссией при Правительстве РК, КАЗақпарат, –Алматы, 2014, –452 с.

ОТВЕТЫ

Ответы упражнений

Задания I.1.1

1. Естественные языки;
2. Математический язык;
3. Язык мимики.

Задания I.1.2.

1. Одним сообщением передается одно содержание;
2. Одним сообщением передается несколько содержаний;
3. Несколькими сообщениями передается одно содержание.

Задания I.1.3.

1. Объемный способ и вероятностный способ;
2. 24 бит или 3 байт;
3. $2^8 = 256$.

Задания I.1.4.

1. 4 бит;
2. 6 бит;
3. 8 бит.

Задания I.2.1.

1. Не идентификатор;
2. Идентификатор;
3. Идентификатор.

Задания I.2.2.

1. Действительное число с фиксированной точкой;
2. Комплексное число;
4. Действительное число с плавающей точкой.

Задания I.2.3.

1. 1/6;
2. -16;
3. 15.

Задания I.2.4.

1. $10000001_2=129_{10}$;
2. $129_{10}=201_8$;
3. $1952_{10}=7A0_{116}$.

Задания I.2.5.

1. Цепочка из начальных прописных букв казахского языка;
2. Цепочка из цифр;
3. Цепочка из смешанных знаков.

Задания I.2.6.

1. АЛАТАУ;
2. ЖЕР;
3. X ⊃ Y.

Задания I.2.7.

1. ложно;
2. ложно;
3. истина.

Задания I.2.8.

1. 1;
2. 1;
3. 0.

Задания I.3.1.

1. Строки, векторы, массивы;
2. Связные списки, графы, деревья, стеки, очереди, деки;
3. Линейные, нелинейные связные списки.

Задания I.3.2.

1. $L=\{a,b,c,d,e,f,h,g,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z\}$;
2. $D = \{0,1,2,3,4,5,6,7,8,9\}$;
3. $X = \{x \mid x > 0 \ \& \ x \in N\}$.

Задания I.3.3.

1. Множество решений этого уравнения пусто;
2. Множество *B* есть подмножество множества *A*;
3. A ⊂ B.

Задания I.3.4.

1. Астана, здесь все буквы из казахского алфавита;
2. Массив в информатике соответствует матрице в математике.

Пусть заданы двумерные матрицы $A=\begin{pmatrix}1 & 3\\ 5 & 7\end{pmatrix}$, $B=\begin{pmatrix}2 & 4\\ 6 & 8\end{pmatrix}$.

Определена операция сложение элементов этих матриц:

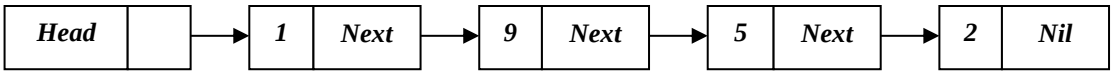
$$A+B=\begin{pmatrix}1 & 3\\ 5 & 7\end{pmatrix}+\begin{pmatrix}2 & 4\\ 6 & 8\end{pmatrix} = \begin{pmatrix}1+2 & 3+4\\ 5+6 & 7+8\end{pmatrix} = \begin{pmatrix}3 & 7\\ 11 & 15\end{pmatrix}$$

3. В таблице будет 6 столбцов, а количество строк будет на единицу больше, чем число студентов в группе, так как одна строка отводится для записи наименования 6 полей:

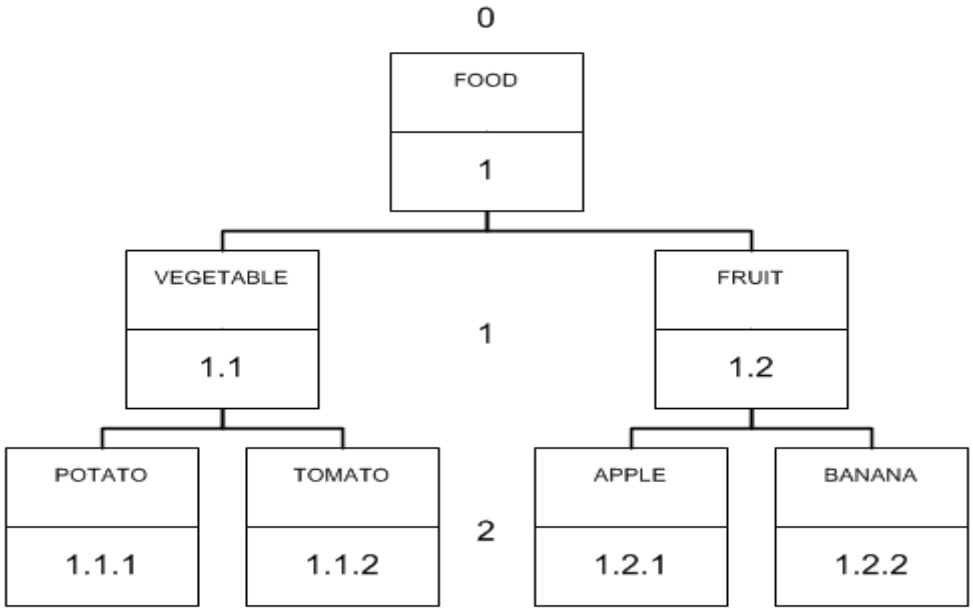
№	Фамилия	Имя	Отчество	Пол	Национальность

Задания I.3.5.

1. Сначала определим структуру с именем Data, содержащая поля типов данных необходимых элементов (в нашем случае поле целого типа), затем определим структуру с именем List, содержащую поле типа Data и поле адреса следующего элемента next. Это позволит изменять структуру Data с собственно данными, никак не затрагивая при этом основную структуру List. Создаваемый список графически изображается ниже:



2. Для представления иерархического списка используется способ, который представляет путь от самого высокого уровня к самым низким уровням:



3. В двухсвязном списке каждый элемент имеет поля с данными и два указателя: один указатель хранит адрес предшествующего элемента списка, второй — адрес последующего элемента. Для работы с двухсвязным списком используются два указателя,

хранящие адреса начала и конца такого списка.

Задания I.3.6.

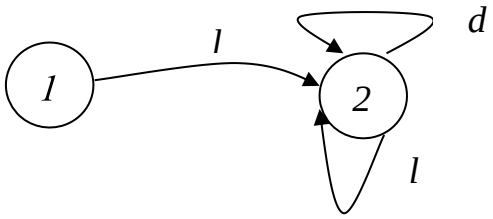
1. Для требуемой хеш-функции областью определения будет множество строк {Астана, Алматы, Қарағанды, Шымкент, Тараз, Қызылорда,...}, занимающих разное количество байтов в памяти, а областью значения – множество чисел {1, 2, 3, 4, 5, 6,...}, каждый из которых будет занимать только 4 байта.

2. Для требуемой хеш-функции областью определения будет множество строк с разными длинами, представляющих идентификаторы (последовательность букв и цифр, начинающаяся из буквы) с разными длинами, а областью значения – множество индексов, каждый из которых имеет фиксированную длину.

3. Для требуемой хеш-функции областью определения будет множество строк с разными длинами, представляющих ФИО студентов, а областью значения – множество двенадцатизначных целых чисел.

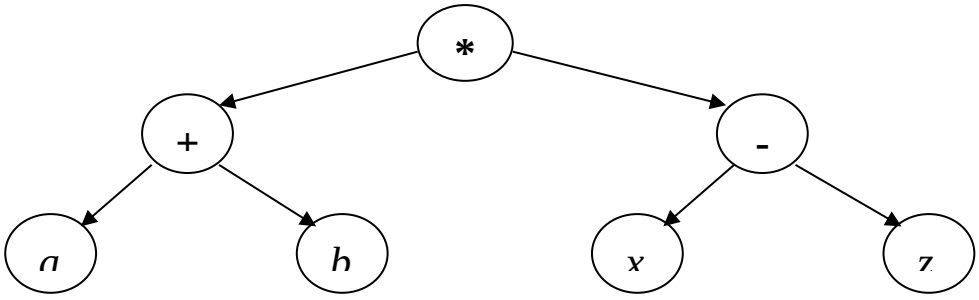
Задания I.3.7.

1. Если букву обозначить через *l*, а цифру – через *d*, то для описания структуры идентификатора можно строить такой граф.



