

Министерство сельского хозяйства РФ
ФГБОУ ВПО
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ
УНИВЕРСИТЕТ»

Факультет прикладной информатики
Кафедра компьютерных технологий и систем

К. С. Галиев, Е. К. Печурина

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ

*Учебно-методическое пособие
для студентов-бакалавров,
изучающих «Информатику»*

Под редакцией профессора В.И. Лойко

Краснодар
2013

УДК 004(078)

ББК 32.81

Г15

Рецензенты:

Г. А. Аршинов – доктор технических наук, профессор
(Кубанский государственный аграрный университет);

Е. В. Луценко – доктор экономических наук, профессор
(Кубанский государственный аграрный университет).

Галиев К. С.

Г15 Основы алгоритмизации и программирования: учеб.-метод.
пособие / К. С. Галиев, Е. К. Печурина. – Краснодар: КубГАУ,
2013. – 94 с.

Учебно-методическое пособие посвящено одному из главных разделов дисциплины «Информатика» - основам разработки алгоритма решения задач и технологии написания текста исходного кода программы. Данное пособие предназначено прежде всего тем студентам-бакалаврам, у которых в названии направления подготовки отсутствуют слова «компьютерных технологий и систем». К ним относятся студенты факультетов: энергетики и электрификации; архитектурно-инженерного и др.

УДК 004(078)

ББК 32.81

© К. С. Галиев, Е. К. Печурина, 2013

© ФГБОУ ВПО «Кубанский государственный
аграрный университет», 2013

Оглавление

ВВЕДЕНИЕ	5
1 ОПИСАНИЕ АЛГОРИТМОВ.....	8
1.1 Понятие алгоритма	8
1.2 Формы описания алгоритмов.....	8
1.3 Свойства алгоритмов	10
1.4 Элементы блок-схемы	10
1.5 Структуры алгоритмов	13
1.6 Примеры циклических структур.....	20
1.7 Вложенные циклы.....	23
1.8 Об эффективности алгоритма	25
1.9 Алгоритм укрупненный и подробный.....	33
1.10 Контрольные вопросы:	34
2 ПРИМЕРЫ СОСТАВЛЕНИЯ БЛОК-СХЕМ	36
2.1 Решение квадратного уравнения	36
2.2 Вычисления в одномерной последовательности	37
2.5 Вычисление значений функции	39
2.6 Поиск <i>max</i> и <i>min</i>	42
2.7 Сортировка чисел.....	43
2.8 Задания для упражнения	51
3 ОСНОВЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА ПАСКАЛЬ	53
3.1 Основные понятия	53
3.2 Операторы языка Паскаль	56
3.3 Функции и процедуры	64
3.4 Замечание о структуре текста на языке Паскаль	66
4 ПРИМЕРЫ ПРОГРАММ НА ЯЗЫКЕ ПАСКАЛЬ	67
4.1 Задача о временах года	67
4.2 Квадратное уравнение	68

4.3 Суммирование чисел	69
4.4 Задача о пробежке.....	72
4.5 Вычисление значений функции	72
4.6 Поиск максимального числа	74
4.7 Поиск <i>max</i> и <i>min</i>	75
4.8 Сортировка чисел.....	76
4.9 Двумерный массив (матрица)	78
4.10 Задания для программирования.....	80
5 ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ	82
5.1 Понятие среды программирования.....	82
5.2 Описание IDE Borland Pascal	82
5.3 Порядок работы в Borland Pascal	85
ЗАКЛЮЧЕНИЕ.....	89
Литература:	90
Приложение 1	91
Приложение 2.....	93

ВВЕДЕНИЕ

В настоящее время компьютеры используются практически во всех сферах деятельности человека. Компьютер является аппаратно-программным комплексом. При этом программное обеспечение является неотъемлемой и важнейшей частью компьютера. При всём многообразии программ и программных комплексов у них есть одна общая черта – технологии разработки.

В данном учебном пособии рассматриваются основные (элементарные) вопросы разработки алгоритмов и программ с использованием алгоритмического языка. Одной из наиболее важных целей данного пособия является привитие студентам навыков алгоритмического мышления. Зачем?

Во-первых, в соответствии с Федеральным государственным образовательным стандартом высшего профессионального образования (ФГОС ВПО) подготовки бакалавра, практически по всем направлениям, выпускник должен обладать рядом общекультурных компетенций, в том числе, владением культурой мышления, способностью к постановке цели и выбору путей ее достижения (ОК-1). Для обладания профессиональными компетенциями (ПК) в части информационных технологий изучается дисциплина «Информатика» [11]. К основным разделам содержания дисциплины относятся, также в числе прочих: алгоритмизация и программирование; языки программирования высокого уровня; программное обеспечение и технологии программирования (см. Приложения 1,2).

Во-вторых, алгоритмы ведут нас по жизни постоянно и как бы незаметно. По сути дела, любая инструкция (к сотовому телефону, стиральной машине, автомобильному навигатору и т.п.) - это алгоритм. И умение читать такие алгоритмы - необходимое условие для приспособления человека, тем более с высшим образованием, к условиям жизни в современном мире.

В-третьих, по определению, алгоритм - это последовательность команд, предназначенная исполнителю, для решения поставленной задачи. Исполнителем алгоритма может быть не только человек, но и компьютер, автоматическое устройство и т.п. Для такого исполнителя алгоритм должен быть составлен весьма подробно и очень четко, чтобы можно было выполнить все команды "не раздумывая". Студент, изучающий информатику, обязан иметь представление о формах описания алгоритма и технологии разработки программы для компьютера.

В первой части учебного пособия приводятся понятие алгоритма, способы описания алгоритма, примеры составления блок-схем.

Во второй части описываются основные операторы алгоритмического языка Паскаль, приводятся примеры составления исходного кода программ, описывается работа в среде программирования Borland Pascal для разработки компьютерной программы.

Учебное пособие предназначено для студентов-бакалавров, изучающих дисциплины «Информатика», «Информационные технологии».

ЧАСТЬ 1.

АЛГОРИТМИЗАЦИЯ

1 ОПИСАНИЕ АЛГОРИТМОВ

1.1 Понятие алгоритма

Понятие алгоритма – одно из основных в информатике и программировании. Напомним, что информатика занимается вопросами автоматизированной обработки информации, а автоматизация осуществляется вычислительной машиной, работающей по заранее составленному алгоритму. Алгоритм – это инструкция для достижения цели. Программирование – это написание инструкции в виде команд для вычислительной машины.

Алгоритмом называется точная и понятная исполнителю инструкция для решения задачи при известных исходных данных.

Это определение алгоритма следует запомнить и обратить внимание на каждое слово, а именно, 1) алгоритм – это инструкция, 2) инструкция для решения задачи, 3) инструкция для исполнителя, 4) инструкция точная и понятная, 5) исходные данные известны.

Исполнителем алгоритма может быть человек или вычислительная машина. Для вычислительной машины алгоритм составляется человеком. Поэтому, человек должен сначала сам разобраться с ходом решения задачи, чтобы затем поручить решение задачи машине, написав для нее инструкцию в виде машинных команд. Вычислительная машина или компьютер (computer – вычислитель) - это одно и то же, т.е. синонимы.

1.2 Формы описания алгоритмов

Каждому типу исполнителя (человеку или компьютеру) соответствует своя форма описания алгоритма. Различают 4 формы описания алгоритма:

- 1) в виде обычного текста, желательно написанного по пунктам;
- 2) в виде чертежной схемы, называемой блок-схемой;
- 3) на специальном языке, называемым алгоритмическим языком;
- 4) в виде последовательности машинных команд, называемых компьютерной программой.

Первые две формы описания алгоритма предназначены для исполнителя-человека. Четвёртая – компьютерная программа – предназначена для исполнителя-компьютера. Третья форма описания - на алгоритмическом языке - является промежуточной, она разрабатывается человеком и она способна быть воспринятой, «прочитанной» компьютером. Компьютер может перевести текст с алгоритмического языка в свои машинные команды и стать исполнителем алгоритма.

Вначале ход решения задачи на уровне идеи описывается обычным текстом. Затем в более продуманной форме описывается блок-схемой. Блок-схема является лаконичным и строгим описанием алгоритма, не допускающим разночтения; блок-схема понимается только однозначно. Содержание блок-схемы переписывается алгоритмическим языком в специальный текст, называемый листингом программы или исходным кодом программы. Листинг программы переводится (транслируется или компилируется) в машинные команды и получается компьютерная программа.

Первые две формы описания алгоритма называются *алгоритмизацией*, последние две формы описания алгоритма называются *программированием*. Поэтапная разработка алгоритма и его реализация в виде компьютерной программы, называется алгоритмизацией и программированием.

Напомним, всё это нужно для того, чтобы заставить машину (компьютер) решать наши задачи вместо нас. Замена человека машиной для выполнения определенного вида работ называется автоматизацией. В нашем случае это вычислительные работы, это стремление автоматизировать обработку информации.

1.3 Свойства алгоритмов

При разработке алгоритма следует исходить из того, чтобы он обладал следующими свойствами:

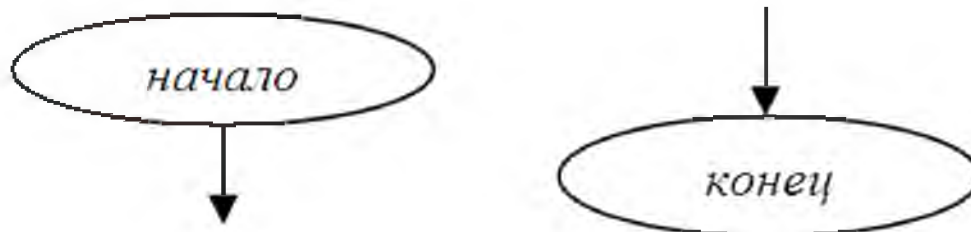
- *Понятность* — алгоритм должен включать только те понятия (слова, конструкции, схемы, операторы, команды), которые исполнителю понятны и доступны, которые входят в его систему команд. Например, алгоритм решения квадратного уравнения для исполнителя ученика 8 класса должен быть написан более подробно, чем для ученика 11 класса или студента вуза.
- *Дискретность* — алгоритм, т.е. ход решения задачи, должен быть разбит на отдельные действия, шаги. Каждое следующее действие должно выполняться только после завершения предыдущего действия.
- *Однозначность* — алгоритм должен всегда выдавать одинаковый результат при одних и тех же исходных данных. Иногда это свойство называют детерминированностью, т.е. отсутствием вероятностного развития событий.
- *Конечность (завершенность, результативность)* — алгоритм должен выдать результат за конечное число шагов.
- *Массовость* — алгоритм мог быть применен для решения не только одной конкретной задачи, но и класса однотипных задач. Другими словами, алгоритм должен без переделки позволять многократное решение задачи с разными вариантами исходных данных. Например, решение квадратного уравнения с различными значениями коэффициентов.

1.4 Элементы блок-схемы

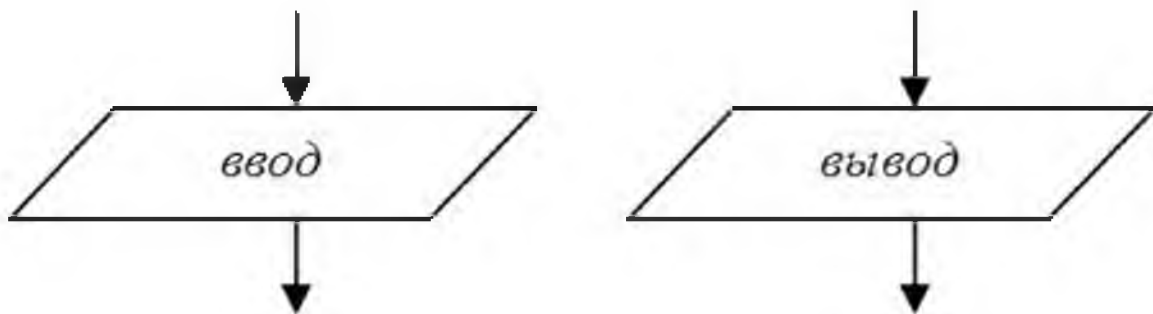
Блок-схемой называется графическое изображение алгоритма, т.е. описание хода решения задачи с помощью геометрических фигур и прямых линий.

Блок-схема строится с использованием следующих фигур:

1) **эллипс** - показывает начало или конец алгоритма (имеет один выход **или** один вход). Разрешается использовать английские слова *begin* - *начало*, *end* - *конец*.

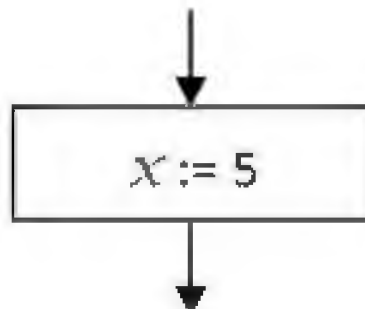


2) **параллелограмм** - показывает ввод или вывод данных (имеет один вход **и** один выход). Можно использовать английские слова *read* - *ввод*, *write* - *вывод*.

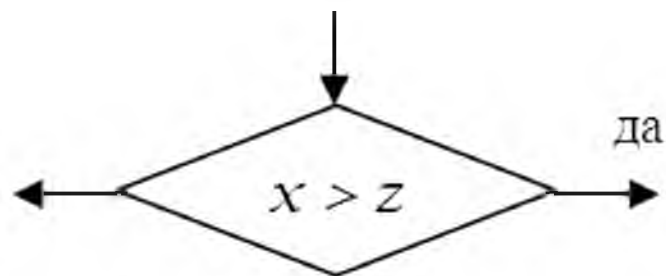


3) **прямоугольник** - показывает вычисления, действия, присваивания (имеет один вход **и** один выход). Введем обозначения:

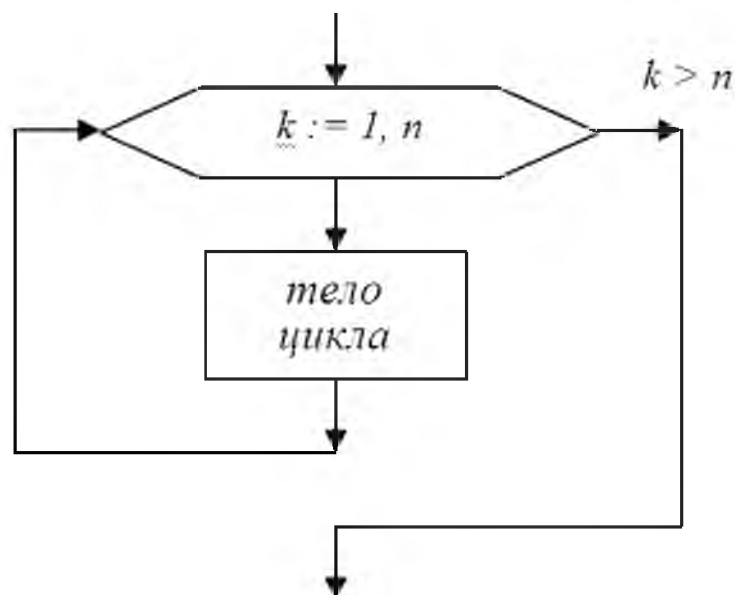
- один символ $=$ (равно) означает сравнение, равенство;
- два символа $:=$ (двоеточие и равно) означает присваивание.



4) **ромб** - показывает проверку условия; ответом является «да» или «нет» (имеет один вход и два выхода). Один из выходов снабжается надписью «да», второй выход может иметь или не иметь надпись «нет». Иногда слова «да-нет» заменяют значками «+» и «-».



5) **шестиугольник** - показывает изменение счетчика при циклических вычислениях (имеет два входа и два выхода). Многократно повторяющаяся часть называется «**телом цикла**». Запись $k := 1, n$ означает, что k изменяется от 1 до n , т.е. $k = 1, 2, 3, \dots, n$. Тело цикла выполняется n раз. При $k > n$ происходит выход из цикла. Изменение счетчика можно также записать в виде $k := k1, k2, h$; где $k1$ - начальное значение; $k2$ - последнее значение; h - шаг изменения. Если $h=1$, то его можно не указывать.



Примечания: 1) внутри тела цикла не разрешается изменение счетчика k ; 2) шестиугольник называют блоком модификации.

Приведенные 5 фигур вполне достаточны для составления блок-схем в учебных целях. Полный перечень геометрических фигур, используемых в блок-схемах, приводится в стандартах ГОСТ 19.701-90, ГОСТ 19.002-80, ГОСТ 19.003-80.

1.5 Структуры алгоритмов

При разработке алгоритма могут встретиться следующие три структуры хода вычислений. Эти структуры алгоритмов носят названия: 1) *линейный*, 2) *ветвление*, 3) *цикл*. Структуры алгоритмов можно наглядно показать с помощью блок-схемы.

1.5.1 Линейная структура

Структура алгоритма называется *линейной*, когда все действия выполняются последовательно одно за другим, как бы в одну линию (рисунок 1.1). Например, вычисление суммы и произведения двух чисел. Опишем алгоритм: 1) ввести числа a и b , т.е. узнать их значения; 2) вычислить сумму $sum = a + b$ и произведение $pr = a * b$; 3) вывести результат вычислений.

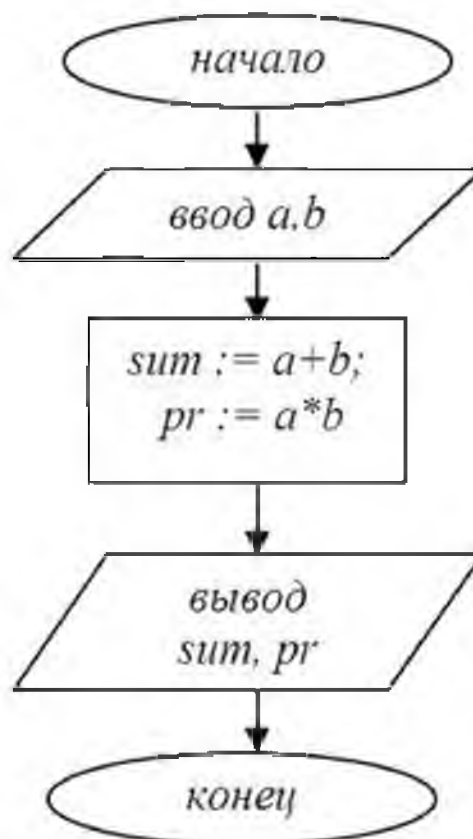


Рисунок 1.1 – Пример линейной структуры.

1.5.2 Структуры ветвления

Структура алгоритма называется разветвляющейся, если в ходе выполнения алгоритма производится проверка некоторого условия и от результата проверки условия («да» или «нет») дальнейшее действие идет по одной или другой ветви. Различают 4 вида структуры ветвления, которым присваивают следующие короткие названия: 1) «если-то», 2) «если-то-иначе», 3) «выбор», 4) «выбор-иначе».

Рассмотрим подробно эти структуры ветвления.

1.5.2.1 Структура ветвления «если-то». Если проверка условия показывает «да», то выполняется указанное действие (рисунок 1.2). Например, если $x > 0$, то принять $y = 1/x$.

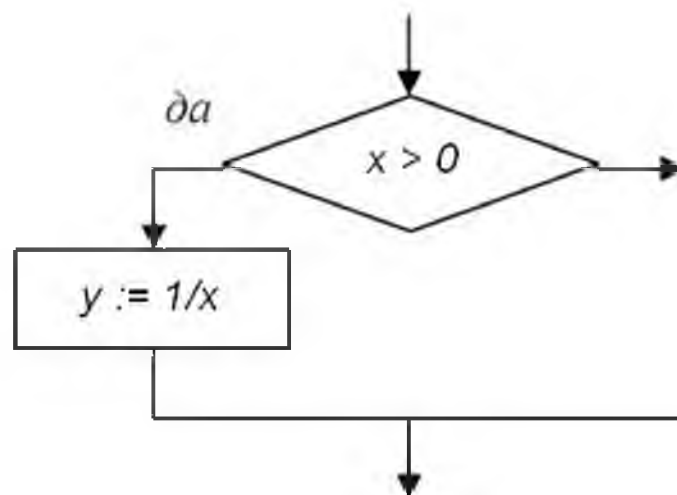


Рисунок 1.2 – Структура ветвления «если-то».

1.5.2.2 Структура ветвления «если-то-иначе». Если проверка условия показывает «да», то выполняется действие1, иначе выполнится действие2 (рисунок 1.3). Например, если первое число больше второго, то максимальным будет первое число, иначе максимальным будет второе число. Опишем алгоритм: если $a > b$, то $\max := a$, иначе $\max := b$.

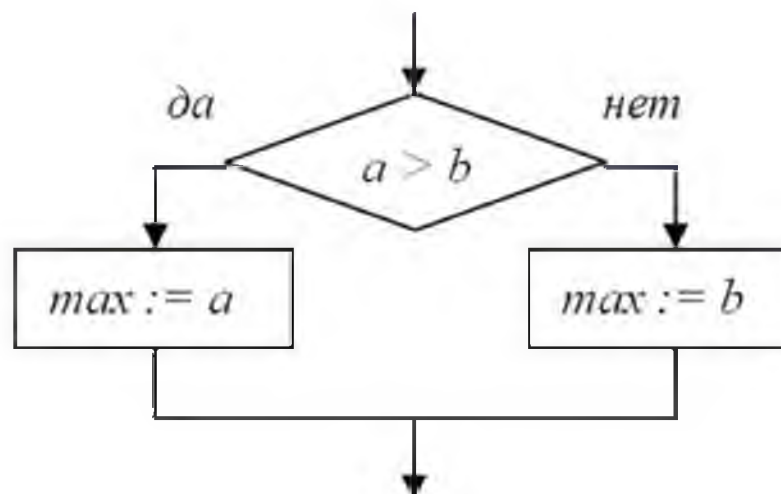


Рисунок 1.3 – Структура ветвления «если-то-иначе».

1.5.2.3 Структура ветвления «выбор». Конструкция «выбор» формулируется так: При выполнении условия1, осуществляется действие1; при выполнении условия2, осуществляется действие2; и так далее; при выполнении условияN, осуществляется действиеN (рисунок 1.4).

