ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
Государственное образовательное учреждение высшего профессионального образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

И.П. Степанова, Е.С. Чердынцев

ИНФОРМАТИКА

Часть 2

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2010

УДК 681.322.06
С79

Степанова И.П.

С79 Информатика. Часть 2: учебное пособие / И.П. Степанова, Е.С. Чердынцев; Томский политехнический университет. − Томск: Изд-во Томского политехнического университета, 2010. − 157 с.

Пособие содержит описание основных алгоритмов решения задач на примере управляющих конструкций алгоритмических языков *Turbo Pascal* и *C++*.

Предназначено для иностранных студентов первого курса.

УДК 681.322.06

Рецензенты

Доктор технических наук, профессор ТУСУРа
Ю.П. Ехлаков

Кандидат технических наук, доцент ТУСУРа
Н.Ю. Хабибуллина

Кандидат технических наук, доцент ТПУ
О.Б. Фофанов

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, ЕДИНИЦ И ТЕРМИНОВ

HTML (англ. *HiperText Markup Language)* – это язык разметки гипертекста, разработанный для размещения информации в *Internet*;

PHP (англ. *Hypertext Preprocessor*) - это язык программирования, который предназначен для быстрого создания динамических *Web*-страниц;

XML (англ. *Extensible Markup Language*) – это расширяемый язык разметки, разработанный специально для размещения информации в *World Wide Web;*

ПП прикладная программа;

ЭВМ электронная вычислительная машина - в настоящее время называют компьютером (от англ. *computer* – вычислитель).

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, ЕДИНИЦ И ТЕРМИНОВ **3**
ВВЕДЕНИЕ **7**
ГЛАВА 1. АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ **9**
1.1. История появления языков программирования **9**
1.2. Развитие языков программирования **11**
1.3. Контрольные вопросы **11**
ГЛАВА 2. СТРУКТУРА ПРОГРАММЫ. ЧИСЛОВЫЕ И ЛОГИЧЕСКИЙ ТИПЫ ДАННЫХ **12**
2.1. Структура программы **12**
2.2. Числовые и логический типы данных **13**
2.2.1. Целочисленный тип данных 13
2.2.2. Вещественный тип данных 13
2.2.3. Логический тип данных – булевский 14
2.3. Контрольные вопросы **15**
ГЛАВА 3. ПРОГРАММЫ С ЛИНЕЙНОЙ СТРУКТУРОЙ **16**
3.1. Оператор присваивания **16**
3.1.1. Обычная скобочная запись арифметического выражения 19
3.1.2. Польская запись 19
3.1.3. Задание значений с помощью датчика случайных чисел 20
3.2. Операторы ввода **21**
3.2.1. Операторы ввода на языке *Turbo Pascal* 21
3.2.2. Операторы ввода на языке *C++* 22
3.3. Операторы вывода **22**
3.3.1. Операторы вывода на языке *Turbo Pascal* 22
3.3.2. Операторы вывода на языке *C++* 25
3.4. Контрольные вопросы **25**
ГЛАВА 4. АЛГОРИТМЫ ВЕТВЛЕНИЯ **27**
4.1. Оператор безусловного перехода `GOTO` **28**
4.2. Оператор условного перехода `IF ... THEN ... ELSE` **28**
4.3. Оператор выбора **34**
4.3.1. Оператор выбора `CASE` на языке *Turbo Pascal* 34
4.3.2. Оператор выбора `switch` и `break` на языке *C++* 36
4.4. Контрольные вопросы **37**
ГЛАВА 5. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ **38**
5.1. Конечные циклы **38**
5.1.1. Операторы цикла в языке *Turbo Pascal* 39
5.1.2. Операторы цикла в языке *C++* 43

5.2. Алгоритмы обработки значений переменных в цикле 45
5.3. Итерационные циклы 49
5.4. Вложенные циклы 54
5.5. Контрольные вопросы 56
ГЛАВА 6. ОБРАБОТКА ЭЛЕМЕНТОВ МАССИВОВ 57
6.1. Одномерные массивы 57
6.1.1. Способы задания одномерных массивов на языке *Turbo Pascal* 58
6.1.2. Ввод, вывод значений элементов одномерного массива на языке *Turbo Pascal* 59
6.1.3. Алгоритмы обработки элементов одномерного массива 60
6.2. Двумерные массивы 71
6.2.1. Способы описания двумерного массива на языке *Turbo Pascal* 72
6.2.2. Ввод значений элементов двумерного массива на языке *Turbo Pascal* 74
6.2.3. Вывод значений элементов матрицы 75
6.2.4. Алгоритмы обработки значений элементов двумерного массива 76
6.2.5. Матричная алгебра 83
6.3. Контрольные вопросы 87
ГЛАВА 7. ОБРАБОТКА СИМВОЛЬНЫХ И СТРОКОВЫХ ДАННЫХ 89
7.1. Символьный тип данных 89
7.2. Строковый тип данных 92
7.3. Контрольные вопросы 99
ГЛАВА 8. ПОДПРОГРАММЫ: ПРОЦЕДУРЫ, ФУНКЦИИ. РЕКУРСИЯ 101
8.1. Подпрограммы на языке *Turbo Pascal* 101
8.1.1. Процедуры 101
8.1.2. Функции 104
8.2. Подпрограммы-функции на языке *C++* 107
8.3. Рекурсия 108
8.4. Обработка массивов, строковых переменных в подпрограммах на языке *Turbo Pascal* 110
8.5. Контрольные вопросы 114
ГЛАВА 9. АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА ДАННЫХ 115
9.1. Алгоритмы сортировки элементов одномерного массива 115
9.1.1. Сортировка выбором 115
9.1.2. Сортировка обменом (метод пузырька) 118
9.2. Поиск данных 120

9.3. Контрольные вопросы 122
ГЛАВА 10. МНОЖЕСТВА 123
10.1. Описание множеств 123
10.2. Операции над множествами 124
10.3. Использование множеств 127
10.4. Контрольные вопросы 128
ГЛАВА 11. ЗАПИСИ 129
11.1. Описание записей на языке *Turbo Pascal* 130
11.2. Описание структурной переменной на языке *C*++ 132
11.3. Контрольные вопросы 133
ГЛАВА 12. ФАЙЛЫ 134
12.1. Текстовые файлы на языке *Turbo Pascal* 136
12.2. Типизированные файлы на языке *Turbo Pascal* 140
12.3. Контрольные вопросы 143
ГЛАВА 13. ПОСТРОЕНИЕ ГРАФИКОВ 144
13.1. Графический режим на языке *Turbo Pascal* 144
13.2. Контрольные вопросы 154
ЗАКЛЮЧЕНИЕ 155
ЛИТЕРАТУРА 156

ВВЕДЕНИЕ

Специалисты широко используют персональные ЭВМ во всех сферах своей деятельности. Класс задач, которые решаются с помощью персональных ЭВМ, очень широк - это обработка справочных, статистических и отчётных данных, автоматизация решения управленческих задач и выполнение экономических расчетов, автоматизация проектных и технологических работ, автоматизация процесса обучения и контроля знаний и многое другое.

В процессе обучения студенты всех специальностей изучают основы программирования на одном из алгоритмических языков программирования: *Turbo Pascal*, *Basic* или *C++* (читается как Си-плюс-плюс). Трудность освоения дисциплины программирования связана с двумя проблемами: освоение алгоритмов обработки данных и изучение самого алгоритмического языка для описания алгоритма решения задачи.

Пособий по языкам программирования имеется большое количество. Но издания, в котором были бы собраны основные алгоритмы обработки информации, причём, с их детальным описанием, очень мало.

Данное пособие призвано дополнить список учебных пособий по алгоритмизации: приведены алгоритмы обработки различных данных с их детальным описанием, даны для наглядности блок-схемы, примеры по алгоритмам приведены на двух алгоритмических языках: *Turbo Pascal* и *C++*.

При изложении программирования на алгоритмических языках *Turbo Pascal* и *C++* авторы придерживались модифицированных правил записи по Бэкусу[1]:

1. Слова, набранные шрифтом *Courier New*, обозначают зарезервированные (ключевые) слова. При этом для языка *Turbo Pascal* не важно, набраны они прописными (заглавными) или строчными буквами. Поэтому при изложении правил и примеров на языке *Turbo Pascal* запись будет выполняться прописными буквами. Язык *C++* различает прописные и строчные буквы, поэтому зарезервированные слова будут записаны строчными буквами шрифтом *Courier New*.

2. Для наглядности и удобства название определяемой языковой конструкции, которая при записи программы будет заменяться иденти-

[1] Правила записи по Бэкусу, которые в литературе называют *нотацией Бэкуса* (Н. Бэкус – один из авторов языка Алгол-60).

фикатором (именем переменной) или выражением, заключается в угловые скобки «<» и «>».

В пособии рассматриваются следующие разделы программирования:

- история развития алгоритмических языков программирования;
- структура программы, числовые и логические типы данных;
- программы с линейной структурой, где приведены способы записи арифметических выражений: как обычная, так и польская запись; даны операторы ввода и вывода;
- алгоритмы ветвления, где описаны все виды ветвлений: безусловный переход, условный переход, переход по выбору;
- циклические алгоритмы, где рассмотрены как конечные, так и итерационные циклы;
- обработка элементов массивов, где описаны максимально полно алгоритмы обработки как одномерных, так и двумерных массивов, а также дана матричная алгебра;
- обработка символьных и строковых данных, где максимально полно приведены процедуры и функции обработки этих данных;
- подпрограммы; даны описания процедуры и функции, а также рассматривается рекурсия;
- алгоритмы сортировки и поиска данных; в данном разделе рассматриваются только два алгоритма: сортировка выбором и сортировка обменом;
- множества на языке *Turbo Pascal*;
- записи на языке *Turbo Pascal* и структуры на языке *C++*;
- файлы на языке *Turbo Pascal*;
- построение графиков на языке *Turbo Pascal*.

Глава 1.

АЛГОРИТМИЧЕСКИЕ ЯЗЫКИ ПРОГРАММИРОВАНИЯ

1.1.История появления языков программирования

Языки программирования условно разделяют на поколения:

- первое поколение, к которому относят *машинные* языки;
- второе поколение, к которому относят языки *ассемблеры*;
- третье поколение – это алгоритмические языки;
- четвёртое поколение языков - это языки управления программным обеспечением, то есть генераторы программ;
- пятое поколение – это языки программирования для решения задач, которые связаны с искусственным интеллектом.

Свой *машинный язык* существует для каждой электронной вычислительной машины. Команды представляются в машинных языках последовательностью нулей (0) и единиц (1). Команды из последовательности нулей и единиц человеку читать невозможно.

В *языках ассемблера*, языках второго поколения, используются символические обозначения вместо цифрового адреса и мнемонические (символьные) коды операций. Пример программы на языке ассемблера приведён на рис. 1.

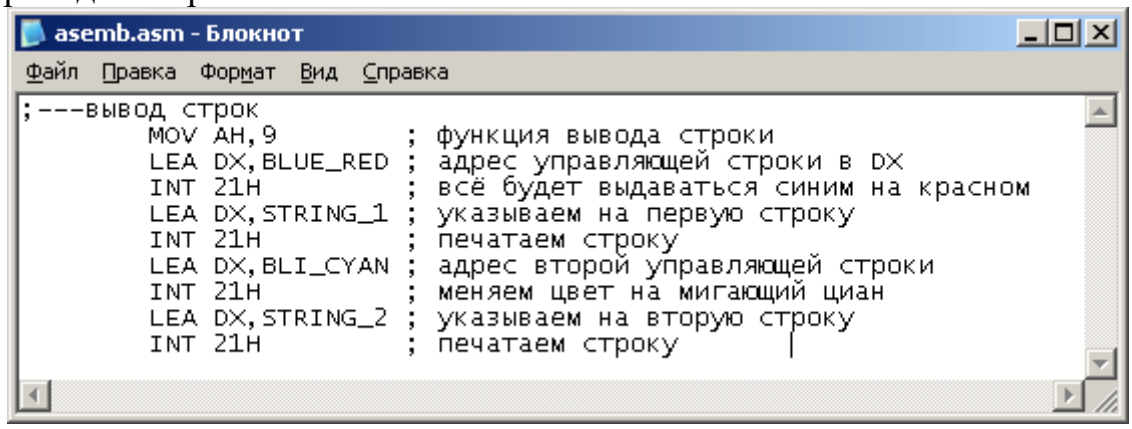


Рис. 1. Пример программы на языке ассемблера

Языки третьего поколения – это алгоритмические языки:

- Фортран (*FORTRAN*[2]*),* который появился в 1955 году и инструкции которого стали понятнее пользователям;

[2] *FORTRAN – FORmula TRANslator* – алгоритмический язык для научных расчётов.

- Алгол (*Algoritmic Language*), который появился в 1960 г. и который также как и *FORTRAN,* был ориентирован на научные применения;

- Бейсик (*Basic*)[3], который появился в 1963 г. и который позже (в 1981 году) стал основным языком микрокомпьютеров благодаря своей простоте;

- Паскаль (*Pascal),* который создал Никлаус Вирт в 1969 году в Федеральном техническом университете (Швейцария);

- Ада (*Ada*), который создан в 1980 году и который унаследовал качества языка *Pascal* и дополнительно приобрёл другие свойства: системное программирование и параллельность;

- *C* (читается как Си), который создал в 1970 году Денис Ритчи, сотрудник американской лаборатории *AT&T Bell Laboratories*.

Алгоритмический язык *C* – это язык общего назначения, который можно использовать для написания различных программ. Язык программирования *C* был написан для создания и поддержки операционной системы *UNIX*.

Особенность языка *C* заключается в том, что он является языком высокого уровня, сохранив многие черты языка низкого уровня. Как и язык ассемблера (язык низкого уровня), язык программирования *C* может непосредственно управлять памятью компьютера. С другой стороны, *C* обладает чертами языков высокого уровня, поэтому программы на языке *C* читать и создавать легче, чем программы на языке ассемблера.

Бьярн Страуструп из американской лаборатории *AT&T Bell Laboratories* разработал в 1980 году на основе языка *C* язык программирования *C++*. Основное отличие языка *C++* от *C* заключается в реализации объектно-ориентированного подхода программирования - чрезвычайно мощного современного способа программирования.

В настоящее время широкое распространение получили реализации языка в системах *Visual C++*, *Borland C++ x.x*, Borland C++ Builder x и др. Материал настоящего учебного пособия ориентирован на версию *Borland C++* 3.1.

Четвёртое поколение языков - это языки управления программным обеспечением, то есть генераторы программ.

К пятому поколению языков относятся языки Лисп[4], который разработал Маккарти в 1961 г., и Пролог[5], который был разработан в университете Люмини в 1973 г. Оба языка программирования созданы для

---

[3] Бейсик – *Basic –Beginner`s All purpose Symbolic Instructions Code* - универсальный код символических инструкций для начинающих

[4] Лисп – *LISP – LISt Processing language*

[5] Пролог – *PROLOG – PROgrammation end LOGique*

задач, которые связаны с искусственным интеллектом. Эти языки хорошо приспособлены к представлению и использованию знаний, как в экспертных системах, так и в узких областях: геологии, медицине, распознаванию речи.

1.2.Развитие языков программирования

С течением времени часть языков программирования развивалась, а другая часть утратила свою актуальность и сегодня представляет лишь чисто познавательный интерес. Развитие языков программирования и их использование связано со следующими факторами:

- наличие среды программирования, которая поддерживает разработку приложений на конкретном языке программирования;
- удобство сопровождения и тестирования программ;
- стоимость разработки новой прикладной программы с применением конкретного языка программирования;
- четкость конструкций языка программирования;
- применение объектно-ориентированного подхода.

Так в 90-х годах с распространением сети *Internet* появляются новые языки программирования, которые ориентированы на создание серверных приложений, такие как *Java*, *Perl* и *PHP*, языки описания документов – *HTML* и *XML*.

Новый язык программирования *Java* появился в 1995 году в корпорации *Sun Microsystems* (создан Кеном Арнольдом и Джеймсом Гослингом). Отличительной особенностью языка *Java* является компиляция в код некоей абстрактной (вымышленной) машины, для которой затем пишется эмулятор (*Java Virtual Machine*) для реальных систем.

Специалисты считают, что языки программирования должны развиваться в сторону абстракции, то есть не зависеть от архитектуры конкретной ЭВМ. Повышение уровня абстракции влечет за собой повышение уровня надежности программирования.

1.3. Контрольные вопросы

1. Почему языки программирования разделяют на поколения?
2. Назовите языки программирования первого поколения.
3. Назовите языки программирования второго поколения.
4. В чём особенность алгоритмических языков третьего поколения?
5. Назовите новые современные языки программирования.

Глава 2.