Здесь с вершины 1 к вершине 2 идет путь с весом *l*, это означает, что идентификатор начинается с буквы. Дальше с вершины 2 опять к вершине 2 идут пути с весами *l* и *d* означают, что в остальных местах могут быть только буква или цифра, при этом длина не ограничена.

2. Дерево для выражения (*a* + *b*) * (*x* -*z*) строится так:



Задания I.3.8.

1. Магазин пистолета: последний заряженный патрон выстреливается первым.

2. В очереди за хлебом: кто пришел раньше всех, тот покупает первым.

3. Двухсторонняя сборка и разборка поезда на железной дороге: вагоны могут сцепляться и отцепляться с двух сторон поезда.

Задания II.1.1.

1. 2^{24};

2. 32 бит;

3. Единицы измерения объема памяти будут степенями двойки, каждая следующая единица на 1024 байтов больше, чем предыдущая.

Задания II.1.2.

1. Принцип адресности памяти, принцип единства данных и команд, принцип программного управления, принцип дискретности работы компьютера.

2. Регистр адреса команды, регистр команды, регистр исходных и результирующих данных, регистр признака результата, регистр особого случая.

3. Рабочим циклом компьютера называется работа во время выполнения одной команды в программе.

Задания II.1.3.

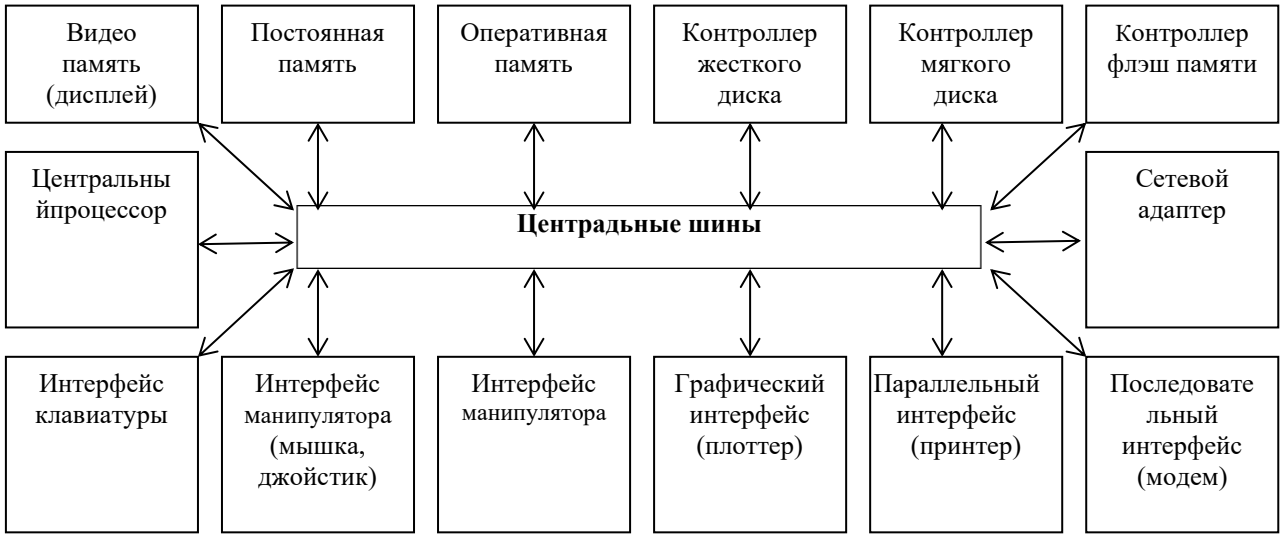
1. Элементной базой компьютеров первого поколения являются электронные лампы.

2. Программным обеспечением компьютеров второго поколения были мониторная система, транслятор, ассемблер, программные пакеты, макросы.

3. В пятом поколении компьютера Prolog должен реализоваться аппаратно и использоваться как машинный язык для создания искусственного интеллекта.

Задания II.1.4.

Структура персонального компьютера:



Задания II.2.1.

1. $\underset{A}{14}2\,43\;\underset{E}{14}2\,43\;\underset{A}{14}2\,43$;

2. $\underset{2}{14}2\,43\;\underset{0}{14}2\,43\;\underset{1}{14}2\,43\;\underset{4}{14}2\,43$;

3. $\underset{D}{14}2\,43\;\underset{E}{14}2\,43\;\underset{F}{14}2\,43$.

Задания II.2.2.

1. $63_{10} = 111111_2$, следовательно

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

2. $-0{,}11101 * 2^5$:

1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0	1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	

Задания III.1.1.

1. Ввод коэфициентов уравнения; вычислить дискриминант; если дискриминант меньше нуля, то уравнение не имеет решение; если дискриминант равен нулю, то уравнение имеет одно решение; если дискриминант больше нуля, то уравнение имеет два решения.

2. Исходные данные для приготовления чая: кипяченная вода в посуде, сухой чай, сахар, молоко, результат чай сладкий с молоком.

3. Результаты экзаменов в каждом семестре.

Задания III.1.2.

1. Определить значения A, B
   Если A>B, то A – больше, иначе B – больше;
2. Задать значения A, B
   Если A<B, то A – меньше, иначе B – меньше;
3. Задать значения A, B; C = (A+B)/2.

Задания III.2.1.

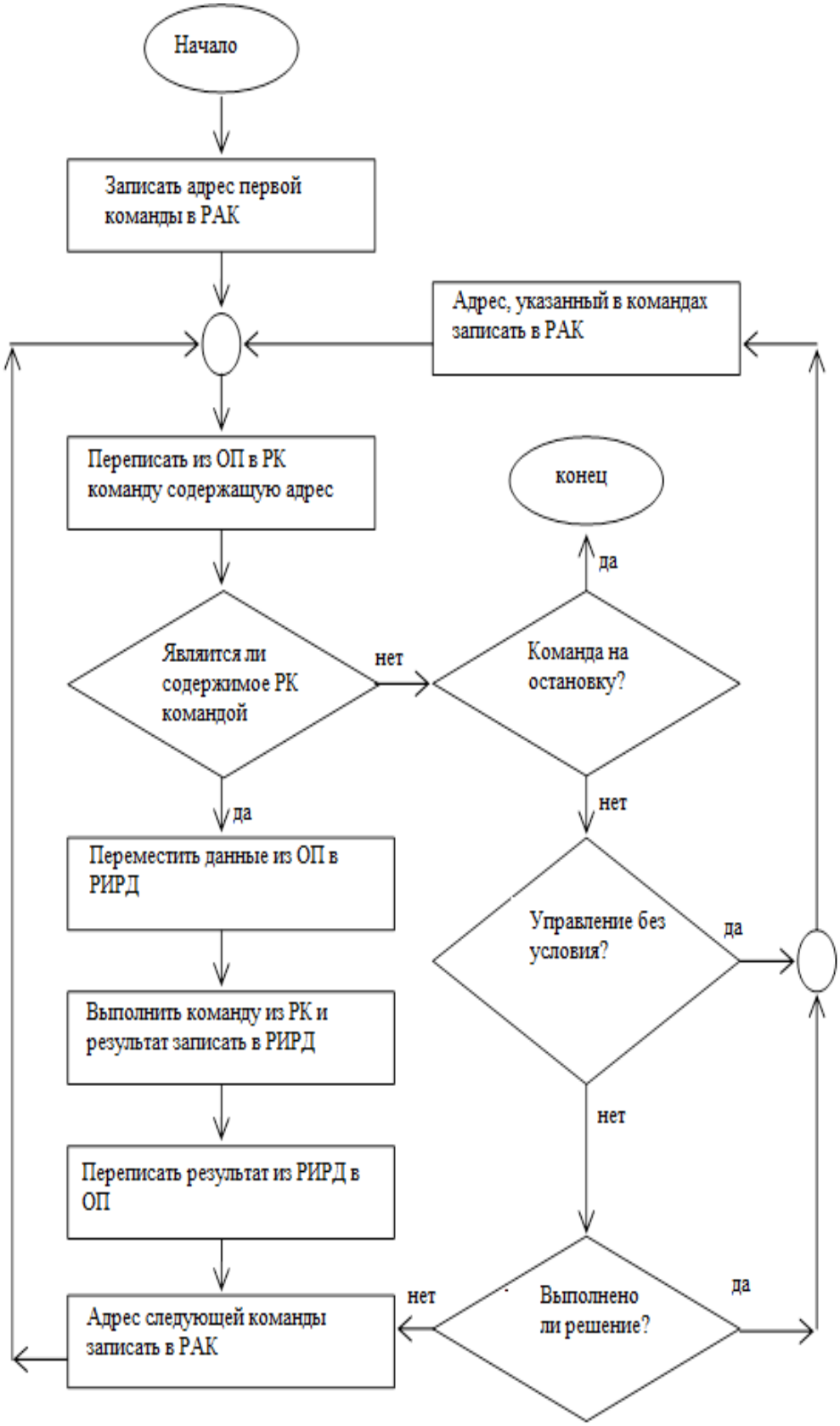
1. НАЧАЛО
   S ← 0
   ввод чисел N, M
   ПОВТОРЕНИЕ
      S ← S+N
      M← M-1
  ДО M=0
  вывод результата S
 КОНЕЦ

2. НАЧАЛО
   S ← 1
   ввод чисел M, N
   ПОВТОРИТЬ
      S ← S*N
      M← M-1
  ДО M=0
  вывод результата S
 КОНЕЦ

3. НАЧАЛО
   S ← 1
   ввод числа N
      S ← (N*N) / 2
  вывод результата S
 КОНЕЦ

Задания III.2.2.