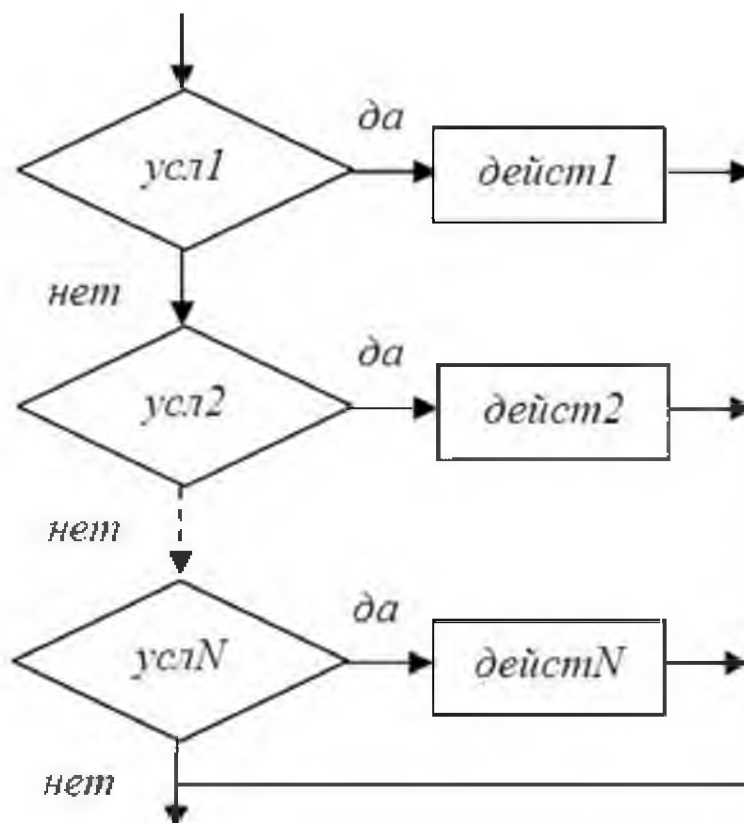


Рисунок 1.4 – Структура ветвления «выбор».

1.5.2.3 Структура ветвления «*выбор-иначе*». Конструкция «*выбор-иначе*» формулируется так: При выполнении условия1, осуществляется действие1; при выполнении условия2, осуществляется действие2; и так далее; при выполнении условияN, осуществляется действиеN; иначе осуществляется действиеN2 (рисунок 1.5).

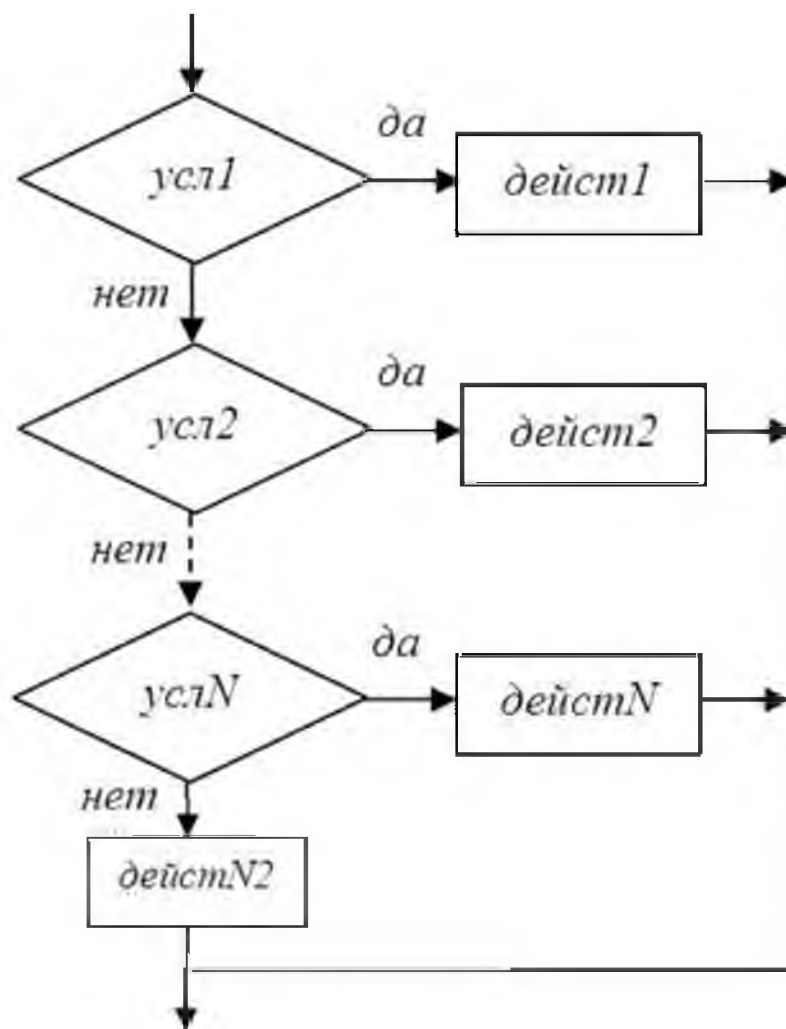


Рисунок 1.5 – Структура ветвления «*выбор-иначе*».

Пример 1.1. При заданном номере месяца h , указать время года. Времена года это весна, лето, осень, зима. При неправильно заданном месяце, указать на ошибку (рисунок 1.6).

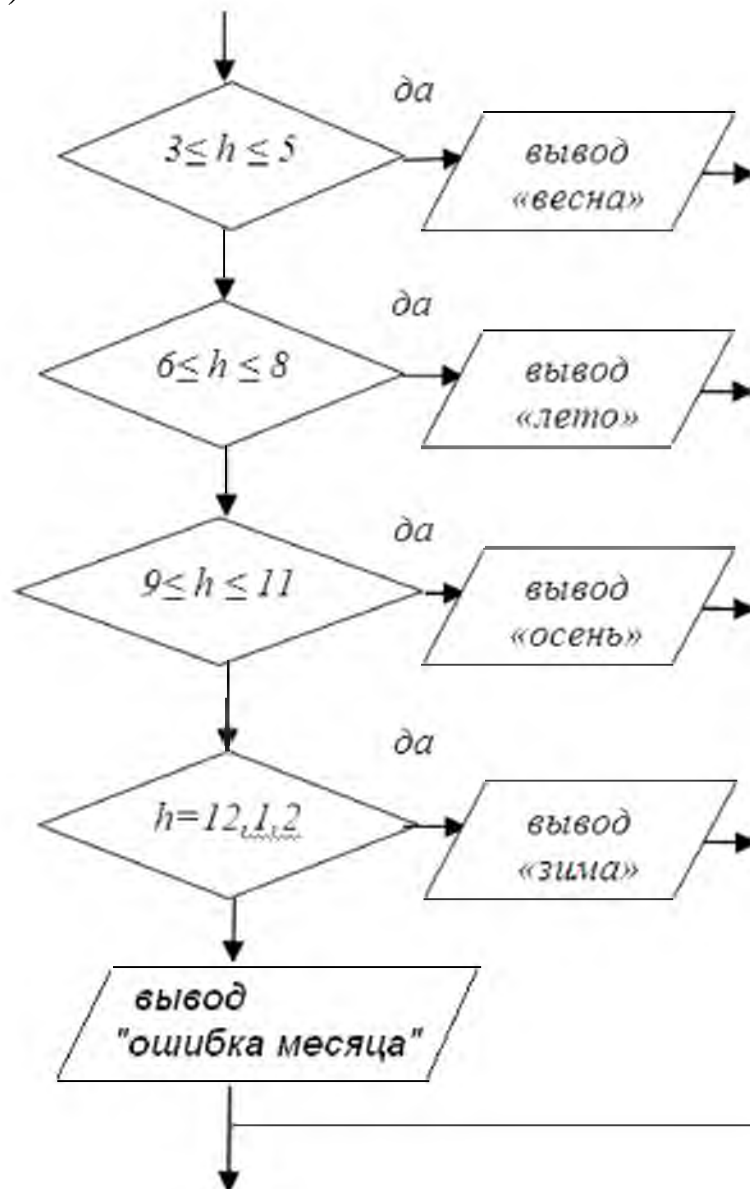


Рисунок 1.6 – Фрагмент блок-схемы к примеру 1.1.

1.5.3 Циклическая структура

Структура алгоритма называется *циклической*, когда многократно выполняется некоторая группа однотипных вычислений. Многократно повторяющаяся часть называется *телом цикла*. Циклический алгоритм всегда состоит из двух

частей: первая часть – **тело цикла**, вторая часть – **условие** повторения цикла. От взаимного расположения этих частей различают три циклические структуры: 1) цикл с предусловием, 2) цикл с постусловием, 3) цикл с параметром.

1.5.3.1 Цикл с *предУсловием*. Условие повторения цикла располагается перед телом цикла (рисунок 1.7).

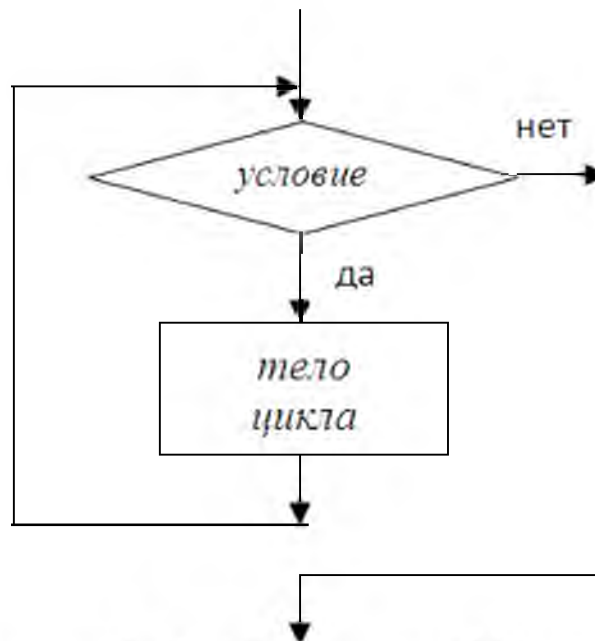


Рисунок 1.7 – Цикл с *предУсловием*.

1.5.3.2 Цикл с *постУсловием*. Условие повторения цикла располагается после тела цикла (рисунок 1.8).

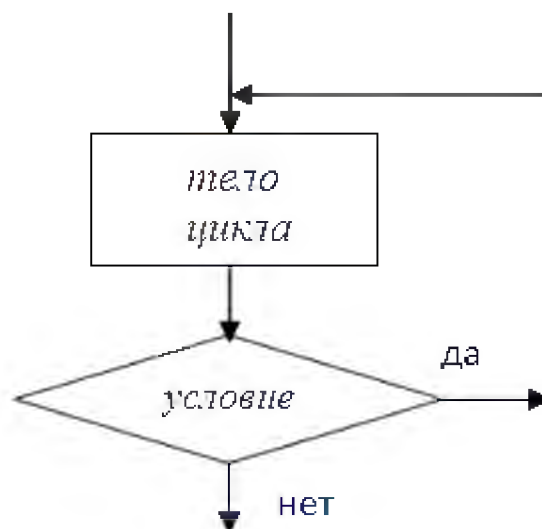


Рисунок 1.8 – Цикл с *постУсловием*.

1.5.3.3 Цикл с параметром. Параметром цикла является переменная, которая отсчитывает количество повторений тела цикла. Чаще всего записывается $k := 1, n$; что означает изменение k от 1 до n с шагом 1 (рисунок 1.9).

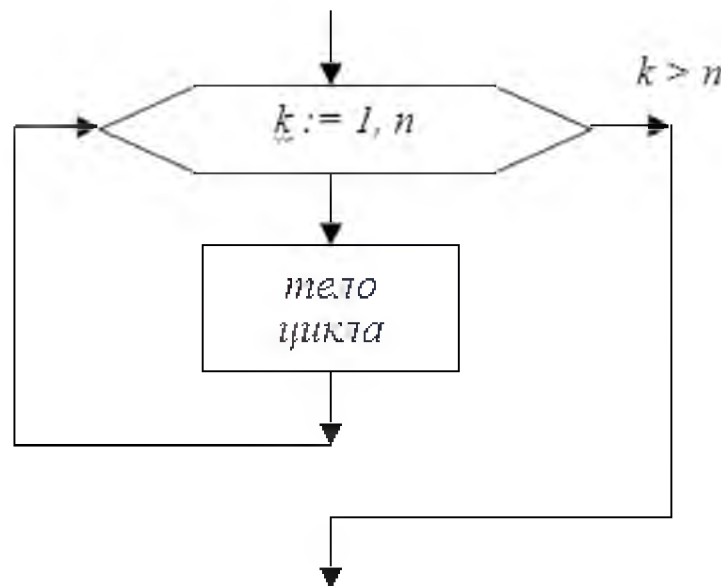


Рисунок 1.9 – Цикл с параметром.

Тело цикла выполняется n раз; при $k > n$ происходит выход из цикла. Параметр цикла можно также записать в виде $k := k1, k2, h$; где $k1$ - начальное значение; $k2$ - последнее значение; h - шаг изменения. Если $h=1$, то его можно не указывать. Внутри тела цикла не разрешается изменение параметра k .

Относительно этих трех разновидностей циклических структур отметим следующее. Циклы с предусловием и постусловием применяются при *итерационных вычислениях*, когда заранее не известно количество повторений тела цикла. При известном количестве повторений тела цикла применяют цикл с параметром.

1.6 Примеры циклических структур

1.6.1 Суммирование чисел

Пример 1.2. Вычислить сумму из n заданных чисел.

$$S = a_1 + a_2 + \cdots + a_n = \sum_{k=1}^n a_k$$

Перед составлением алгоритма рассмотрим следующую аналогию. В некоторую корзину S надо сложить n штук заданных чисел (типа бочонок лото). Вначале следует очистить корзину, т.е. $S=0$. Затем в корзину положить первое число; получим $S := S+a_1$ (читается так: к прежнему значению S прибавить a_1 и получить новое значение $S=0+a_1=a_1$). Затем опять $S := S+a_2$ (к прежнему S прибавить a_2 и получить новое значение $S=a_1+a_2$) и т.д. заполнять нашу корзину.

В этой задаче количество вычислений типа $S := S+a_k$ заранее известно, их n штук. Запись $S := S+a_k$ означает, что сначала надо выполнить вычисления справа, т.е. $S+a_k$, затем полученный результат присвоить левой переменной S .

Теперь нетрудно описать алгоритм вычисления: 1) ввод n , т.е. узнать количество заданных чисел, 2) $S := 0$, 3) организовать цикл с параметром $k := 1, n$ и телом цикла из двух действий: ввод a_k и вычисление $S := S+a_k$, 4) вывод суммы S (рисунок 1.10).

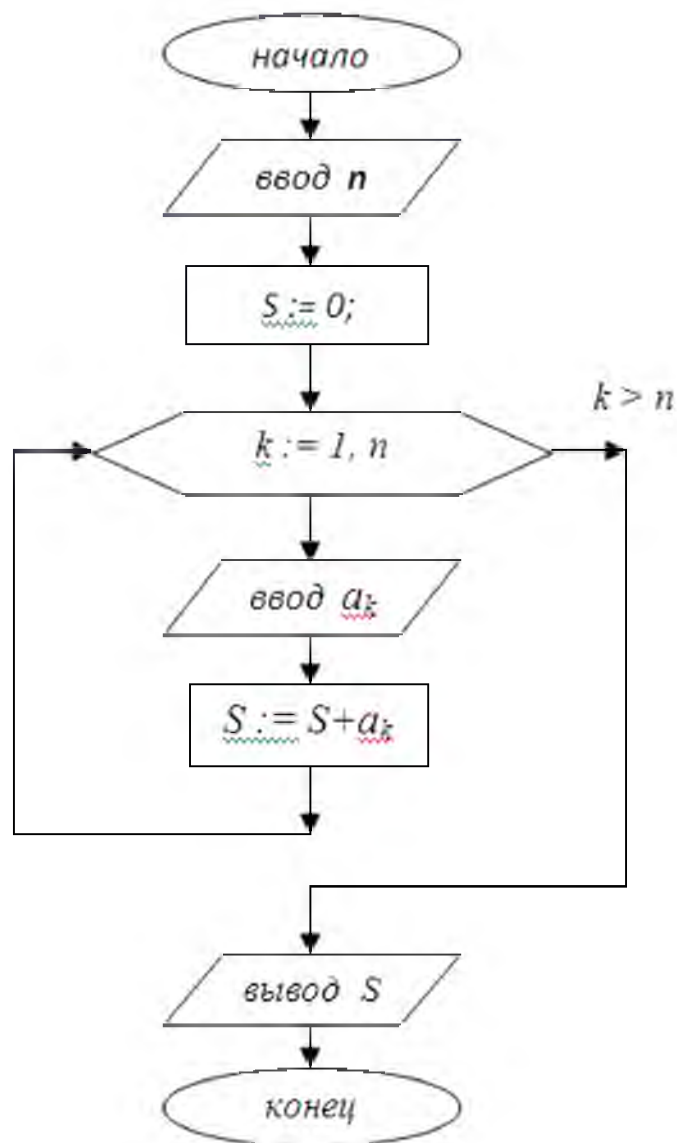


Рисунок 1.10 – Блок-схема к примеру 1.2.

1.6.2 Задача о пробежке

Пример 1.3. Спортсмен на тренировке пробегает в первый день расстояние $S = S_1$. В каждый следующий день увеличивает длину пробежки на pr процентов по отношению к вчерашнему, т.е. приращение расстояния составляет величину $p \cdot S$, где коэффициент $p = pr/100$. Таким образом, расстояние пробежки следующего дня равно $S + p \cdot S$. Возникает вопрос, сколько потребуется дней, чтобы пробежать расстояние S_n ?

Например, $S_1=500$ м, $p=0,1$ ($pr=10\%$); $S_n=2000$ м. Через сколько дней длина пробежки будет 2000 м ?

Эта задача итерационного характера, здесь заранее неизвестно количество циклических вычислений $S := S + p*S$. Известно лишь условие завершения цикла, это $S \geq S_n$.

Опишем алгоритм вычислений: 1) ввод S_1 , p , S_n ; 2) в первый день $k:=1$; $S := S_1$; 3) в следующие дни $k := k+1$; $S := S + p*S$; 4) если $S \geq S_n$, то выход из цикла, иначе переход к 3-му пункту; 5) вывод искомого количества дней $n := k$.

Рассмотрим выполнение алгоритма по шагам.

1) Ввод $S_1=500$, $p=0,1$; $S_n=2000$.

2) $k := 1$, $S := S_1=500$;

3) $k := k+1 = 1+1 = 2$, $S := S + p*S = 500+0,1*500 = 550$;

4) $k := k+1 = 2+1 = 3$, $S := S + p*S = 550+0,1*550 = 605$;

5) $k := k+1 = 3+1 = 4$, $S := S + p*S = 605+0,1*605 = 665,5$;

17) $k := k+1 = 15+1 = 16$, $S := S + p*S = 1898,7+0,1*1898,7 = 2088,6$;

Таким образом, результат будет достигнут в 16-й день (расстояние более 2000 м).

Блок-схема решения данной задачи представлена на рисунке 1.11.

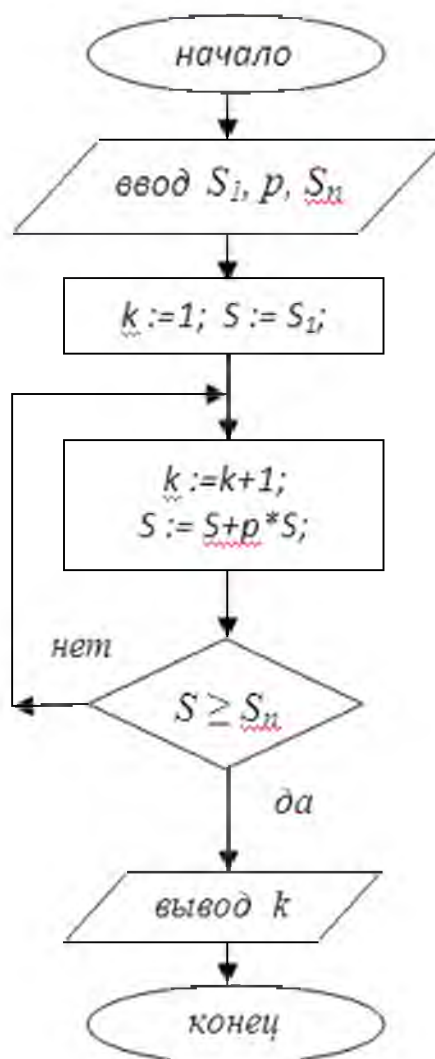


Рисунок 1.11 – Блок-схема к примеру 1.3.

1.7 Вложенные циклы

Может оказаться так, что тело цикла содержит внутри себя некоторый циклический процесс. В таком случае говорят о вложенных циклах. Уровень вложения может быть различным.

Например, вычисление суммы чисел в матрице описывается вложенным циклом в 2 уровня: внешний цикл описывает перемещение по строкам, а внутри строки перемещение по столбцам представляет собой внутренний цикл.

Пример 1.4. Обозначим матрицу $A = \{ a_{ij} \}$, где $i := 1, n$; $j := 1, m$; n, m – размеры матрицы; i – номер строки; j – номер столбца; S – сумма элементов матрицы.

$$A = \{ a_{ij} \} = \begin{Bmatrix} a_{11} & a_{12} & \dots & a_{1,i} & \dots & a_{1m} \\ a_{21} & a_{22} & \dots & a_{2,i} & \dots & a_{2,m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{i,1} & a_{i,2} & \dots & a_{i,i} & \dots & a_{i,m} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,i} & \dots & a_{n,m} \end{Bmatrix} \quad S = \sum_{i=1}^n \sum_{j=1}^m a_{ij}$$

Сумму чисел матрицы вычисляем по аналогии с примером 1.2. Сначала выделяем строку и суммируем числа в этой строке. Строка i меняется от 1 до n . Внутри строки элементы меняются по индексу j от 1 до m . Перемещения по строкам образуют внешний цикл, перемещения внутри строки образуют внутренний цикл (рисунок 1.12).

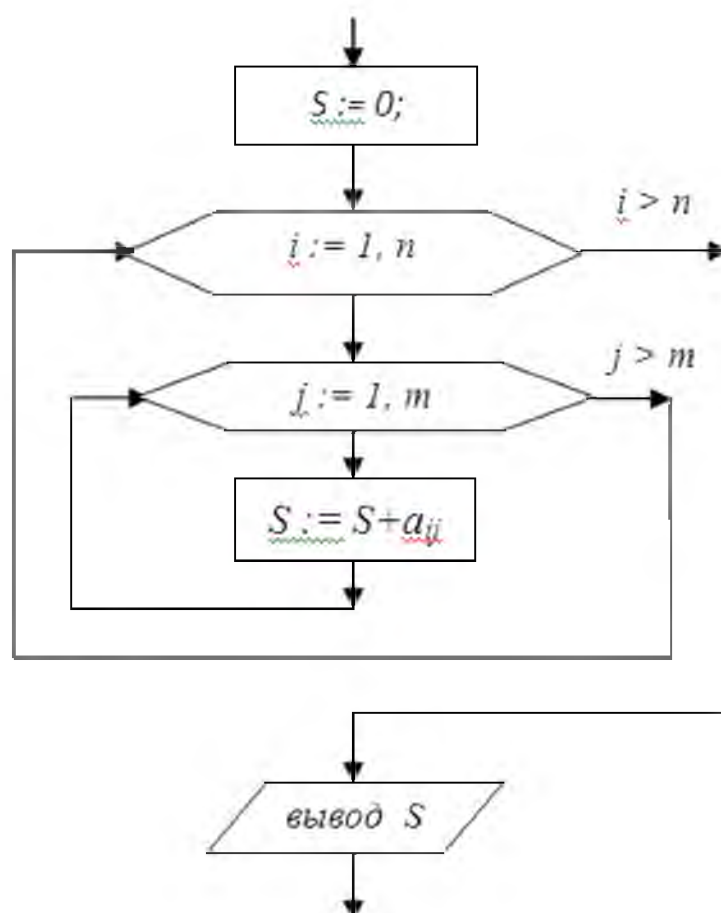


Рисунок 1.12 – Фрагмент блок-схемы к примеру 1.4.