СТРУКТУРА ПРОГРАММЫ. ЧИСЛОВЫЕ И ЛОГИЧЕСКИЙ ТИПЫ ДАННЫХ

2.1. Структура программы

Любая программа состоит из основных блоков:

- блока начала, в который входит задание констант и задание типов обрабатываемых переменных;
- блока задания или ввода значений обрабатываемых переменных;
- блока обработки значений обрабатываемых переменных;
- блока вывода значений результатов обработки;
- блока окончания программы.

Блочная структура любой программы приведена на рис. 2.

Рис. 2. Блочная структура любой программы

2.2. Числовые и логический типы данных

При объявлении переменной необходимо указать её *тип*. Тип переменной описывает набор значений, которые переменная может принимать, и действия, которые могут быть над ней выполнены.

В языках программирования различают типы данных:

- целочисленный, который обрабатывает целые значения;
- вещественный, который обрабатывает действительные (дробные) значения.

2.2.1. Целочисленный тип данных

В *Turbo Pascal* тип переменной отделяется от идентификатора двоеточием (:).

В *Turbo Pascal* и *C++* реализованы типы целых данных, которые приведены в табл. 1.

Таблица 1

Целочисленный тип данных

Целочисленный тип данных		Наименование	Диапазон допустимых значений	Количество байт для внутреннего хранения
Turbo Pascal	*C++*			
byte		байт	от 0 до 255	1байт
integer	int = short int	целое	от –32767 до 32767	2 байта
longint	long = long int	длинное целое	$\text{от} -2\cdot10^9 \text{ до} 2\cdot10^9$	4 байта
shortint		короткое целое	от –127 до 127	1 байт
word	unsigned int	слово	от 0 до 65535	2 байта

Далее в примерах программ на *Turbo Pascal* будет использован только целочисленный тип `INTEGER`, а на *C++* - `int`.

Примеры чисел целого типа приведены через запятую, причём в случае типа `INTEGER` максимально возможное число +32767 и минимальное число –32767:

-32767, -20, -5, 0, 5, 20, 444, 6666, 32000, +32767 .

2.2.2. Вещественный тип данных

В *Turbo Pascal* и *C++* реализованы типы вещественных данных, которые приведены в табл. 2.

Далее в программах на *Turbo Pascal* будет использован только вещественный тип `REAL`, а на *C++* - либо `float`, либо `double`.

Примеры чисел вещественного типа приведены через запятую. При записи использован десятичный способ изображения числа, причём в записи дробная часть отделяется *десятичной точкой:*

-22.1234, -5.5, 0, 5.5, 22.1234, 6666.6, 88888.8888, 444444.5678 .

Таблица 2

Вещественный тип данных

Вещественный тип данных		Наименование	Диапазон допустимых значений	Число значащих цифр	Количество байт для внутреннего хранения
Turbo Pascal	*C++*				
comp		длинное целое	от $-9\cdot10^{18}$ до $9\cdot10^{18}$	19	8 байт
double	double	с двойной точностью	от 10^{-308} до 10^{308}	16	8 байт
extended	long double	с повышенной точностью	от 10^{-4931} до 10^{4931}	18	10 байт
real		вещественный	от 10^{-38} до 10^{38}	12	6 байт
single	float	с одинарной точностью	от 10^{-38} до 10^{38}	8	4 байта

2.2.3. Логический тип данных – булевский

Логический (булевский) тип данных имеет стандартный описатель *BOOLEAN*. Переменные и константы булевского типа принимают только одно из двух значений, которые определяются стандартными константами *TRUE* (истина) и *FALSE* (ложь). В операциях сравнения для переменных логического типа значение *FALSE<TRUE*.

Логические операции выполняются над операндами булевского типа. Список логических операций приведён в табл. 3.

Оформление логических операций в языке *C++*:

- логическое отрицание НЕ (*NOT*) – это знак «!»;
- логическое «И» (*AND*) - это знак «&&»;
- логическое «ИЛИ» (*OR*) - это знак «||».

Таблица 3

Логические операции

Операция	Пример		Значение A	Значение B	Результат	Название
	Turbo Pascal	*C++*				
NOT (!)	*NOT* A	!A	True False		False True	Логическое отрицание
AND (&&)	A *AND* B	A && B	True True False False	True False True False	True False False False	Логическое «И»
OR (||)	A *OR* B	A || B	True True False False	True False True False	True True True False	Логическое «ИЛИ»
XOR	A *XOR* B		True True False False	True False True False	False True True False	Исключающее «ИЛИ»

2.3. Контрольные вопросы

1. Назовите основные блоки любой программы.
2. Назовите числовые типы данных.
3. Приведите примеры описания целых данных на языке *Turbo Pascal* (на языке *C++*).
4. Приведите примеры описания вещественных данных на языке *Turbo Pascal* (на языке *C++*). .
5. Назовите логический тип данных.
6. Назовите логические операции.

Глава 3.

ПРОГРАММЫ С ЛИНЕЙНОЙ СТРУКТУРОЙ

Наиболее наглядный способ представления алгоритма – это изображение его в виде схемы, то есть последовательности блоков, которые предписывают выполнение определённых действий, и связей между ними.

По характеру связей между блоками различают следующие алгоритмы:

- алгоритмы *линейной* структуры;
- алгоритмы *ветвления* (*разветвляющейся* структуры);
- алгоритмы *циклической* структуры.

Алгоритм линейной структуры – это алгоритм, в котором операции выполняются последовательно друг за другом, то есть в программе будут присутствовать только операторы ввода данных, вывода данных и операторы присваивания.

3.1. Оператор присваивания

Оператор присваивания используется для присва́ивания значения переменной. Оператор присваивания заменяет текущее значение переменной, стоящей слева, новым значением, которое определяется арифметическим выражением справа.

Арифметические операции

Арифметическое выражение может состоять из числовых констант, переменных, стандартных функций и операций над ними. В табл. 4 приведены символы, которые используются для обозначения операций во всех алгоритмических языках программирования: сложение (+), вычитание (−), умножение (*) и деление (/). Дополнительные символы, которые используются для обозначения операций в языке *Turbo Pascal* и в языке *C++* приведены в табл. 5.

Таблица 4

Арифметические операции

Операция	Действие	Пример
+	Сложение	A+B
-	Вычитание	A−B
*	Умножение	A*B
/	Деление	A/B

Таблица 5

Дополнительные операции в *Turbo Pascal* и *C++*

Операция		Действие	Пример	
Turbo Pascal	*C++*		*Turbo Pascal*	*C++*
`DIV`[6]		Целочисленное деление	`A DIV B`	
`MOD`[7]	%	Остаток от целочисленного деления	`A MOD B`	`A%B`

Значение выражения `X/Y` (деление) всегда будет вещественное. Если *Y* равно нулю, то возникает ошибка вычисления, и программа прерывается.

Значение выражения `I DIV J` представляет собой математическое частное (*целую часть*) от `I/J`, которое округлено до значения целого типа. Если *J* равно нулю, то возникает ошибка вычисления, и программа прерывается.

Операторы `MOD` и *%* возвращают *остаток*, полученный при делении двух операндов, то есть:

```
I MOD J = I - (I DIV J) * J .
```

Знак результата оператора `MOD` будет тем же, что и знак *I*. Если *J* равно нулю, то возникает ошибка вычисления, и программа прерывается.

Арифметическое выражение должно быть совместимо по присваиванию с типом переменной в левой части или типом значения, которое возвращает функция в качестве результата. В табл. 6 приведены типы операндов и результата в арифметических операциях на примере языка *Turbo Pascal.*

Таблица 6

Тип операндов и тип результата в арифметических операциях

Знак операции	Выражение	Действие	Тип операндов	Тип результата
+	A+B	сложение	real integer real, integer	real integer real
-	A-B	вычитание	real integer real, integer	real integer real
*	A*B	умножение	real integer real, integer	real integer real
/	A/B	деление	real integer real, integer	real real real

[6] *DIV* - англ. *divide,* которое переводится как делить.
[7] *MOD* - англ. *modulo,*что означает определение остатка.

Для указания порядка выполнения операций в арифметическом выражении используются круглые скобки.

Стандартные функции

В языке *Turbo Pascal* и в языке *C++* определены стандартные функции, которые приведены в табл. 7. Аргументом функции может быть любое арифметическое выражение, которое заключено в круглые скобки.

Таблица 7

Стандартные функции *Turbo Pascal* и *C++*

Запись в математике	Реализуемое действие	Запись в *Turbo Pascal*	Запись в *C++*[8]
$\|x\|$	абсолютное значение X	`abs(x)`	`Abs(x)`
$\cos x$	косинус X	`cos(x)`	`cos(x)`
$\sin x$	синус X	`sin(x)`	`sin(x)`
arctg x	арктангенс X	`arctan(x)`	`atan(x)`
e^x	экспонента X	`exp(x)`	`exp(x)`
π	пи	`pi`	
$\sqrt{x}$	корень квадратный X	`sqrt(x)`	`sqrt(x)`
x^2	квадрат X	`sqr(x)`	`pow(x,2)`
$\ln x$	логарифм натуральный X	`ln(x)`	`Log(x)`

Формула 1 Пример 1. Составить программу вычисления функции:

$$Y = \sin x + \cos x \cdot \operatorname{arctg} x - \sqrt{x} / x^2 + |\pi - x| - \ln x$$

Значение x в программе задавать стандартной функцией `random(N)`, которую называют *датчиком случайных чисел*.

{Программа по условию примера 1 на языке *Turbo Pascal*}

```
VAR X:INTEGER;  ;{X объявлено как целое}
    Y:REAL;       {Y объявлено как вещественное}
 BEGIN
  X:=RANDOM(1000));  {получено случайное число из интервала [0,1000]}
  Y:=SIN(X)+COS(X)*ARCTAN(X);
  Y:=Y-SQRT(X)/(X*X)+ABS(PI-X)-LN(X);
```

[8] Для языка *C++* это только часть функций

```
    WRITELN('X = ',X:4,Y = ',Y:10:4);
     END.  {конец программы}
```

/*Программа по условию примера 1 на языке *C++*/

```
#include<iostream.h>
#include<math.h>
  void main()
    { const pi=3.14;
       int x;
       float y;
       x=random(1000);
       cout<<"x="<<x<<'\n';
       y=sin(x)+cos(x)*Atan(x);
       y=y-sqrt(x)/(x*x)+Abs(pi -x)-Log(x);
       cout<<"y="<<y;
       getch();
    }
```

3.1.1. Обычная скобочная запись арифметического выражения

Арифметическое выражение представляет собой последовательность символов. В арифметическом выражении могут быть имена переменных, встроенные функции, аргумент которых обязательно должен заключаться в круглые скобки, константы и арифметические операции: сложение (+), вычитание (−), умножение (*) и деление (/).

При выполнении арифметического выражения по умолчанию используется приоритет (главенство) арифметических операций как в математике. Для изменения порядка выполнения операций в арифметическом выражении можно использовать круглые скобки.

Формула 2 Пример 2. Обычная скобочная запись арифметического выражения:

$(R + H)/2 * 3.14$.

3.1.2. Польская запись

В пункте 3.1.1. приведён широко распространённый способ записи арифметических выражений с использованием круглых скобок и дополнительных соглашений о приоритетности арифметических операций. Но в математике и вычислительной науке такой способ (с использованием

круглых скобок и дополнительных соглашений о приоритетности арифметических операций) неудобен.

Польский учёный Ян Лукасевич (польск. Jan Lukasiewicz) предложил другой способ записи выражений, в котором операция стоит до или после операндов, а не разделяет их. При этом вводится вспомогательный символ, который называют *разделителем*. Этот разделитель, например, литера «_», должен служить для отделения одного операнда от другого

Запись выражения, в которой операция стоит до операндов, называется *польской префиксной записью*, а запись выражения, в которой операция стоит после операндов, называется *польской суффиксной записью*.

Пример польской суффиксной записи выражения приведено в табл. 8.

Таблица 8

Пример польской суффиксной записи выражения

Выражение	Польская суффиксная запись
$(R+H)/2*3.14$	$R\_H\_+\_2\_/\_3.14*$

3.1.3. Задание значений с помощью датчика случайных чисел

Во многих языках программирования задать значение переменной можно с помощью специальной встроенной функции, которую называют *датчиком случайных чисел*[9]. В *Turbo Pascal и C++* – это функция *RANDOM(N)*, где *N* – верхнее значение целого интервала от 0 до N.

Формула 3 Пример 3. Составить программу, в которой необходимо получить случайным образом целые значения из интервалов:

- от 10 до 99 (двухзначные числа);
- от 100 до 999 (трёхзначные числа);
- от 1000 до 9999 (четырёхзначные числа);
- от 10000 до 32767 (пятизначные числа).

{Программа по условию примера 3 с использованием функции `RANDOM(N)` на языке *Turbo Pascal*}

```
VAR A,B,B,D:INTEGER; { A,B,B,D объявлены как целые}
 BEGIN
  RANDOMIZE; {нарушили регулярную[10] последовательность случайных чисел}
  A:=10+RANDOM(90);      { двухзначные числа от 10 до 99 }
  WRITELN('A = ',A:10);
```

[9] Датчик случайных чисел = генератор (создатель) псевдослучайных чисел.

[10] Регулярная = повторяющаяся

```
    B:=100+RANDOM(900);   { трёхзначные числа от 100 до 999}
    WRITELN('B = ',A:10);
    C:=1000+RANDOM(9000); { четырёхзначные числа от 1000 до 9999}
    WRITELN('C = ',C:10);
    D:=10000+RANDOM(22768); {пятизначные числа от 10000 до 32767}
    WRITELN('D = ',D:10);
   END.
```

/*программа по условию примера 3 с использованием функции RANDOM(N) на языке *C*++*/

```
#include<iostream.h>
#include<math.h>
  void main()
   {  int a,b,c,d;
      a=10+random(90); cout<<"\a= "<<a<<'\n';
      b=100+random(900); cout<<"b="<<b<<'\n';
      c=1000+random(9000); cout<<"c="<<c<<'\n';
      d=10000+random(22768); cout<<"d="<<d<<'\n';
      getch();
   }
```

3.2. Операторы ввода

3.2.1. Операторы ввода на языке *Turbo Pascal*

Для ввода значений данных в память персонального компьютера с клавиатуры предназначен оператор *READ* либо *READLN*, с помощью которого переменные получают значения во время выполнения программы.

Оператор имеет следующий вид:

либо `READ(a1,a2, ...,an);` либо `READLN(a1,a2, ...,an);`

где *a1,a2, ...,an* – это переменные, последовательно принимающие значения, которые вводятся с клавиатуры и которые отражаются на экране монитора.

Следует знать о вводе данных следующее:

1. Значения данных вводятся после набора на экране всей программы и запуска её на выполнение.

2. Если вводятся числа, то они отделяются друг от друга пробелом или несколькими пробелами.

3. При вводе количество вводимых числовых значений должно соответствовать количеству параметров в операторе *READ*.

4. Перед вводом данных с помощью оператора *READ* рекомендуется давать поясняющий текст с помощью оператора *WRITE*. Этим самым устанавливается диалог между пользователем и программой.

Оператор ввода имеет следующие назначения:

1. `READ(a1,a2, ...,an);` – по данному оператору переменные `a1,a2, ...,an` последовательно получают вводимые значения.

2. *`READLN(a1,a2, ...,an);`* – по данному оператору переменные *`a1,a2, ...,an`* последовательно получают вводимые значения, после чего происходит переход на новую строку. Если за этим оператором последует снова оператор ввода, то он будет ждать данные в новой строке.

3. *`READLN;`* – это пустой оператор ввода, который используется в программах для небольшой остановки при их выполнении.

3.2.2. Операторы ввода на языке *C++*

C++ имеет средство ввода для всех типов данных – *`cin>>`*. Стандартный поток ввода *`cin`* вместе с двумя символами «больше» (*>>*), которые называются *оператором извлечения*, служит для считывания данных с клавиатуры.

Стандартный поток ввода *`cin>>`* не требует указания адреса переменной для числовых и символьных данных, а требует только имя переменной. Поток ввода *`cin>>`* самостоятельно определяет тип данных на основании вводимой информации.

```
/*Фрагмент программы ввода целочисленного значения с клавиа-
  туры в переменную a*/
int a;
cin>>a;
```

3.3. Операторы вывода

3.3.1. Операторы вывода на языке *Turbo Pascal*

Для вывода значений данных из памяти персонального компьютера на экран монитора предназначен оператор *WRITE* либо *WRITELN*.

Оператор имеет вид

либо `WRITE(b1,b2,...,bn);` либо `WRITELN(b1,b2,...,bn);`

где `b1,b2, ...,bn` – являются переменными, константами, выражениями или строками символов, заключёнными в апострофы.

Оператор вывода имеет назначения:

1. `WRITE(b1,b2, ...,bn);` - данный оператор осуществляет вывод на экран монитора значений *b*1,*b*2, ...,*b*n .