1.



2. НАЧАЛО

S ← 0

N ввод числа

ПОВТОРИТЬ

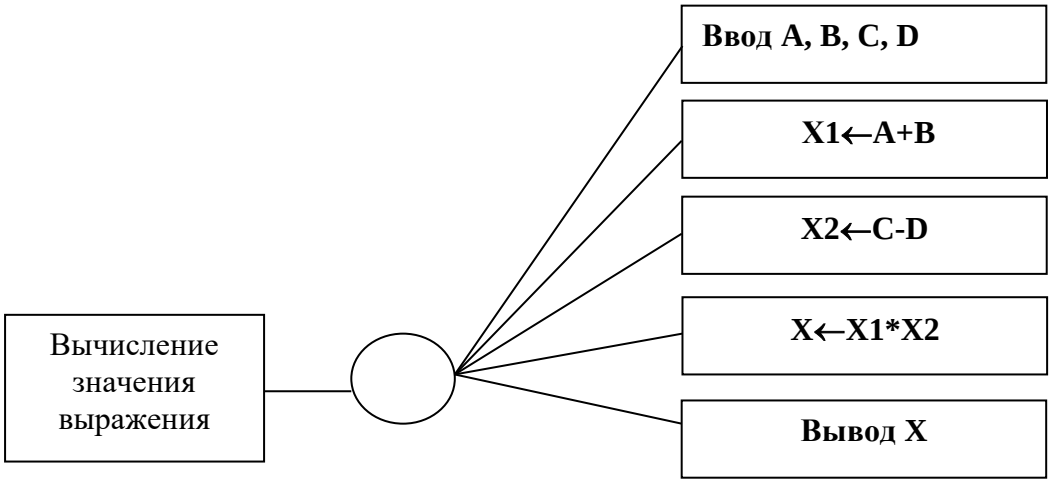
S ← S+N

S >100 ДО

S вывод результата

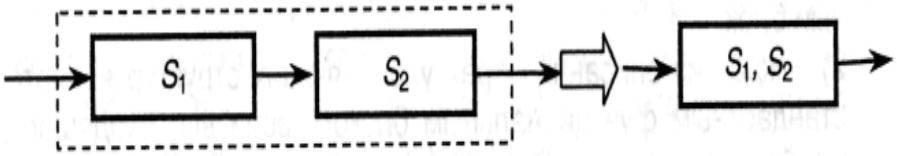
КОНЕЦ

3.



Задания III.3.1.

1.

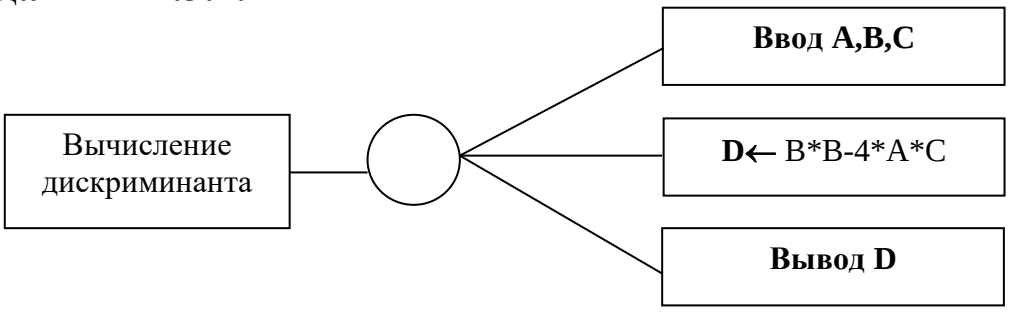


2. Простое ветвление, альтернативное ветвление, многозначное ветвление.

3. Понятность, простота, проверяемость, модифицируемость.

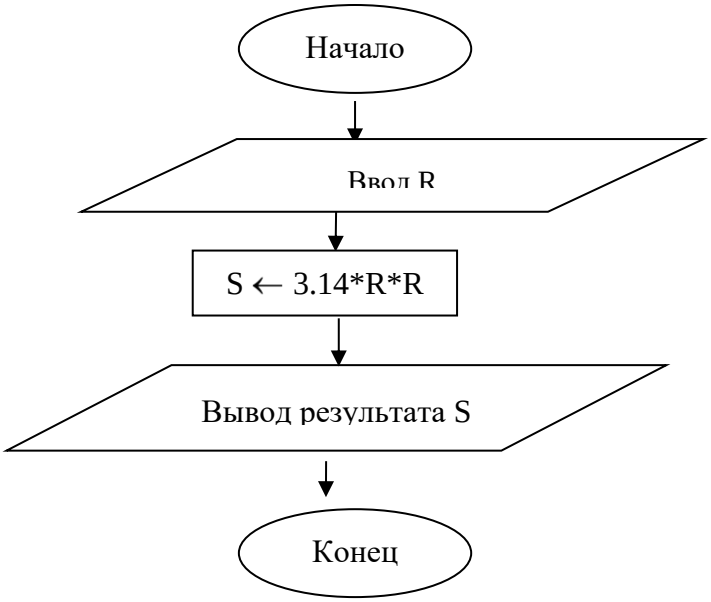
Задания III.3.2.

1.

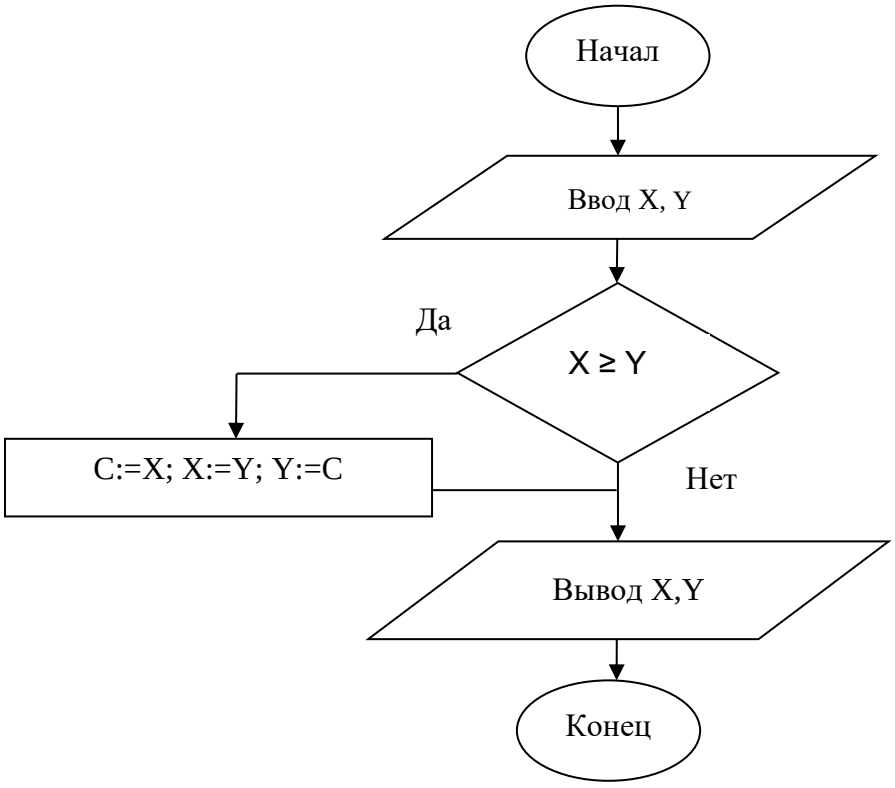


2. **АЛГ** ПЛОЩАДЬ
НАЧАЛО
ввод H, B
S = B* H/ 2
вывод S
КОНЕЦ

3.