1.8 Об эффективности алгоритма

Выше были указаны основные свойства алгоритма. Обобщением этих свойств является *эффективность* алгоритма. Рассмотрим это понятие на примере нахождения максимального числа *max* среди заданных чисел.

Пример 1.5. Нахождения максимального числа *max* из двух чисел *a*, *b*. Алгоритм: 1) ввод чисел, 2) сравнение двух чисел, 3) вывод результата (рисунок 1.13).

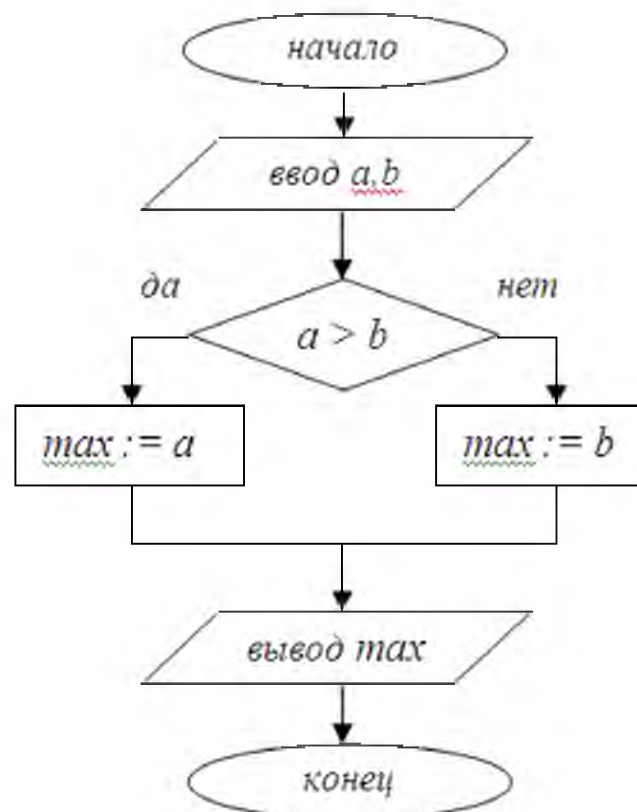


Рисунок 1.13 – Поиск *max* (алгоритм «если-то-иначе»).

Пример 1.6. Нахождения максимального числа *max* из трех чисел *a*, *b*, *c*. Алгоритм: 1) ввод чисел; 2) сравнение двух чисел *a* и *b*; 3) если направление «да», то сравнение *a* и *c*, иначе сравнение *b* и *c*; 4) нахождение *max* из двух чисел; 5) вывод результата.

Для трех чисел (например, 3,4,5) возможны шесть разных вариантов (var) перестановок чисел.

<i>a</i>	5	5	4	4	3	3
<i>b</i>	3	4	3	5	4	5
<i>c</i>	4	3	5	3	5	4
<i>Var</i>	1	2	3	4	5	6

На блок-схеме (рисунок 1.14) выносками показаны номера вариантов сочетаний чисел в данном направлении после выполнения сравнения.

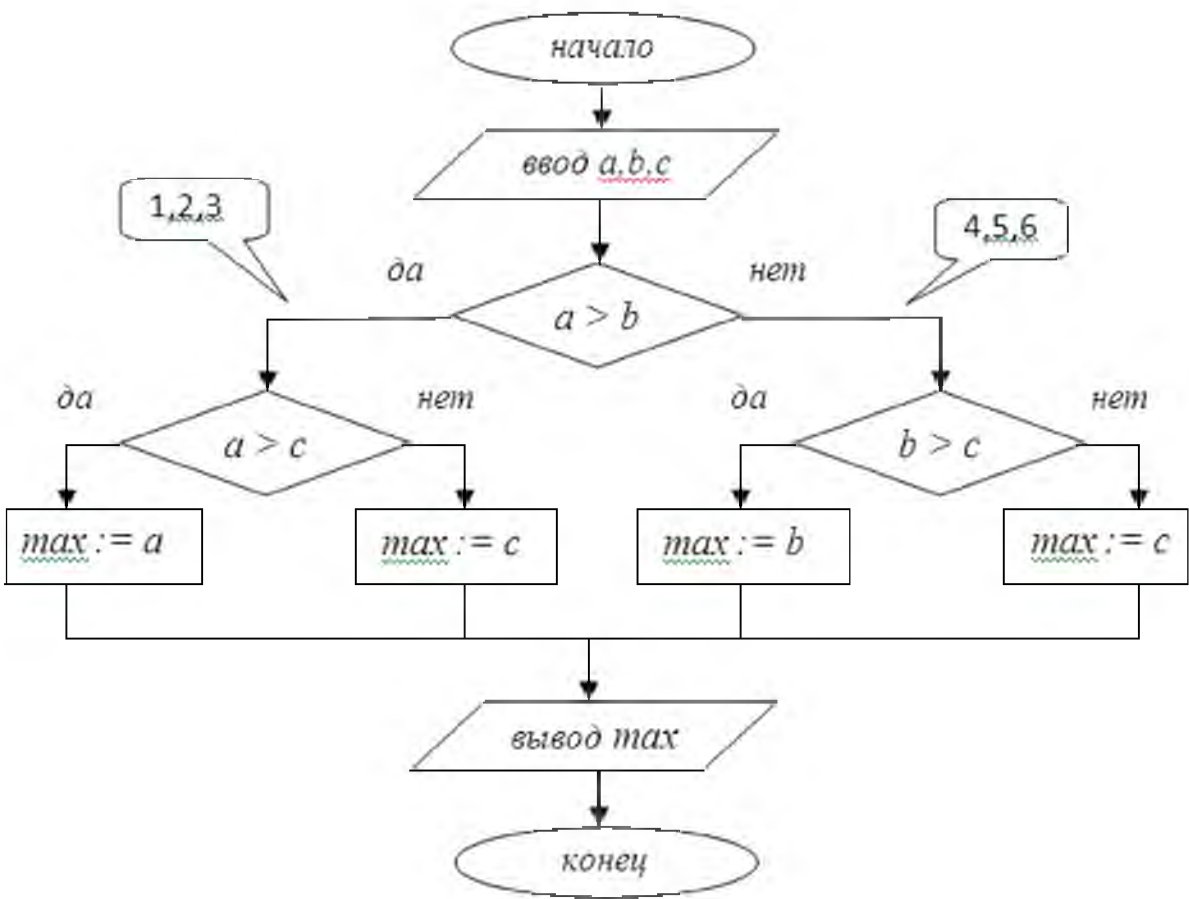


Рисунок 1.14 – Поиск *max* (алгоритм «если-то-иначе»).

Составленная блок-схема не содержит ошибок и приводит к правильному результату. Однако такая блок-схема нам не нравится: она кажется сложной для восприятия и содержит

дублирующий блок с присвоением. Тем не менее, в связи с отсутствием ошибок, продолжим рассуждения.

Пример 1.7. Нахождения максимального числа \max из 4-х чисел a, b, c, d . Здесь возможны 24 варианта перестановок чисел.

<i>a</i>	6	6	6	6	6	6	5	5	5	5	5	5
<i>b</i>	5	5	4	4	3	3	6	6	4	4	3	3
<i>c</i>	4	3	5	3	4	5	4	3	3	6	4	6
<i>d</i>	3	4	3	5	5	4	3	4	6	3	6	4
<i>Var</i>	1	2	3	4	5	6	7	8	9	10	11	12

<i>a</i>	4	4	4	4	4	4	3	3	3	3	3	3
<i>b</i>	6	6	5	5	3	3	6	6	5	5	4	4
<i>c</i>	3	5	3	6	6	5	4	5	6	4	6	5
<i>d</i>	5	3	6	3	5	6	5	4	4	6	5	6
<i>Var</i>	13	14	15	16	17	18	19	20	21	22	23	24

Рассуждая аналогично предыдущему примеру, будем строить следующий алгоритм: 1) ввод 4-х чисел; 2) на первом уровне сравнение двух чисел $a > b$; 3) в каждом из направлений «да-нет» по 12 вариантов сочетаний чисел; если направление «да», то в этой ветви на втором уровне сравнение $a > c$; 4) на третьем уровне два сравнения $a > d$ (8 вариантов чисел) и $c > d$ (4 варианта чисел); и т.д. (рисунок 1.15).

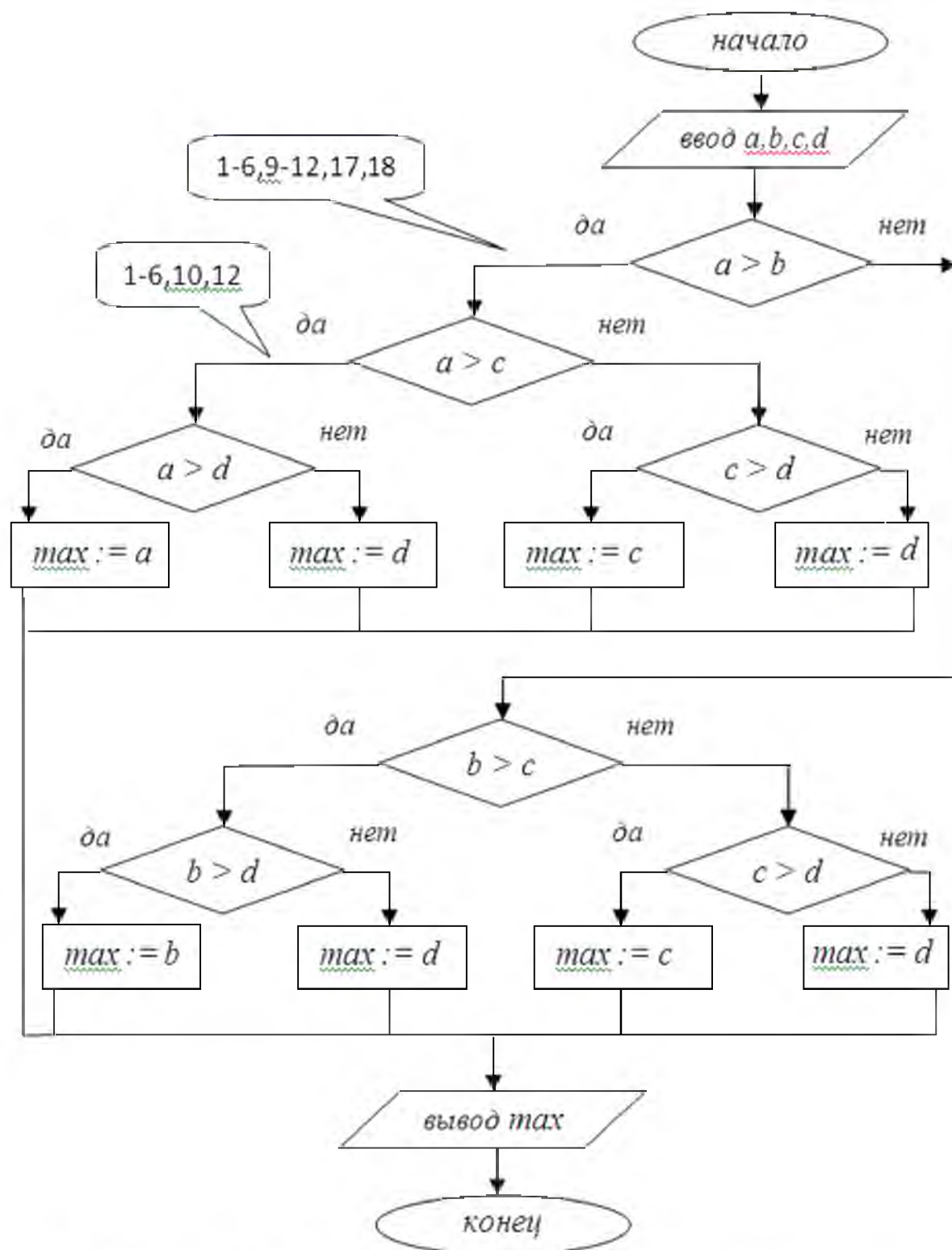


Рисунок 1.15 – Поиск *max* (алгоритм «если-то-иначе»).

Эта блок-схема не содержит ошибок, однако она сложна для восприятия. Можно окончательно запутаться при построении блок-схемы поиска *max*, например, из 15 чисел. Делаем вывод:

поиск *max* по алгоритмам, приведенным в примерах 1.5-1.7, **неэффективен**.

При поиске *max* из двух чисел, неэффективность алгоритма не замечается и не проявляется. Принятая за основу идея поиска *max* из двух заданных чисел, с использованием структуры ветвления «если-то-иначе», оказывается неудачной для большего количества чисел. Приведенная идея поиска *max* путем сравнения заданных чисел между собой по схеме «если-то-иначе», оказалась неэффективной.

Пример 1.8. Рассмотрим другую идею поиска *max* из двух чисел. Опишем алгоритм: 1) ввести оба числа; 2) первое число принять за *max*; 3) второе число сравнить с *max*, используя ветвление «если-то»; 4) вывод *max* (рисунок 1.16).

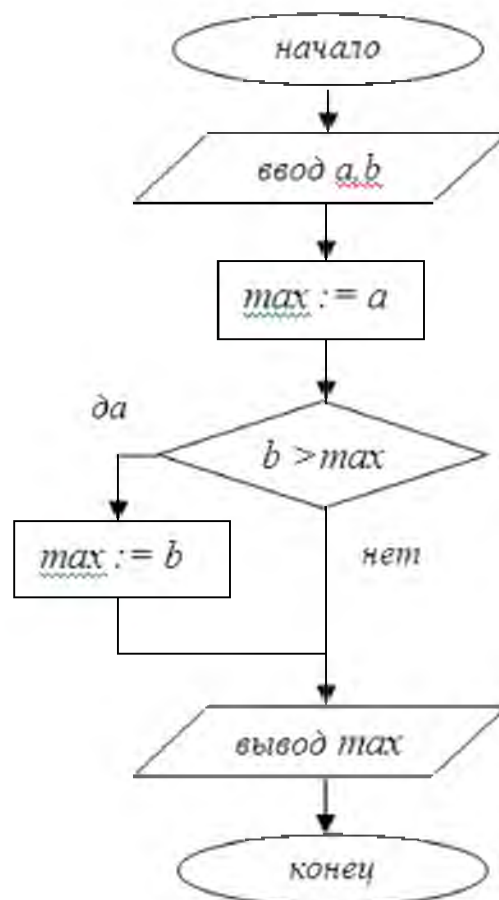


Рисунок 1.16 – Поиск *max* (алгоритм «если-то»).

Пример 1.9. Поиск *max* из трех чисел. Опишем алгоритм: 1) ввести числа; 2) первое число принять за *max*; 3) второе число сравнить с *max* по схеме «если-то»; 4) аналогично, третье число сравнить с *max*; 5) вывод *max* (рисунок 1.17).

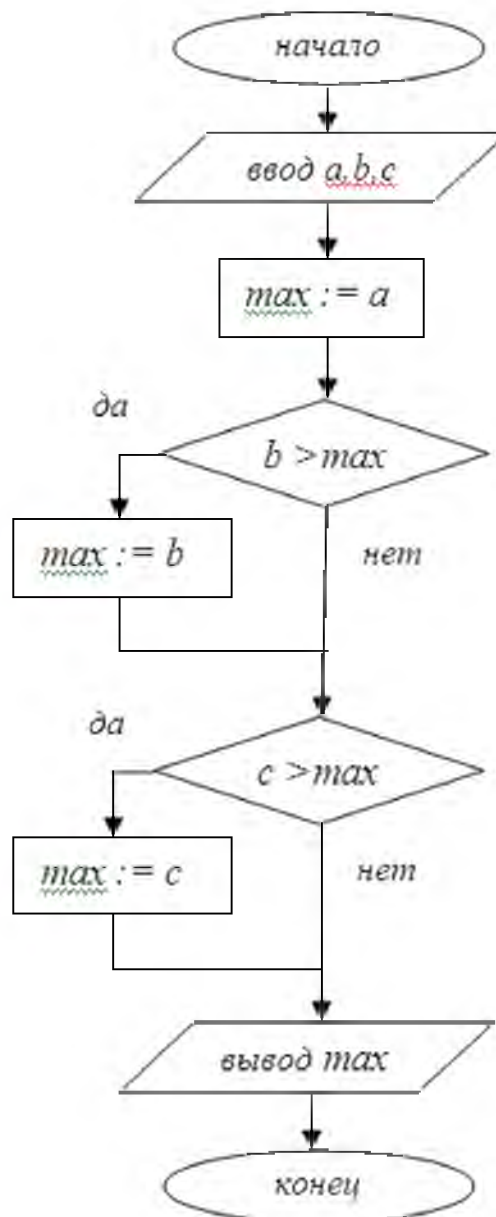


Рисунок 1.17 – Поиск *max* (алгоритм «если-то»).

Теперь сравните алгоритмы поиска *max* в примерах 1.5, 1.6 и в примерах 1.8, 1.9. Эффективность последних алгоритмов очевидна — они чрезвычайно просты для восприятия и понимания. Основная идея следующая: первое число принимается за *max*, затем остальные числа поочередно

сравниваются с max по схеме «если-то»; сравниваются числа не между собой, а сравнивается очередное число с тем значением max , которое определилось на предыдущем шаге.

Пример 1.10. Поиск max из n чисел: $a_1, a_2, \dots, a_k, \dots, a_n$; Опишем алгоритм: 1) ввод n ; 2) ввод a_1 ; $max := a_1$; 3) организовать цикл с переменной k , которая изменяется от 2 до n , т.е. $k := 2, n$ и с телом цикла: ввод a_k и сравнение $a_k > max$ по схеме «если-то»; 4) вывод max . Всё очень просто и понятно (рисунок 1.18).

Таким образом, будем считать алгоритм эффективным, если он компактный и простой для восприятия и вычисления.

В книге Дональда Кнута «Искусство программирования», приводится следующее определение: «Алгоритм считается эффективным, если все его операции достаточно просты для того, чтобы их можно было точно выполнить в течение конечного промежутка времени с помощью карандаша и бумаги» [1].

Разработка эффективного алгоритма является творческой работой, направленной на поиск ответа кратчайшим путем.

Пример 1.11. Известен случай из биографии великого математика Карла Фридриха Гаусса. Учитель младших классов дал задачу: сложить числа от 1 до 100. Ученики, кроме одного Карла, стали суммировать поочередно все числа (по схеме нашего примера 1.2). Юный Гаусс обратил внимание на то, что сумма двух крайних чисел дают одинаковый результат: $1+100=101$; $2+99=101$; $3+98=101$, и т.д.; таких слагаемых 50 штук, половина всех чисел. Гаусс первым дал ответ $= 101*50 = 5050$. Алгоритм Гаусса для вычисления суммы целых чисел от 1 до n по формуле $S=(1+n)*n/2$ оказался более эффективным, чем схема примера 1.2 (рис.1.2). Количество вычислений по схеме примера 1.2 зависит от n , а по алгоритму Гаусса только одно вычисление по формуле.

Примечание: формула Гаусса верна только для непрерывного ряда целых чисел от 1 до n . Алгоритм же суммирования по схеме примера 1.2 может применяться для произвольного набора чисел, в этом смысле, он наиболее эффективен.

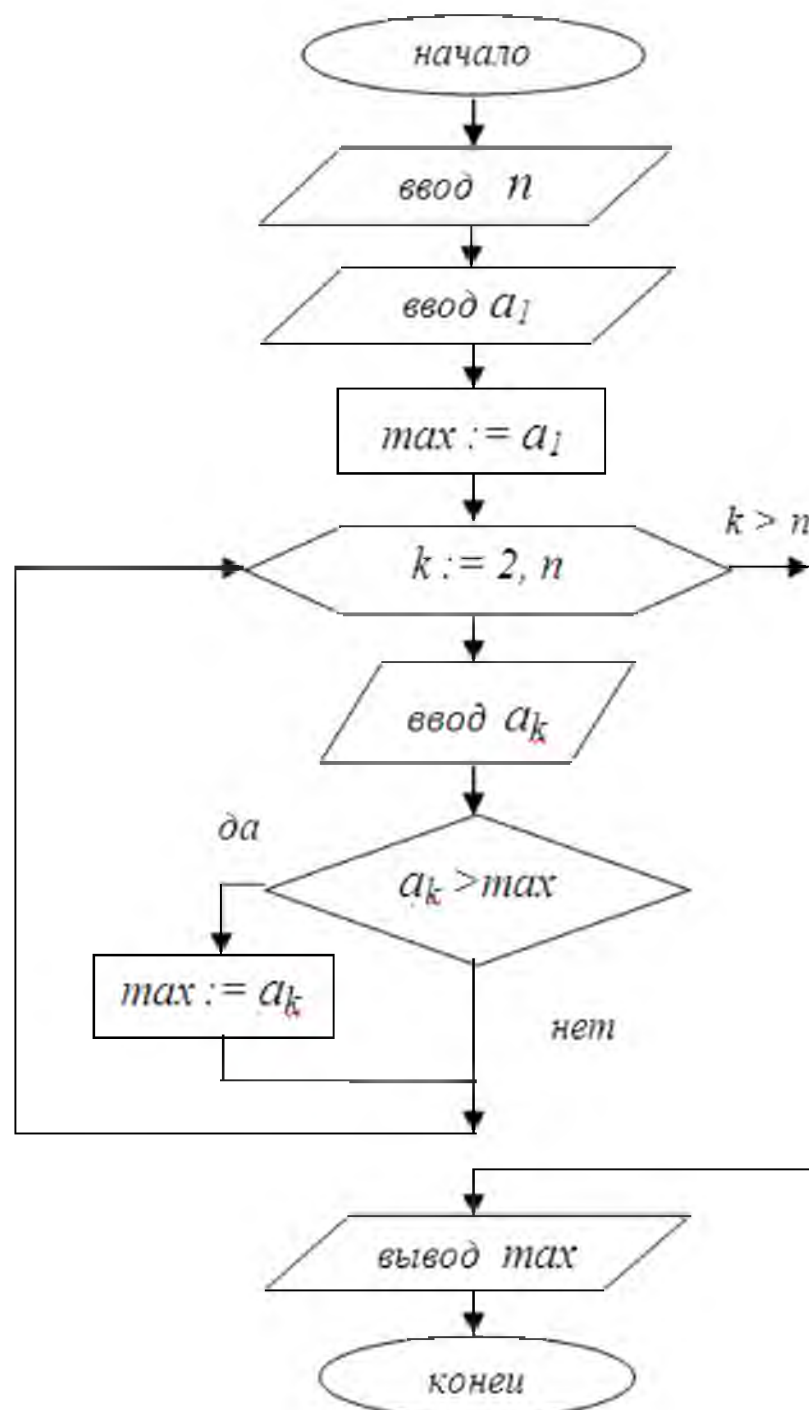


Рисунок 1.18 – Блок-схема к примеру 1.10.
(поиск max из n чисел).