2. `WRITELN(b1,b2, ...,bn);` - данный оператор осуществляет сначала вывод на экран монитора значений *b*1,*b*2, ...,*b*n , а затем переход на новую строку.

3. `WRITELN;` - данный оператор осуществляет пропуск строки на экране.

Для того чтобы результаты выполнения программы были наглядными и «красивыми», в операторах вывода предусмотрены *форматы*. Формат указывается в операторе вывода вслед за идентификатором выводимого данного через двоеточие. Форматы зависят от типа данных.

Форматы данных

Форматирование данных при выводе будем рассматривать по типам данных: целого и вещественного.

Тип данного – *целый*.

Вывод данных целого типа допускается с форматом и без него. Для наглядности результатов можно использовать формат. Формат указывается в операторе вывода вслед за идентификатором выводимого данного через двоеточие.

Для данных целого типа формат имеет вид

`X:N`,

где `X` – это данное целого типа (константа, переменная, выражение);

N – это количество знаков в поле выводимого числа, включая знак числа.

Если формат задан бóльшим числом, чем цифр в числе, то перед числом выводится дополнительное количество пробелов (числа выравниваются справа).

```
{Программа с иллюстрацией вывода целого числа с форматированием}
VAR K:INTEGER; {переменная K описана как целая}
 BEGIN                  {начало вычислительного блока}
  K:=91;
  WRITELN('K=',K:8); {выводим на печать значение 91 в поле из 8
                                  знаков}
 END.
```

Тип данного – *вещественный***.**

Константа вещественного (действительного) типа может быть представлена на языке *Turbo Pascal* в двух видах: числом с фиксированной точкой и числом с плавающей точкой.

Число *с фиксированной точкой* изображается десятичным числом с дробной частью, которая отделяется от целой с помощью точки. Например: `-77.77`, `-11.11`, `22.22`, `123.456`.

В математике для изображения очень больших и малых чисел используется запись числа с десятичным порядком. На языке *Turbo Pascal* числа также могут изображаться с десятичным порядком, в этом случае они имеют вид

`mEp`,

где `m` – это *мантисса* числа, в качестве которой могут быть действительные числа с фиксированной точкой, содержащие знаки «+» или «-»;

`p` – это *порядок* числа, в качестве которого могут быть только целые числа, содержащие знаки «+» или «-»;

`E` – признак записи числа с плавающей точкой.

Формула 4 Пример 4. Представить вещественное число 987 как число с плавающей точкой.

Варианты ответов:

0.987E+03, 9.87E+02, 98.7E+01, 987E+00, 9870E-01, 98700E-02 .

Представление вещественного числа в конкретной программе можно получить, перемещая положение десятичной точки в мантиссе (точка «плывёт») и одновременно изменяя величину порядка.

Вывод данных вещественного типа допускается с форматом и без него. Если отсутствует формат, то число выводится с плавающей точкой – *мантисса, E, порядок*. Для наглядности результатов предусмотрены форматы. Формат указывается в операторе вывода вслед за идентификатором выводимой переменной через двоеточие.

Для данных вещественного типа формат имеет вид

`X:N:L`,

где `X` – это данное вещественного типа (константа, переменная, выражение);

`N` – это количество знаков в поле выводимого числа, включая знак числа, целую часть, точку и дробную часть;

`L` – это поле дробной части.

```
{Программа вывода вещественного числа с форматированием на
 языке Turbo Pascal}.
VAR S:REAL; {переменная S описана как вещественная}
  BEGIN      {начало вычислительного блока}
    S:=91.123456;
    WRITELN('S=',S:10:2);  {выводим на печать значение
                            91.123456 в поле из 10 позиций, включая
                            точку, и двух (2) знаков после неё}
 END.
```

3.3.2. Операторы вывода на языке *C++*

C++ имеет средство вывода для всех типов данных – `cout<<`. Стандартный поток вывода `cout` вместе с двумя символами «меньше» (<<), которые называются *оператором вставки*, служит для отображения значений констант и переменных или символов на экране без использования указателей формата.

Оператор вставки «<<» указывает `cout` отобразить информацию, которая помещена после них. Информация может быть представлена в виде имени константы, имени переменной или в виде набора символов, которые обязательно надо заключать в кавычки. Чтобы перейти к новой строке, надо добавить управляющий код `"\n"` (см. фрагмент программы).

```
/*Фрагмент программы вывода на экран целочисленного значения
  переменной a1 и вещественного значения y*/
int a1; float y;
a1=55; y=44.44;
cout<<"a1="<<a1<<"\n";
cout<<"y="<<y;
```

3.4. Контрольные вопросы

1. Какой алгоритм называется алгоритмом линейной структуры?
2. Из чего может состоять арифметическое выражение?
3. Назовите арифметические операции, которые используются в языках программирования.
4. Назовите дополнительные арифметические операции в языке *Turbo Pascal* (в языке *C++*).

5. Объясните, в чём состоит операция целочисленного деления на языке *Turbo Pascal.*

6. Объясните, в чём состоит операция «остаток от целочисленного деления» на языке *Turbo Pascal* (на языке *C++*).

7. Перечислите стандартные функции на языке *Turbo Pascal* (на языке *C++*).

8. Что означает термин «Обычная скобочная запись арифметического выражения»?

9. В чём отличие польской записи арифметического выражения?

10. Как называется стандартная функция `RANDOM(N)` и в чём её назначение?

11. С помощью какого оператора вводятся значения данных на языке *Turbo Pascal* (на языке *C++*)?

12. С помощью какого оператора выводятся значения данных на языке *Turbo Pascal* (на языке *C++*)?

13. Зачем используется форматирование выводимых значений в операторе вывода на языке *Turbo Pascal*?

Глава 4.

АЛГОРИТМЫ ВЕТВЛЕНИЯ

Рассмотрим задачу, в которой встречается следующее условие: **ЕСЛИ** днём погода будет хорошая, **ТО** буду играть в футбол, **ИНАЧЕ**, **ЕСЛИ** днём будет дождь, **ТО** буду готовиться по математике; вечером буду готовить ужин.

Блок-схема задачи приведена на рис.3.

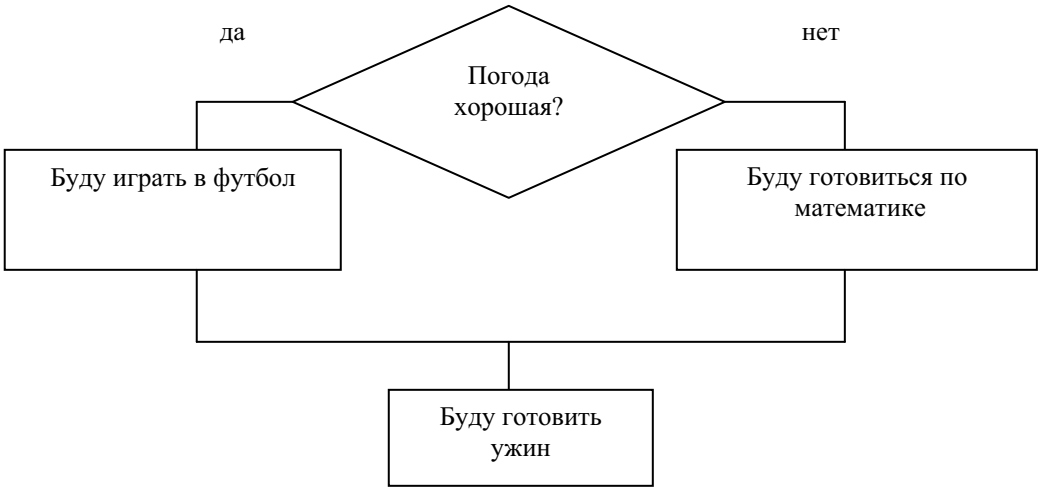


Рис. 3. Блок-схема разветвляющегося алгоритма

Алгоритм ветвления (разветвляющейся структуры) - это алгоритм, в котором требуется организовать выбор выполнения последовательности действий в зависимости от каких-либо условий.

Для организации алгоритма ветвления в языке *Turbo Pascal* используются операторы:

- оператор безусловного перехода `GOTO`;
- оператор условного перехода `IF`;
- оператор выбора `CASE`.

Для организации алгоритма ветвления в языке *C++* используются операторы:

- оператор безусловного перехода `goto`;
- оператор условного перехода `if`;
- оператор выбора `switch`.

4.1. Оператор безусловного перехода GOTO

Оператор безусловного перехода GOTO означает «перейти к» и применяется в случаях, когда после выполнения некоторого оператора надо выполнить не следующий по порядку, а другой, отмеченный меткой оператор.

Оператор GOTO можно применить в следующих случаях:

- для обработки ошибок, которые могут возникнуть в программе;
- для многократного выполнения программы;
- для раннего (преждевременного) выхода из цикла.

{Программа многократного ввода значений переменной (с использованием оператора безусловного перехода GOTO) на языке *Turbo Pascal*}

```
LABEL lab1;
  VAR Z:REAL;
    BEGIN
      lab1: WRITELN('вводи одно значение Z');
            READLN(Z);
            {операторы вычислений}
            GOTO lab1;
    END.
```

4.2. Оператор условного перехода IF … THEN … ELSE

Оператор условного перехода имеет следующий вид:

IF <*логическое_выражение*> THEN <*оператор*-1>
ELSE <*оператор*-2>;

где <*логическое_выражение*> состоит из выражения слева, соединённого *операцией отношения* с выражением или константой справа.

Для записи логических выражений используются операции отношения (табл. 9) и логические операции (табл. 10).

Таблица 9

Операции отношения

Действие	Операция		Пример записи	
	Turbo Pascal	*C++*	*Turbo Pascal*	*C++*
равно	=	==	A=B	A==B
не равно (≠)	<>	!=	A<>B	A!=B
меньше	<	<	A<B	A<B

больше	>	>	A>B	A>B
меньше или равно (≤)	<=	<=	A<=B	A<=B
больше или равно (≥)	>=	>=	A>=B	A>=B

Таблица 10

Логические операции

Действие	Операция		Пример записи	
	Turbo Pascal	*C++*	*Turbo Pascal*	*C++*
отрицание “НЕ”	NOT	!	NOT A	!A
логическое “И”	AND	&&	A AND B	A&&B
логическое “ИЛИ”	OR	||	A OR B	A||B
исключающее “ИЛИ”	XOR		A XOR B	

Условный оператор читается так:

ЕСЛИ (IF) <*логическое выражение*> **ИСТИННО** (TRUE), **ТО** (THEN) выполняется *оператор*-1 (идём по веточке «ДА»); **ИНАЧЕ** (ELSE) **ЕСЛИ** (IF) *логическое выражение* **ЛОЖНО** (FALSE) выполняется *оператор*-2 (идём по веточке «НЕТ»).

Формула 5Пример 5. Составить программу вычисления выражения

$$Y = \begin{cases} X^3 - 3, & если \quad X > 0 \\ 2 - X^2, & если \quad X \le 0 \end{cases}$$

Блок-схема разветвляющегося алгоритма решения задачи по условию примера 5 приведена на рис.4.

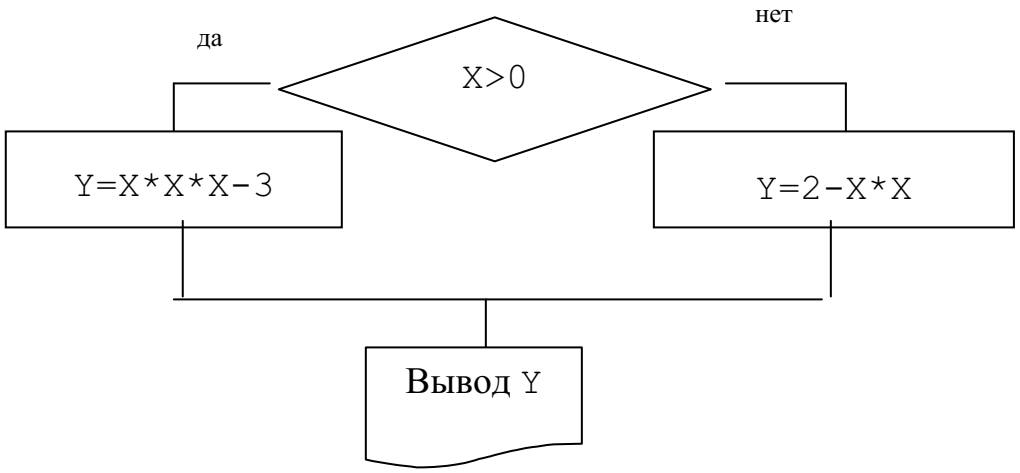


Рис. 4. Блок-схема разветвляющегося алгоритма

{Фрагмент программы по блок-схеме рис. 4 на языке *Turbo Pascal*}

```
IF X>0 THEN Y:=Y=X*X*X-3
        ELSE Y:=Y=2-X*X;
WRITELN('Y=',Y);
```

/*Фрагмент программы по блок-схеме рис. 4 на языке *C*++*/

```
if x>0 y=x*x*x-3;
       else y=2-x*x;
cout<<"y="<<y;
```

Способы оформления условного оператора `IF` (примеры даны на языке *Turbo Pascal*).

1. Оператор `IF` может включать составной оператор, состоящий из ключевого слова `BEGIN`, после которого идёт последовательность операторов, и ключевого слова `END`. Составные операторы могут использоваться после ключевых слов `THEN` и `ELSE` (рис. 5).

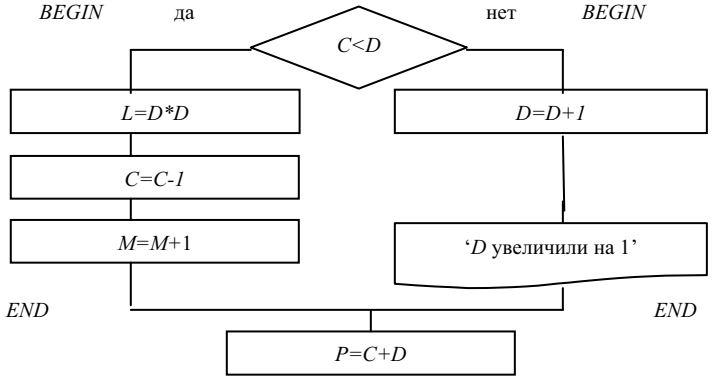


Рис. 5. Блок-схема разветвляющегося алгоритма

В таких случаях формат условного оператора будет следующим:

```
IF <логическое_выражение> THEN
                          BEGIN
                            {оператор-1T; . . .оператор-nT;}
                          END
                    ELSE
                          BEGIN
                            {оператор-1E; . . . оператор-nE;}
                          END;
```

Условный оператор IF с использованием составных операторов можно представить схемой, которая приведена на рис. 6.

Внутри составных операторов могут быть условные операторы, содержащие простые или составные операторы.

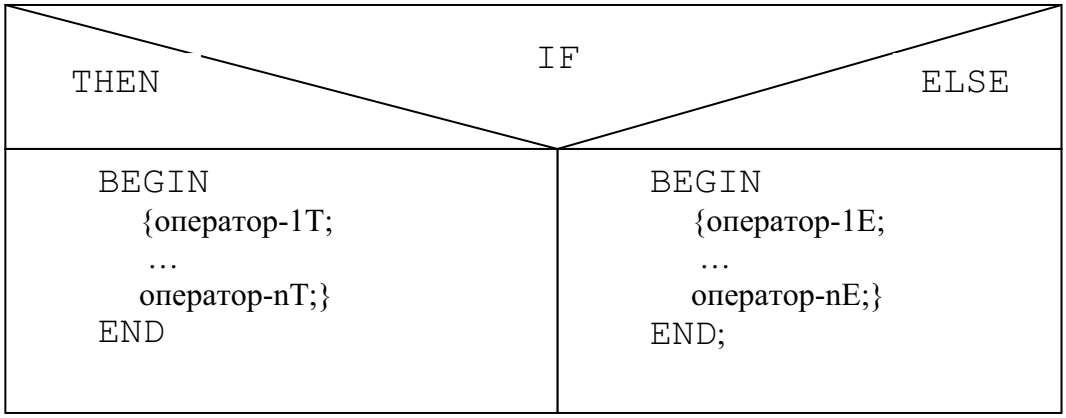


Рис. 6. Схема условного оператора IF с использованием составных операторов

{Фрагмент программы по блок-схеме рис. 5}

```
IF C<D THEN BEGIN
                L:=D*D;
                C:=C-1
                M:=M+1;
              END
       ELSE BEGIN
                D:=D+1;
                WRITELN('D увеличили на 1');
              END;
P:=C+D;
```

2. Условный оператор допускает краткую форму, которая соответствует блок-схеме рис. 7.