Задания III.3.3.

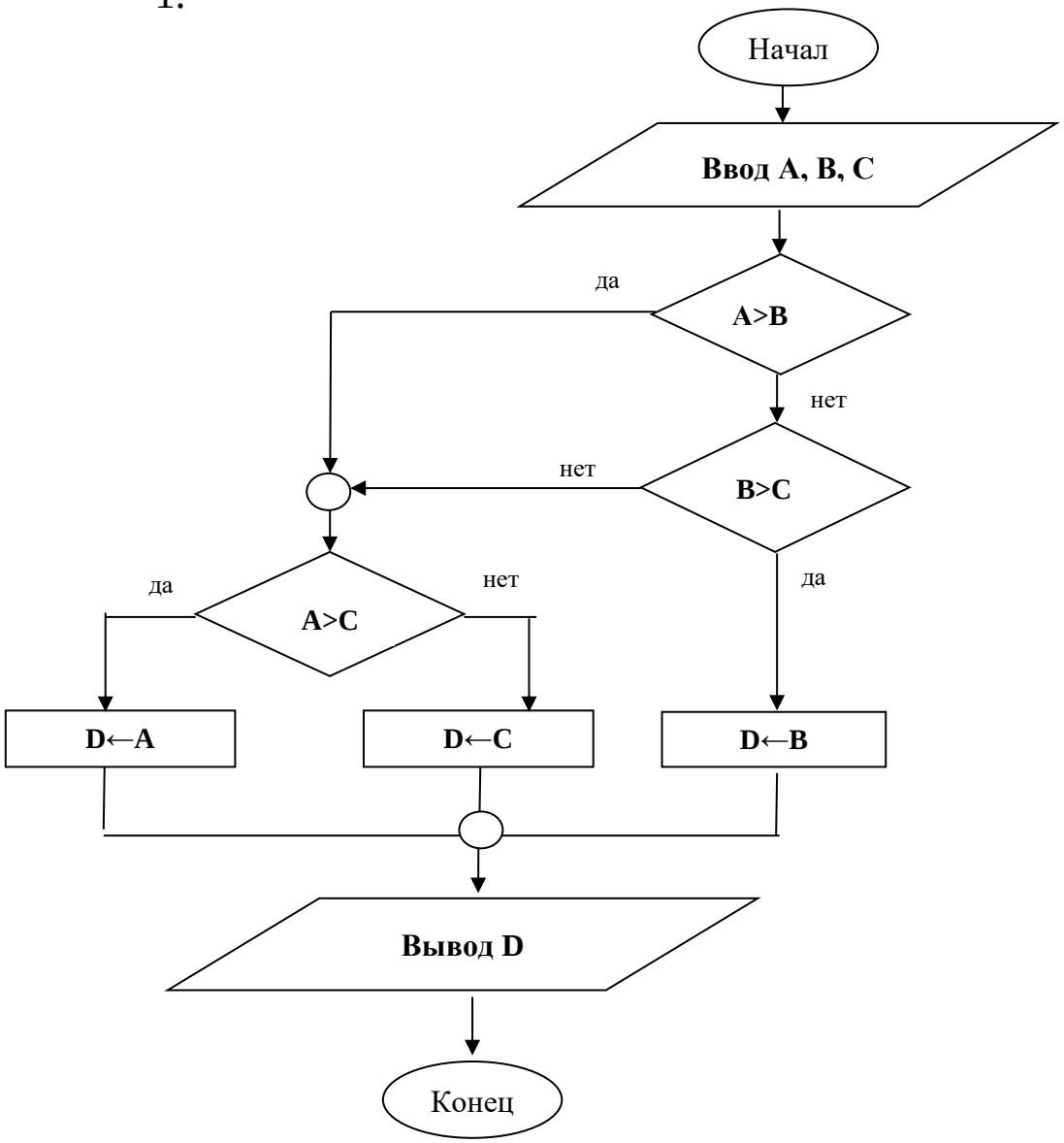


Задания III.3.4.

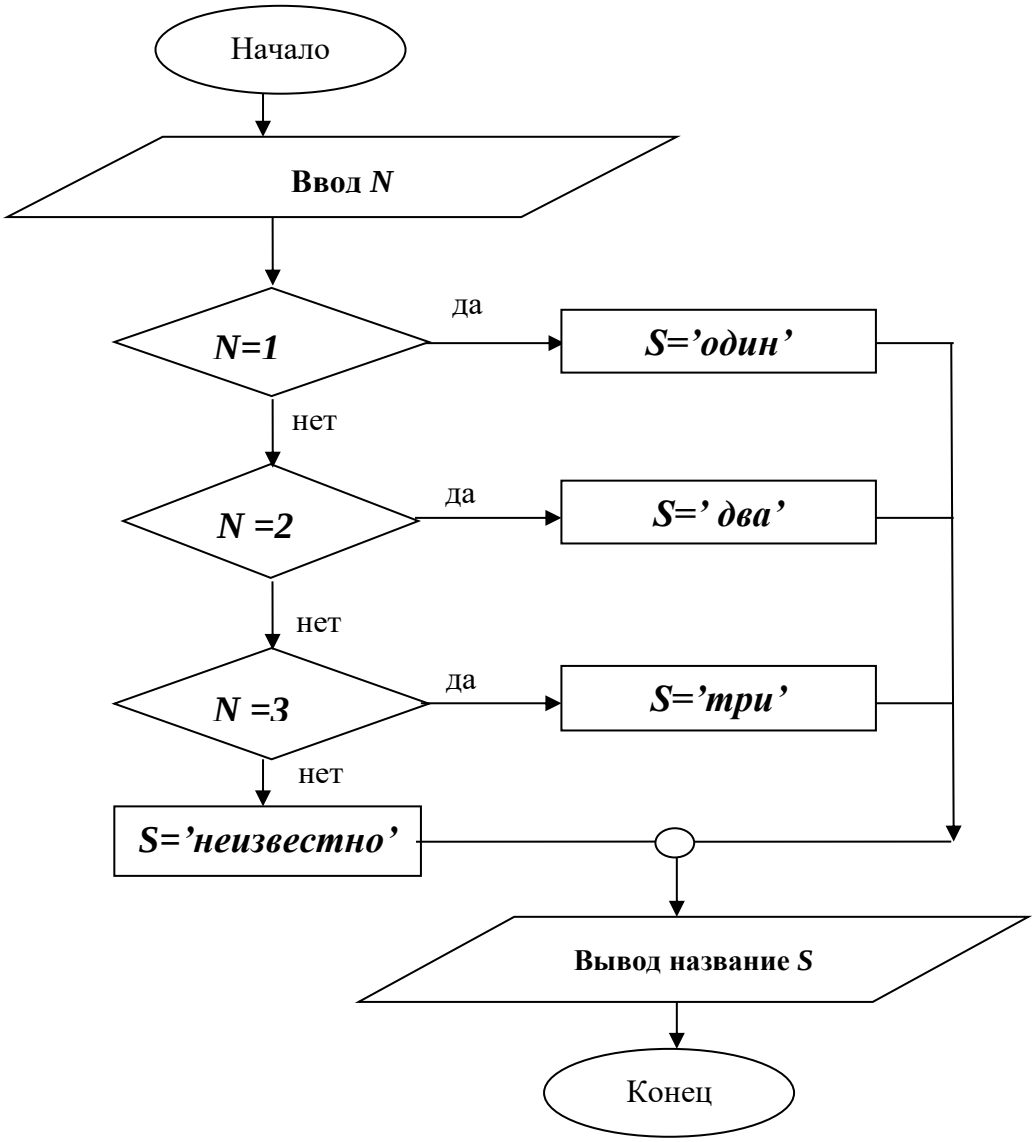
1. АЛГ МИНМАХ
   ВВОД *X*, *Y*
   НАЧАЛО
   ЕСЛИ *X*>*Y* ТО *A*=*X*
   ИНАЧЕ *B*=*Y*
   ВЫВОД *A*, *B*
   КОНЕЦ
2. Альтернативное ветвление
3. R=9.