Примеры 1.8-1.11 завершают рассмотрение вопроса об эффективности алгоритма.

1.9 Алгоритм укрупненный и подробный

В блок-схемах фигура прямоугольника указывает на выполнение вычислений, связанных с присваиванием или на выполнение группы действий. В частности, прямоугольником показывается тело цикла – многократно повторяющаяся часть. Иногда, для удобства восприятия алгоритма, некоторую его часть обозначают прямоугольником и затем эту часть показывают отдельно в подробном исполнении.

Например, блок-схема вычисления суммы чисел в примере 1.2 показана подробно. Разложим эту схему на две части: укрупненная схема и подробная ссылка на укрупненный блок (рисунок 1.19).

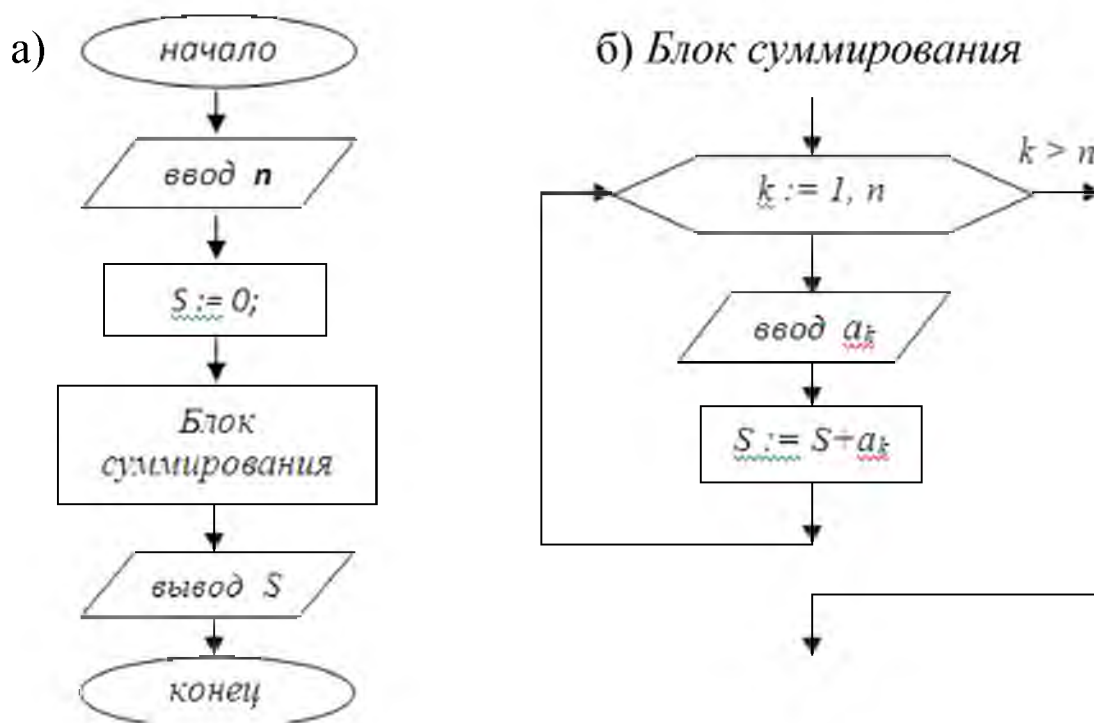


Рисунок 1.19 – Блок-схема суммирования n чисел;

а) укрупненная схема,

б) подробная схема блока суммирования.

Известно, что чем более подготовлен исполнитель для решения задачи, тем менее подробно можно описывать алгоритм. Например, решение квадратного уравнения (определение корней) для школьника-восьмиклассника следует описать более подробно, чем для 11-классника. Как было сказано, это следует из уровня подготовки исполнителя. Исполнитель-человек обладает определенным багажом знаний, способен логически мыслить и ему порой достаточно подсказать идею, чтобы прийти к верному решению. Исполнитель-компьютер – это техническое устройство (набор элементов из железа, пластмассы, проводов и т.п.), по существу «нечто безмозглое», ничего самостоятельно «думать и решать» не способен; компьютер может только неукоснительно выполнять последовательность простейших команд, заранее составленных человеком. Поэтому для исполнителя-компьютера необходимо составлять подробнейший алгоритм действий на машинном языке.

Таким образом, для исполнителя-человека алгоритм составляется либо в укрупненном, либо в подробном виде, в зависимости от его уровня знаний, а для исполнителя-компьютера алгоритм всегда составляется в подробном виде, в форме компьютерной программы.

1.10 Контрольные вопросы:

1. Что называется алгоритмом ?
2. На что обратить внимание в определении понятия алгоритм?
3. Какими основными свойствами должны обладать алгоритмы?
4. Исполнитель алгоритма.
5. Формы описания алгоритма.
6. Какая форма описания алгоритма является наиболее продуманной и подробной? и почему?
7. Что называется алгоритмизацией?
8. Что называется программированием?
9. Что такое блок-схема?

10. Какие вычислительные структуры могут содержать алгоритмы?
11. Какова структура линейного алгоритма?
12. Из каких блоков составляется линейная структура алгоритма?
13. Какая характерная особенность разветвляющейся структуры алгоритма?
14. Из каких блоков составляется разветвляющаяся структура алгоритма?
15. На какие виды подразделяется структура ветвления?
16. Какой вычислительный процесс называется циклическим?
17. Из каких двух частей состоит циклическая структура алгоритма?
18. На какие три вида подразделяется циклическая структура?
19. Для каких задач следует использовать цикл с параметром?
20. В каких задачах используется цикл с предусловием или постусловием?
21. Что означают записи $k := 1, n$ или $k := k1, k2, h$?
22. Что означают записи $S := S+a$ и $S = a+b$?
23. Что значит вложенный цикл?
24. Что понимается под эффективностью алгоритма?

2 ПРИМЕРЫ СОСТАВЛЕНИЯ БЛОК-СХЕМ

2.1 Решение квадратного уравнения

Пример 2.1. Определить корни квадратного уравнения

$$ax^2 + bx + c = 0; \quad a \neq 0;$$

Опишем алгоритм: 1) ввести коэффициенты a, b, c ; 2) вычислить дискриминант $d = b^2 - 4ac$; 3) если $d \geq 0$, то вычислить корни

$$x_1 = \frac{-b + \sqrt{d}}{2a}; \quad x_2 = \frac{-b - \sqrt{d}}{2a};$$

и вывести их значения, иначе вывести сообщение «корней нет» (рисунок 2.1).

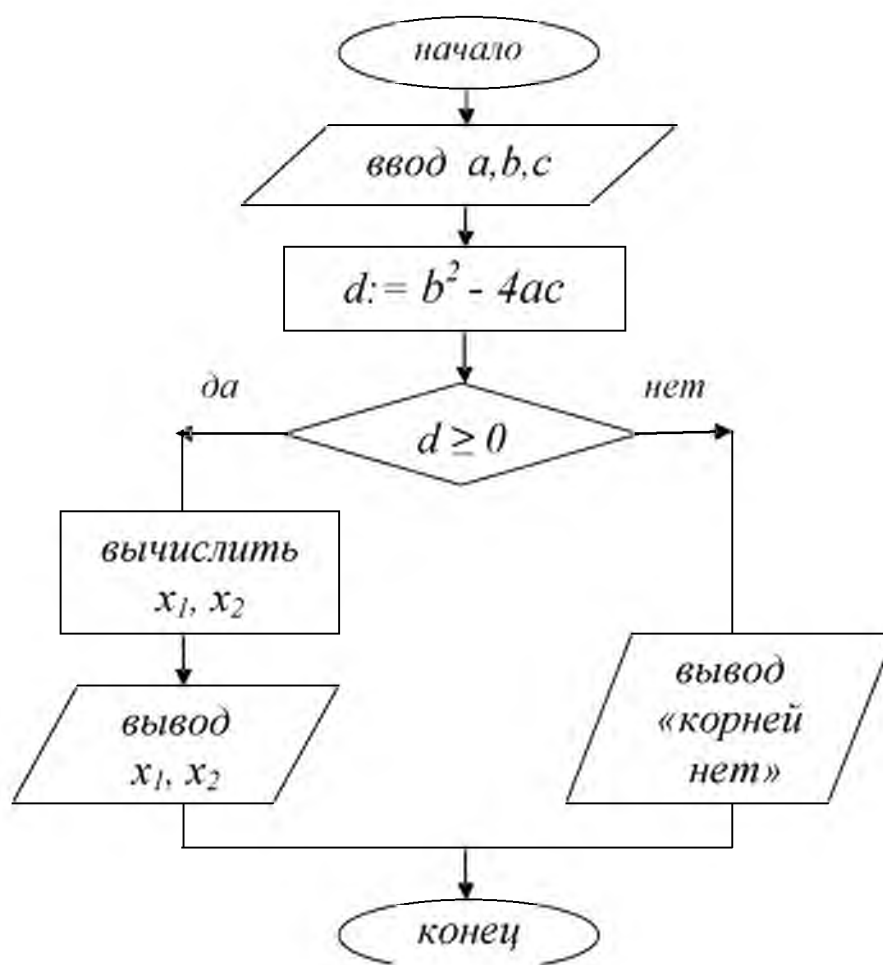


Рисунок 2.1 – Решение квадратного уравнения

2.2 Вычисления в одномерной последовательности

Пример 2.2. Для заданной последовательности чисел найти сумму всех чисел, сумму отрицательных чисел и их количество, сумму положительных чисел и их количество.

Обозначения: n – количество чисел; $a_1, a_2, \dots, a_k, \dots, a_n$ – последовательность заданных чисел; $k := 1, n$ – индекс k меняется от 1 до n ; S – сумма всех чисел; S_1, t – сумма и количество отрицательных чисел; S_2, p – сумма и количество положительных чисел; $S_2 = S - S_1$; $p = n - t$;

За основу берем схему суммирования n чисел (см. рисунок 1.10) и добавляем блок суммирования отрицательных чисел, при этом суммируем также их количество. Блок-схема данной задачи приведена на рисунке 2.2.

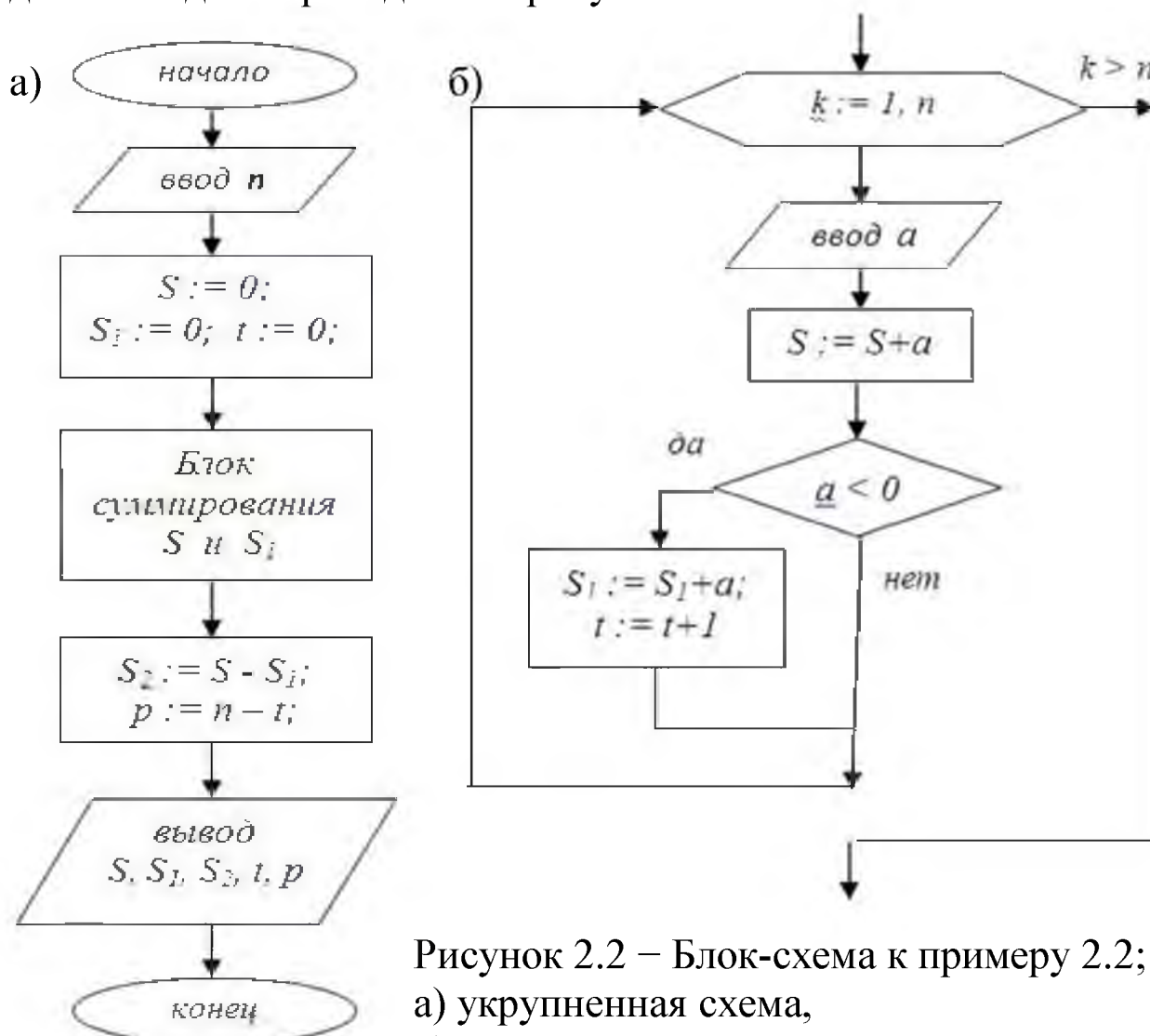


Рисунок 2.2 – Блок-схема к примеру 2.2;
а) укрупненная схема,
б) подробная схема блока суммирования.

Пример 2.3. Вычислить сумму следующего ряда

$$S = \sum_{k=1}^n a_k, \quad \text{где } a_k = \frac{5k}{3+k^2} > 10^{-1};$$

Данный ряд убывающий и ограниченный. Найти сумму ряда и количество членов n .

Опишем алгоритм: 1) вначале примем $k=0$, сумма $S=0$; 2) берем $k := k+1$; 3) вычисляем a_k ; 4) если $a_k > 10^{-1}$, то вычислить $S := S + a_k$ и перейти к пункту 2, иначе переход к пункту 5; 5) вывод S и k ; последнее значение k есть n .

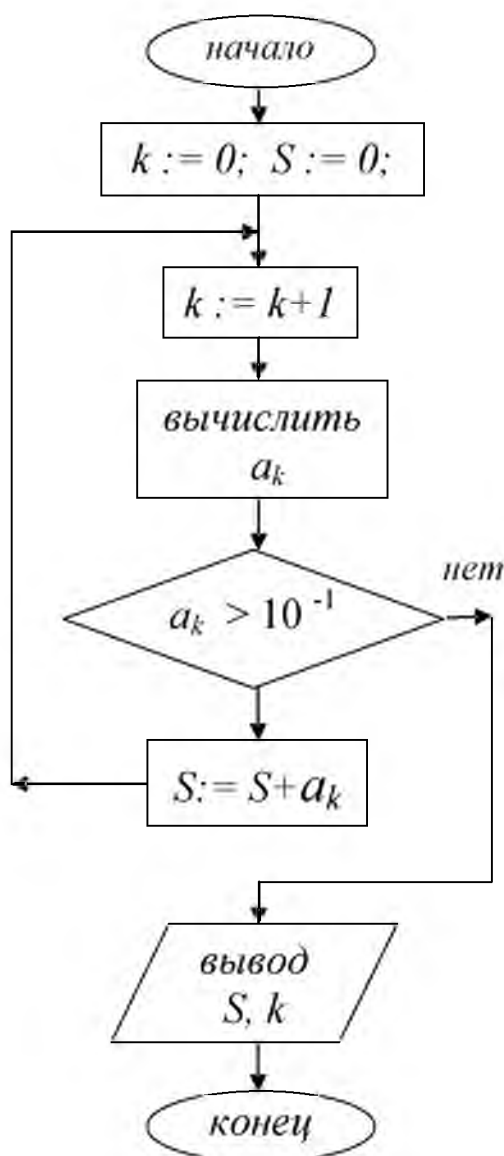


Рисунок 2.3 – Блок-схема к примеру 2.3.

Пример 2.4. Заданы две последовательности чисел $a_k, b_k, k := 1, n$. Вычислить элементы третьей последовательности по формуле $c_k = a_k - 2b_k$.

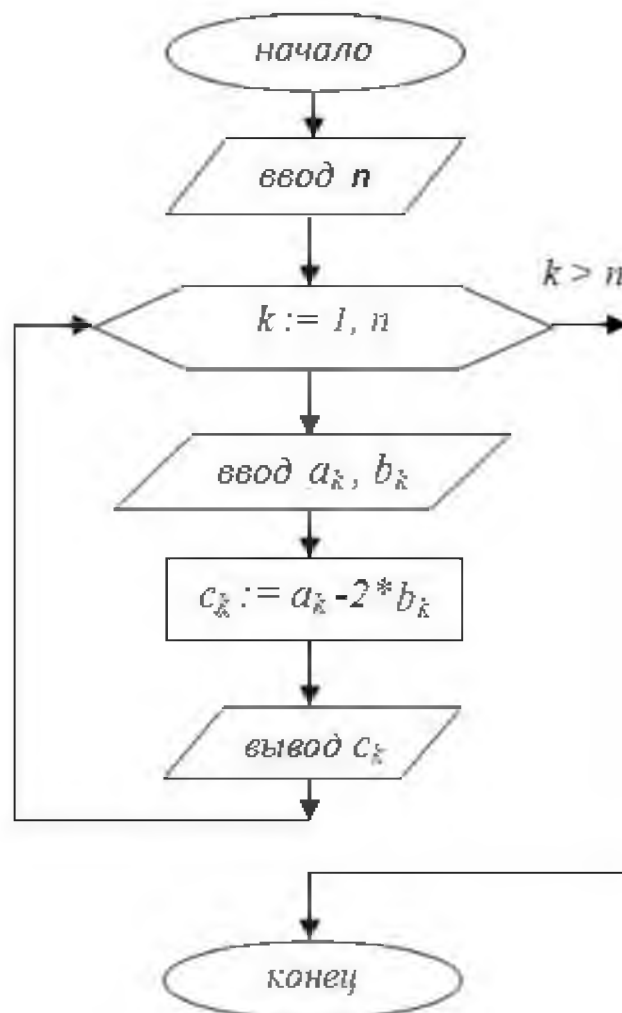


Рисунок 2.4 – Блок-схема к примеру 2.4.

2.5 Вычисление значений функции

Пример 2.5. Вычислить значения следующей функции. На отрезке $x = [12; 14]$ с шагом $h = 0,2$ задана функция

$$y = \begin{cases} 5 + x, & \text{если } x \leq 13 \\ x \cdot x, & \text{если } x > 13 \end{cases}$$

1	2	3	4	...	n	k
---	---	---	---	-----	---	---

12

13

14

x

Здесь $k := 1, n$ - номера точек на оси x ;

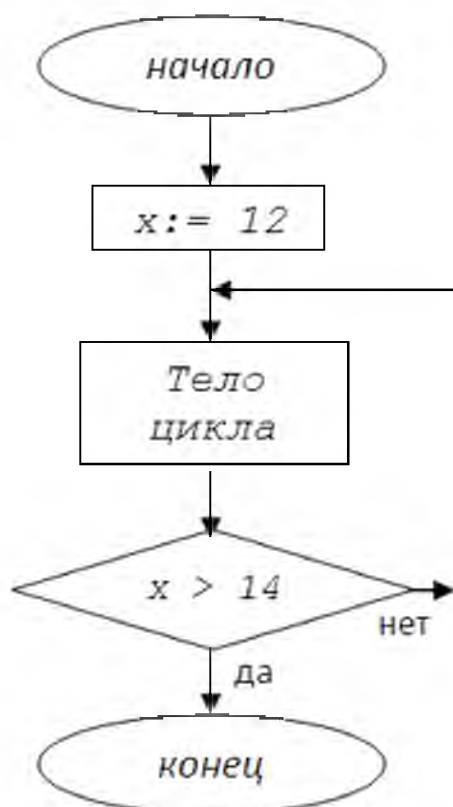
$n = (14 - 12)/0,2 + 1 = 11$ - количество точек на оси.

Вычислить значения функции в заданных точках.

Решение: Значения x в точках разбиения оси известны: $x = 12$; $x := x + 0,2$; и т.д. до $x = 14$. Для каждого значения x надо вычислить y . Это циклический алгоритм. Телом цикла является вычисление y по верхней или нижней формуле.

Покажем все три циклические структуры: 1) цикл с предусловием, 2) цикл с постусловием, 3) цикл с параметром (рис. 2.5 – 2.6).