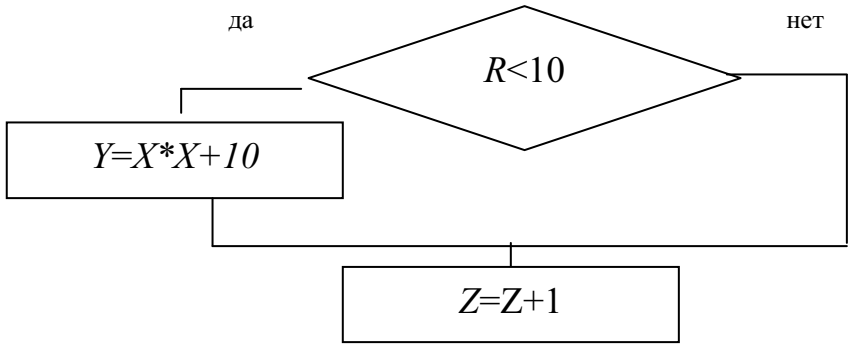


Рис. 7. Блок-схема разветвляющегося алгоритма

Формат условного оператора IF по блок-схеме рис. 7 будет следующим:

```
IF <логическое_выражение> THEN
                          BEGIN
                             {оператор-1T; . . .
                              оператор-nT;}
                          END;
```

{Фрагмент программы по блок-схеме рис. 7}

```
IF R<10 THEN Y:=X*X+10;
Z:=Z+1;
```

Для разветвляющегося алгоритма рис. 7 условный оператор читается так: **ЕСЛИ** условие R<10 ИСТИННО (*TRUE*), **ТО** вычисляется значение *Y* и происходит передача управления на вычисление значения Z:=Z+1. ЕСЛИ условие R<10 ЛОЖНО (*FALSE*), то есть R>=10, то сразу выполнится оператор присваивания Z:=Z+1.

3. Записать условный оператор IF по следующему условию:

$$Z=\begin{cases}12-X^2, & \text{если} \quad X>0;\\ 85 \quad\;\; , & \text{если} \quad X=0;\\ X^2-12, & \text{если} \quad X<0.\end{cases}$$

Алгоритм решения приведён на блок-схеме рис. 8.

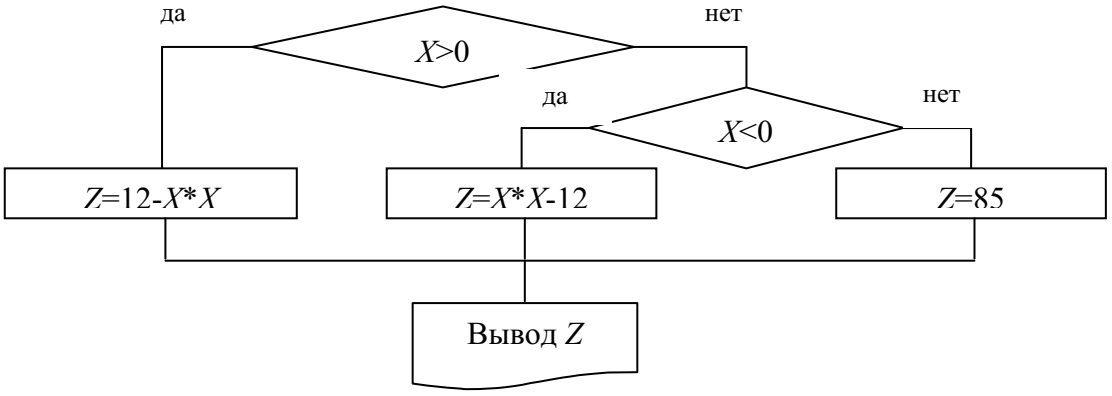


Рис. 8. Блок-схема разветвляющегося алгоритма

{Первый способ записи фрагмента программы по блок-схеме рис. 8}

```
IF X>0 THEN Z:=12-X*X
       ELSE IF X<0 THEN Z:=X*X-12
```

```
                    ELSE Z:=85;
WRITELN('Z= ',Z);
```

{Второй способ записи фрагмента программы по блок-схеме рис. 8}

```
IF X>0 THEN Z:=12-X*X;
IF X<0 THEN Z:=X*X-12;
IF X=0 THEN Z:= 85;
WRITELN('Z= ',Z);
```

4. Записать условный оператор `IF` по следующему условию:

$$Z=\begin{cases}50-X^2, & \text{если} \quad X>10;\\ 50 \quad\quad , & \text{если} \quad X<=5;\\ X^2-50, & \text{если} \quad 5<X\le 10\end{cases}$$

Алгоритм решения представлен блок-схемой рис. 9.

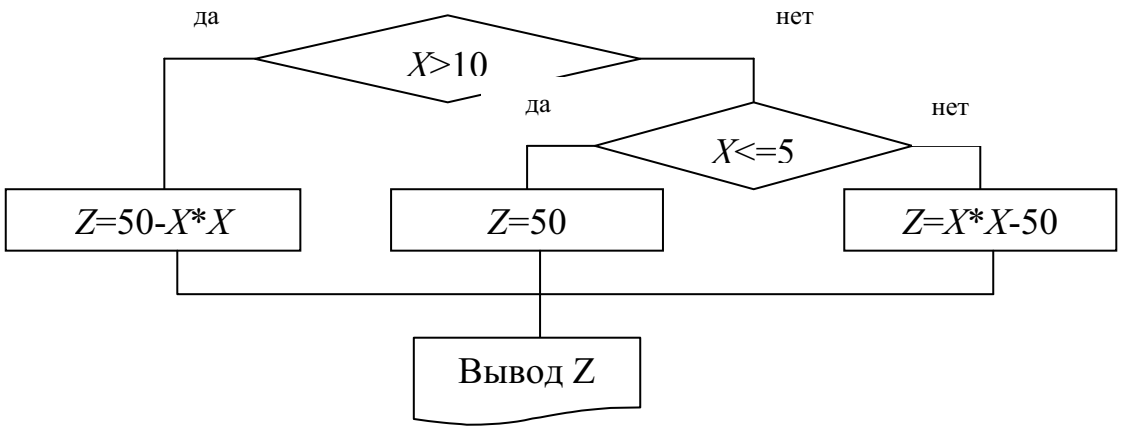


Рис. 9. Блок-схема разветвляющегося алгоритма

{Первый способ записи фрагмента программы по блок-схеме рис. 9}

```
IF X>10 THEN Z:=50-X*X
        ELSE IF X<=5 THEN Z=50
                     ELSE Z:=X*X-50;
WRITELN('Z= ',Z);
```

{Второй способ записи фрагмента программы по блок-схеме рис. 9}

```
IF X>10 THEN Z:=50-X*X;
IF X<=5 THEN Z:=50;
IF (X>5) AND (X<=10) THEN Z:=X*X-50;
WRITELN('Z= ',Z);
```

Во втором способе в записи <*логического_выражения*> используется логическая операция AND, приведённая в табл. 10.

4.3. Оператор выбора

Оператор выбора позволяет выбрать один вариант из многих продолжений программы. Выбор осуществляется по значению <*ключа выбора*>, который называют ещё *селектором.* <*Ключ выбора*> – это выражение типа INTEGER или CHAR.

4.3.1. Оператор выбора CASE на языке *Turbo Pascal*

Оператор выбора имеет следующий вид:

```
CASE <ключ выбора> OF
    <константа выбора-1>: BEGIN <операторы-1> END;
    <константа выбора-2>: BEGIN <операторы-2> END;
                          …
    <константа выбора-N>: BEGIN <операторы- N > END
    ELSE
                BEGIN <операторы-ELSE> END;
END;
```

Оператор выбора CASE работает следующим образом:

- вначале вычисляется значение выражения <*ключа выбора*>;
- затем в последовательности <*список выбора*> отыскивается такой оператор, которому предшествует <*константа выбора*>, равная вычисленному значению <*ключа выбора*>. Найденный оператор выполняется, после чего оператор выбора завершает свою работу.

Если в <*списке выбора*> не будет найдена нужная константа, то управление передаётся оператору, стоящему за ключевым словом ELSE.

Оператор выбора CASE можно представить схемой, которая приведена на рис. 10.

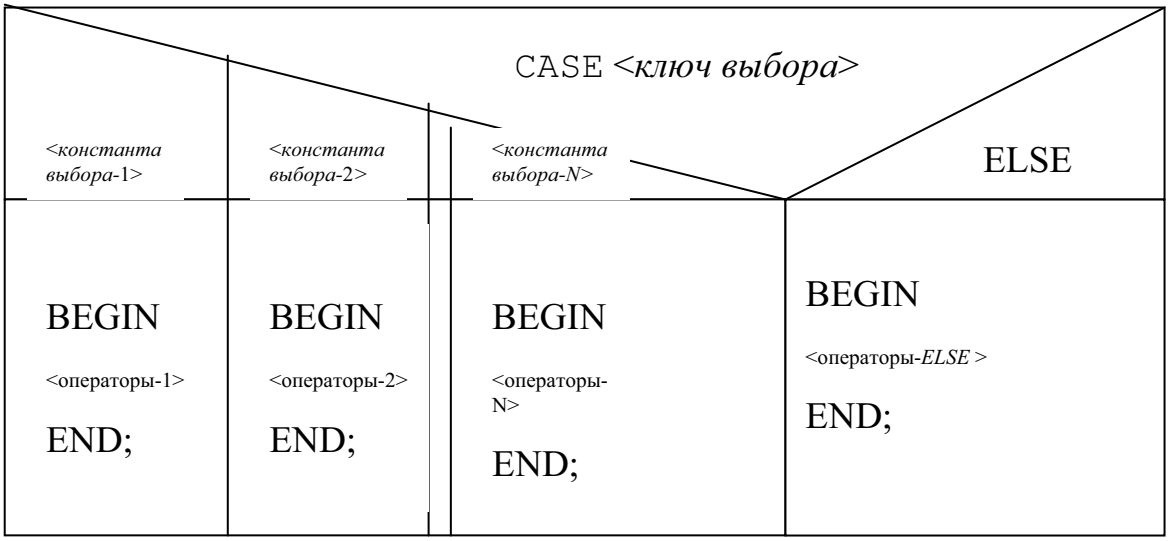


Рис. 10. Схема оператора выбора *CASE*

Формула 6Пример_6. Составить программу вычисления выражения

$$Y = \begin{cases} X^2 - 10, & \text{если} \quad X = 1; \\ 20 \cdot X, & \text{если} \quad X = 2; \\ 30 + X, & \text{если} \quad X = 3. \end{cases}$$

Блок-схема разветвляющегося алгоритма решения задачи приведена на рис. 11.

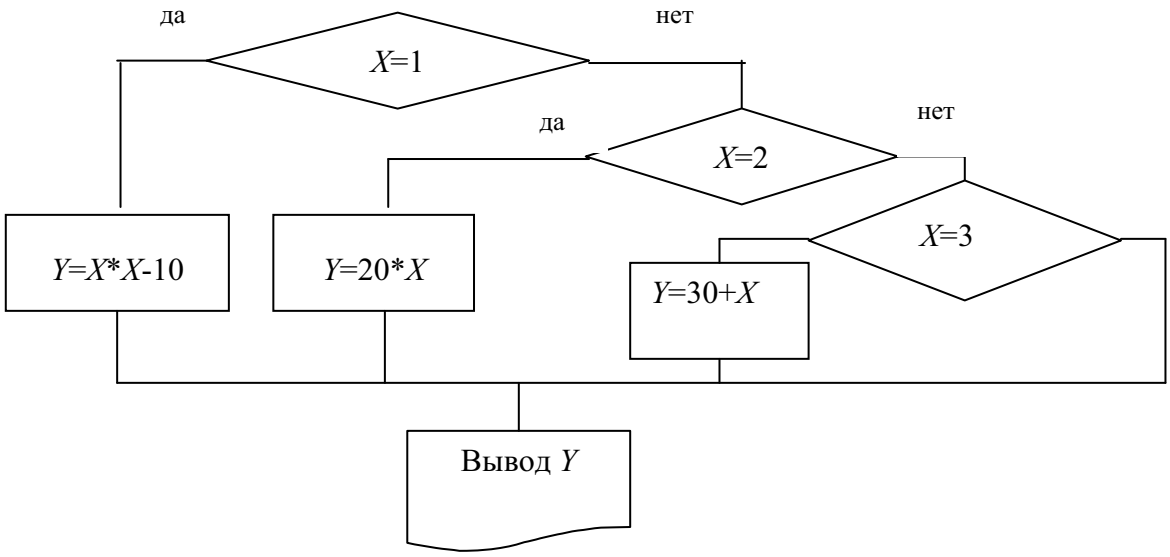


Рис. 11. Блок-схема разветвляющегося алгоритма

{Фрагмент программы по блок-схеме рис. 11}

```
CASE X OF
    1: BEGIN
         X:=X*X-10;
         END;
    2: BEGIN
         Y:=20*X;
         END;
    3: BEGIN
         Y:=30+X;
         END:
END;
WRITELN('Y=',Y:8:2);
```

4.3.2. Оператор выбора `switch` и `break` на языке *C++*

Оператор выбора `switch` имеет следующий вид:

`switch(`<*ключ выбора*>**)**

{

  `case` <*константа выбора*-1>: <оператор-1>; `[break;]`

  `case` <*константа выбора-i*>: <оператор-i>; `[break;]`

  …

  `case` <*константа выбора-N*>: <оператор- *N* >; `[break;]`

  `[default; <оператор-N+1 >;]`

}

Оператор `switch` выполняется по следующему правилу: сначала вычисляется значение выражения <*ключа выбора*>, при этом тип значения должен быть одним из целых: *int*, *unsigned int*, *long int, long unsigned* и *char*. Вычисленное значение <*ключа выбора*> сравнивается со значениями констант или константных выражений <*константа выбора*-1>, <*константа выбора-i*>, …, <*константа выбора-N*>. При совпадении значения <*ключа выбора*> с <*константа выбора-i*> выполняется оператор <оператор-i>.

Управление будет передано сразу другому оператору после оператора `switch`, если в *i*-ой ветви присутствует оператор `break`. Если оператор `break` будет отсутствовать, то будут выполняться операторы в ветвях *i*+1, *i*+2 и так далее до тех пор, пока в них не встретится оператор `break` или не будет выполнен оператор <оператор-*N*+1 >.

Если значение <*ключа выбора*> не совпало ни с одной из констант, то выполнится оператор в ветви, которая помечена `default`.

При отсутствии ветви `default` будет выполняться оператор, который следует за `switch`.

```
     /*Фрагмент программы по блок-схеме рис. 11*/
switch(x)
  {
    Case 1: y=x*x-10; break;
    Case 2: y=20*x; break;
    Case 3: y=30+x; break;
  }
cout<<"y="<<y;
```

4.4. Контрольные вопросы

1. Какой алгоритм называется алгоритмом ветвления (разветвляющейся структуры)?
2. В каких случаях используется оператор безусловного перехода `GOTO`?
3. Какие ветвления в программе описывает оператор условного перехода `IF`?
4. В каких случаях лучше использовать в программе оператор выбора: на языке *Turbo Pascal* – `CASE`, на языке *C++* - `switch`?
5. Приведите примеры различных способов оформления условного оператора `IF`.
6. Приведите обозначения операций отношения (сравнения) в условном операторе `IF`.
7. Приведите обозначения логических операций.
8. Опишите этапы работы оператора выбора: на языке *Turbo Pascal* – `CASE`, на языке *C++* - `switch`.

Глава 5.

ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ

Часто при решении задач приходится многократно выполнять одни и те же действия или вычислять по одним и тем же математическим зависимостям при различных значениях входящих в них значений переменных. Такие многократно повторяемые участки вычислительного процесса называются *циклами*.

Циклический алгоритм – это алгоритм, в котором выполняются одни и те же действия при регулярно[11] изменяющейся переменной цикла.

Использование циклов позволяет существенно сократить схему алгоритма и длину соответствующей ему программы.

Циклы принято различать:

- циклы с заданным числом повторений (*конечные* циклы);
- циклы с неизвестным числом повторений (*итерационные* циклы).

5.1. Конечные циклы

Рассмотрим задачу, в которой необходимо обработать следующую ситуацию: подготовиться к экзамену, затратив на это 60 минут. Алгоритм задачи оформим блок-схемой (рис. 12).