Задания III.3.5.

1.



2.



3. Многозначное ветвление.

Задания III.3.6.

```
АЛГ ОН
НАЧАЛО
    I:=1
    ПОКА I < 11
     ВЫВОД I
     I:=I+1
КОНЕЦ
```

Задания III.3.7.

24

Задания III.3.8.

АЛГ МАХ

M ← 0

ДЛЯ *V*=1,15 ШАГ 1

ПОВТОРИТЬ

ВВОД *X*

ЕСЛИ *M*<*X* ТО *M:=X*

КОНЕЦ

ВЫВОД *M*

ЗАВЕРШЕНИЕ

Задания III.4.1.

1. $2^8 = 256$;

2. Программа методом компиляции работает быстрее, но занимает много памяти, методом интерпретации занимает мало место памяти, но медленнее работает.

3. Сначала с алгоритмического языка на машинный язык, затем выполняет программу на машинном языке.

Задания III.4.2.

1. Сначала вычисляется значение выражения справа от знака присваивания, затем это значение присваивается переменной слева от знака присваивания.

2. Процедурный, функциональный, логический, продукционный и объектный.

3. АЛГ ДИСКР

ВВОД *A, B, C*

НАЧАЛО

D ← *B*B - 4*A*C*

ВЫВОД *D*

КОНЕЦ

Задания III.4.3.

1. Длина высоты треугольника вычисляется по программе:

program *высота* (*input*, *output*) ;

var *a*, *b*, *c*, *h* : *real;*

begin *read* (*a*, *b*, *c*)*;*

```
        h:=2/a*sqrt(p*(p–a) *(p–b) *(p–c));
        write (h)
    end.
```

2. Биномиальный коэффициент вычисляется по программе.

```
function c (n:integer, k:integer); real;
    begin
        c:= fact(n)/(fact(k)*fact(n-k));
    end.
```

Задания III.4.4.

Значение (CAR X) равно 6.

Значение (CDR X) равно (7 8).

Значение (CONS X Y) равно (((3 4)5)(6)).

Задания III.4.5.

сын(*X, Y*) :– ***отец***(*Y, X*)**.**

сын(*X, Y*) :– ***мать***(*Y, X*)**.**

дочь(*X, Y*) :– ***отец***(*Y, X*)**.**

дочь(*X, Y*) :– ***мать***(*Y, X*)**.**

внук(*X, Z*) :– ***сын***(*X, Y*), ***сын***(*Y, Z*)**.**

внук(*X, Z*) :– ***сын***(*X, Y*), ***дочь***(*Y, Z*)**.**

внучка(*X, Z*) :– ***дочь***(*X, Y*), ***сын***(*Y, Z*)**.**

внучка(*X, Z*) :– ***дочь***(*X, Y*), ***дочь***(*Y, Z*)**.**

Задания III.4.6.

a. /12/

b. /9/

c. /12/

d. /4/

Задания III.4.7.

1. Площадь полной поверхности и объем усеченного конуса вычисляется по программе:

```
package kz.enu.inf;
import static java.lang.Math.*;
public class Calculate {
  private int r1 = 20;
  private int r2 = 15;
 private int l = 13;
```

```
 private int h = 12;
public void Find() {
        double S, V;
        S = PI*(pow(r1,2)+(r1+r2)*l+pow(r2,2));
        V=PI*h*(pow(r1,2)+pow(r2,2)+r1*r2)/3;
        System.out.println("S=" + S+" V=" + V );

 }
 public static void main(String[] args) {
 Calculate obj = new Calculate();
        obj.Find();
 }
}
```

2 Координаты точки, делящий отрезок AB, соединяющий две точки $A(x_1, y_1)$ и $B(x_2, y_2)$ в отношении $m:n$ определяются по программе:

```
package kz.enu.inf;
import static java.lang.Math.*;
public class Calculate {
  private int x1 = 5;
  private int y1 = 15;
 private int x2 = -5;
 private int y2 = 20;
 private int m = 2;
 private int n = 3;
public void Find() {
        double x, y, k;
        k=m/n;
 x=(x1+k*x2)/(1+k);
 y=(y1+k*y2)/(1+k);
 System.out.println("x=" + x+" y=" + y );

 }
 public static void main(String[] args) {
 Calculate obj = new Calculate();
        obj.Find();
 }
}
```

Задания IV.1. 1

1. Для сортировки исходными данными являются элементы однотипного массива, между их значениями должны быть определены отношения сравнения ("=", ">", "<", ">=", "<="); результатами эти же элементы, но их значения должны расположиться в порядке возрастания или убывания.

2. Внутренняя сортировка проводится в оперативной памяти, а внешняя сортировка - на внешней памяти.

3. Сортировка облегчает поиск информации, упорядочивает элементы в структуре данных.

Задания IV.1.2.

1. Фильтрация элемента заканчивается, когда находится элемент $a[i]$, ключ которого меньше, чем ключ x и доступна левая часть последовательности.

2. Сортируемый массив ***a*** подразделяется на исходную последовательность $a[1], a[2], \ldots, a[n]$ и готовую последовательность $a[1], a[2], \ldots, a[i-1]$, сортировка осуществляется совершением шагов.

3. Исходная последовательность подразделяется на две части: одна часть упорядочена, а другая часть не упорядочена.

Задания IV.1.3.

1. Сортировка начинается выбором элемента с самым маленьким ключом и обменивается с первым элементом.

2. На каждом шаге рассматривается только один текущий элемент заданной последовательности.

3. Число сравнений ключей не зависит от начального порядка ключей.

Задания IV.1.4.

1. Основана на принципе сравнения и обмена соседних элементов пока все элементы массива не отсортируются.

2. Эта последовательность 25 37 42 54 75 91 06 отсортируется за один шаг при сортировке методом пузырька.

3. Эта последовательность 19 31 43 52 69 87 05 отсортируется за один шаг при шейкер- сортировке.

Ответы вопросов

Вопросы I.1.1.

1. Информация – представление свойств и отношений материальных и нематериальных объектов и субъектов окружающего мира независимо от сознания человека.

2. Естественные языки, математический язык, музыкальный язык, язык глухонемых.

3. Язык программирования можно отнести к языкам общения.

Вопросы I.1.2.

1. Между передатчиком и приемником должно быть предварительное соглашение о виде (представлении) и содержании сообщения.

2. Сообщение можно передавать с помощью некоторого языка.

3. Отношение между сообщением и содержанием неоднозначно.

Вопросы I.1.3.

1. Бит (bit).

2. Джон фон Нейман и Клод Шеннон.

3. Кратно 2.

Вопросы I.1.4.

1. Неопределенность измеряется энтропией.

2. Имеется однозначное соответствие.

3. Нужно определить энтропию.

Вопросы I.2.1.

1. Числовой, символьный и логический.

2. Конечная последовательность, которая состоит из букв и цифр и начинается с буквы.

3. Переменная величина имеет много значений, постоянная величина – только одно значение.

Вопросы I.2.2.

1. Целые, вещественные и комплексные числа.

2. Вещественные с фиксированной и плавающей точкой.

3. Вещественной и мнимой части.

Вопросы I.2.3.

1. Операции пишутся перед операндами.

2. Операции пишутся между операндами.

3. Операции пишутся после операндов.

Вопросы I.2.4.

1. Системой счисления называется способ записи чисел с помощью цифр и множество правил.

2. Позиционные и непозиционные системы счисления.

3. Число представляется в различной системе счисления.

Вопросы I.2.5.

1. Значением символьной величины является цепочка, образованная из букв, цифр или специальных знаков.

2. Длина символьной величины |ABCD| равна четырем

3. Длина пустой цепочки равна нулю.

Вопросы I.2.6.

1. Операции над символьными величинами подразделяются на две группы: операции конструирования, операции деления.

3. Итерация определяется с помощью конкатенации и дизъюнкции.

1. Пустой цепочкой называют абстрактную цепочку, которая не содержит ни одного символа.

Вопросы I.2.7.

1. К логическим значениям относятся «истина» и «ложь»;

2. «истина» или «да» или « 1 » или «+»;

«ложь» или «нет» или « 0 » или «–».

3. Истина.

Вопросы I.2.8.

1.

1/A	B	A∨B
0	0	0
0	1	1
1	1	1
1	1	1

2. Операцию «Коньюнкция» («и»).

Вопросы I.3.1.