а) цикл с постУсловием



б) тело цикла

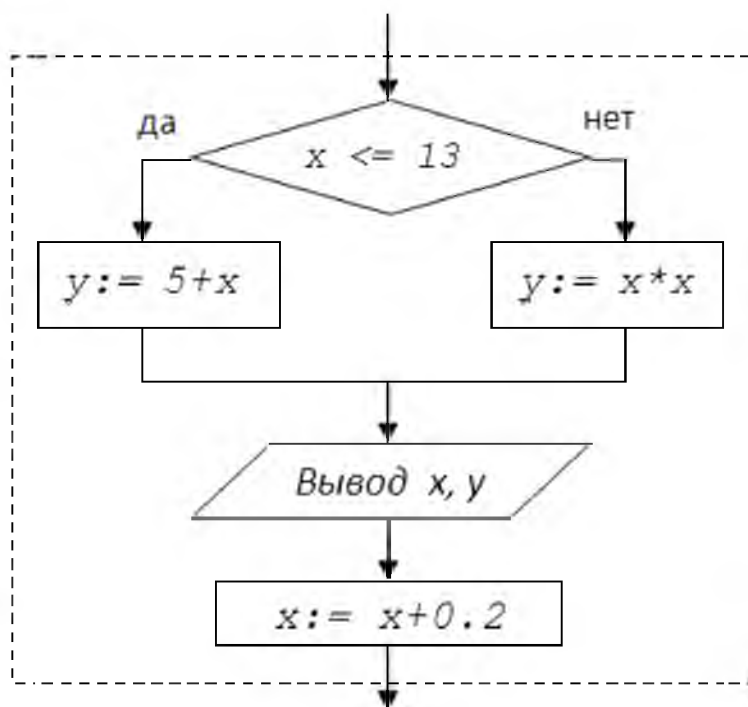
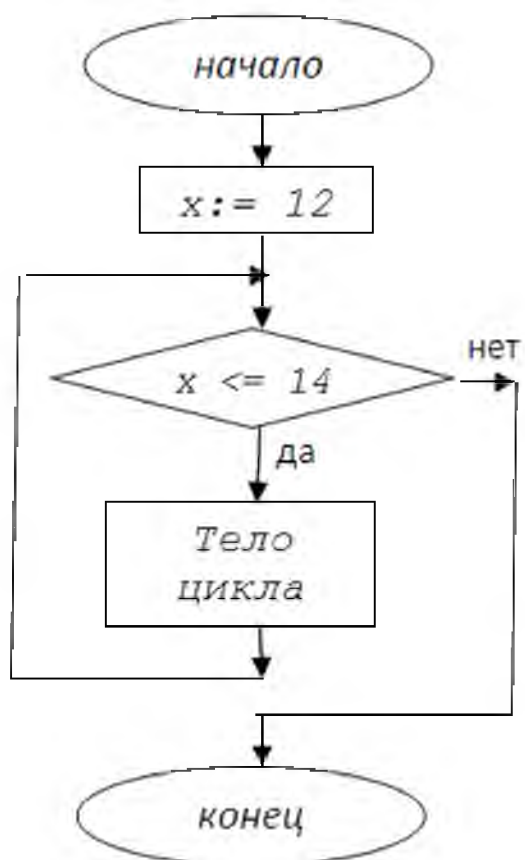


Рисунок 2.5 – Блок-схема к примеру 2.5

а) цикл с предУсловием



б) цикл с параметром k

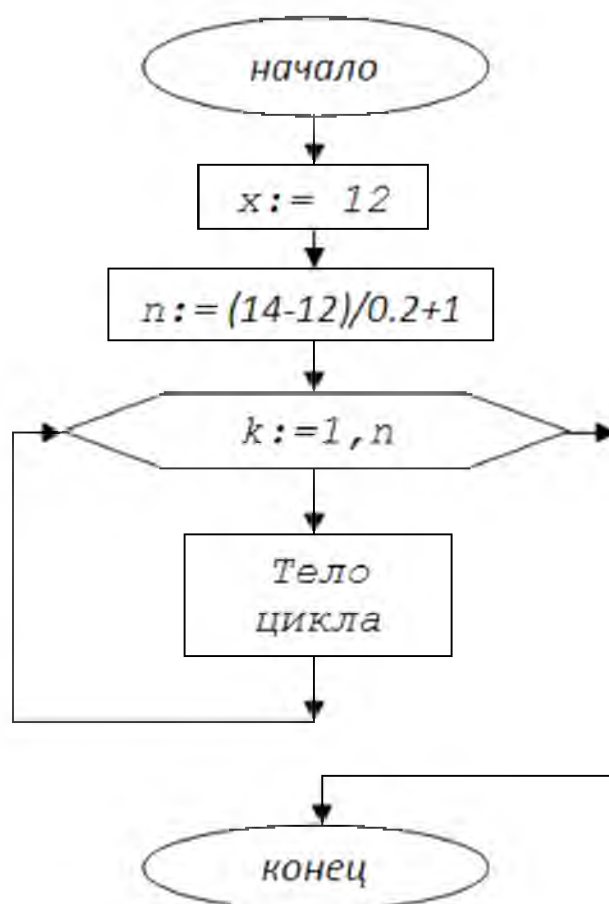


Рисунок 2.6 – Блок-схема к примеру 2.5

2.6 Поиск \max и \min

Пример 2.6. Найти \max , \min и их местоположение в последовательности из n чисел a_k , индекс $k := 1, n$;

Обозначим:

r - местоположение \max , т.е. $\max = a_r$;

t - местоположение \min , т.е. $\min = a_t$;

Алгоритм поиска \max был рассмотрен в примере 1.10. Добавим в тот алгоритм поиск \min и индексов t, r . В начале принимаем $\max = a_1, \min = a_1$, значит $r = 1, t = 1$ (рис. 2.7).

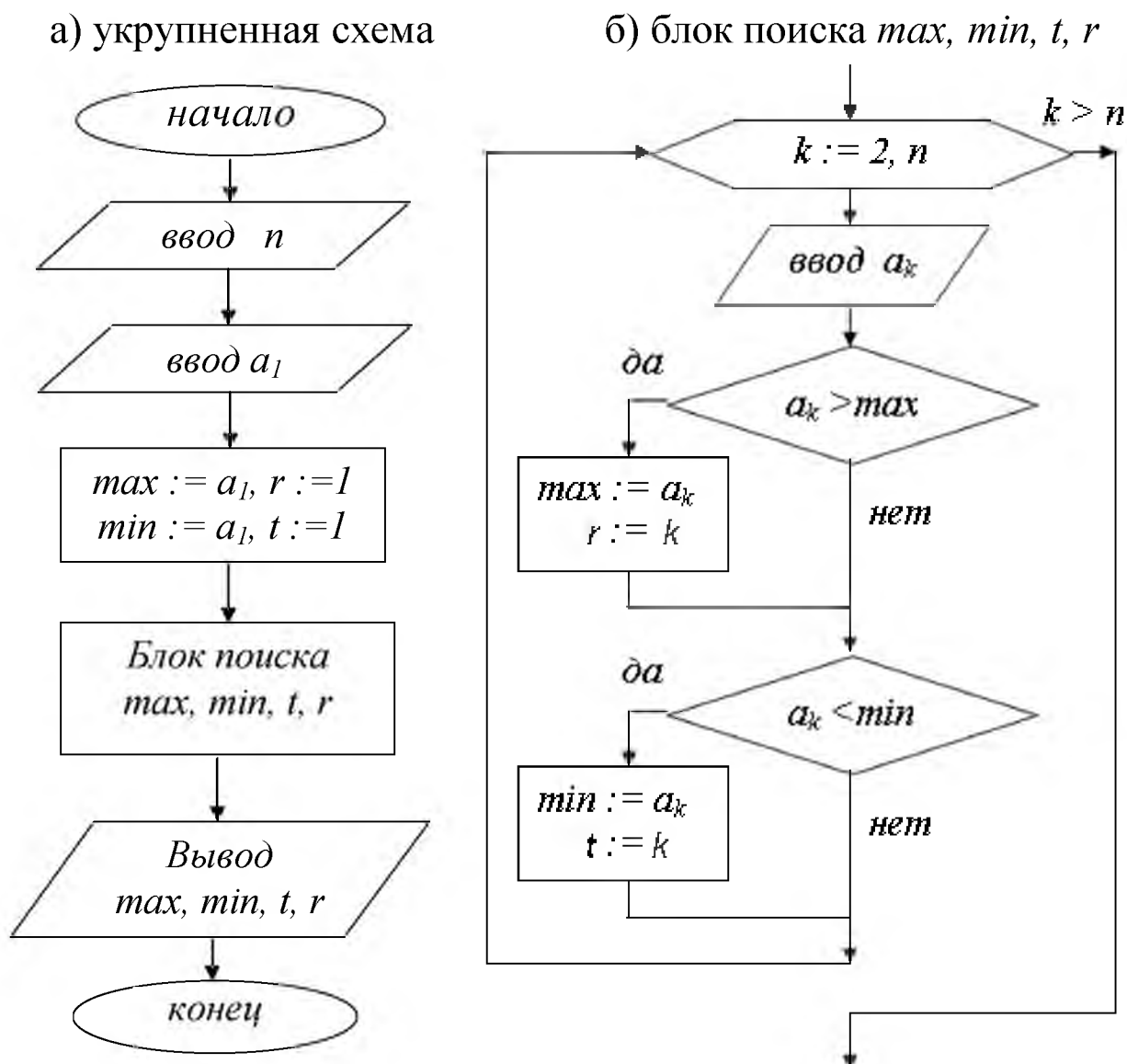


Рисунок 2.7 – Поиск \max, \min, t, r ;

2.7 Сортировка чисел

Сортировка чисел – это упорядочение заданного массива чисел по их значению (по возрастанию или по убыванию). Сортировка достигается перестановкой чисел местами. Рассмотрим два способа сортировки (например, по возрастанию):

- линейная сортировка,
- пузырьковая сортировка.

1. Метод *линейной сортировки* заключается в том, чтобы в массиве найти *min* и его обменять с первым числом. Затем выполняются аналогичные действия с укороченным массивом, кроме первого элемента. Всего таких действий, с поиском *min* и перестановкой чисел, на единицу меньше размера массива. В учебнике [2] метод линейной сортировки назван *методом прямой выборки*.

2. Идея пузырьковой сортировки основана на том, что более «легкий» элемент «всплывает выше» по сравнению с соседним элементом. Сравниваются два соседних числа с конца массива и при необходимости переставляются местами. Самый «легкий» элемент (*min*) оказывается на первом месте. Затем выполняются аналогичные действия с укороченным массивом (кроме первого элемента). Всего таких действий на единицу меньше размера массива.

Пример 2.7. Рассмотрим одномерный массив из n чисел: $a_1, a_2, a_3, \dots, a_n$; или $a = \{a_k; k := 1, n; \}$. Здесь запись $k := 1, n$ означает, что k изменяется от 1 до n , т.е. $k = 1, 2, 3, \dots, n$.

2.7.1 Линейная сортировка (по возрастанию).

Пусть $n=7$ (смотри исходный массив, рисунок 2.8).

1. В строке ищем *min* и его номер ячейки t . В исходной строке $min = a_5 = -7$; ячейка $t=5$.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	
4	8	-5	7	-7	9	6	исходный массив
-7	8	-5	7	4	9	6	(1) этапы
	-5	8	7	4	9	6	(2) преобразований
		4	7	8	9	6	(3) $\downarrow i$
			6	8	9	7	(4) $i = 1, 2, \dots, n-1$
				7	9	8	(5)
					8	9	(6)
-7	-5	4	6	7	8	9	отсортирован

Рисунок 2.8 – Перестановка при линейной сортировке.

2. Делаем перестановку чисел между ячейками 1 и t . Сначала в ячейку $t=5$ запишем первое число, затем в первую ячейку запишем \min .

Для четкого понимания выполняемых действий, примем два следующих обозначения:

- первое – равенство, обозначим символом $=$ (*равно*);
- второе - замена, т.е. присваивание нового значения, обозначим двумя символами $:=$ (двоеточие и равно).

Тогда, перестановка запишется так:

$a_t = a_5 := a_1 = 4$; $a_1 := \min = -7$. Или $a_t := a_1$; $a_1 := \min$.

Данная строка читается так: ячейку с минимальным числом a_t равным a_5 , заменили числом a_1 , равным 4; число a_1 заменили \min , равным -7. Или a_t заменили на a_1 ; a_1 заменили на \min .

В результате получим $a_5 = 4$, $a_1 = -7$ (смотри первый этап преобразования, рисунок 2.8).

3. Аналогично рассмотрим оставшуюся часть массива, т.е. кроме первой ячейки (смотри этапы преобразований):

(2-этап): первый элемент остатка - это $a_2 = 8$;

$\min = a_3 = -5$; ячейка $t = 3$;

перестановка $a_t := a_2$; $a_2 := \min$; результат $a_3 = 8$, $a_2 = -5$;

(3-этап): первый элемент остатка - это $a_3 = 8$,

$\min = a_5 = 4$; ячейка $t = 5$;

перестановка $a_t := a_3$; $a_3 := \min$; результат $a_5 = 8$, $a_3 = 4$;

(4-этап): первый элемент остатка - это $a_4 = 7$,

$\min = a_7 = 6$; ячейка $t = 7$;

перестановка $a_t := a_4$, $a_4 := \min$; результат $a_7 = 7$, $a_4 = 6$;

(5-этап): первый элемент остатка - это $a_5 = 8$,

$\min = a_7 = 7$; ячейка $t = 7$;

перестановка $a_t := a_5$, $a_5 := \min$; результат $a_7 = 8$, $a_5 = 7$;

(6-этап): первый элемент остатка - это $a_6 = 9$,

$\min = a_7 = 8$; ячейка $t = 7$;

перестановка $a_t := a_6$, $a_6 := \min$; результат $a_7 = 9$, $a_6 = 8$;

4. Таким образом, всего $n-1 = 6$ этапов преобразований. На каждом этапе $i = 1, 2, \dots, n-1$ выполняются два действия:

- первое – поиск \min и номера ячейки t ,
- второе – перестановка $a_t := a_i$, $a_i := \min$; (рисунок 2.9).

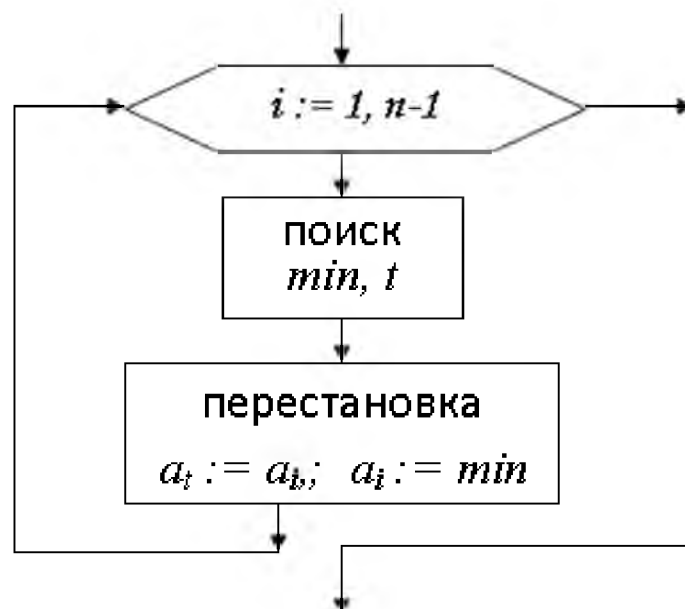


Рисунок 2.9 – Преобразования массива при линейной сортировке.

Подробный алгоритм поиска \min и t рассмотрен в предыдущем примере 2.6 (см. рисунок 2.7). В нашем случае сортировки чисел будет отличие в алгоритме поиска;

связано это с тем, что у нас выполняется перестановка чисел, в процессе которой мы должны обозревать сразу все числа, и поэтому необходимо сначала ввести все числа в виде массива данных. Индекс элемента массива будем указывать в квадратных скобках.

5. Алгоритм линейной сортировки по возрастанию представлен укрупненной блок-схемой на рисунке 2.10 и подробными схемами на рисунке 2.11.

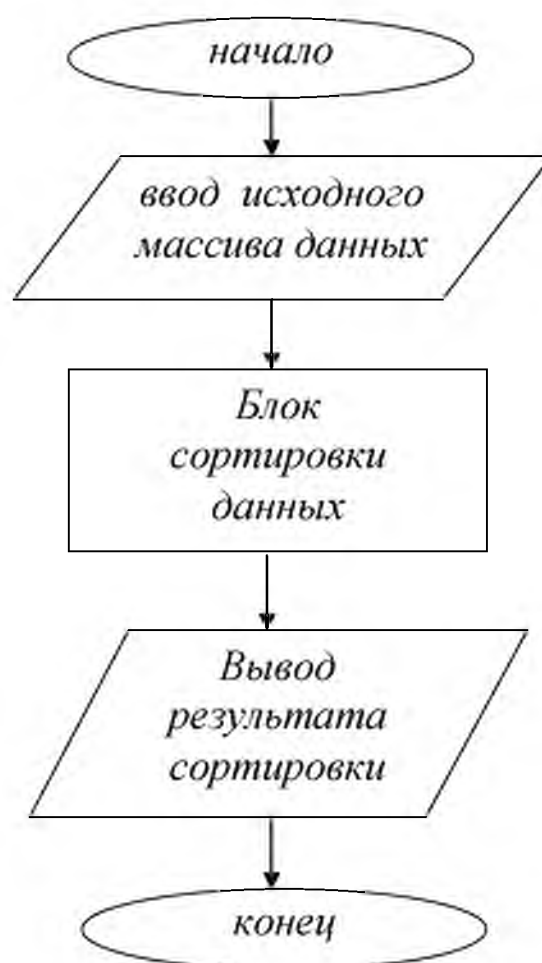


Рисунок 2.10 – Укрупненная блок-схема сортировки.

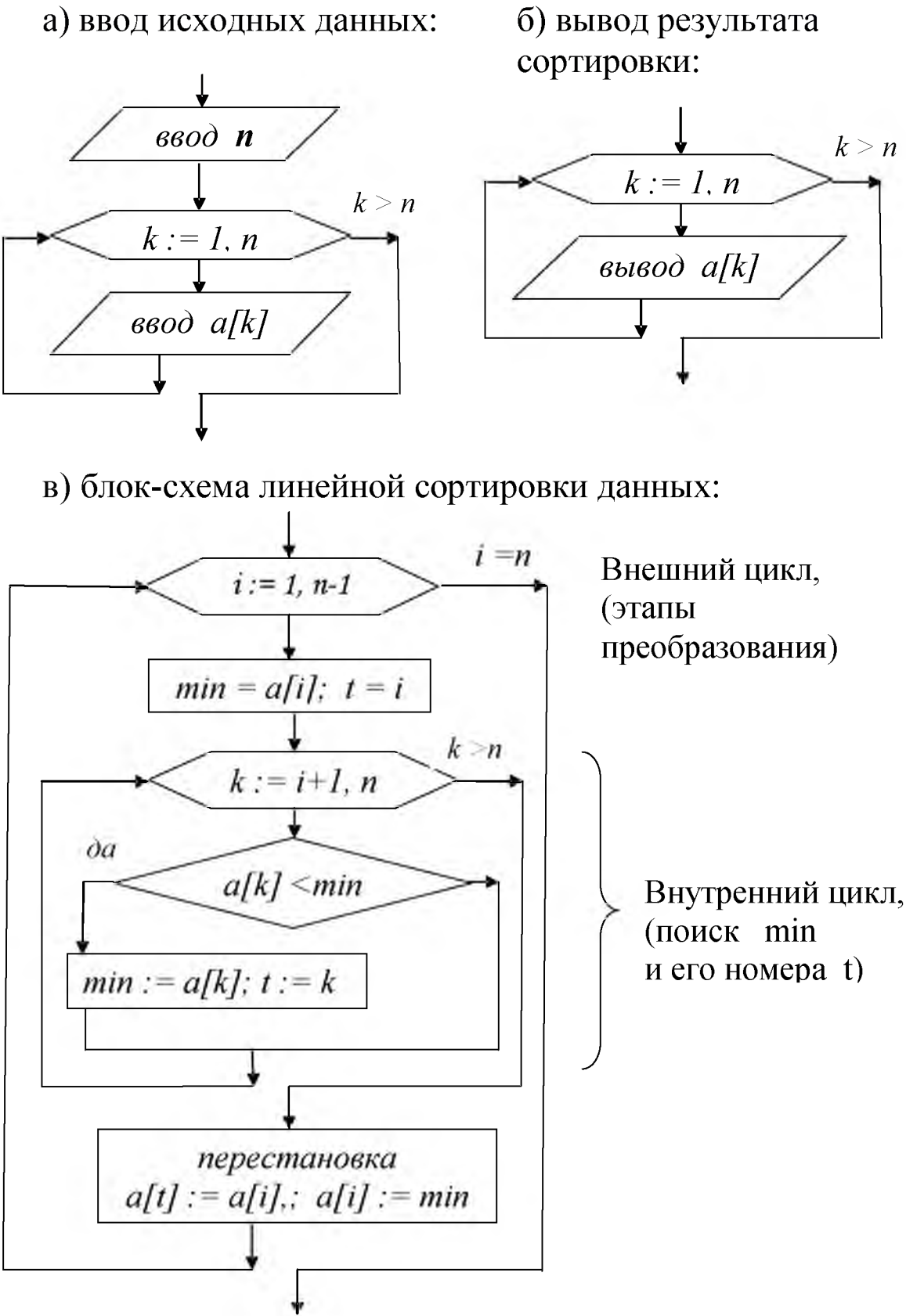


Рисунок 2.11 – Фрагменты схемы сортировки (приложение к рисунку 2.10).

2.7.2 Пузырьковая сортировка (по возрастанию).

Исходные данные те же.

a_1	a_2	a_3	a_4	a_5	a_6	a_7	
4	8	-5	7	-7	9	6	исходный массив

1. С конца массива сравниваются два соседних числа и при необходимости переставляются местами – меньшее значение ставится впереди. Самый «легкий» элемент $min=-7$ оказывается на первом месте.

Перестановка чисел представляет собой «проталкивание легкого элемента налево». Ввиду того, что методика сортировки названа «пузырьковой», а пузырек всплывает наверх, то слово «налево» заменим на «наверх». Этап преобразования можно показать следующей схемой (рисунок 2.12):

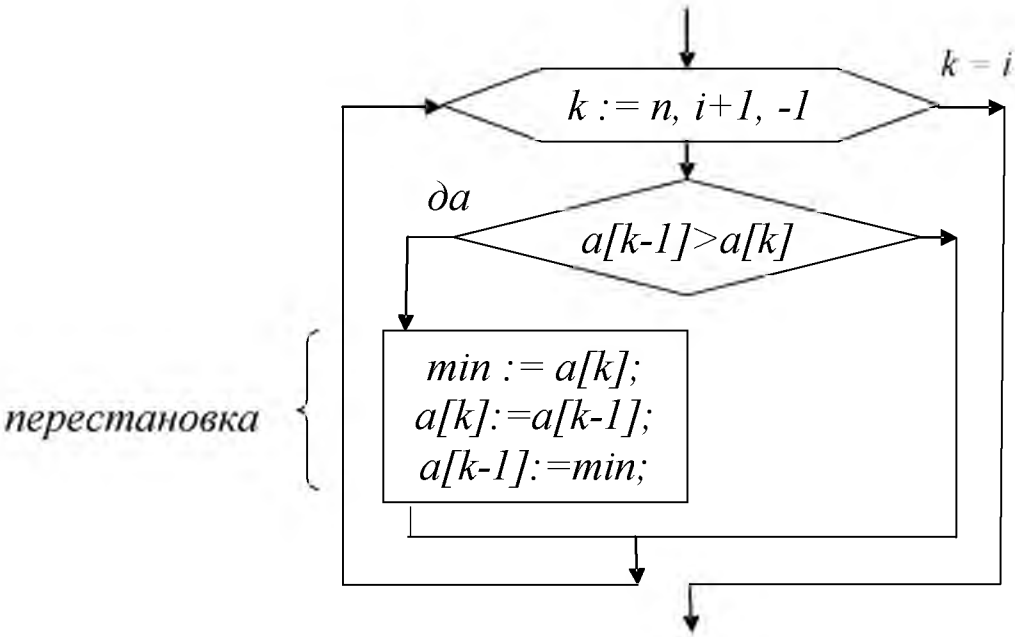


Рисунок 2.12 – Блок-схема циклического «проталкивания легкого элемента наверх».