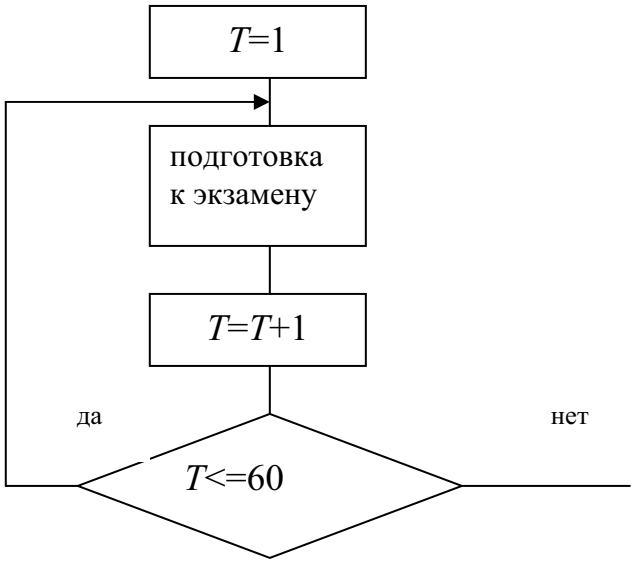


Рис. 12. Блок-схема циклического процесса

[11] Регулярно – это значит, что задана закономерность изменения.

Блок-схема цикла с заданным числом повторений приведена на рис. 13.

Для операторов цикла с заданным числом повторений характерны следующие особенности:

1. Имеется *<переменная_цикла>*.

2. *<Переменной_цикла>* должно быть присвоено *<начальное значение>* до входа в циклическую часть.

3. Внутри цикла *<переменная_цикла>* должна изменяться на *<шаг_приращения>*.

4. *<Переменная_цикла>* должна иметь *<конечное значение>* для организации выхода из цикла.

5. Наличие условного оператора на достижение *<конечного значения>*. Если *<конечное значение>* не достигнуто, то управление автоматически передаётся оператору, стоящему за оператором начала цикла.

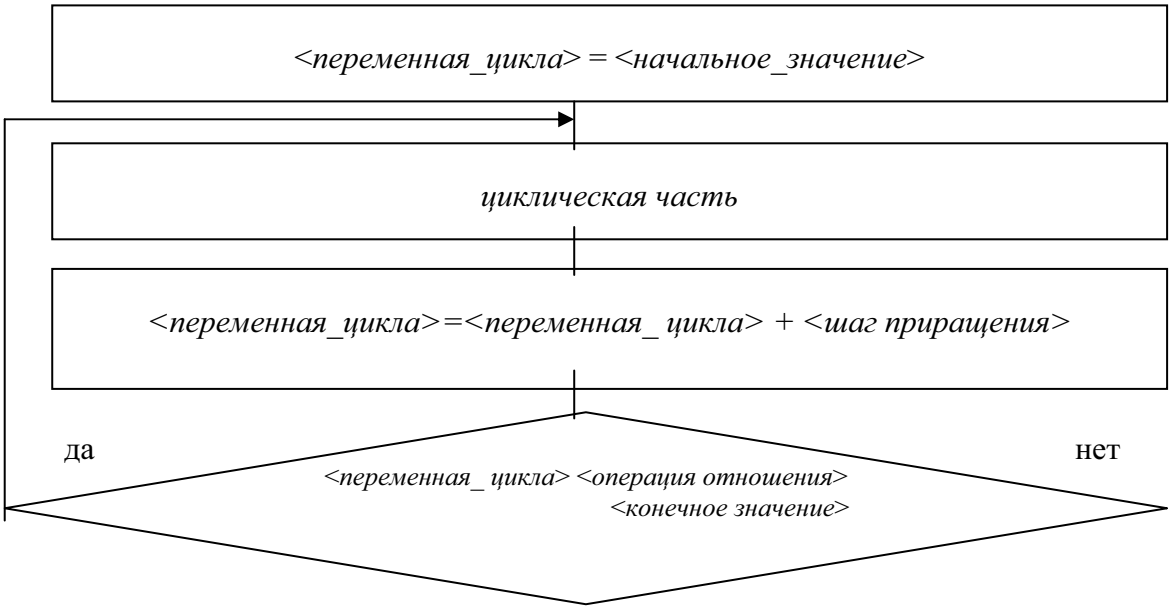


Рис. 13. Блок-схема цикла с заданным числом повторений

5.1.1. Операторы цикла в языке *Turbo Pascal*

В алгоритмическом языке *Turbo Pascal* имеется три вида операторов цикла:

1. Оператор цикла с параметром `FOR...TO...DO...`
   либо
   `FOR...DOWNTO...DO...`

2. Оператор с предварительным условием `WHILE...DO`.

3. Оператор цикла с последующим условием REPEAT...UNTIL.

Оператор цикла с параметром FOR...TO...DO

Оператор цикла с параметром FOR…TO…DO можно применить, если:

- <*начальное значение*>, <*конечное значение*> – это выражения целого типа;
- <*начальное значение*> не больше (≤) <*конечного значения*>;
- <шаг приращения> равен +1;
- <*переменная цикла*> объявлена в разделе VAR как переменная целого типа – INTEGER или CHAR.

Оператор цикла с параметром FOR…TO имеет следующий вид:

FOR <*переменная цикла*>**:=**<*начальное значение*> TO
<*конечное значение*> DO BEGIN
{операторы циклической части}
END;

где FOR – переводится как «для»;
TO – переводится как «до»;
DO – переводится как «выполнить».

{Программа по блок-схеме рис. 12}

```
VAR T:INTEGER;
 BEGIN
  FOR T:=1 TO 60 DO BEGIN
                WRITELN('подготовка к экзамену');
                      END;
 END.
```

Оператор цикла с параметром FOR...DOWNTO...DO

Оператор цикла с параметром FOR…DOWNTO…DO можно применить, если:

- <*начальное значение*>, <*конечное значение*> – это выражения целого типа;
- <*начальное значение*> не меньше (≥) <*конечного значения*>;
- <*шаг приращения*> равен (-1);
- <*переменная цикла*> объявлена в разделе VAR как переменная целого типа – INTEGER или CHAR.

Оператор цикла с параметром FOR...DOWNTO имеет следующий вид:

FOR <*переменная цикла*>**:=**<*начальное значение*> DOWNTO
<*конечное значение*> DO BEGIN
{операторы циклической части}
END;

где FOR – переводится как «для»;
DOWNTO – переводится как «с вычитанием 1 до»;
DO – переводится как «выполнить».

{Программа по блок-схеме рис. 12}

```
VAR T:INTEGER;
 BEGIN
  FOR T:=60 DOWNTO 1 DO BEGIN
             WRITELN('подготовка к экзамену');
                        END;
 END.
```

Оператор цикла с предварительным условием WHILE..DO

Оператор цикла с предварительным условием (предусловием) WHILE… DO применяется, если:

- или <*начальное значение*>, или/и <*конечное значение*> - это выражения вещественного типа;
- или/и <*шаг приращения*> не равен ($\neq$) ± 1.

Оператор цикла с предварительным условием WHILE…DO имеет следующий вид:

<*переменная цикла*>:=<*начальное значение*>;
WHILE <*переменная цикла*><*знак отношения*><*конечное значение*> DO
BEGIN
{операторы циклической части}
<*переменная_цикла*>:=<*переменная_цикла*>+<*шаг приращения*>;
END;
где WHILE – переводится как «пока».

Оператор цикла WHILE…DO действует следующим образом:

- <переменной цикла> присваивается <начальное значение>;

• перед входом в цикл предварительно проверяется значение логического выражения, стоящего за ключевым словом WHILE. Пока оно ИСТИННО (TRUE), выполняются операторы циклической части;

• в циклической части обязательно должен присутствовать оператор изменения значения <*переменной цикла*>. По окончании циклической части управление вновь будет передано оператору WHILE;

• выход из цикла произойдет, как только логическое выражение за WHILE станет ЛОЖНЫМ (FALSE).

```
{Программа по блок-схеме рис. 12}
VAR T:INTEGER;
 BEGIN
  T:=1; {присваивание начального значения переменной цикла T}
  WHILE T<=60 DO BEGIN
    WRITELN('подготовка к экзамену');
    T:=T+1; {изменение переменной цикла T на шаг +1}
                  END;
 END.
```

Внимание! После оператора WHILE записывается логическое выражение для *выполнения* цикла!

Оператор цикла с последующим условием REPEAT...UNTIL

Оператор цикла с последующим условием (постусловием) *REPEAT ... UNTIL* применяется, если:

- или <*начальное значение*>, или/и <*конечное значение*> – выражения вещественного типа;

- или/и <*шаг приращения*> не равен ($\neq$) ± 1.

Оператор цикла с последующим условием *REPEAT ... UNTIL* имеет следующий вид:

<*переменная цикла*>:=<*начальное значение*>;
REPEAT
{операторы циклической части}
<*переменная_цикла*>:=<*переменная_цикла*>+<*шаг приращения*>;
UNTIL <*переменная_цикла*> <*операция отношения*> <*конечное значение*>;

где *REPEAT* – переводится как «повтор»;
UNTIL – «до тех пор».

Оператор цикла *REPEAT ... UNTIL* действует следующим образом:

- <переменной цикла> присваивается <начальное значение>;
- операторы циклической части выполняются повторно до тех пор, пока значение логического выражения, стоящего за ключевым словом *UNTIL*, ЛОЖНО (*FALSE*);
- в циклической части обязательно должен присутствовать оператор изменения значения <*переменной цикла*>;
- выход из цикла произойдет, как только логическое выражение за *UNTIL* станет ИСТИННЫМ (*TRUE*).

```
{Программа по блок-схеме рис. 12}
VAR T:INTEGER;
 BEGIN
  T:=1; {присваивание начального значения переменной цикла T}
  REPEAT
   WRITELN('подготовка к экзамену');
   T:=T+1; {изменение переменной цикла T на шаг +1}
  UNTIL T>120;
 END.
```

Внимание! После оператора *UNTIL* записывается логическое выражение для *выхода* из цикла!

5.1.2. Операторы цикла в языке *C*++

В алгоритмическом языке *C*++ имеются три оператора цикла:

1. Оператор цикла с предусловием **while**.
2. Оператор цикла с постусловием **do**.
3. Оператор параметрического цикла **for(...)**.

Все три оператора построены по схеме

<*заголовок_цикла*>
<*тело_цикла*>

Во всех трёх операторах цикла <*тело_цикла*> – это либо отдельный, либо составной оператор, то есть последовательность операторов, которые заключены в операторные фигурные скобки {}.

Цикл **while** (цикл с предусловием) имеет вид:

while (<*выражение_условие*>)
<*тело_цикла*>

В качестве *<выражения_условия>* используются логическое или арифметическое выражение. Если оно истинно, то есть равно 1, то *<тело_цикла>* выполняется до тех пор, пока *<выражение_условие>* не станет ложным, то есть равным 0.

Проверка истинности выражения осуществляется до каждого выполнения *тело_цикла.* Поэтому, если при первом вычислении *<выражения_условия>* получается значение 0 (ЛОЖНОЕ - *FALSE*), то *<тело_цикла>* не выполнится ни разу.

```
 /*Фрагмент программы по блок-схеме рис. 12*/
t=1;
while (t<=60)
  {   cout<<'подготовка к экзамену';
      t++
  }
```

Цикл **do** (цикл с постусловием) имеет вид:

do

<тело_цикла>

while (*<выражение_условие>*)

В качестве *<выражения_условия>* используются логическое или арифметическое выражение, как и в цикле **while**. В цикле **do** *<тело_цикла>* всегда выполнится по крайней мере один раз. После каждого выполнения *<тела_цикла>* проверяется истинность *<выражения_условия>,* и если оно равно 0 (ЛОЖНОЕ - *FALSE*), то цикл заканчивается. Если *<выражение_условие>* равно 1, то есть истинно, то цикл выполняется вновь.

```
 /* Фрагмент программы по блок-схеме рис. 12*/
t=1;
do
  {   cout<<'подготовка к экзамену';
      t++
  }
while (t<=60);
```

Цикл **for** (цикл параметрический) имеет вид:

for(*<выражение_1>;<выражение_условие>;<выражение_3>*)

<тело_цикла>

Первое и третье выражения в операторе **for** могут состоять из нескольких выражений, которые разделяются запятыми.

<*Выражение_1*> определяет действия, которые выполняются до начала цикла, то есть задаёт начальные условия для цикла.

<*Выражение_условие*> – это логическое или арифметическое выражение. Оно определяет условие окончания или продолжения цикла. Если оно истинно, то есть равно 1, то выполняется <*тело_цикла*>, а затем вычисляется <*выражение_3*>.

<*Выражение_3*> задаёт необходимые для следующей итерации изменения параметров или любых других переменных <*тела_цикла*>. После выполнения <*выражения_3*> вычисляется истинность <*выражения_условия*>, и всё повторяется.

Таким образом, <*выражение_1*> вычисляется только один раз, а <*выражение_условие*> и <*выражение_3*> вычисляются после каждого выполнения тела цикла. Цикл продолжается до тех пор, пока не станет ложным <*выражение_условие*>.

```
/* Фрагмент программы по блок-схеме рис. 12*/
for(t=1;t<=60;t++)
 {   cout<<'подготовка к экзамену';
 }
```

5.2. Алгоритмы обработки значений переменных в цикле

Вычисление суммы значений переменной в цикле

В математике *суммирование* значений переменной записывается выражением

$$S = \sum_{i=1}^{10} y .$$

При программировании сумма будет вычисляться по формуле

$$S=S+Y,$$

где S – переменная, которая используется для накопления суммы;

Y – слагаемое.

По выражению $S=S+Y$ каждое новое значение S получается из предыдущего добавлением очередного слагаемого.

Суммирование значений переменной оформляется в цикле!

Перед началом цикла начальное значение суммы нужно обязательно обнулить ($S=0$), то есть очистить содержимое ячейки по адресу S, так как при выделении ячеек памяти под переменные, которые описаны в начале программы, очистки содержимого ячеек не происходит.

Вычисление произведения значений переменной в цикле

В математике *произведение* значений переменной записывается выражением

$$P = \prod_{i=1}^{10} y \ .$$

При программировании произведение будет вычисляться по формуле

$$P=P*Y,$$

где P – переменная, которая используется для накопления произведения;

Y – сомножитель.

По выражению $P=P*Y$ каждое новое значение P получается из предыдущего умножением на очередной сомножитель.

Произведение значений переменной оформляется в цикле!

Перед началом цикла начальное значение произведения приравниваем единице, то есть P=1, чтобы не влиять на результаты перемножения.

Подсчёт количества значений переменных по заданному условию

Подсчёт количества значений переменных по заданному условию ведётся в промежуточной переменной, например *CH*, которую назовём *счётчиком*, по формуле $CH=CH+1$. По выражению $CH=CH+1$ каждое новое значение *CH* получается из предыдущего добавлением единицы. До начала подсчёта значение счётчика обнуляем, то есть $CH=0$.

*Пример_*. Составить программу вычисления среднего арифметического 20-ти значений переменной *W*, произведения 20-ти значений переменной *W*, не равных нулю, подсчёта количества значений *W*, равных 1. Значения *W* вводить с клавиатуры.

Для записи программы введём следующие обозначения:

S – промежуточная переменная, где накапливается сумма;

P – промежуточная переменная, где накапливается произведение;

CH – промежуточная переменная, где накапливается количество заданных значений;

I – переменная цикла, изменяющаяся от 1 до 20-ти.

Блок-схема алгоритма приведена на рис. 14.