1. Структуры данных по связям между элементами

подразделяются на линейные структуры данных и нелинейные структуры данных.

2. Нелинейные структуры данных по взаимной связи между элементами делятся на иерархические, сетевые, табличные.

3. Структуры данных по способу представления подразделяются на физические и логические структуры данных.

Вопросы I.3.2.

1. Количество элементов множества называют его мощностью.

2. Множество определяется указанием или описанием всех элементов.

3. Длина множества пустых цепочек равна единице.

Вопросы I.3.3.

1. Объединение двух множеств.

2. Пересечение двух множеств.

3. Разность двух множеств.

Вопросы I.3.4.

1. Таблицы это статические несвязные структуры данных, состоящие из элементов индексированного одного множества или прямых произведений нескольких индексированных множеств.

2. Массив состоит из элементов, для которых указываются имена и индексы.

3. В многомерном массиве количество измерений равно числу индексов.

Вопросы I.3.5.

1. Линейный односвязный список это динамическая линейная структура данных, состоящая из элементов одного типа, связанных между собой последовательно посредством указателей.

2. Кольцевой список это линейный односвязный или двухсвязный список, в котором указатель последнего элемента должен указывать на первый элемент.

3. Иерархический список связная структура данных, каждый элемент которой может быть также началом списка следующего подуровня иерархии и многосвязным.

Вопросы I.3.6.

1. При достаточно маленьком размере хеш-таблицы по отношению количеству ключей или при плохой хеш-функции.
2. Если удаляется некий элемент из хеш-таблицы и его ячейка в таблице в процессе вставки окажется заполненной.
3. Для определения шага смещения адреса используется другая хеш-функция, вместо линейного смещения на одну позицию.

Вопросы I.3.7.

1. Путь в графе - это последовательность ребер, ведущая от одной вершины к другой, чтобы каждые два соседних ребра имеют общую вершину, и никакое ребро не встречается более одного раза.
2. Дерево это граф, в котором все вершины связаны, а пути незамкнуты, т.е. связный граф без циклов и без петель.
3. Двоичное дерево это дерево, в котором каждая вершина имеет не более двух потомков.

Вопросы I.3.8.

1. В стеке определены операции построение стека, добавление элемента, удаление элемента, просмотр элемента в вершине стека без удаления, проверка состояния стека и очистка стека.
2. В очереди определены операции создание очереди, добавление элемента в конец очереди, удаление элемента из головы очереди и очистка очереди.
3. В деке определены операции добавление элемента с конца дека, добавление элемента с начала дека, удаление элемента с конца дека, удаление элемента с начала дека, определение размера дека и очистка дека.

Вопросы II.1.1.

1. Компьютеры в зависимости от вида сообщения информации подразделяются на аналоговые или дискретные компьютеры.
2. В классической архитектуре компьютера двойными стрелками обозначен передачи данных и команд..
3. В компьютере адресное слово сохраняет адреса, а машинное слово - данные.

Вопросы II.1.2.

1. Разрядность процессора это объемное количество информации для выполнения одной операции, она равна емкости соответствующего регистра и измеряется в битах.

2. Частотой процессора называется количество выполняемых тактов (работа при выполнении одной команды) за одну секунду.

3. Скоростью процессора называется количество выполняемых операций за одну секунду, она измеряется в операция/секунд (о/с).

Вопросы II.1.3.

1. Элементной базой компьютеров являются электронные элементы (лампы, транзистры, интегральные схемы и др.), используемые для создания их основных устройств (процессоров и др.).

2. В настоящее время известны шесть поколений компьютеров.

3. Қомпьютеры шестого поколения будут создаваться на основе нейронных сетей, нечеткой логики и теории квантового вычисленияи и генетических алгоритмов.

Вопросы II.1.4.

1. Монитор, системный блок, клавиатура.

2. Микропроцессор.

3. Первый универсальный микропроцессор Intel–8080 был создан в 1974 году.

Вопросы II.2.1.

1. Логические данные представляются в памяти компьютера одним битом: 0 – ложь, 1 – истина.

2. Символьные данные представляются в памяти компьютера последовательностью длиной 8-бит или 16-бит, состоящей из 0 и 1.

3. ANSI 8-битовый стандарт, а Unicode 16-битовый стандарт.

Вопросы II.2.2.

1. Целое число представляется в двоичной системе счисления в одном машинном слове, при этом знак числа размещается в первом бите: 0 – знак «+», а 1 – знак «-».

2. Вещественное число с плавающей точкой представляется в двоичной системе счисления в одном машинном слове: в самом

левом бите записывается знак мантиссы, после представляется сама мантисса (количество байтов зависит от разрядности машинного слова), затем в одном бите пишется знак порядка, а в конце в одном байте размещается сам порядок.

3. Вещественное число с фиксированной точкой перед представлением в компьютере преобразуется в вещественное число с плавающей точкой в нормальной форме, затем представляется как в вещественное число с плавающей точкой.

Вопросы III.1.1.

1. Преобразовывать данные для достижения заранее определенные цели.

2. На вход обработки даются исходные данные

3. Результат это данные, полученные после завершения выполнения обработки.

Вопросы III.1.2.

1. Алгоритм это конечная последовательность понятных и точных указаний о том, что какие операции и в каком порядке нужно выполнять для решения любой задачи заданного класса за конечное число шагов.

2. Понятность, точность, дискретность, конечность, массовость.

3. Протокол выполнения алгоритмв это список исходных данных и результатов для каждого шага.

Вопросы III.2.1.

1. Вербальный и графический алгоритмические языки.

2. Алфавит вербального алгоритмического языка состоит из латинских и русских букв, арабских и римских цифров, скобок, знаков операций, зарезирвированных слов.

3. НАЧАЛО, КОНЕЦ, ВВОД, ВЫВОД, ЕСЛИ, ТО, ИНАЧЕ, ВЫБОР, ТОГДА, ВЫПОЛНИТЬ, ПОВТОРИТЬ, ПОКА, ДО, ДЛЯ, ШАГ, АЛГОРИТМ, ЗАВЕРШИЛСЯ.

Вопросы III.2.2.

1. Имеется два вида графических алгоритмических языков: язык блок-схем и древовидный язык.

2. Алфавит языка блок-схем состоит из овала, параллелограмма, прямоугольника, ромба, окружности и стрелки.

3. Обход дерева показывает порядок разложения действий на простые и порядок их выполнения в алгоритме.

Вопросы III.3.1.

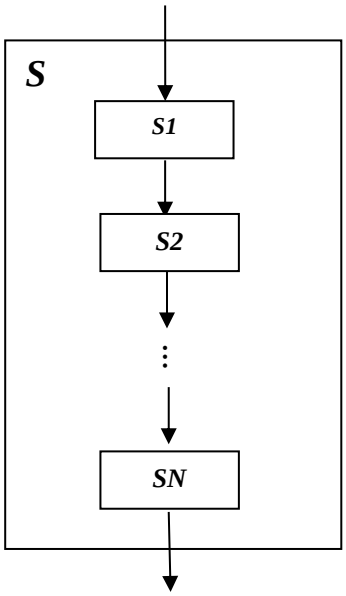
1. Последовательность объединяет несколько функциональных блоков в один блок.

2. Ветвление организует выполнение одного из двух функциональных блоков S_1 и S_2 в зависимости от значения логического условия.

3. Повторение организует многократное повторение выполнения функционального блока.

Вопросы III.3.2.

1.



2.

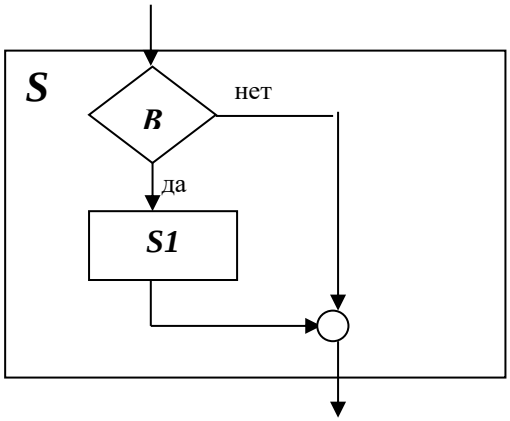
: **S** тело сложного действия
НАЧАЛО
< ***S1*** >
< ***S2*** >
...
< ***SN***>
КОНЕЦ

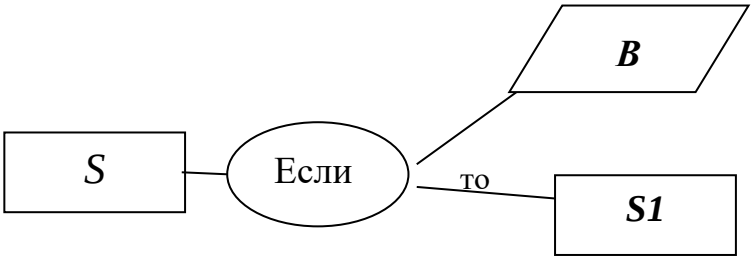
3. Для сложного действия ***S***, образованного из последовательности действий S1, S2,..., SN исходными данными являются исходные данные ***S1***, результатом – результат ***SN***.