Здесь $k := n, i+1, -1$ означает, что k изменяется от n до $i+1$ с шагом -1 , т.е. k убывает до значения $i+1$. На первом этапе $i=1$ параметр цикла k принимает значения $7, 6, 5, \dots, 2$.

2. Аналогично рассмотрим оставшуюся часть массива, т.е. кроме первой ячейки (смотри этапы преобразований, рисунок 2.13):

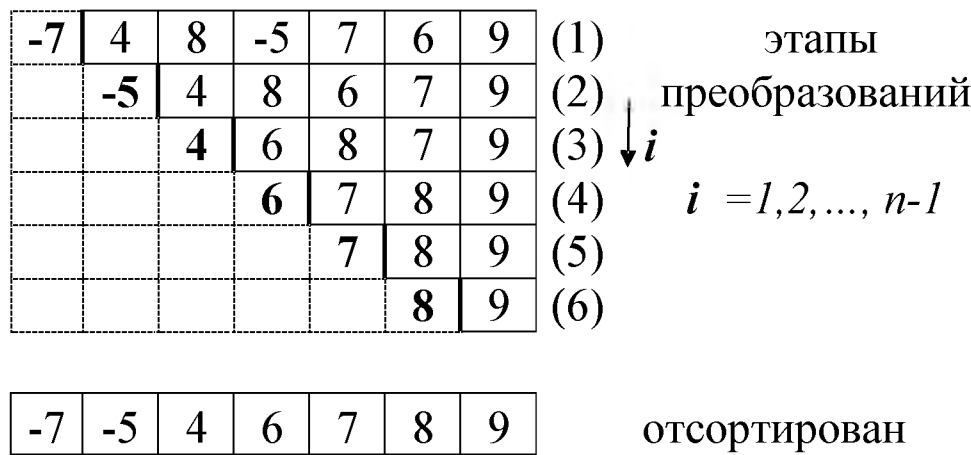


Рисунок 2.13 – Преобразование массива при пузырьковой сортировке.

Алгоритм этапов преобразований можно показать следующей схемой (рисунок 2.14):

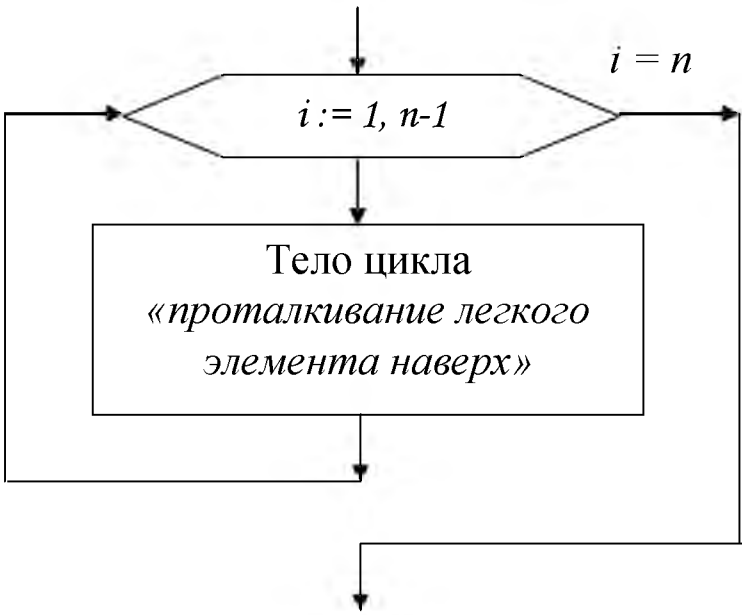


Рисунок 2.14 – Блок-схема этапов преобразований.

3. Окончательный алгоритм сортировки может быть представлен укрупненной схемой (см. рисунок 2.10) и

подробной блок-схемой самой пузырьковой сортировки по возрастанию (рисунок 2.15).

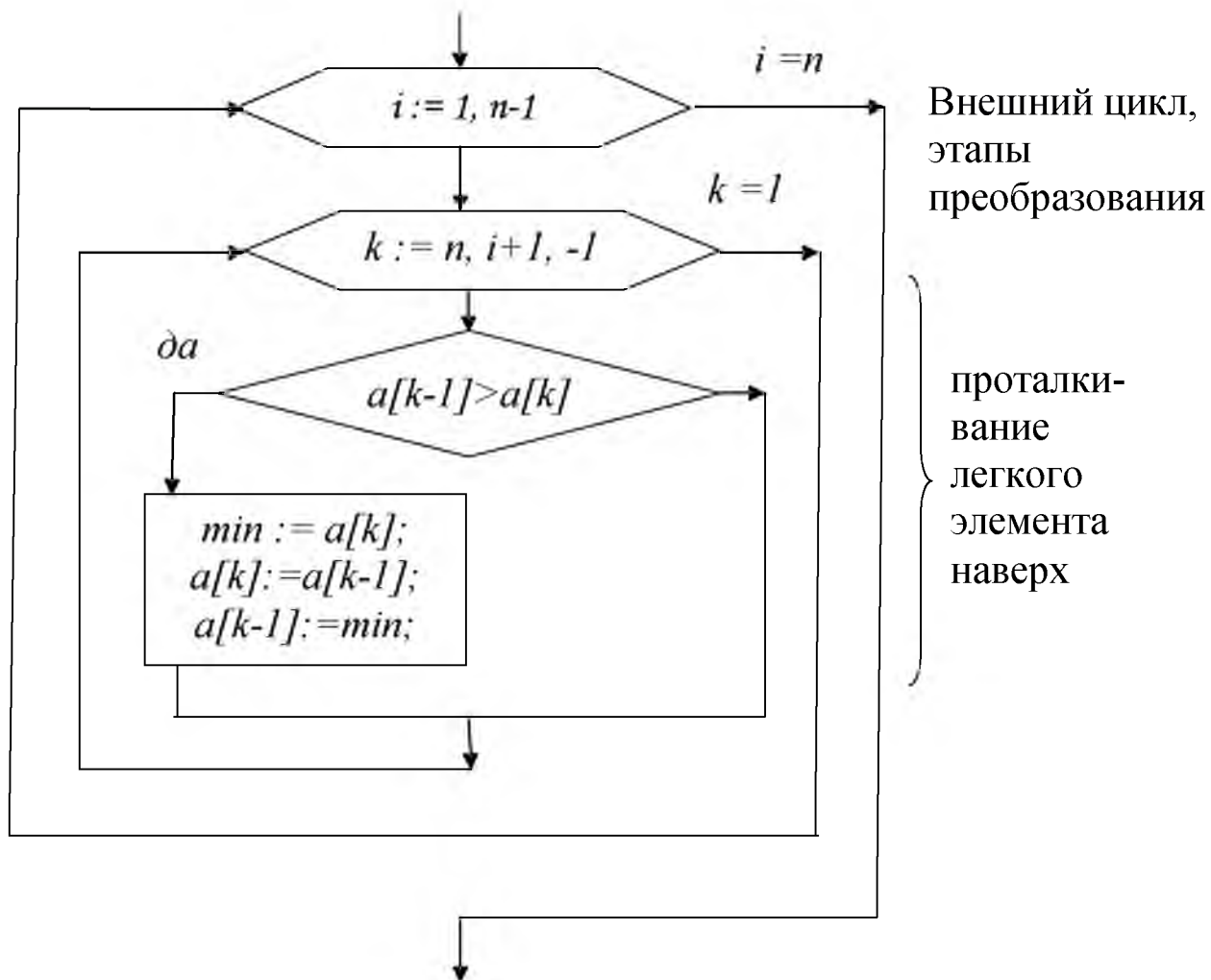


Рисунок 2.15 – Блок-схема пузырьковой сортировки

2.8 Задания для упражнения

Составить блок-схему к следующим задачам.

2.8.1. Даны два одномерных массива чисел. Найти среднее арифметическое значение в каждом массиве.

2.8.2. Задан одномерный массив целых чисел, содержащий несколько нулей. Найти количество нулей, номера первого и последнего нуля в массиве.

2.8.3. Задан одномерный массив целых чисел, содержащий несколько нулей. Удалить из массива нулевые элементы без нарушения порядка их следования.

2.8.4. Задан одномерный массив чисел. Расположить числа в обратном порядке.

2.8.5. Задана функция $y = f(x)$ на отрезке $a \leq x \leq b$. Отрезок разбит на участки с шагом h . Вывести таблицу значений x, y для каждой точки разбиения отрезка.

2.8.6. Задан двумерный массив чисел (матрица размером 4×5). Составить блок-схему ввода заданных чисел в матрицу.

2.8.7. Задан двумерный массив чисел (матрица размером 4×5). Составить блок-схему вывода элементов матрицы в виде таблицы.

2.8.8. Задан двумерный массив чисел (матрица размером 4×5). Составить фрагмент блок-схемы вычисления суммы чисел во второй строке.

2.8.9. Задан двумерный массив чисел (матрица размером 4×5). Составить фрагмент блок-схемы вычисления среднего значения каждой строки.

2.8.10. Задан двумерный массив чисел (матрица размером 4×5). Составить фрагмент блок-схемы нахождения max и его адреса.

ЧАСТЬ 2.

ПРОГРАММИРОВАНИЕ

3 ОСНОВЫ АЛГОРИТМИЧЕСКОГО ЯЗЫКА ПАСКАЛЬ

Программированием называется составление описания алгоритма (хода решения задачи) на специальном алгоритмическом языке. Широко известны следующие алгоритмические языки: Бейсик - Basic, Паскаль - Pascal, Си - C, и др.[10]. Алгоритмический язык является искусственно придуманным (похожим на английский) с ограниченным набором зарезервированных слов и символов. Как в любом языке, в нём следует выделить три составляющие: алфавит, синтаксис и семантику.

- Алфавит – это используемый в языке набор символов и зарезервированных слов.
- Синтаксис – это правило написания предложений.
- Семантика – это осмысление написанного предложения.

Текст, написанный на алгоритмическом языке, называется *листингом* программы, или *исходным кодом* программы. Этот текст затем переводится (транслируется или компилируется) в машинные команды и получается компьютерная программа.

Рассмотрим алгоритмический язык Паскаль, чтобы познакомиться с основами программирования. Рассмотрим лишь основы языка для написания ряда учебных программ. Подробное изложение языка приведено в [4-9].

3.1 Основные понятия

Алфавит языка:

- *a, b, ..., z, A, ..., Z* – английские буквы;
- *0, 1, ..., 9* – цифры;
- *+, -, *, /, =, <, >, (,), [,], {, }, : , ; , _ , ' , . , -* набор символов;
- *а, б, ..., я, А, ..., Я* – русские буквы; используются внутри одинарных кавычек или внутри фигурных скобок;

- *program, begin, end, const, var, real, integer, readln, writeln, if, then, else, for, to, do, while, repeat, until, function, procedure, ...*, и др. - набор зарезервированных слов; это не полный перечень, но нам пока достаточно.

Идентификаторы – это обозначение объектов (переменных или постоянных величин и имён функций и процедур). Любой идентификатор начинается с английской буквы, затем опять буквы или цифры. Например: *a, d, alc, a22, max, min*. Регистр букв (прописная или строчная) не играет роли: *max, MaX, mAX* – обозначают одно и то же. Верхние или нижние индексы в идентификаторах не допускаются.

Структура текста. Текст состоит из строк, длина строки не более 127 символов. Во всем тексте, в том числе в формулах, не допускаются верхние или нижние индексы. Если формула содержит горизонтальную дробную черту, то она заменяется символом слеш (/) – наклонная черта, чтобы запись не выходила за верхний и нижний уровень строки.

Рассмотрим написание чисел и некоторых выражений на языке Паскаль (таблица 3.1).

Таблица 3.1
Написание чисел и выражений на языке Паскаль

обычная запись	на Паскале	комментарий
-25,47	-25.47	дробная часть отделяется точкой
0, 038	0.038 или .038	ноль целых можно не указывать
$\frac{4}{5}$	4/5	запись в пределах высоты строки
$25,47 \cdot 10^4$	25.47E4	10^4 заменяется на E4 без умножения (без точки)

Продолжение таблицы 3.1

Написание чисел и выражений на языке Паскаль

$\frac{a-b}{(x+y)z}$	$(a-b)/((x+y)*z)$	Знаменатель взят в скобки
x^2 ; \sqrt{a}	$x*x$ или $\text{sqr}(x)$; $\text{sqrt}(a)$	Стандартные функции возведения в квадрат и извлечения квадратного корня
$x \in [a, b)$	$(x \geq a) \text{ and } (x < b)$	<i>and</i> - это логическое И

Текст на Паскале начинается со слова *program* и названия программы, затем идет раздел описаний меток, констант, переменных, функций и процедур, затем начинается тело программы со слова *begin* и завершается словом *end*. (*end* с точкой). Каждое предложение **заканчивается точкой с запятой (;)**.

Приведем пример простейшей программы, которая вводит два числа, вычисляет сумму и произведение чисел, определяет максимальное число и выводит на экран результаты.

```

program primer;           {заголовок программы}
var a, b, sum, pr, max : integer; {описание переменных}
begin                     {начало тела программы}
    write('Введите два числа: '); {вывод сообщения на экран}
    readln (a,b);          {ввод данных с клавиатуры}
    sum := a + b;  pr := a*b;
    if a >= b then max :=a else max :=b;
    writeln ('Сумма= ', sum);    {вывод результата расчета}
    writeln ('Произведение=', pr);
    writeln ('Максимум= ', max);
end.      (* конец программы *)

```

Здесь, в листинге программы, внутри фигурных скобок приводится комментарий, который не будет включен в саму

компьютерную программу после трансляции листинга. Комментарий можно также вставлять внутри круглых скобок с звездочкой (см. комментарий рядом с *end*).

3.2 Операторы языка Паскаль

Оператором называется команда, которая выполняет определенное действие. Написание предложения с оператором требует строгого соблюдения синтаксиса (правила записи).

3.2.1 Оператор присваивания

Присваиванием называется вычисление некоторого значения и присваивание его переменной. Оператор присваивания обозначается $:=$ (двоеточие и равно, пишется слитно без пробела). Например: *sum := a + b; k := k+1; x := y; t:=0;* Слева обязательно только одна переменная. Справа выражение или переменная или число. Нельзя слева писать выражение. $a*x + b := 0;$ - это неправильно.

3.2.2 Объявление переменной

Переменные, используемые при решении задачи, описываются перед телом программы (перед *begin*). Для объявления переменных используется слово *var*. Например,

var a, b, sum, pr, max : integer;

Конструкция записи (синтаксис) следующая: 1) вначале слово *var*; 2) затем список переменных, разделенных запятыми; 3) двоеточие; 4) тип данных, 5) точка с запятой.

Тип данных — это возможные значения, которые может принимать переменная. Наиболее часто встречаются следующие типы данных: *integer* — целые числа; *real* — вещественные, т.е. числа с дробной частью; *string* — строка, т.е. набор букв, символов; *array* — массив данных.

3.2.3 Ввод данных с клавиатуры

Исходные данные для решения задачи вводятся через клавиатуру. Переменным надо присвоить начальные значения

(или исходные данные), используя зарезервированное слово *readln*. Синтаксис:

readln (список переменных);

Например, *readln (a,b);* Список внутри скобок разделяется запятыми.

Примечание: данный оператор реализует блок ввода данных (параллелограмм) в блок-схеме.

3.2.4 Вывод данных на экран

По ходу решения задачи, переменные принимают конкретные значения, которые можно вывести на экран.

Синтаксис этой процедуры допускает следующие формы записи:

1) ***writeln (список данных);***

Например, *writeln (a,b);* Элементом списка может быть не только переменная, но и текст, заключенный в апострофы (одинарные кавычки), например, *writeln ('Сумма= ' , sum);* После вывода всех данных, курсор на экране переходит на начало следующей строки.

2) ***write (данные);***

После вывода данных, курсор на экране остается в конце строки.

3) ***writeln;*** Вывод пустой строки.

4) ***Форматированный вывод.*** Рассмотрим программу, которая дважды выводит на экран значения 4-х переменных:

```
a= -234,3; b= 0,00567; i= 8; j= -9;
program primer1;
var a,b:real; i,j:integer;
begin
  a:= -234.3; b:=0.00567; i:=8; j:= -9;
  writeln (a, b, i, j);
  writeln (a:7:2, b:8:3; i:4, j:6);
end.
```

На экран будут выведены числа оператором `writeln (a, b, i, j)` в следующем виде:

17 знакомест (ячеек, позиций)														17 знакомест																						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	1	1	2
-	2	.	3	4	3	0	0	0	0	0	0	0	E	+	0	2		5	.	6	7	0	0	0	0	0	0	0	0	E	-	0	3	8	-	9
<i>a</i>																	<i>b</i>																	<i>i</i>	<i>j</i>	

Здесь, в таблице в первой и второй строках показано, что для чисел *a*, *b* отведены по 17 позиций, для *i*, *j* – одна и две позиции.

Числа *a*, *b* на экран выведены в экспоненциальной форме:

$$a = -2.3430000000E+02 = -2,343 \cdot 10^2 = -234,3;$$

$$b = 5.6700000000E-03 = 5,67 \cdot 10^{-3} = 0,00567;$$

$$i = 8; j = -9;$$

Оператор `writeln (a:7:2, b:8:3; i:4, j:6)` выводит числа в форматированном виде:

7 знакомест							8 знакомест								4 знакомест				6 знакомест					
1	2	3	4	5	6	7	1	2	3	4	5	6	7	8	1	2	3	4	1	2	3	4	5	6
-	2	3	4	.	3	0				0	.	0	0	6				8					-	9
<i>a</i>							<i>b</i>								<i>i</i>				<i>j</i>					

Здесь, для числа *a* выделено 7 позиций, из них 2 позиции для дробной части; для числа *b* выделено 8 позиций, из них 3 позиции для дробной части (дробная часть 0,00567 округляется до 0,006, чтобы уместить в 3 позиции):

$$a = -234.30; b = 0.006; i = 8; j = -9;$$

Для форматированного вида следует указать количество позиций для числа. Синтаксис следующий:

Переменная : ВсегоПозиций : ПозДляДробнойЧасти.

Примечание: оператор `writeln()` реализует блок вывода данных (параллелограмм) в блок-схеме.

3.2.5 Операторы ветвления

- 1) Оператор структуры «если-то» (смотри 1.5.2.1).

Синтаксис:

if условие *then* оператор-действие;

Семантика: Если (*if*) условие=истина (да), то (*then*) выполняется оператор-действие. Например,

if $a > b$ *then* $max := a$; Если $a > b$, то $max := a$;

- 2) Оператор структуры «если-то-иначе» (смотри 1.5.2.2).

Синтаксис:

if условие *then* оператор1 *else* оператор2;

Перед *else* не ставится точка с запятой.

Семантика: Если (*if*) условие=истина, то (*then*) выполняется оператор1, иначе (*else*) выполнится оператор2. Например,

if $a > b$ *then* $max := a$ *else* $max := b$;

Если $a > b$, то $max := a$, иначе $max := b$;

- 3) Оператор структуры «выбор» (смотри 1.5.2.3).

Синтаксис:

case переменная *of*
 список1 : действие1;
 список2 : действие2;
 ...
 списокN : действиеN;
end;

Семантика: Если значение *переменной* совпадает с одним из элементов *списка*, то выполнится соответствующее *действие*.

Например:

case h *of*
 3,4,5 : *writeln* ('весна');
 6,7,8 : *writeln* ('лето');
 9,10,11 : *writeln* ('осень');
 12,1,2 : *writeln* ('зима');
end;

К примеру, если $h = 8$, то вывод сообщения «лето».

4) Оператор структуры «выбор-иначе» (смотри 1.5.2.4).

Синтаксис:

```
case переменная of  
список1 : действие1;  
.....  
списокN : действиеN;  
else : действиеN2;  
end;
```

Семантика: Если значение *переменной* совпадает с одним из элементов *списка*, то выполняется соответствующее *действие*, иначе выполняется *действиеN2*.

Например: *case h of*

```
3,4,5 : writeln ('весна');  
6,7,8 : writeln ('лето');  
9,10,11 : writeln ('осень');  
12,1,2 : writeln ('зима');  
else : writeln ('ошибка месяца');  
end;
```

К примеру, если $1 > h > 12$, то вывод сообщения «*ошибка месяца*».

5) Оператор безусловного перехода.

Конструкция этого оператора состоит из трех частей:

- в разделе описаний служебным словом ***label*** указывается идентификатор метки, например, ***label m1***;
- в теле программы служебным словом ***goto m1*** осуществляется переход на требуемое место;
- требуемое место помечается меткой ***m1***.

Пример, использующий оператор безусловного перехода, приведен в пункте 4.1.1.

3.2.6 Операторы цикла

Для организации циклического процесса вычислений имеются три конструкции:

- 1) Цикл с предУсловием (см. пункт 1.5.3.1).

Синтаксис:

while условие *do* телоЦикла;

Семантика: **Пока** условие=истина, **выполняется** телоЦикла; если условие=ложь, то выход из цикла.

Например (рис. 2.6,в),

while $x \leq 14$ *do* телоZikla;

- 2) Цикл с постУсловием (см. пункт 1.5.3.2).

Синтаксис:

Repeat телоЦикла *until* условие;

Семантика: **Повторять** телоЦикла, **пока** условие=ложь; если условие=истина, то выход из цикла.

Например (рис. 2.5,а),

repeat телоZikla *until* $x > 14$;

- 3) Цикл с параметром (см. пункт 1.5.3.3).

Синтаксис:

for $k := k1$ *to* kN *do* телоЦикла;

Здесь k – переменная, параметр цикла; $k1$ – начальное значение параметра; kN – конечное значение параметра; Величины k , $k1$, kN должны быть целого типа.

Семантика: **Для** параметра цикла k , который изменяется **от** $k1$ **до** kN с шагом=1, **выполняется** телоЦикла. При $k > kN$ будет выход из цикла.

Например,

for $k := 1$ *to* n *do* телоZikla;

Здесь, часть конструкции, *for k:=1 to n* заставляет принимать значения $k=1,2,3,\dots,n$ и каждый раз выполнять *teloZikla*.

Если параметр цикла убывает с шагом = -1, то синтаксис будет следующим

for k:= kN downto k1 do телоЦикла;

Примечание. Внутри тела цикла запрещается изменять значения k , $k1$, kN .

3.2.7 Типы данных (продолжение)

Кроме распространенных числовых типов данных *integer* – целые числа и *real* – вещественные, рассмотрим еще два типа.