{программа по блок-схеме рис. 14 на языке *Turbo Pasal*}

```
VAR I,CH:INTEGER;
    S,P,W:REAL;
 BEGIN
  S:=0;   {обнуление суммы}
  P:=1;   {произведение приравниваем 1}
  CH:=0; {обнуление счётчика}
  FOR I:=1 TO 20 DO BEGIN
       WRITELN('вводи W, равное 1 или неравное 1');
       READLN(W);
       S:=S+W;
       IF W<>0 THEN P:=P*W;
       IF W=1 THEN CH:=CH+1;
                     END;
  WRITELN('S/20=',S/20,' P=',P,' CH=',CH);
  END.
```

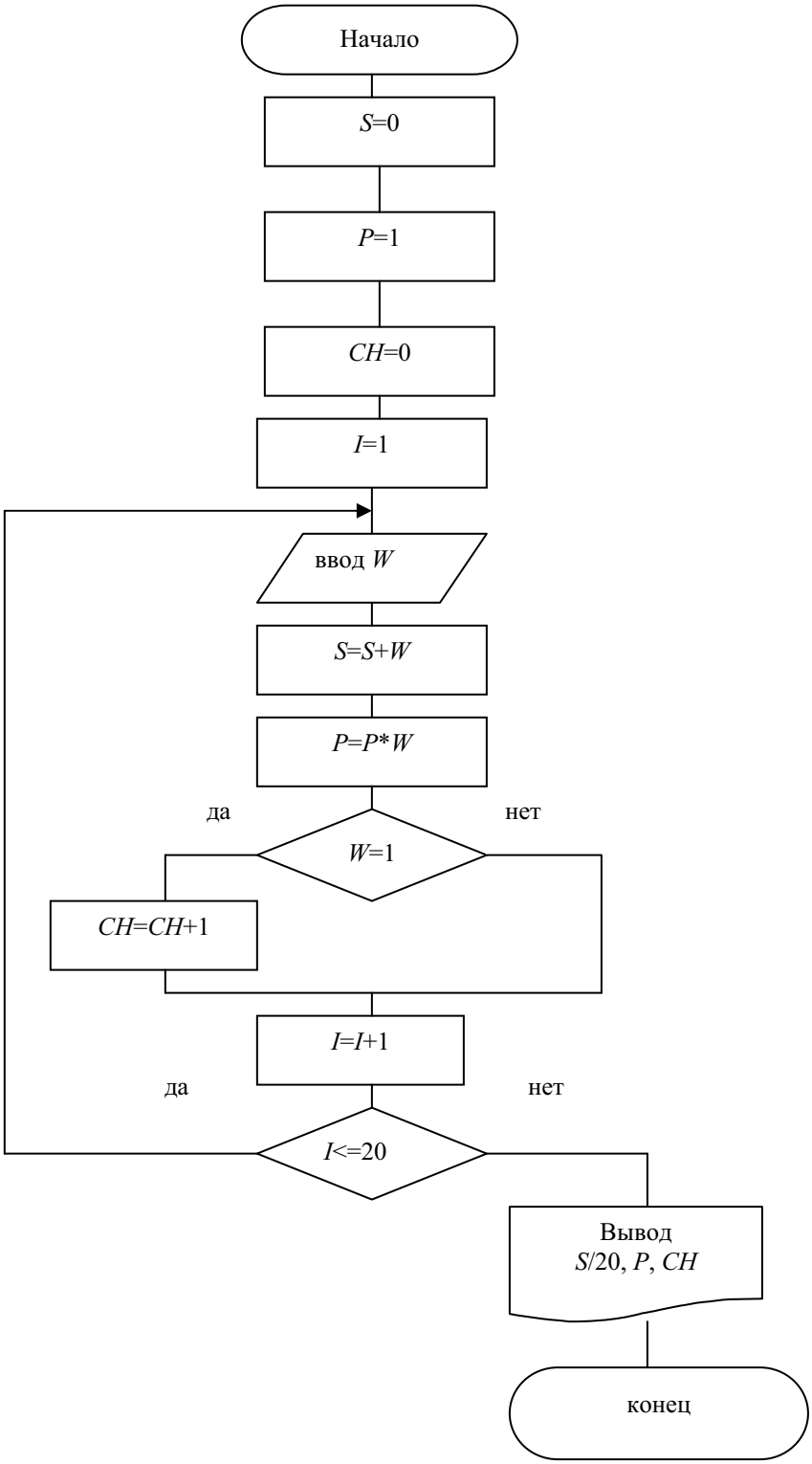


Рис. 14. Блок-схема алгоритма нахождения суммы, произведения, подсчёта количества единичных значений 20-ти значений переменной *W*

{программа по блок-схеме рис. 14 на языке *C*++}

```
#include <iostream.h>
```

```
#include <math.h>
void main()
{    float w;
     float s=0;
     float p=1;
     int i;
     int ch=0;
for (i=1;i<=20;i++)
   {cout<<"w="; cin>>w;
     s=s+w;
     if (w!=0) then p=p*w;
     if (w==1) then ch=ch+1;
   }
     cout<<"s/20="<<s/20;
     cout<<"p="<<p;
     cout<<"ch="<<ch;
getch();
}
```

5.3. Итерационные циклы

При выполнении некоторых программ количество этапов вычислений заранее неизвестно, например, при вычислении значения некоторой переменной с заданной точностью ε в программах, составленных по алгоритмам численных методов. В таких алгоритмах используют итерационные[12] циклы, так как конечное число этапов вычислений (итераций) заранее неизвестно.

На рис. 15 приведена блок-схема алгоритма итерационного цикла.

В блок-схеме рис. 15 использованы следующие обозначения переменных:

eps - точность вычисления переменной X;

Xtek - текущее приближение переменной X; вычисляется по зависимости $Xtek = F(Xpred)$;

Xpred - предыдущее приближение переменной X;

D - модуль разности между предыдущим значением X и текущим значением X, то есть $D = |Xpred - Xtek|$;

N - счетчик количества итераций.

[12] Итерация – это этап выполнения алгоритма.

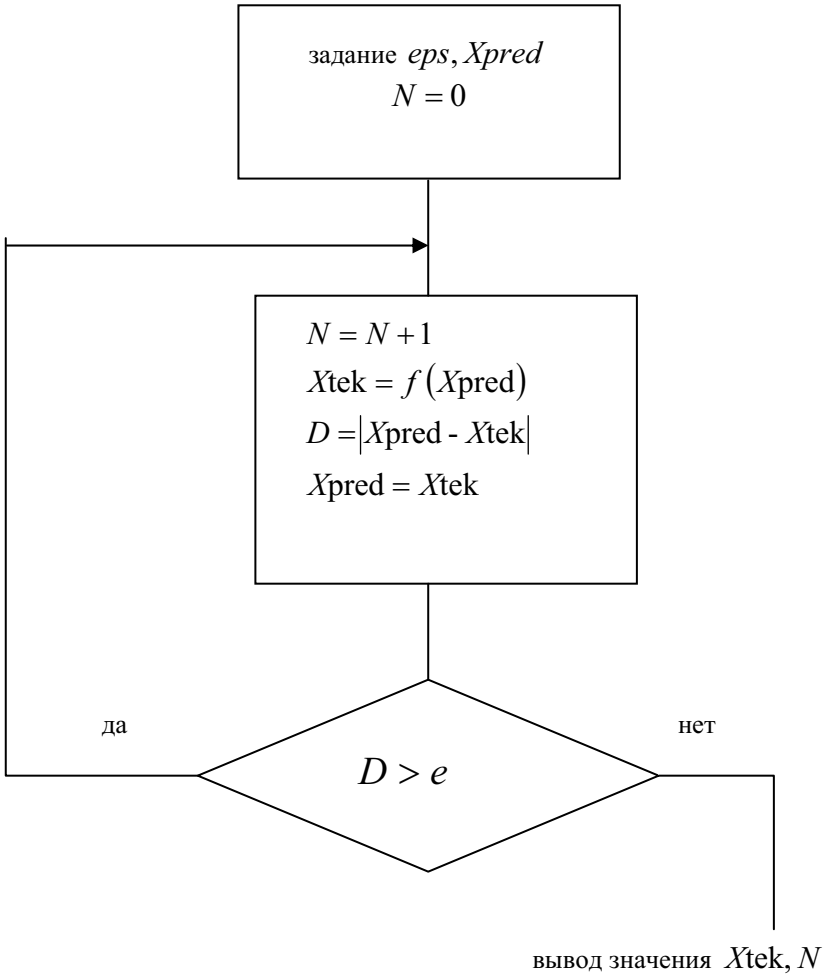


Рис. 15. Блок-схема алгоритма итерационного цикла

Для записи итерационных циклов наиболее подходит оператор с предусловием *WHILE*.

{Запись итерационного цикла на языке *Turbo Pascal*}

```
VAR EPS,XTEK,XPRED,D:REAL;
    N:INTEGER;
  BEGIN
    EPS:=0.0001;
    XPRED:=<начальное значение>;
    XTEK:=F(XPRED);
    D:=ABS(XPRED-XTEK);
    N:=0;
    WHILE D>=EPS DO BEGIN
       XPRED:=XTEK;
```

```
      N:=N+1;
      XTEK:=F(XPRED);
      D:=ABS(XPRED-XTEK);
                    END;
   WRITELN('XTEK =',XTEK,'N=',N);
 END.
```

*Пример_* . Вычислить кубический корень, используя рекуррентную формулу Ньютона:

$$y_{n+1} = \frac{2 \cdot y_n + x / y_n^{\,2}}{3},$$

где y_n - предыдущее значение кубического корня на итерации n;

y_{n+1} - текущее значение кубического корня на итерации $(n+1)$;

$y_0 = 1.$

Вычисления продолжать до тех пор, пока

$|y_{n+1} - y_n| >= \varepsilon,$ где ε - заданная точность вычислений.

```
 {Программа вычисления кубического корня на языке Turbo Pascal}
CONST EPS=0.001;
 VAR X,YN,YN1:REAL;
     N:INTEGER;
  BEGIN
    WRITELN('Введи x='); READLN(X);
    N:=1;
    YN:=1;
    YN1:=(2*YN+X/(YN*YN))/3;
    WHILE ABS(YN-YN1)>=EPS DO BEGIN
      YN:=YN1;
      N:=N+1;
      YN1:=(2*YN+X/SQR(YN))/3;
                          END;
    WRITELN('Корень  кубический  из',X:10:2,'равен',
         YN1:10:4,'с точностью',EPS:10:4);
  END.
```

*Пример_* . Вычислить приближённое значение

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

для заданного значения x и заданной точности ε. Вычисления будем продолжать до тех пор, пока очередной член ряда остаётся больше за-

данной точности ε. В примере используется величина ***n!***, которая называется «*n* факториал» и значение которой равно $(1*2*3*...*n)$. Значение *n!* с увеличением *n* быстро растёт. В частности, для любого *x* можно найти *n* такое, что $n!> x^n$.

Перед началом программирования приведём зависимость *Xtek* элемента последовательности от *Xpred* элемента в табл. 13 для того, чтобы вывести формулу зависимости текущего элемента от предыдущего, то есть $Xtek = F(Xpred)$.

Таблица 11 – Вывод формулы зависимости $Xtek = F(Xpred)$

Номер итерации (шага) - n	Значение *Xtek*	$Xtek = F(Xpred)$
n=0	1	
n=1	$\frac{x}{1!}=\frac{1\cdot x}{n}$	$Xtek = Xpred\cdot\frac{x}{n}$
n=2	$\frac{x^2}{2!}=\frac{x\cdot x}{1\cdot n}$	$Xtek = Xpred\cdot\frac{x}{n}$
n=3	$\frac{x^3}{3!}=\frac{x^2\cdot x}{1\cdot 2\cdot n}$	$Xtek = Xpred\cdot\frac{x}{n}$
…	…	…
n=i	$\frac{x^i}{i!}=\frac{x^{i-1}\cdot x}{1\cdot 2\cdot ...(i-1)\cdot i}$	$Xtek = Xpred\cdot\frac{x}{n}$

Таким образом, зависимость $Xtek = F(Xpred)$ для рассматриваемого примера выведена: для получения текущего значения необходимо умножать предыдущий элемент на выражение (x/n), то есть в цикле вычислять $(Xtek = Xtek*(x/n)$.

Для рассматриваемого примера введём следующие обозначения переменных:

Xtek - очередной член ряда;

eps - точность вычисления текущего значения переменной *Xtek*;

sum - результат суммирования членов ряда;

n - номер члена ряда.

{программа с использованием оператора цикла *WHILE* на языке *Turbo Pascal* }

```
CONST eps=0.00001;
  VAR sum,Xtek,x:REAL;
      n:INTEGER;
   BEGIN
```

```
    WRITELN('ввод x'); READLN(x);{задано значение x}
    sum:=1; n:=0;
    Xtek:=1;
    WHILE ABS(Xtek)>=eps DO BEGIN
     n:=n+1;
     Xtek:= Xtek*x/n;
     sum:=sum+Xtek;
                                END;
    WRITELN('sum=',sum:10:6,'       n=',n);
    READLN;
  END.
```

Ниже приведены три фрагмента программ на языке *C++* для примера_ с использованием различных операторов цикла. Во фрагментах отсутствуют: блок определения переменных, операторы ввода исходных данных, вывода результатов.

{первый фрагмент программы с использованием оператора цикла *while* с предусловием на языке *C++*}

```
n=2;
sum:=1;
Xtek:=x;
while(Xtek>=eps||Xtek<=-eps)
{sum=sum+Xtek;
Xtek=Xtek*x/n;
n++;
}
```

{второй фрагмент программы с использованием оператора цикла *while* с постусловием на языке *C++*}

```
n=1;
sum:=0;
Xtek:=1;
do {sum=sum+Xtek;
Xtek=Xtek*x/n;
n++;
}
while(Xtek>eps||Xtek<-eps)
```

{третий фрагмент программы с использованием оператора цикла *for* на языке *C++*}

```
n=2;
sum:=1;
Xtek:=x;
for( ;Xtek>eps||Xtek<-eps;)
{sum=sum+Xtek;
Xtek=Xtek*x/n;
n++;
}
```

5.4. Вложенные циклы

Циклы могут быть вложены один в другой. При использовании вложенных циклов необходимо составлять программу таким образом, чтобы внутренний цикл полностью укладывался в циклическую часть внешнего цикла. Внутренний цикл может также, в свою очередь, содержать другой внутренний цикл (циклы).

Структуру вложенных циклов рассмотрим на примере.

Пример.

Вычислить значения функции

$Y=2*K+N$

при N, изменяющемся от 1 до 2 с шагом 1;

K, изменяющемся от 2 до 8 с шагом 2.

Все вычисления примера сведём в табл. 14. Блок-схема алгоритма вычислений приведена на рис. 16.

Таблица 12 - Вычисление значений функции $Y=2*K+N$

N	K	Y
1	2	$y_1=$
1	4	$y_2=$
1	6	$y_3=$
1	8	$y_4=$
2	2	$y_5=$
2	4	$y_6=$
2	6	$y_7=$
2	8	$y_8=$

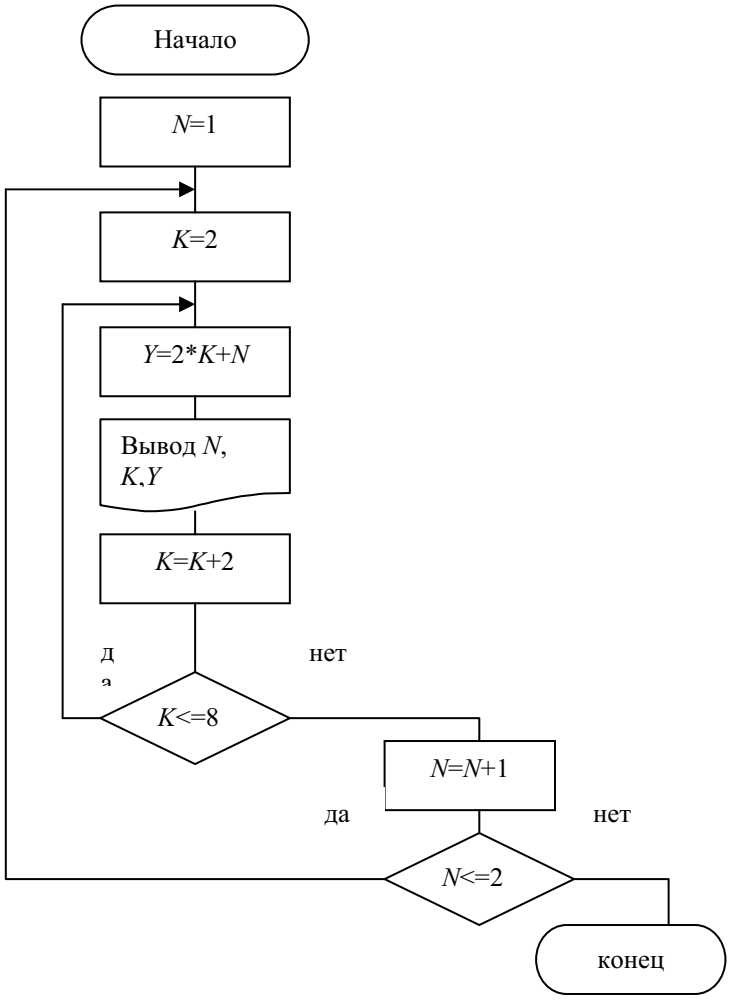


Рис. 16. Блок-схема алгоритма с вложенными циклами

Обсудим варианты использования операторов цикла для переменных *N* и *K*.

Переменная *N* изменяется от целого значения, равного 1, до целого значения, равного 2. Шаг приращения равен +1. Для переменной *N* можно применить оператор *FOR*.

Переменная *K* изменяется от целого значения, равного 2, до целого значения, равного 8. Но шаг приращения равен +2. Поэтому для переменной *K* можно применить либо оператор *WHILE* ..., либо оператор *REPEAT...UNTIL*. В программе будем использовать оператор *WHILE* ...

```
     {Программа по блок-схеме рис. 16}
VAR N:INTEGER;
    K,Y:REAL;
 BEGIN
  FOR N:=1 TO 2 DO BEGIN {начало цикла по N}
```

```
   K:=2; {начало цикла по K}
     WHILE K<= 8 DO BEGIN
                  Y:=2*K+N;
                  WRITELN('N=',N,' K=',K,' Y=',Y);
                  K:=K+2;
                      END; {окончание цикла WHILE по K}
                   END; {окончание цикла FOR по N}
END.
```

5.5. Контрольные вопросы

1. Какой алгоритм называется циклическим?
2. Назовите виды циклов.
3. Назовите признаки конечного цикла.
4. Назовите операторы цикла, изучаемые в языке программирования.
5. В чём отличие оператора цикла с предусловием от оператора цикла с постусловием?
6. Как работает оператор цикла с предусловием?
7. Как работает оператор цикла с постусловием?
8. Как работает оператор цикла с параметром FOR?
9. Как вычисляется сумма заданного количества значений переменной в математике и в программировании?
10. Как вычисляется произведение заданного количества значений переменной в математике и в программировании?
11. Как ведётся подсчёт различных значений простой переменной, значения которой задаются в цикле?
12. Опишите работу итерационного цикла.
13. В каких случаях используется итерационный цикл?
14. Объясните работу вложенных циклов.