Вопросы III.3.3.

1. : ***S*** тело сложного действия

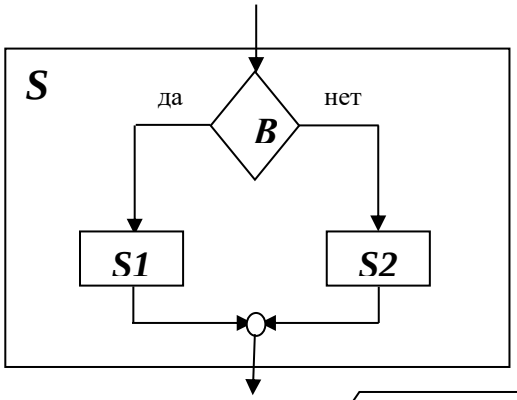
ЕСЛИ <***B*** > ТО < ***S1*** >

2. 

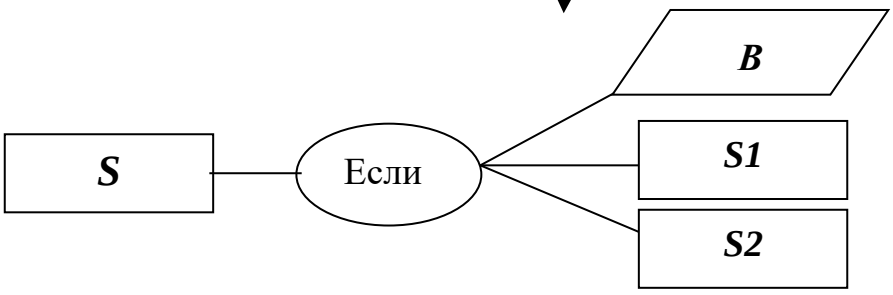
3. 

Вопросы III.3.4.

1. Язык блок-схемы:



2. Древовидный язык:



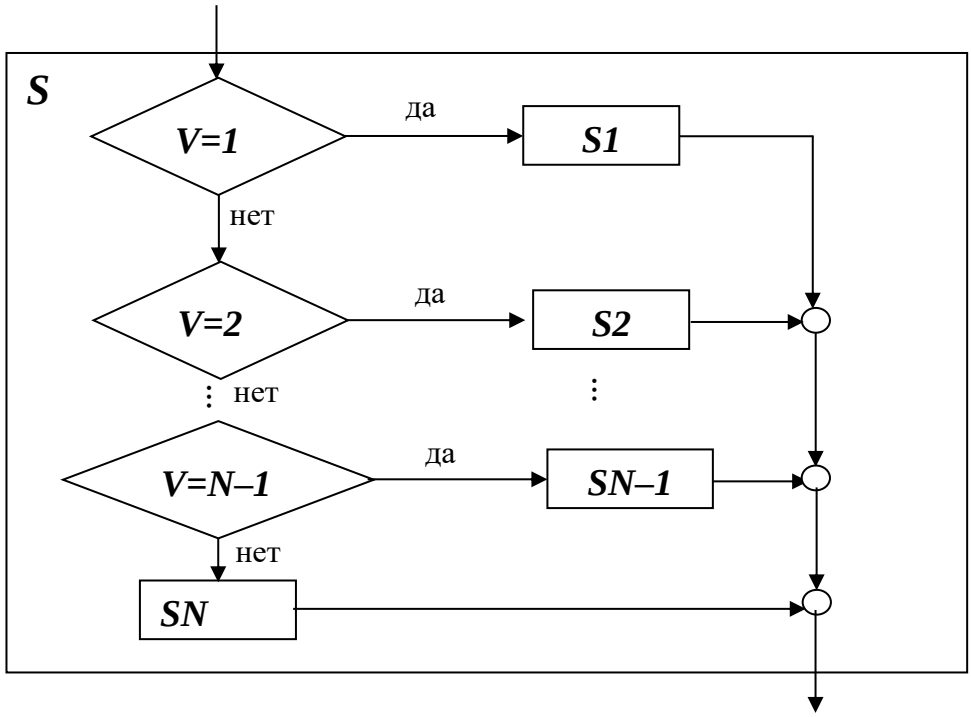
3. Вербальный язык:

ЕСЛИ <**B** > ТО < **S1** >

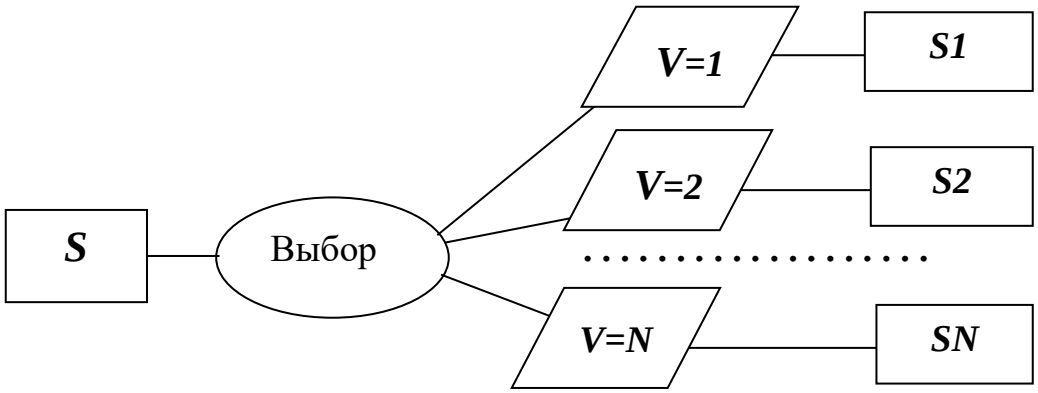
ИНАЧЕ < **S2** >

Вопросы III.3.5.

1. Язык блок-схемы:



2. Древовидный язык:



3. Вербальный язык:

ВЫБОР
V*= *1 ТОГДА ***S1***

V* = *2 ТОГДА ***S2***

. . . . . .

V* = *N ТОГДА ***SN***

Вопросы III.3.6.

1. Сложное действие ***S*** повторения с предусловием показывает, что в зависимости от условия ***B*** повторяется выполнение действия ***S1*** или ничего не выполняется.

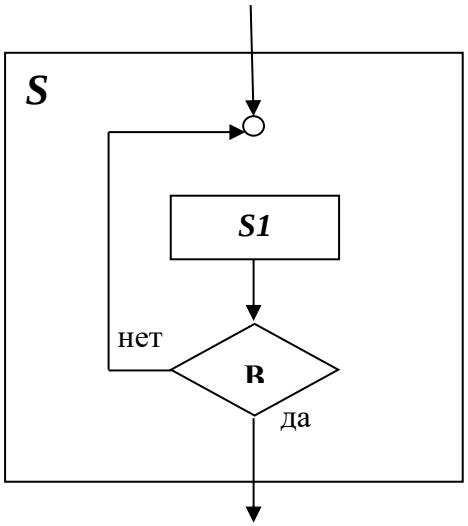
2. В повторении с предусловием имеются два случая: проверяемое условие может выполниться или не выполниться.

3. Если в повторении с предусловием проверяемое условие выполняется не зависимо от тела повторения, то тело повторения может повторяться бесконечно.

Вопросы III.3.7.

1. Сложное действие **S** повторения с постусловием показывает, что в зависимости от условия ***B*** повторяется выполнение или прекращается выполнение действия ***S1***.

2. Язык блок-схемы:



3. При выполнении повторения с постусловием, если условие не поменяет свое значение, то возможны два случая: если оно заранее истино, то тело повторения выполняется только один раз; если оно заранее ложно, то тело выполняется бесконечно.

Вопросы III.3.8.