3.2.7.1 Тип данных строка – *string*.

Переменная этого типа может содержать не более 255 символов в строке. Если заранее известно, что такие длинные строки не будут использоваться, то можно указать длину строки в квадратных скобках: *string[40]*.

Например, *var fam, name, fio : string[40];*

fam := 'Иванов'; name := 'Вася'; fio := fam + name;

Значения строковых данных необходимо заключать в апострофы (одинарные кавычки). Строковые данные можно складывать, например, в переменной *fio* будет текст «ИвановВася». Чтобы отделить слова, надо вставить пробел и написать *fio := fam + ' ' + name;* Здесь между апострофами находится один пробел. Тогда в переменной *fio* будет текст «Иванов Вася».

3.2.7.2 Тип данных массив – *array*.

Массивом называется последовательность однотипных данных. Элементами массива могут быть другие типы данных: либо только *integer*, либо только *real*, либо только *string*, и другие.

Синтаксис типа *array* состоит из 4 частей: 1) слово *array*, 2) диапазон индексов в квадратных скобках, 3) слово *of*, 4) тип элементов, 5) точка с запятой.

Например,

```
var v : array [1..50] of integer;
    w : array [1..30] of real;
    a : array [1..25, 1..4] of integer;
```

Диапазон индексов обозначается двумя точками между крайними значениями. Здесь переменные

v — описывает одномерный массив из 50 целых чисел; элементами массива являются *v[1]*, *v[2]*, ..., *v[50]*;

w — описывает одномерный массив из 30 вещественных чисел; элементами массива являются *w[1]*, *w[2]*, ..., *w[30]*;

a — описывает двумерный массив (матрицу) из 25 строк и 4 столбцов; элементами массива являются целые числа. Элемент массива обозначается *a[i,j]*, где *i:=1,25* — номер строки, *j:=1,4* — номер столбца. Или

```
a[1,1], a[1,2], a[1,3], a[1,4],
a[2,1], a[2,2], a[2,3], a[2,4],
.....
a[25,1], a[25,2], a[25,3], a[25,4].
```

Например, таким массивом можно описать оценки студентов в сессию (25 студентов, 4 дисциплины). К примеру, *a[7,3]:=5* будет означать, что 7-й по списку студент получил на 3-м экзамене оценку 5. Более подробно о типе массив и других типах данных можно узнать в [2].

3.2.8 Операторные скобки

В структурах ветвления или в циклических структурах бывает необходимо выполнить не одно действие, а группу действий (см. рисунки 2.1, 2.2). В таких случаях группу операторов надо объединить в блок, заключив их в операторные скобки. Роль операторных скобок выполняют слова *begin* и *end*. Если нужную группу операторов не объединять в блок, то будет выполняться только первый оператор группы, что приведёт к ошибочным результатам.

Например, фрагмент листинга к задаче вычисления значений функции (см. рисунок 2.6,г):

```
for k:=1 to n do
  begin
    if x<=13 then y:=5+x else y:=x*x;
    writeln (x, y);
    x:= x+0.2;
  end;
```

3.3 Функции и процедуры

Функции и процедуры представляют собой обособленный фрагмент программы, оформленные в виде отдельного блока с именем, которые выполняют определенные вычисления. Функции и процедуры, как правило, имеют аргументы. Разница между функцией и процедурой следующая: функция вычисляет значение одной величины и возвращает её через своё имя; процедура вычисляет значения нескольких величин и возвращает их через аргументы или через глобальные переменные. Глобальная переменная объявляется вне процедуры и используется внутри процедуры.

В языке Паскаль имеется большой набор стандартных функций, например, $\sin(x)$, $\cos(x)$,

$\text{sqrt}(x)$ – извлечение квадратного корня из x ;

$\text{sqr}(x)$ – возведение x в квадрат; и другие функции, а также процедуры, например,

$\text{gotoXY}(x,y)$ – перемещает курсор в заданную позицию экрана.

3.3.1 Функции

Синтаксис:

```
function fname (arg1 : tip1, arg2 : tip2,...) : tipRez;
begin
  операторы тела функции;
  fname :=***;
end;
```


Здесь *fname* – имя функции; *arg1*, *arg2* – аргументы; *tip1*, *tip2* – тип данных аргументов; *tipRez* – тип данных результата вычислений, т.е. тип функции. Тело функции заключено в операторные скобки *begin* и *end* с точкой-запятой. В конце тела функции обязательно надо присвоить имени результат вычисления.

Например, функция вычисления максимального из двух чисел:

```
function max2 (a, b : integer) : integer;
begin
    max2 := a; if b > max2 then max2 := b;
end;
```

Пример функции вычисления максимального из трёх чисел:

```
function max3 (a, b, c : integer) : integer;
begin
    max3 := a;
    if b > max3 then max3 := b;
    if c > max3 then max3 := c;
end;
```

3.3.2 Процедуры

Синтаксис:

```
procedure fproc (arg1 : tip1, arg2 : tip2, ...) ;
begin
    операторы тела процедуры;
end;
```

Здесь *fproc* – имя процедуры; *arg1*, *arg2* – аргументы; *tip1*, *tip2* – тип данных аргументов. Тело процедуры заключено в операторные скобки *begin* и *end* с точкой-запятой.

3.4 Замечание о структуре текста на языке Паскаль

Текст на языке Паскаль состоит из строк, в каждой из которых не более 127 символов. Например, задача, приведенная в пункте 3.1, «ввести два числа, вычислить сумму и произведение чисел, определить максимальное число и вывести на экран результаты», может быть решена следующей последовательностью операторов:

```
program primer; var a,b,sum,pr,max:integer;begin readln (a,b);  
sum:=a+b; pr:=a*b; if a>=b then max:=a else max:=b;  
writeln (sum,pr,max);end.
```

Этот текст на языке Паскаль не содержит ошибок и компьютер выдаст правильный результат. Однако такой текст не удобен, во-первых, для восприятия при чтении, во-вторых, для практического применения при расчете на компьютере. Для устранения указанных неудобств, текст на языке Паскаль располагают более короткими строками, используют отступы в строках, некоторые фрагменты текста снабжают комментариями, перед вводом данных добавляют оператор вывода подсказки, вывод результатов снабжают поясняющим текстом. Числовые данные надо выводить в форматированном виде. В конце пункта 3.1 приведен именно такой рекомендуемый текст на языке Паскаль для решения указанной задачи.

4 ПРИМЕРЫ ПРОГРАММ НА ЯЗЫКЕ ПАСКАЛЬ

Приведены примеры программ для задач, рассмотренных в главах 1,2 при составлении блок-схем. Если имеется блок-схема, то исходный код программы пишется очень легко; надо каждый блок описать языком программирования.

4.1 Задача о временах года

При заданном номере месяца h , указать время года. Времена года это весна, лето, осень, зима. При неправильно заданном месяце, указать на ошибку (см. пример 1.1, рисунок 1.1).

4.1.1 Решение задачи по схеме рис.1.1:

```
program primer4_1a;  
label m1;  
var h:integer;  
begin  
  write ('Введите h= '); readln(h);  
  if (h>=3) and (h<=5) then  
    begin writeln('весна'); goto m1; end;  
  if (h>=6) and (h<=8) then  
    begin writeln('лето'); goto m1; end;  
  if (h>=9) and (h<=11) then  
    begin writeln('осень'); goto m1; end;  
  if (h=12) or (h=1) or (h=2) then  
    writeln('зима') else writeln('ошибка месяца');  
m1: end.
```

4.1.2 Решение задачи без оператора goto :

```
program primer4_1b;  
var h:integer;
```

```

begin
  write ('Введите h= '); readln(h);
  if (h>=3) and (h<=5) then writeln ('весна');
  if (h>=6) and (h<=8) then writeln ('лето');
  if (h>=9) and (h<=11) then writeln ('осень');
  if (h=12) or (h=1) or (h=2) then writeln('зима');
  if (h<1) or (h>12) then writeln('ошибка месяца');
end.

```

4.1.3 Решение задачи с оператором «выбор-иначе»:

```

program primer4_1c;
var h:integer;
begin
  write ('Введите h= '); readln(h);
  case h of
    3,4,5   : writeln('весна');
    6,7,8   : writeln('лето');
    9,10,11: writeln('осень');
    12,1,2  : writeln('зима');
    else    : writeln('ошибка месяца');
  end;
end.

```

На экран будут выведены
результаты в следующем виде:

Введите h= 5 весна Введите h= 9 осень Введите h= 14 ошибка месяца
--

4.2 Квадратное уравнение

Блок-схема решения приведена на рисунке 2.1 (см. пример 2.1).

```

program primer4_2;
var a,b,c,d,x1,x2:real;
begin
  writeln('Введите a,b,c= '); readln(a,b,c);
  d:= b*b-4*a*c;

```

```

if d>=0 then
  begin
    x1:=(-b+sqrt(d))/(2*a);
    x2:=(-b-sqrt(d))/(2*a);
    writeln('x1= ',x1:7:3);
    writeln('x2= ',x2:7:3);
  end
else writeln('Корней нет');
end.

```

Например:

Введите a,b,c= 1 -5 6 x1= 3.000 x2= 2.000 Введите a,b,c= 1 2 3 Корней нет

4.3 Суммирование чисел

4.3.1 Сумма из *n* заданных чисел

(см. пример 1.2, рисунок 1.2).

```

program primer4_3a;
var a,S:real; n,k:integer;
begin
  write('Введите n='); readln(n);
  S:=0;
  for k:=1 to n do
    begin
      write('a',k,'='); readln(a);
      S:=S+a;
    end;
  writeln('S =', S:6:2);
end.

```

Например:

Введите n=5 a1= 3 a2= 6 a3= -5 a4= -2 a5= 10 S = 12.00
--

4.3.2 Выделить сумму отрицательных чисел

Найти сумму всех чисел S , сумму отрицательных чисел $S1$ и их количество t , сумму положительных чисел $S2$ и их количество p (см. пример 2.2, рисунок 2.2).

```
program primer4_3b;
var a,S,S1,S2:real; t,p,n,k:integer;
begin
  write('Введите n='); readln(n);
  S:=0; S1:=0; t:=0;
  for k:=1 to n do
    begin
      write('a',k,'='); readln(a);
      S:=S+a;
      if a<0 then begin S1:=S1+a; t:=t+1; end;
    end;
  writeln('S =', S:6:2);
  writeln('S1=',S1:6:2,' t=',t);
  writeln('S2=',S-S1:6:2,' p=',n-t);
end.
```

Например:

Введите n=5
a1= 3
a2= 6
a3= -5
a4= -2
a5= 10
S = 12.00
S1= -7.00 t=2
S2= 19.00 p=3

4.3.3 Суммирование убывающего ряда
(см. пример 2.3, рисунок 2.3).

```
program primer4_3c;
const c=0.1;
label zikl;
var a,S:real; k:integer;
```

```

begin
  k:=0; S:=0;
  zikl: k:=k+1;
  a:=5*k/(3+k*k*k);
  if a>c then begin S:=S+a; goto zikl; end;
  writeln('S=',S:5:2);
  writeln('k=',k-1);
end.

```

Результаты расчета: S= 3,39; n=7.

Если перед оператором `k:=0` вставить оператор `writeln('k a S')`, а после оператора `S:=S+a` вставить оператор `writeln(k,a:7:3,S:7:2)`, то на экран будут выведены значения слагаемых и соответствующая сумма.

k	a	S
1	1.250	1.25
2	0.909	2.16
3	0.500	2.66
4	0.299	2.96
5	0.195	3.15
6	0.137	3.29
7	0.101	3.39
S= 3.39		
n=7		

4.3.4 Формирование новой последовательности

Заданы две последовательности $a_k, b_k, k := 1, n$. Вычислить элементы третьей последовательности по формуле $c_k = a_k - 2b_k$ (см. пример 2.4, рисунок 2.4).

```

program primer4_3d;
var a,b,c:real; n,k:integer;
begin
  write(', Введите n='); readln(n);
  for k:=1 to n do
    begin
      write('a',k,'= '); readln(a);
      write('b',k,'= '); readln(b);
      c:=a-2*b;
      writeln('c',k,c:6:2);
    end;
  readln;
end.

```

Введите n=4
a1= 8
b1= 2
c1 4.00
a2= 16.5
b2= 3.65
c2 9.20
a3= 9
b3= 4
c3 1.00
a4= 11
b4= 3
c4 5.00

4.4 Задача о пробежке

Расстояние пробежки в 1-й день S_1 , коэффициент увеличения расстояния в следующие дни p . Через сколько дней будет достигнуто расстояние S_n ? (см. пример 1.3, рисунок 1.3).

```
program primer4_4;
var S1,p,Sn,S : real; k :integer;
begin
  write('Введите S1= '); readln(S1);
  write('Введите p= '); readln(p);
  write('Введите Sn= '); readln(Sn);
  k:=1; S:=S1;
  repeat k:=k+1; S:=S+p*S; until S >= Sn;
  writeln('n= ',k);
end.
```

Пример расчета:

Введите S1=	500
Введите p=	0.1
Введите Sn=	2000
n=	16

Если перед словом **until** вставить оператор **writeln(k:2,S:9:1)**, то будут выведены дни и расстояния.

2	550.0
3	605.0
4	665.5
5	732.0
6	805.3
7	885.8
8	974.4
9	1071.8
10	1179.0
11	1296.9
12	1426.6
13	1569.2
14	1726.1
15	1898.7
16	2088.6

4.5 Вычисление значений функции

Вычислить значения функции

$$y = \begin{cases} 5 + x, & \text{если } x \leq 13 \\ x \cdot x, & \text{если } x > 13 \end{cases}$$

на отрезке $x = [12; 14]$ с шагом 0,2 (см. пример 2.5, рисунки 2.5, 2.6).


```

program primer4_5a; {цикл с предУсловием}
var x,y:real;
procedure teloZikla;
begin
  if x<=13 then y:=5+x else y:=x*x;
  writeln(x:4:1, y:9:1);
  x:= x+0.2;
end;
begin
  writeln('  x          y');
  writeln('-----');
  x:=12;
  while x<=14 do teloZikla;
end.

```

x	y
12.0	17.0
12.2	17.2
12.4	17.4
12.6	17.6
12.8	17.8
13.0	18.0
13.2	174.2
13.4	179.6
13.6	185.0
13.8	190.4
14.0	196.0

На экран будет выведено следующие:

Если строку с оператором **while** заменить на строку
 repeat teloZikla until x>14;
 то будет реализован цикл с постУсловием.

```

program primer4_5b; {цикл с параметром}
var x,y:real; k,n:integer;
procedure teloZikla;
begin
  if x<=13 then y:=5+x else y:=x*x;
  writeln(x:4:1, y:9:1);
  x:= x+0.2;
end;
begin
  writeln('  x          y');
  writeln('-----');
  x:=12;
  n:=round((14-12)/0.2)+1;
  for k:=1 to n do teloZikla;
end.

```

4.6 Поиск максимального числа

4.6.1 Нахождения максимального числа из двух чисел.

```
program primer4_6a;    {блок-схема на рисунке 1.13}
var a,b,max :integer;
begin
  write('Введите a= '); readln(a);
  write('Введите b= '); readln(b);
  if a>b then max:=a else max:=b;
  writeln('max = ',max);
end.
```

Если строку с оператором ветвления заменить строкой
max:=a; if b>max then max:=b;
то получим реализацию по блок-схеме на рисунке 1.16.

4.6.2 Нахождения максимального числа из трех чисел.

```
program primer4_6b;    {блок-схема на рисунке 1.14}
var a,b,c,max :integer;
begin
  write('Введите a= '); readln(a);
  write('Введите b= '); readln(b);
  write('Введите c= '); readln(c);
  if a>b then
    begin if a>c then max:=a else max:=c; end
  else
    begin if b>c then max:=b else max:=c; end;
  writeln('max = ',max);
end.
```

Если строки с оператором if-then-else заменить на
max:=a;
if b>max then max:=b;
if c>max then max:=c;

то получим реализацию по блок-схеме на рисунке 1.17 (эффективный алгоритм).

4.6.3 Поиск *max* из *n* чисел

```

program primer4_6c; {блок-схема на рисунке 1.18}
var a,max :real; k,n :integer;
begin
  write('Введите n='); readln(n);
  write('a1= '); readln(a);
  max:=a;
  for k:=2 to n do
    begin write('a',k,'= '); readln(a);
      if a>max then max:=a;
    end;
  writeln('max = ',max:6:2);
end.

```

Пример с результатами расчета.

Введите n=8
a1= 3
a2= -4
a3= 5
a4= -2
a5= 12
a6= 6
a7= 9
a8= 7
max = 12.00

4.7 Поиск *max* и *min*

Найти *max*, *min* и их местоположение в последовательности из *n* чисел. Обозначения: $max = a_r$; $min = a_t$; (см. пример 2.6).

```

program primer4_7; {блок-схема на рисунке 2.7}
var a,k,n, max,r, min,t :integer;
begin
  write('Введите n='); readln(n);
  write('a1= '); readln(a);
  max:=a; r:=1; min:=a; t:=1;
  for k:=2 to n do
    begin
      write('a',k,'= '); readln(a);
      if a>max then begin max:=a; r:=k; end;

```

```

    if a<min then begin min:=a; t:=k; end;
  end;
  writeln('max = ',max, ' r= ',r);
  writeln('min = ',min, ' t= ',t);
end.

```

Пример с результатами расчета:

Введите n=8
a1= 3
a2= -4
a3= 5
a4= -2
a5= 12
a6= 6
a7= 9
a8= 7
max = 12 r= 5
min = -4 t= 2

4.8 Сортировка чисел

4.8.1 Линейная сортировка по возрастанию

(см. пример 2.7.1, рисунки 2.10, 2.11).

```

program sort1;
var a:array[1..100] of integer;
    i,k,t,n,min :integer;
begin
  (* ВВОД ИСХОДНЫХ ДАННЫХ *)
  write('Кол-во чисел n= '); readln(n);
  for i:=1 to n do
    begin write('a',i,'= '); readln(a[i]); end;
  (* ВЫВОД ИСХОДНЫХ ДАННЫХ *)
  writeln; writeln('Исходный массив:');
  for i:=1 to n do write(a[i]:5); writeln;
  (* ВНЕШНИЙ ЦИКЛ *)
  for i:=1 to n-1 do
    begin
      min:=a[i]; t:=i;
      (* ВНУТРЕННИЙ ЦИКЛ *)
      for k:=i+1 to n do
        if a[k]<min then
          begin min:=a[k]; t:=k; end;
      (* ПЕРЕСТАНОВКА *)
      a[t]:=a[i]; a[i]:=min;
    end;
  end;

```

```
    (* вывод результата сортировки *)  
    writeln; writeln('Отсортированный массив:');  
    for i:=1 to n do write(a[i]:5); writeln;  
end.
```

4.8.2 Пузырьковая сортировка по возрастанию (см. пример 2.7.2, рисунки 2.10, 2.15).

```
program sort2;  
var a:array[1..100] of integer;  
    i,k,t,n,min :integer;  
begin  
    (* ввод исходных данных *)  
    write('Кол-во чисел n= '); readln(n);  
    for i:=1 to n do  
        begin write('a',i,'= '); readln(a[i]); end;  
    (* вывод исходных данных *)  
    writeln; writeln('Исходный массив:');  
    for i:=1 to n do write(a[i]:5); writeln;  
    (* внешний цикл *)  
    for i:=1 to n-1 do  
        for k:=n downto i+1 do  
            if a[k-1]>a[k] then  
                begin  
                    min:=a[k]; a[k]:=a[k-1]; a[k-1]:=min;  
                end;  
        (* вывод результата сортировки *)  
        writeln; writeln('Отсортированный массив:');  
        for i:=1 to n do write(a[i]:5); writeln;  
    end.
```

Пример расчета:

Количество чисел n= 9									
a[1]=	15								
a[2]=	3								
a[3]=	-7								
a[4]=	0								
a[5]=	-4								
a[6]=	2								
a[7]=	1								
a[8]=	10								
a[9]=	9								
Исходный массив:									
15	3	-7	0	-4	2	1	10	9	
Отсортированный массив:									
-7	-4	0	1	2	3	9	10	15	

4.9 Двумерный массив (матрица)

Для матрицы размером $n \cdot m$ выполнить следующие действия:

- 1) ввод элементов (a_{ij});
- 2) вывод элементов в виде матрицы (таблицы);
- 3) вычислить сумму всех элементов (S);
- 4) вычислить сумму элементов первого столбца (S_1);
- 5) вычислить сумму элементов второй строки (S_2);
- 6) найти максимальный элемент и его индексы (max, t, p);
- 7) найти минимальный элемент второй строки и его индексы (min_2, g, h).

```

program matrica;
const n=4; m=5;
var a:array[1..n,1..m] of integer;
    i,j,s,s1,s2,max,t,p,min2,g,h : integer;
begin
    {Ввод элементов матрицы}
    writeln('Введите элементы матрицы: ');

```

```
for i:=1 to n do for j:=1 to m do
  begin
    write('a[',i,',',j,']= '); readln(a[i,j]);
  end;
{Вывод элементов матрицы}
writeln('Вывод матрицы:');
for i:=1 to n do
  begin
    for j:=1 to m do write(a[i,j]:4); writeln;
  end;
{сумма всех элементов матрицы}
s:=0;
for i:=1 to n do
  for j:=1 to m do s:=s+a[i,j];
writeln('s= ',s);
{сумма элементов первого столбца}
s1:=0; for i:=1 to n do s1:=s1+a[i,1];
writeln('s1= ',s1);
{сумма элементов второй строки}
s2:=0; for j:=1 to m do s2:=s2+a[2,j];
writeln('s2= ',s2);

{max и его индексы}
max:= a[1,1]; t:=1; p:=1;
for i:=1 to n do for j:=1 to m do
  if a[i,j] > max then
    begin max:= a[i,j]; t:=i; p:=j; end;
writeln('max= ',max,' t= ',t,' p= ',p);

{min2 и его индексы}
min2:= a[2,1]; g:=2; h:=1;
for j:=1 to m do
  if a[2,j] < min2 then
    begin min2:= a[2,j]; h:=j; end;
writeln('min2= ',min2,' g= ',g,' h= ',h);
end.
```

4.10 Задания для программирования

Задание 1. Запишите следующие числа, арифметические и логические выражения на языке Pascal.