Глава 6.

ОБРАБОТКА ЭЛЕМЕНТОВ МАССИВОВ

Массив в программировании – это конечная коллекция однотипных[13] данных. Массив обозначается одним именем.

Тип данных элементов массива может быть следующим:

- целым;
- вещественным;
- символьным;
- строковым;
- записью.

Ниже рассмотрена обработка элементов одномерных и двумерных массивов.

6.1. Одномерные массивы

Пусть дана совокупность действительных чисел: 2.1, 33.2, -2.1, которую назовём ***одномерным массивом*** и которую обозначим одним именем, например, *W*.

Каждый элемент одномерного массива обозначается именем массива с индексом, где индекс обозначает номер элемента в последовательности.

Действительные числа рассматриваемой совокупности можно представить размещёнными в одноэтажном доме, причём, каждое число в отдельной комнате. Адрес всего дома и будет именем массива. Тогда номер комнаты будет играть роль индекса, по которому элементы массива упорядочены (рис. 17).

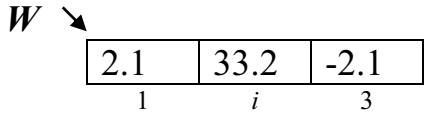


Рис. 17. Размещение элементов массива в памяти

В математике индекс указывается, либо ниже имени массива, например: W_1, W_2, W_3, либо заключается в круглые скобки, например:

[13] Однотипные – это данные одного типа

$W(1)$, $W(2)$, $W(3)$. Тогда в общем виде для рассматриваемого примера можно записать $W(\mathrm{i})$, где $i = \overline{1,3}$.

В языках *Turbo Pascal* и *C++* индекс заключается в квадратные скобки, то есть алгебраической нотации $A(i)$ отвечает `A[i]` при записи на языке *Turbo Pascal* и `a[i]` на языке *C++*.

6.1.1. Способы задания одномерных массивов на языке *Turbo Pascal*

Если в программе будет использоваться одномерный массив, то он должен быть обязательно задан одним из трёх способов:

- либо в разделе описания переменных - `VAR`;
- либо в разделе констант - `CONST`;
- либо в разделе типов - `TYPE`.

1. Задание одномерного массива в разделе описания переменных – `VAR`

Описание одномерного массива в разделе `VAR` рассмотрим на примере.

Формула 7Пример 7. Задан массив $Q(10)$, элементы которого могут принимать любые вещественные значения, что означает: задан одномерный массив Q, состоящий из 10-ти элементов.

```
VAR Q:ARRAY[1..10] OF REAL;{задан одномерный массив Q,
                            состоящий из 10-ти элементов вещественного типа}
```

Здесь `ARRAY` – переводится как «массив»;

`OF` – переводится как «из»;

`1..10` – тип индекса, который указывает границы его изменения. Две горизонтальные точки между значением нижней границы индекса (в нашем случае 1) и значением верхней границы индекса (в нашем случае – это 10) ставить обязательно!

Если несколько массивов имеют одинаковый тип индексов и одинаковый тип данных, то в описании можно объединить массивы в список.

```
{Задание списком трёх массивов F,D,R, каждый из которых содержит по 10 элементов действительных чисел}
VAR F,D,R:ARRAY[1..10] OF REAL;
```

2. Задание одномерного массива в разделе констант - CONST

Задание одномерного массива в разделе CONST рассмотрим на примере.

Формула 8Пример 8. Задан одномерный массив $Q(10)$, элементы которого принимают следующие вещественные значения:

1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7, 8.8, 9.9, 10.10.

```
{Задание элементов одномерного массива Q(10) в разделе CONST
  перечислением их значений, которые отделяются друг от друга
  запятыми, а весь список значений заключён в круглые скобки}
CONST R:ARRAY[1..10] OF REAL=(1.1,2.2,3.3,4.4,5.5,
                              6.6,7.7,8.8,9.9,10.10);
```

Над элементами массива можно производить все операции, которые допустимы над простыми переменными заданного типа. Элементы массива могут стоять как в левой части оператора присваивания, так и в выражениях правой части.

3. Задание одномерного массива в разделе типов - TYPE

Тип данных может определяться идентификатором типа. Раздел описания типов данных начинается зарезервированным словом TYPE, за которым следует одно или несколько определений типов, разделённых точкой с запятой. Задание одномерного массива в разделе TYPE рассмотрим на примере.

```
TYPE VEKTOR:ARRAY[1..10] OF REAL;
  VAR F,D,R: VEKTOR; {заданы списком три массива F,D,R
    действительных чисел, каждый из которых содержит по 10
    элементов как в описании массива VEKTOR раздела TYPE}
```

Знание способа задания массивов в разделе типов необходимо в программах с использованием подпрограмм обработки элементов массива.

6.1.2. Ввод, вывод значений элементов одномерного массива на языке *Turbo Pascal*

Ввод значений элементов одномерного массива

Ввод значений элементов одномерного массива не требуется, если массив задан в разделе CONST!

Ввод значений элементов одномерного массива осуществляется только в цикле, где в качестве переменной цикла выступает индекс элемента массива. Так как индекс массива имеет целые значения границ и его изменение при переходе от текущего индекса к соседнему индексу равно +1, то для записи оператора изменения индекса рекомендуется использовать оператор FOR.

```
{Программа ввода значений элементов одномерного массива
 R(10)}
VAR R:ARRAY[1..10] OF REAL; {задан одномерный массив R,
                состоящий из 10-ти элементов вещественного типа}
    I:INTEGER; {задана переменная цикла, которая будет использована в качестве индекса элемента массива}
BEGIN
 WRITELN('вводи 10 значений элементов массива R через пробел');
 FOR I:=1 TO 10 DO READ(R[I]);
END.
```

Вывод значений элементов одномерного массива

Вывод значений элементов одномерного массива осуществляется только в цикле, где в качестве переменной цикла выступает индекс элемента массива.

```
{Программа вывода значений элементов одномерного массива
 R(10)}
VAR R:ARRAY[1..10] OF REAL;
    I:INTEGER;
BEGIN
 WRITELN('вводи 10 значений элементов массива R через пробел');
 FOR I:=1 TO 10 DO READ(R[I]);
 WRITELN('вывод столбиком 10 значений элементов массива R');
 FOR I:=1 TO 10 DO
                WRITELN('R[',I,']=', R[I]:8:2);
END.
```

6.1.3. Алгоритмы обработки элементов одномерного массива

Суммирование, произведение элементов одномерного массива

В математике ***суммирование*** числовых значений элементов одномерного массива (вектора) описывается выражением

$$S = \sum_{i=1}^{10} x_i \ .$$

При программировании сумма значений элементов одномерного массива (вектора) будет вычисляться по формуле

$$S = S + x_i \ ,$$

где S – переменная, которая используется для накопления суммы;

x_i – значение i-ого элемента одномерного массива.

По выражению $S = S + x_i$ каждое новое значение S получается из предыдущего добавлением очередного значения элемента одномерного массива.

Суммирование значений элементов одномерного массива оформляется в цикле!

Перед началом цикла начальное значение суммы нужно обязательно обнулить (S=0), то есть очистить содержимое ячейки по адресу S.

В математике ***произведение*** числовых значений элементов одномерного массива (вектора) описывается выражением

$$P = \prod_{i=1}^{10} x_i \ .$$

При программировании произведение значений элементов одномерного массива (вектора) будет вычисляться по формуле

$$P = P * x_i \ ,$$

где P – переменная, которая используется для накопления произведения;

x_i – значение i-ого элемента одномерного массива.

По выражению $P = P * x_i$ каждое новое значение P получается из предыдущего умножением на очередной сомножитель.

Произведение значений элементов одномерного массива оформляется в цикле!

Перед началом цикла начальное значение произведения приравниваем единице, то есть P=1, чтобы не влиять на результаты перемножения.

Формула 9Пример 9. Найти сумму, произведение, подсчитать количество единичных значений 10-ти значений элементов одномерного массива $X(10)$. Блок-схема алгоритма приведена на рис. 18.

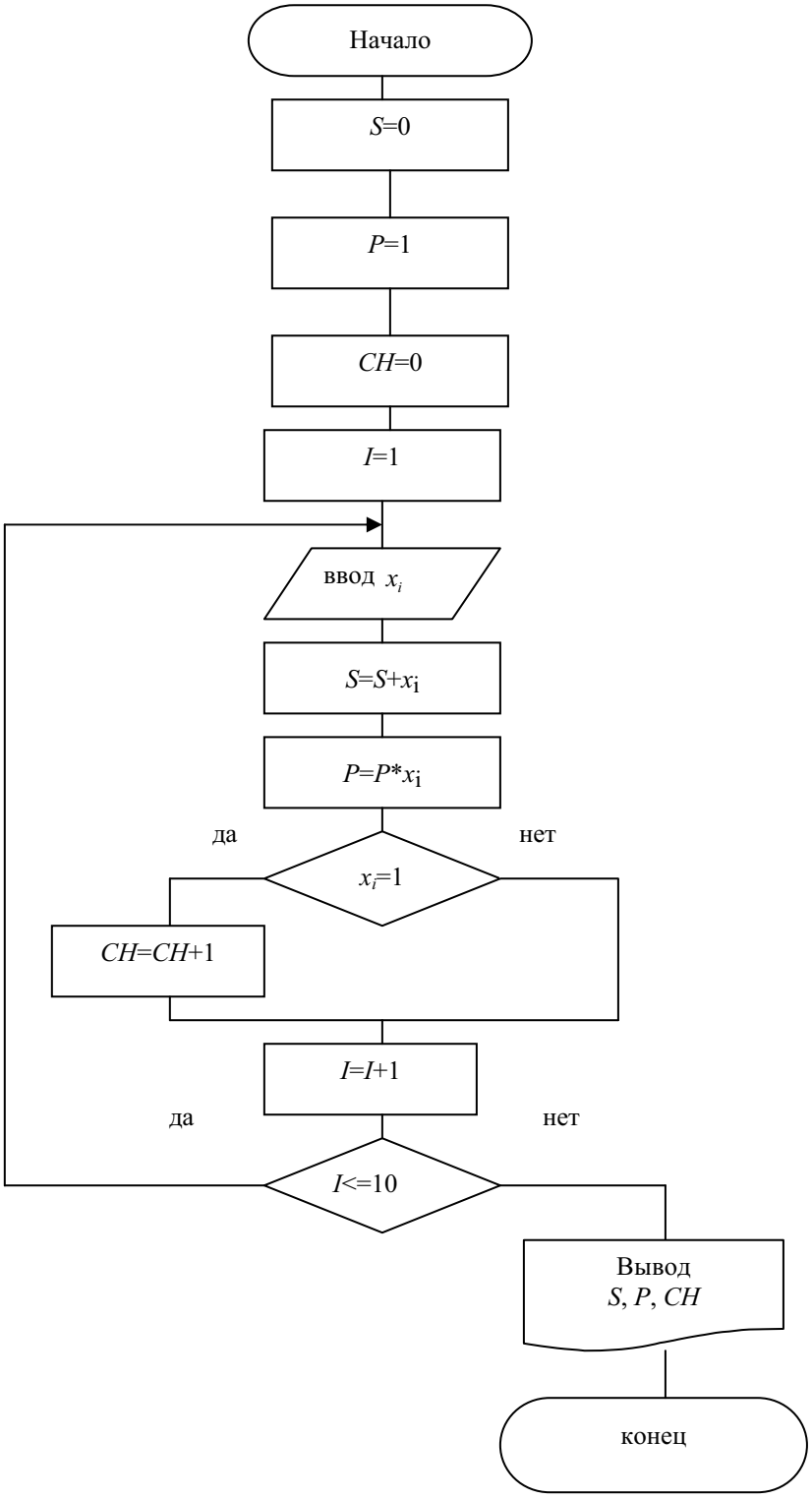


Рис. 18. Блок-схема алгоритма нахождения суммы, произведения, подсчёта количества выборочных значений элементов массива *Y*

```
{Программа определения суммы, произведения, количества единичных значений 10-ти значений элементов одномерного массива X(10) на языке Turbo Pascal}
VAR X:ARRAY[1..10] OF REAL;
    I,CH:INTEGER;
    S,P:REAL;
 BEGIN
  S:=0; {обнуление суммы}
  P:=1; {произведение приравниваем 1}
  CH:=0; {обнуление счётчика}
  WRITELN('вводи 10 значений по одному на строке');
  FOR I:=1 TO 10 DO BEGIN
           READLN(X[I]);
           S:=S+X[I];
           IF X[I]<>0 THEN P:=P*X[I];
           IF X[I]=1 THEN CH:=CH+1;
                  END;
  WRITELN('S=',S:7:2,' P=',P:7:2,' CH=', CH);
 END.
```

Поиск минимального (максимального) значения элемента и его местоположения в одномерном массиве

Поиск наибольшего (максимального) или наименьшего (минимального) значения элемента в одномерном массиве осуществляется последовательным сравнением значений.

Для нахождения *наибольшего* значения элемента задаём во вспомогательной переменной, например *XMAX*, начальное значение, равное очень маленькому числу – (-*MAXINT*, где *MAXINT* – зарезервированная константа, возвращающая максимальное целое число, то есть 32767). Если необходимо определить к тому же местоположение найденного максимального значения, то лучше за начальное максимальное значение принять значение первого элемента массива, а во вспомогательной переменной, например *IMAX*, запомнить местоположение первого элемента. При этом поиск в цикле описывается формулой:

$$XMAX = \begin{cases} X_i, & \text{если } X_i > XMAX, \quad IMAX = i; \\ XMAX, & \text{если } X_i \le XMAX. \end{cases}$$

Для нахождения *наименьшего* значения элемента задаём во вспомогательной переменной, например *XMIN*, начальное значение, равное очень большому числу – *MAXINT*, где *MAXINT* – зарезервированная

константа, возвращающая максимальное целое число, то есть 32767. Если необходимо определить к тому же местоположение найденного минимального значения, то лучше за начальное максимальное значение принять значение первого элемента массива, а во вспомогательной переменной, например *IMIN*, запомнить местоположение первого элемента. При этом поиск в цикле описывается формулой:

$$XMIN = \begin{cases} X_i, & \text{если } X_i < XMIN, \quad IMIN = i; \\ XMIN, & \text{если } X_i \geq XMIN. \end{cases}$$

Формула 10 Пример 10. Составить программу для нахождения максимального и минимального значений элементов массива $X(10)$ и их порядковых номеров.

```
{Программа нахождения максимального и минимального значений элементов массива X(10) и их порядковых номеров на языке Turbo Pascal}
VAR X:ARRAY[1..10] OF REAL;
    IMAX,IMIN,I:INTEGER;
    XMAX,XMIN:REAL;
 BEGIN
  WRITELN('вводи 10 значений через пробел');
  FOR I:=1 TO 10 DO READ(X[I]);
  XMAX:=A[1];{задаём начальное значение максимального элемента}
  IMAX:=1;    {запоминаем местоположение первого максимального элемента}
  XMIN:=A[1]; {задаём начальное значение минимального элемента}
  IMIN:=1; {запоминаем местоположение первого минимального элемента}
  FOR I:=1 TO 10 DO BEGIN
    IF X[I]>XMAX THEN BEGIN
      XMAX:=A[I]; {заменяем значение максимального элемента}
      IMAX:=I; {запоминаем местоположение максимального элемента}
                          END;
      IF X[I]<XMIN THEN BEGIN
      XMIN:=A[I]; {заменяем значение минимального элемента }
      IMIN:=I; {запоминаем местоположение минимального элемента}
                          END;
                       END; {окончание цикла по I}
  WRITELN('XMAX=',XMAX:7:2,'IMAX=',IMAX);
  WRITELN('XMIN=',XMIN:7:2,'IMIN=',IMIN);
 END.
```

Подсчёт количества отрицательных, положительных, нулевых значений элементов одномерного массива

Подсчёт количества отрицательных, положительных, нулевых значений элементов одномерного массива рассмотрим на примере.

Формула 11 Пример 11. В массиве $X(10)$ подсчитать количество нулей, отрицательных и положительных элементов.