1. Параметрическое повторение не выполняется, если начальное значение параметра не меньше конечного значения.

2. В параметрическом повторении число выполнений тела повторения определяется начальным и конечным значениями праметра и значением шага: при каждом выполнении тела

повторения начальному значению параметра добавляется значение шага до достижения конечного значения параметра.

3. Параметрическое повторение применяется в случае, когда число повторения заранее известно.

Вопросы III.4.1.

1. Транслятор нужен для автоматического перевода с искусственного алгоритмического языка на машинный язык.
2. Самый низкий уровень у машинного языка.
3. На логическом исчислении основан язык Prolog.

Вопросы III.4.2.

1. В существующих трансляторах реализованы интерпретация и компиляция.
2. Для облегчения программирования появились языки высокого уровня.
3. Абстракция, полиморфизм, инкапсуляция, наследование.

Вопросы III.4.3.

1. Для объявления переменных применяют специальное слово *var*, а для представления их типов – специальные слова *integer*, *real*, *char*.
2. Описание процедуры начинается со слово **procedure**, затем пишется имя процедуры и в круглой скобке её параметры.
3. Описание функции начинается со слово **function**, затем пишется имя функции и в круглой скобке её параметры, а в конце указывается тип значения функции.

Вопросы III.4.4.

1. Функция
2. В языке Lisp определены следующие стандартные функции: CAR, CDR, CONS, DEFINE, LAMBDA, COND, EQUAL, SETQ.
3. В языке Lisp используются списки.

Вопросы III.4.5.

1. В логическом языке программной единицей является логическое утверждение.
2. Программа дифференцирования алгебраического выражения работает правильно?

Вопросы III.4.6.

1. Программной единицей в продукционных языках является подстановка?

2. В языке Refal имеются следующие стандартные функции: ADD, SUB, MUL, DIV, FIRST, LAST, TYPE.

3. Для построения программы синтаксического анализа на языке Refal нужно применить функции FIRST, LAST, TYPE.

Вопросы III.4.7.

1. Проект Java был представлен корпорацией Sun Microsystems в 1995 году.

2. Приложение Java запускается с помощью виртуальной Java-машины.

3. Идентификаторы - это имена переменных, подпрограмм-функций и других элементов языка программирования.

Вопросы IV.1. 1

1. Сортировка это перестановка мест элементов некоторой структуры данных для их размещения в определенном порядке.

2. Значением функции упорядочения является ключ элемента.

3. Методы сортировки подразделяются на две большие группы: внутрение сортировки и внешние сортировки.

Вопросы IV.1.2.

1. Для проведения сортировки включением нужны входная последовательность *a[1], a[2], ..., a[n]* и готовая последовательность *a[1], a[2], ..., a[i–1].*

2. Число сравнений ключей C_i при 1-м просеивании составляет самое большее *i*-1, самое меньшее 1 и, если предположить, что все перестановки *n* ключей равновероятны, в среднем равно *i/2*.

3. Число сравнений не зависит от начальной упорядоченности элементов.

Вопросы IV.1.3.

1. Сортировка выбором основана на правиле выбора элемента с наименьшим ключом и меняется местами с первым элементом *a*[*1*].

2. Для нахождения элемента с наименьшим ключом с помощью сортировки выбором рассматриваются все элементы заданной

последовательности, и этот элемент отправляется в готовую последовательность.

3. При сортировке выбором число сравнений ключей не зависит от первоначального порядка.

Вопросы IV.1.4.

1. Сортировка обменом основана на принципе сравнения и перестановке двух соседних элементов.

2. В методе пузырька имеется следующая ассиметрия: в отсортированном массиве один «легкий» элемент (пузырек), расположенный в конце «тяжелой» стороны, найдет свое место только за один просмотр, а один «тяжелый» элемент, расположенный в начале «легкой» стороны найдет свое место только за один шаг в каждом просмотре.

3. Шейкер-сортировка в каждом следующем просмотре, чтобы найти пузырек, меняет направление (вверх, вниз).

Ответы тестов

Номера уроков	Тест 1	Тест 2	Тест 3
I.1.1	B	D	A
I.1.2	C	A	B
I.1.3	C	B	C
I.1.4	E	B	A
I.2.1	D	D	A
I.2.2	C	B	E
I.2.3	A	B	B
I.2.4	E	A	B
I.2.5	A	D	D
I.2.6	B	B	E
I.2.7	A	A	B
I.2.8	B	A	C
I.3.1	A	C	E
I.3.2	B	C	C
I.3.3	A	A	A
I.3.4	A	B	A
I.3.5	A	A	B
I.3.6	A	B	A
I.3.7	A	A	A
I.3.8	A	A	A
II.1.1	B	A	B
II.1.2	A	A	A
II.1.3	A	A	A
II.1.4	A	B	A
II.2.1	C	A	A
II.2.2	A	A	A
III.1.1	A	D	A
III.1.2	A	A	E
III.2.1	A	E	A

III.2.2	A	A	A
III.3.1	A	B	B
III.3.2	D	C	A
III.3.3	B	C	B
III.3.4	A	D	A
III.3.5	C	A	C
III.3.6	E	C	C
III.3.7	A	A	C
III.3.8	A	B	B
III.4.1	A	D	A
III.4.2	A	B	C
III.4.3	A	B	C
III.4.4	E	A	D
III.4.5	B	A	A
III.4.6	A	C	E
III.4.7	A	E	C
IV.1.1	A	A	E
IV.1.2	A	D	C
IV.1.3	A	C	E
IV.1.4	A	D	C

ШАРИПБАЙ А.А.

ИНФОРМАТИКА

(Учебник)

Бумага офсетная Формат 60х100 1/16
Плотность 80 гр/м^2. Белизна 90%. Печать РИЗО.
Усл.печ.стр. 15.25. Объем 318.

За содержание учебника отдел издательства не отвечает

в
РК,
Тел.:
233 80
e-mail: evero08mail.ru

Подготовлено к изданию и отпечатано
издательстве “Эверо”
Алматы, ул. Байтурсынова, 22
+7 /727/ 233 83 89, 233 83 43,
45, 233 80 42

ШАРИПБАЙ Алтынбек Амирулы, доктор технических наук, профессор, академик Международной академии информатизации, академик Академии педагогических наук РК, лауреат государственной премии РК, директор НИИ «Искусственный интеллект» ЕНУ им.Л.Н.Гумилева (www.e-ai.kz). Его научными интересами являются теоретические и прикладные проблемы информатики и информационных технологий: теория и технология программирования, автоматизированные системы, искусственный интеллект, компьютерная лингвистика и электронное обучение. В этой области им были разработаны методы формализации семантики продукционных и логических языков программирования, методы автоматизированной верификации программного и аппаратных средств. По результатам этих исследований он защитил докторскую диссертацию на тему «Верификация программных и аппаратных средств вычислительных машин и систем» по специальности «05.13.15 (05.13.11) - Вычислительные машины, системы и сети (Математическое, программное и техническое обеспечение)». Отдельные его научные результаты были внедрены в крупные научные центры: 1976-1979 годы - «Тренажер с языка имитации космических систем» в Летно-испытательском институте, г. Жуковск; 1988-1989 годы «Система верификации цифровых схем» в Научно-производственном центре, г. Зеленоград; 1990-1991 годы «Система параллельного программирования для многопроцессорного вычислительного комплекса» в Научно-исследовательском центре электронной вычислительной техники, г. Москва.

Он опубликовал более 300 научных трудов, издал 5 учебников и более 10 учебных пособий, 2 монографии и 5 терминологических и толковых словарей по информатике и вычислительной технике, получил более 30 свидетельств о государственной регистрации интеллектуальной собственности, участвовал в разработке многих государственных стандартов: 10 - в области информационных технологий, 9 - в области образования.

Под научным руководством А.Шарипбай подготовлены 5 докторов и 8 кандидатов наук, 4 доктора PhD по группе специальностей «Информатика, вычислительная техника и управление». Его научной школой создана математическая теория казахского языка, разработаны методы автоматизированного анализа и синтеза устных и письменных слов и предложений казахского языка, предложена технология создания электронных учебников изданий и др. Эти научные результаты в 1994-2014 годы были применены для создания многих электронных учебных изданий, системы дистанционного обучения казахскому языку, системы распознавания и синтеза казахской речи и других автоматизированных систем, в том числе учебных и экспертных систем, внедренных в различных государственных и негосударственных структурах и организациях РК.