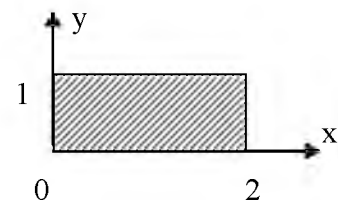
а) $5!$; $6,38$; $\frac{11}{4}$; $-\frac{1}{6}$; $\sqrt{2}$; $5 \cdot 10^6$; $-24,8 \cdot 10^{-7}$

б) $a+bx+cxy$; $[(ax-b)x+c]x-d$; $\frac{ab}{c} + \frac{c}{ab}$; $\frac{x+y}{a_1} \cdot \frac{a_2}{x-y}$

$\frac{1+\frac{x}{2!}+\frac{y}{3!}}{1+\frac{2}{3+xy}}$; $(1+x)^2$; $\cos^2 x^3$; $\sqrt{1+x^2}$;

в) $x \in [0, 1]$; $x \notin [0, 1]$; $x \in [-4, -2)$ или $(2, 4]$;

Запишите условие принадлежности точки (x, y) заштрихованной области



г) $y=1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}$; $f=6,673 \cdot 10^{-8} \cdot \frac{m_1 \cdot m_2}{r^2}$;

Задание 2. Составить программы для следующих задач:

1) вычислить периметр и площадь прямоугольника по заданным длинам двух сторон.

2) вычислить периметр и площадь прямоугольного треугольника по заданным длинам двух катетов.

3) вычислить сумму цифр трехзначного целого числа.

(Подсказка: использовать операции целочисленного деления **div** и **mod**).

4) по введенному часу определяет название времени суток. Будем считать, что с 6 до 9 -утро, с 10 до 17 -день, с 18 до 22 -вечер, с 23 до 5 –ночь.

5) по введенному номеру месяца определяет сезон года.
Сезоны года это весна, лето, осень, зима.

6) определить корни квадратного уравнения $ax^2+bx+c=0$ по заданным коэффициентам **a,b,c**. При отсутствии корней вывести сообщение.

7) выбрать максимальное из трех чисел.

8) ввести первую буквы названия месяца года; определить время года.

9) Вычислить сумму членов ряда $s = 5 - \frac{x^2}{2!} + \frac{x^4}{4!} - \dots$, с точностью 10^{-2} при $x=0,75$.

5 ИНТЕГРИРОВАННАЯ СРЕДА ПРОГРАММИРОВАНИЯ

5.1 Понятие среды программирования

Интегрированной средой программирования (Integrated Development Enviroment - **IDE**) называется пакет программ, предназначенный для разработки новых программ с использованием алгоритмического языка. Название языка указывается в заголовке интегрированной среды (в данном случае, язык Паскаль). IDE позволяет:

- 1) набирать текст на алгоритмическом языке (имеется встроенный текстовый редактор);
- 2) переводить текст с алгоритмического языка в команды процессора (в двоичный код). Такой перевод называется **компиляцией**;
- 3) вставлять в программу математические и др. функции (имеет обширную библиотеку встроенных функций);
- 4) обрабатывать несколько текстов в разных окнах, копировать фрагменты текстов между окнами;
- 5) отлаживать программу (искать и исправлять ошибки);
- 6) запускать готовую программу на выполнение;
- 7) сохранять набранный текст и готовую программу в виде файлов.

5.2 Описание IDE Borland Pascal

Стандартный язык Паскаль был создан в 1970 году швейцарским профессором Никлаусом Виртом специально для обучения программированию студентов университетов. Впоследствии усилиями фирмы Borland язык был доведен до профессионального уровня и получил название Turbo Pascal. Название Borland Pascal — это более полная версия Turbo Pascal (с большим количеством библиотек и исходным кодом стандартной библиотеки).

IDE Turbo/Borland Pascal — это интегрированная среда разработки программного обеспечения на основе языка Паскаль.

Управление средой Borland Pascal осуществляется как мышью, так и клавиатурой. Вызов IDE производится запуском файла **bp.exe**. Вид экрана после запуска среды показан на рисунке 5.1.



Рисунок 5.1 – Вид экрана после запуска Borland Pascal.

Верхняя строка окна содержит главное меню среды, которое активизируется мышью или клавишей F10. Выход из главного меню - Esc. В нижней части окна расположена строка состояния, она отражает также подсказки по использованию управляющих команд в данном режиме. Среднюю часть экрана занимает поле редактора, в котором набирается текст программы и могут открываться окна с ранее созданными текстами. Каждое окно с редактируемым текстом имеет рамку с названием файла и номером окна. В нижней рамке слева указывается положение курсора (номер строки и позиция в строке).

Ниже выборочно показываются наиболее употребительные пункты главного меню (рисунке 5.1).

Меню **Файл**

Новый	
Открыть...	F3
Сохранить	F2
Сохранить как...	
Сохранить все	
Сменить каталог...	
Печать	
Временный выход	
Выход	Alt+X

Меню **Правка**

Откат	Alt+BkSp
Повтор	
Вырезать	Shift+Del
Скопировать	Ctrl+Ins
Вставить	Shift+Ins
Удалить	Ctrl+Del

Меню **Поиск**

Найти...	
Заменить...	
Продолжить поиск	Shift+F7
Перейти к строке...	
К последней ошибке	
Найти ошибку...	
Найти процедуру...	

Меню **Пуск**

Запустить	Ctrl+F9
Обойти подпрограмму	F8
Войти в подпрограмму	F7
До курсора	F4
Параметры...	

Меню **Компиляция**

Компилировать	Alt+F9
Передепать	F9
Выстроить	
Режим...	Реальный
Главный файл...	
Сбросить главный файл	

Меню **Опции**

Компилятор...	
Размеры памяти...	
Компоновщик...	
Отладчик...	
Каталоги...	
Просмотрщик...	
Оболочка	▶
Открыть...	
Сохранить	
Сохранить как...	

Рисунок 5.2 – Элементы меню Borland Pascal.

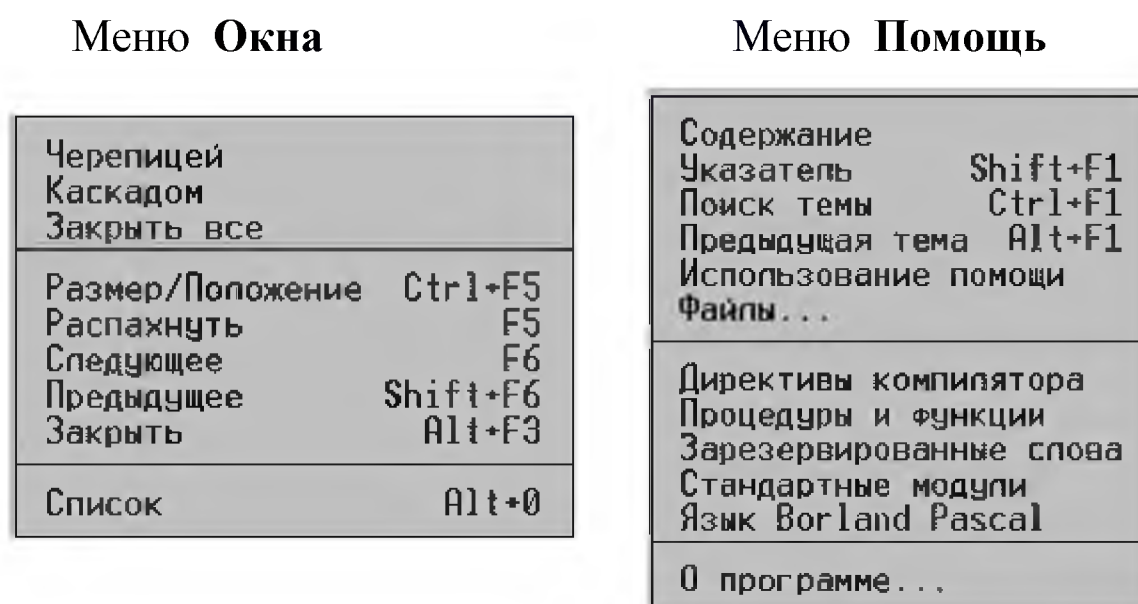


Рисунок 5.2 – Элементы меню (продолжение).

5.3 Порядок работы в Borland Pascal

Опишем порядок действий по созданию компьютерной программы с использованием IDE Borland Pascal (BP).

1. Прежде всего надо разработать алгоритм решения вашей конкретной задачи. Вы должны четко представлять себе ход решения задачи (лучше всего в виде блок-схемы). Это самое главное, именно этому вопросу посвящены предыдущие главы данного пособия.

2. Запустить BP. В наших учебных компьютерных классах запустить BP можно через ярлык на рабочем столе или через файл **bp.exe** в папке **C:\USER\BP**.

3. На рабочем поле синего цвета (меню **Файл–Новый**) набрать текст программы на алгоритмическом языке (рисунок 5.3). Основные приемы редактирования текста:

- переключение режимов вставки/замены символов - *Ins*;
- переключение раскладки клавиатуры: русский - *Правая Ctrl*, латинский - *Левая Ctrl*;
- вставка пустой строки - *Enter*;
- удаление строки – *Ctrl+Y*;

- разделение строки - *Enter*;
- склеивание строк - *Del* в конце первой строки;
- выделение блока (фрагмента) текста – *Shift+стрелки*;
- отмена выделения – *Ctrl+K, K*;
- копирование в позицию курсора - *Ctrl+K, C*;
- перенос в позицию курсора – *Ctrl+K, V*;
- откат – *Alt+BkSp* (Alt+Backspace) или меню **Правка-Откат**.

Напоминаем, что набранный текст должен заканчиваться словом **end.** (end с точкой).

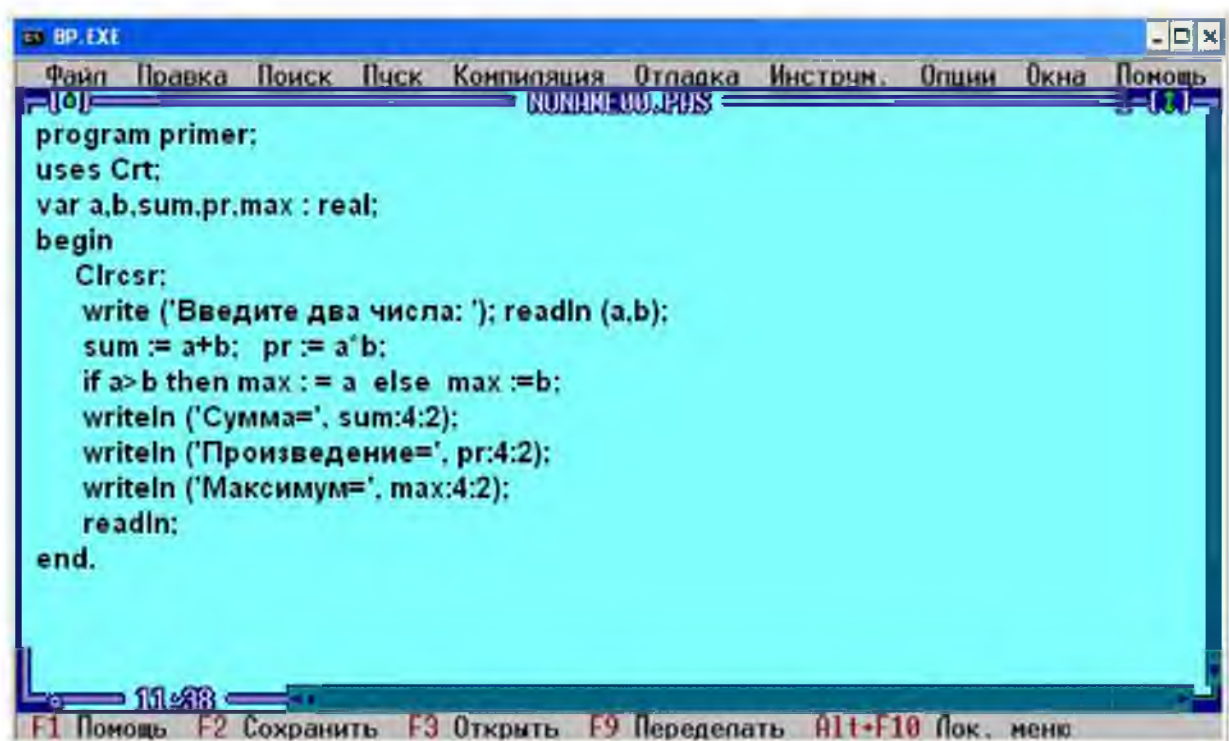


Рисунок 5.3 – Окно с набранным текстом.

4. Сохранить набранный текст, написав имя файла без указания расширения (меню **Файл–Сохранить**, рис. 5.4).

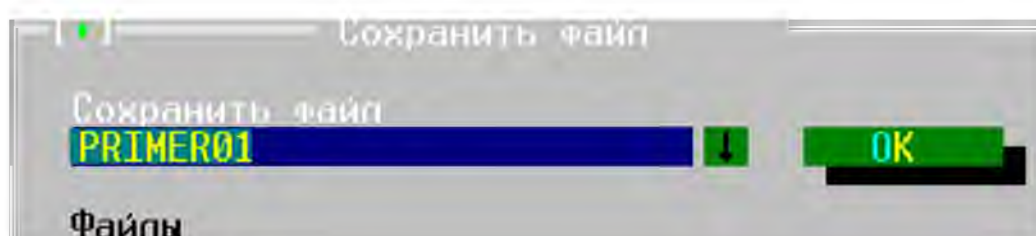


Рисунок 5.4 – Сохранение текста.

Расширение (**.pas**) будет присвоено файлу автоматически. В наших учебных компьютерных классах рекомендуется сохранять файлы в папке **C:\Work**.

5. Выполнить компиляцию (меню **Компиляция—Компилировать**, или **Alt+F9**). Если в тексте нет синтаксических ошибок, то появится приятное сообщение (рисунок 5.5). Если текст содержит ошибки, то курсор встанет возле ошибки и появится сообщение красного цвета с указанием типа ошибки; при этом нажатие **F1** поможет устранить ошибку. Устранив ошибку, повторить компиляцию.

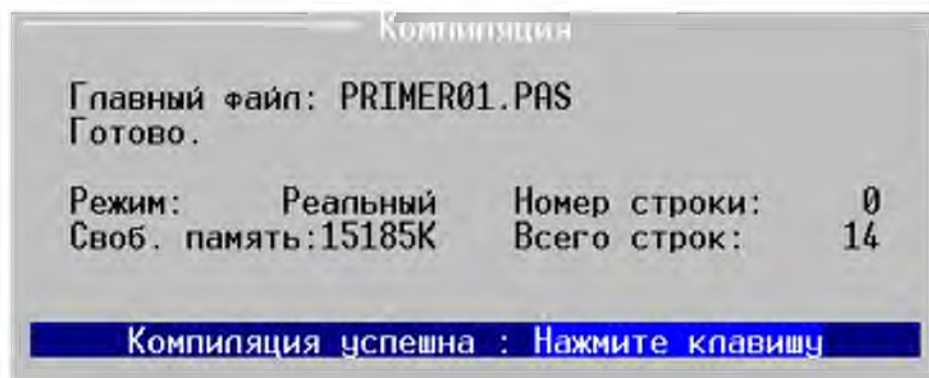


Рисунок 5.5 – Успешная компиляция.

Успешная компиляция свидетельствует о создании готовой компьютерной программы с ранее присвоенным именем и расширением (**.exe**).

6. Выполнить проверку правильности работы созданной программы (меню **Пуск—Запустить**, или **Ctrl+F9**). Появится окно с черным фоном; ввести исходные данные для задачи. Для просмотра результата расчета следует нажать **Alt+F5** или меню **Отладка—Вывод**. Проверить работу программы с разными вариантами исходных данных.

7. Для разработки другой программы надо открыть новое окно (меню **Файл-Новый**), т.е. повторить все действия с пункта 3.

8. Для завершения работы с Borland Pascal нажать **Alt+X** или меню **Файл-Выход**.

Замечание: Для удобства работы с программой рекомендуется добавить в исходный код программы следующие три строки (см. рисунок 5.2):

- 1) Во второй строке (после строки `program`) написать ***uses Crt;***
- это директива подключения библиотеки экранных функций.
- 2) В начало тела программы (после первого ***begin***) написать строку ***Clrscr;*** - это функция очистки экрана от прежних сообщений.
- 3) В конце тела программы (перед ***end.*** с точкой) написать строку ***readln;*** - это оператор обращения к клавиатуре, открывает окно для просмотра результата вычислений. Написание этой строки заменяет нажатие ***Alt+F5*** для просмотра результата.

ЗАКЛЮЧЕНИЕ

Были рассмотрены основные (элементарные) понятия разработки алгоритмов решения простейших задач и составление к ним текстов исходных кодов программ для реализации на компьютере. В качестве языка программирования рассматривался язык высокого уровня Паскаль (элементарный уровень).

Имеется много книг, посвященных алгоритмизации и программированию [1-10], которые предназначены для читателей с соответствующим уровнем подготовки и заинтересованности, и для которых компьютерные технологии составляют профилирующее направление подготовки. Данное пособие предназначено для студентов, направление подготовки которых не предполагает досконального изучения компьютерных технологий, однако знание основных разделов дисциплины «Информатика» для них определено государственными стандартами образования [Приложение 1, 2].

Для углубленного изучения основ алгоритмизации и программирования следует обратиться к специализированной литературе [1-10].

Литература:

1. Дональд Э. Кнут. Искусство программирования, том 1. Основные алгоритмы. - М.: Вильямс, 2008. - 720 с.
2. Лойко В.И. Структуры и алгоритмы обработки данных. Учебное пособие для вузов (электронное издание).- Краснодар: КубГАУ. 2009. - 268 с., ил.
3. Голицина О.Л., Попов И.И. Основы алгоритмизации и программирования. — М: ФОРУМ, 2008. — 432 с.
4. Е. А. Зуев. Программирование на языке Turbo Pascal 6.0, 7.0, М.: Веста, Радио и связь, 1993, — С.376, ISBN 5-256-01218-5
5. Кассера В. Ф. Turbo Pascal 7.0, Диасофт, 2003, ISBN 5-93772-097-0
6. Эллиот Б. Коффман. Turbo Pascal = Turbo Pascal Web Update. — М.: Вильямс, 2005. — С. 896. — ISBN 0-201-35086-6
7. Моргун Александр Николаевич. Справочник по Turbo Pascal для студентов. — М.: Диалектика, 2006. — С. 608. — ISBN 5-8459-1028-5
8. Нэйл Рубенкинг. Turbo Pascal для Windows = Turbo Pascal for Windows. Techniques and Utilites. — М.: Мир, 1993. — С. 535.
9. Фаронов В. В. Turbo Pascal. Наиболее полное руководство. BHV-Санкт-Петербург, 2007. ISBN 5-94157-295-6.
10. www.rsdn.ru - русскоязычный сайт, посвящённый программированию.
11. www.mpei.ru/lang/rus/main/aboutuniversity/umo/dpreparation.asp

Приложение 1

Министерство образования и науки Российской Федерации
Учебно-методическое объединение вузов по образованию в области
энергетики и электротехники

Рекомендовано ректором ГОУ ВПО МЭИ (ТУ)
Серебрянниковым С.Ф. от 06.04.2010г.

Примерная
основная образовательная программа
высшего профессионального образования

направление подготовки
140400 Электроэнергетика и электротехника

Настоящая примерная основная образовательная программа (ПрООП) разработана в соответствии с федеральным государственным образовательным стандартом высшего профессионального образования (ФГОС ВПО) подготовки бакалавра по направлению 140400 Электроэнергетика и электротехника, утвержденным приказом Министра образования и науки Российской Федерации от 9 декабря 2009 года № 710.

Аннотация примерной программы учебной дисциплины
“Информатика”

1. Цели и задачи дисциплины

Целью дисциплины является формирование мировоззрения и развитие системного мышления студентов.

Задачей изучения дисциплины является приобретение студентами практических навыков алгоритмизации, программирования; овладение персональным компьютером на пользовательском уровне, формирование умения работать с базами данных.

2. Требования к уровню освоения содержания дисциплины

Процесс изучения дисциплины направлен на формирование следующих компетенций:

- способность и готовность владеть основными методами, способами и средствами получения, хранения, переработки информации, использовать компьютер как средство работы с информацией (ОК-11);

- способность понимать сущность и значение информации в развитии современного информационного общества, сознавать опасности и угрозы, возникающие в этом процессе, соблюдать основные требования информационной безопасности, в том числе защиты государственной тайны (ОК-15);

- готовность использовать информационные технологии в своей предметной области (ПК-10);

- способность использовать современные информационные технологии, управлять информацией с применением прикладных программ; использовать сетевые компьютерные технологии, базы данных и пакеты прикладных программ в своей предметной области (ПК-19).

В результате изучения дисциплины студент должен:

знать: содержание и способы использования компьютерных и информационных технологий;

уметь: применять компьютерную технику и информационные технологии в своей профессиональной деятельности;

владеть: средствами компьютерной техники и информационных технологий.

3. Содержание дисциплины. Основные разделы

Понятие информации, общая характеристика процессов сбора, передачи, обработки и накопления информации; технические и программные средства реализации информационных процессов; модели решения функциональных и вычислительных задач; алгоритмизация и программирование: языки программирования высокого уровня; базы данных; программное обеспечение и технологии программирования; локальные и глобальные сети ЭВМ; основы защиты информации и сведений, составляющих государственную тайну; методы защиты информации; компьютерный практикум.

Приложение 2

ВЫПИСКА

ФГОС ВПО подготовки бакалавра по направлению

270800 – Строительство

Утвержден Приказом министерства образования и науки РФ
от 18.01.2010г. № 54

5.1. Выпускник должен обладать следующими общекультурными компетенциями (ОК):

- владением культурой мышления, способностью к обобщению, анализу, восприятию информации, постановке цели и выбору путей ее достижения (ОК-1).

5.2. Выпускник должен обладать следующими профессиональными компетенциями (ПК):

- способностью понимать сущность и значение информации в развитии современного информационного общества, сознавать опасности и угрозы, возникающие в этом процессе, соблюдать основные требования информационной безопасности, в том числе защиты государственной тайны (ПК-4);
- владением основными методами, способами и средствами получения, хранения, переработки информации, навыками работы с компьютером как средством управления информацией (ПК-5);
- способностью работать с информацией в глобальных компьютерных сетях (ПК-6);

6. Требования к структуре основных образовательных программ бакалавриата.

Код УЦ ООП (учебный цикл основной образовательной программы): Б.2 - Математический, естественнонаучный и общетехнический цикл. Базовая часть.

В результате изучения базовой части цикла обучающийся должен знать:

- основные понятия информатики, современные средства вычислительной техники, основы алгоритмического языка и технологию составления программ.

Учебное издание

Галиев Карим Сулейманович
Печурина Елена Каримовна

**ОСНОВЫ АЛГОРИТМИЗАЦИИ
И ПРОГРАММИРОВАНИЯ**

Учебно-методическое пособие
для студентов-бакалавров,
изучающих «Информатику»

В авторской редакции

Подписано в печать «__»_____2013 г. Формат $60 \times 84^{1/16}$.
Тираж 200 экз. Усл. печ. л. – 6.0. Уч.-изд. л. – 4,3
Заказ № _____

Типография
Кубанского государственного аграрного университета,
350044, г. Краснодар, ул. Калинина, 13