Для подсчёта количества нулей, отрицательных и положительных элементов введём три дополнительные переменные:

K0 – для подсчёта количества нулей;

KOTR – для подсчёта количества отрицательных элементов;

KP – для подсчёта количества положительных элементов.

Тогда, в зависимости от значения элемента массива (10), будем увеличивать соответствующий счётчик.

```
{Программа подсчёта количества нулей, отрицательных
 и положительных элементов на языке Turbo Pascal}
VAR X:ARRAY[1..10] OF REAL;
    I,K0,KOTR,KP:INTEGER;
  BEGIN
   WRITELN('вводи 10 значений массива X');
   FOR I:=1 TO 10 DO READ(X[I]);
   K0:=0;
   KOTR:=0;
   KP:=0;
   FOR I:=1 TO 10 DO BEGIN
       IF X[I]<0 THEN KOTR:=KOTR+1;
       IF X[I]=0 THEN K0:=K0+1;
       IF X[I]>0 THEN KP:=KP+1;
                      END;
    WRITELN('KOTR=',KOTR:7,'K0=',K0:7,'KP=',KP:7);
   END.
```

Перестановка элементов местами

Перестановка элементов в массиве или между массивами осуществляется с использованием промежуточной переменной.

Формула 12 Пример 12. Поменять местами значения двух элементов: элемента A_3 и элемента A_4 (см. рис. 19).

Для решения воспользуемся дополнительной переменной P. Тогда перестановка будет выглядеть следующим образом:

I. $P = A_3$ – запоминаем в промежуточной переменной значение A_3;

II. $A_3 = A_4$ – пересылаем в A_3 значение A_4;

III. $A_4 = P$ – в A_4 пересылаем значение, хранящееся в промежуточной переменной P, то есть фактически значение A_3.

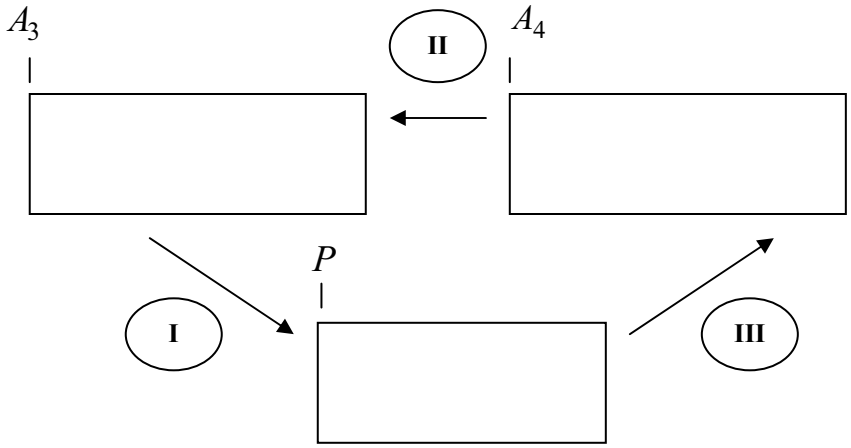


Рис. 19. Схема перестановки значений элементов массива местами

Формула 13 Пример_13. Значения, хранящиеся в массиве $X(10)$, переписать в массив $R(10)$.

```
{Программа перезаписи значений, хранящихся в массиве X(10),
 в массив R(10) на языке Turbo Pascal}
VAR X,R:ARRAY[1..10] OF REAL;
    I:INTEGER;
    P:REAL; {промежуточная переменная}
BEGIN
  WRITELN('вводи 10 значений массива X');
  FOR I:=1 TO 10 DO READ(X[I]);
  WRITELN('вводи 10 значений массива R');
  FOR I:=1 TO 10 DO READ(R[I]);
  FOR I:=1 TO 10 DO BEGIN
    P:=X([I]);{запоминаем в промежуточной переменной значение A(i)}
    X([I]):=R([I]); {пересылаем в X(i) значение R(i)}
    R([I]):=P;{в R(i) пересылаем значение, хранящееся в переменной P}
                END;
FOR I:=1 TO 10 DO WRITELN(X[I]:8:2);
```

```
  FOR I:=1 TO 10 DO WRITELN(R[I]:8:2);
END.
```

Формула 14 Пример 14. Вычислить значение функции *Y*, заданную выражением:

$$y = \begin{cases} X / B_i, & \text{если} \quad X \geq 5 \quad \text{или} \quad B_i = 25; \\ X^2 - B_i^2, & \text{если} \quad X < 5 \quad \text{и} \quad B_i < 25; \\ B_i^3 - X^3 - \pi, & \text{при всех других значениях} \quad X \text{ и } B_i, \end{cases}$$

где *B* – одномерный массив, состоящий из 5-ти элементов;

X – изменяется от +1 до 11-ти с шагом 2.

Блок-схема алгоритма вычисления функции *Y* приведена на рис. 20.

Обсудим варианты использования операторов цикла для переменных *X* и элементов массива *B*.

Индексная переменная *I* массива *B* изменяется от целого значения, равного 1, до целого значения, равного 5-ти. Шаг приращения равен +1. Для индексной переменной *I* необходимо применить оператор FOR...TO.

Переменная *X* изменяется от целого значения, равного +1, до целого значения, равного +11, но шаг приращения равен +2. Поэтому для переменной *X* можно применить либо оператор WHILE..., либо оператор REPEAT...UNTIL. В программе использован оператор WHILE...

```
{Программа вычисления функции Y с использованием вложенных
  циклов и разветвления вычислений на языке Turbo Pascal}
VAR B:ARRAY[1..5] OF REAL;
    I,X:INTEGER;
    Y:REAL;
BEGIN
 WRITELN('вводи 5 значения массива B');
 FOR I:=1 TO 5 DO READ(B[I]);
 FOR I:=1 TO 5 DO BEGIN {начало цикла по I}
  X:=1; {начало цикла по X}
  WHILE X<=11 DO BEGIN
    IF (X>=5) OR (B[I]=25) THEN Y:=X/B[I])
                          ELSE
     IF (X<5) AND (B[I]<25) THEN Y:=X*X-B[I]*B[I]
                            ELSE
                       Y:=B[I]*B[I]*B[I]-X*X*X-PI;
```

```
      WRITELN('B[',I,']=',B[I],'X=',X,'Y=',Y:8:3);
        X:=X+2;
                          END; {окончание цикла по X}
                           END; {окончание цикла FOR по I}
END.
```

Формула 15 Пример 15. Даны два массива $A(4)$ и $B(4)$. Заполнить новый массив $Q(8)$ следующим образом: сначала 2 элемента массива A, затем 2 элемента массива B, вновь 2 элемента массива A, затем 2 элемента массива B.

```
        {Перезапись элементов из массивов A(4) и B(4) в массив Q(8) на
         языке Turbo Pascal}
CONST A:ARRAY[1..4] OF INTEGER=(1,2,5,6);
          B:ARRAY[1..4] OF INTEGER=(3,4,7,8);
   VAR Q:ARRAY[1..8] OF INTEGER;
          I:INTEGER;
      BEGIN
       FOR I:=1 TO 2 DO BEGIN
         Q[I]:=A[I];          {для Q индекс=1,2; для A индекс=1,2}
         Q[2+I]:=B[I];        {для Q индекс=3,4; для B индекс=1,2}
         Q[4+I]:=A[2+I];  {для Q индекс=5,6; для A индекс=3,4}
         Q[6+I]:=B[2+I];  {для Q индекс=7,8; для B индекс=3,4}
                             END; {окончание цикла по I}
       FOR I:=1 TO 8 DO WRITELN('Q[',I,']',Q[I]);
      END.
```

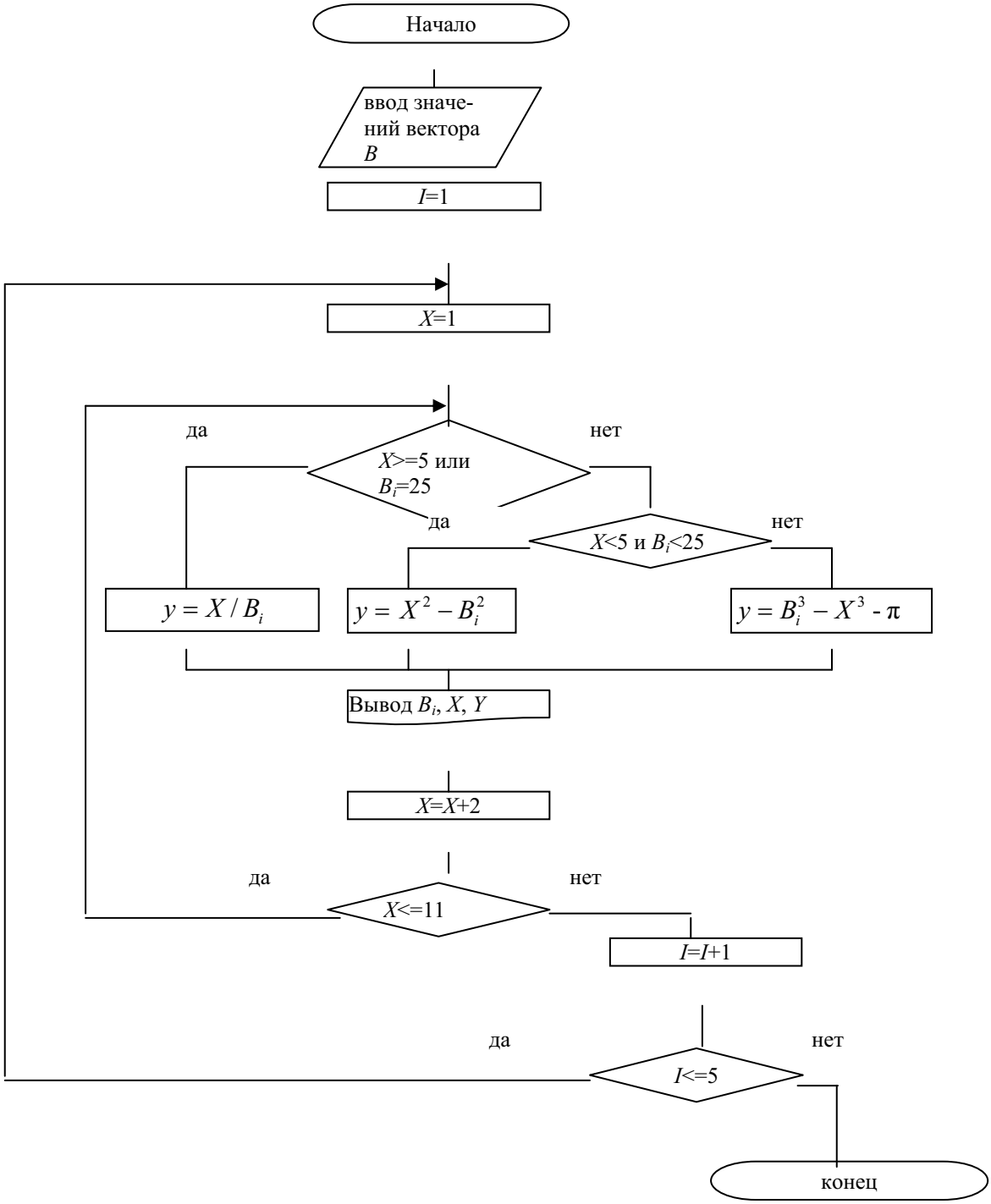


Рис. 20. Блок-схема алгоритма вычисления функции Y с использованием вложенных циклов и разветвления вычислений

Формула 16 Пример 16. Даны три массива $A(2)$, $B(2)$ и $C(2)$. Заполнить массив $Q(6)$ следующим образом: на места, кратные трём, поместить элементы массива A, на предшествующие им места поместить

элементы массива *B*, а перед элементами массива *B* поместить элементы массива *C*.

{Перезапись элементов из массивов *A*(4) и *B*(4) в массив *Q*(8) на языке *Turbo Pascal*}

```
CONST A:ARRAY[1..2] OF INTEGER=(3,6);
      B:ARRAY[1..2] OF INTEGER=(2,5);
      C:ARRAY[1..2] OF INTEGER=(1,4);
  VAR Q:ARRAY[1..6] OF INTEGER;
      I:INTEGER;
   BEGIN
     FOR I:=1 TO 2 DO BEGIN
         Q[3*I]:=A[I];  {для Q индекс=3,6; для A индекс=1,2}
         Q[3*I-1]:=B[I];  {для Q индекс=2,5; для B индекс=1,2}
         Q[3*I-2]:=C[I];  {для Q индекс=1,4; для C индекс=1,2}
                       END;
     FOR I:=1 TO 6 DO WRITELN('Q[',I,']',Q[I]);
   END.
```

Если в массиве элементы должны быть на местах, кратных четырём, то выражение для индекса будет [4*I].

Формула 17 Пример 17. Даны два массива *Z*(3) и *Q*(3), которые состоят из положительных и отрицательных элементов. Заполнить массив *P*(6) следующим образом: сначала положительные элементы массива *Z*, затем положительные элементы массива *Q*. В программе надо воспользоваться дополнительной переменной *CH*, в которой подсчитывать текущее положение элемента в массиве *P*.

{Перезапись элементов из массивов *Z*(3) и *Q*(3) в массив *P*(6) на языке *Turbo Pascal*}

```
CONST Z:ARRAY[1..3] OF INTEGER=(1,2,-4);
      Q:ARRAY[1..3] OF INTEGER=(-5,3,4);
 VAR P:ARRAY[1..6] OF INTEGER;
     CH,I:INTEGER;
   BEGIN
    CH:=0;  {CH – это счетчик количества положительных элементов
              и одновременно индексная переменная массива P(6)}
    {перезапись положительных элементов из массива Z(3)}
    FOR I:=1 TO 3 DO BEGIN
       IF Z[I]>0 THEN BEGIN
```

```
            CH:=CH+1;
            P[CH]:=Z[I];
                END;
               END;
```
{перезапись положительных элементов из массива $Q(3)$}
```
 FOR I:=1 TO 3 DO BEGIN
   IF Q[I]>0 THEN BEGIN
            CH:=CH+1;
            P[CH]:=Q[I];
                END;
               END;
 FOR I:=1 TO CH DO WRITE(P[I];
END.
```

6.2. Двумерные массивы

Двумерный массив – это коллекция однотипных данных вида:

$$A=\begin{pmatrix} a_{11}\,a_{12}\,\ldots a_{1n} \\ a_{21}\,a_{22}\ldots a_{2n} \\ .\ .\ .\ .\ . \\ a_{m1}\ a_{m2}\ldots a_{mn} \end{pmatrix} .$$

Тип данных элементов массива может быть следующим:

- целым;
- вещественным;
- символьным;
- строковым;
- записью.

Матрица – это двумерный массив чисел либо целого, либо вещественного типа.

Матрица характеризуется *размерностью M*N*, то есть произведением числа строк *M* на число столбцов *N*.

Квадратная матрица имеет *M=N* и размерность $m^2=m*m$.

Диагональная матрица является разновидностью квадратной матрицы, у которой все элементы нулевые, кроме диагональных, то есть $A(i,j)\#0$ при $i=j$:

$$A = \begin{pmatrix} a_{11}\,0\;\;0\ldots0 \\ 0\;\;a_{22}\,0\ldots0 \\ .\;\;.\;\;.\;\;.\;\;. \\ 0\;\;0\;\;\;0\ldots a_{mm} \end{pmatrix} \quad .$$

Единичная матрица - это разновидность диагональной матрицы, у которой $A(i,j)=1$ при $i=j$.

Нулевая матрица – это матрица, у которой все $A(i,j)=0$.

Главная диагональ матрицы - это диагональ матрицы, индексы элементов которой равны, то есть $i=j$, например: $A(1,1)$, $A(2,2)$ и т. д. При программировании элементы главной диагонали принято записывать как $A(i,i)$ при $i = \overline{1,N}$.

Побочная диагональ матрицы – это диагональ матрицы, сумма индексов элементов которой равна $(N+1)$, где N – размерность квадратной матрицы; например, для матрицы размерности 5 – это будут элементы $A(1,5)$, $A(2,4)$, $A(3,3)$, $A(4,2)$, $A(5,1)$. При программировании элементы побочной диагонали будут записаны как $A(i,N+1-i)$.

Двумерный массив, как и одномерный массив, обозначается одним именем.

Размеры двумерного массива, как и одномерного массива, должны быть объявлены либо в разделе `VAR`, либо в разделе `CONST`, либо в разделе типов `TYPE`, причём алгебраической нотации $A(i,j)$ отвечает `A[i,j]` при записи на языке *Turbo Pascal* и `a[i][j]` на языке *C++*.

6.2.1. Способы описания двумерного массива на языке *Turbo Pascal*

Если в программе будет использоваться двумерный массив, то он должен быть обязательно задан одним из трёх способов:

- либо в разделе описания переменных - `VAR`;
- либо в разделе констант - `CONST`;
- либо в разделе типов - `TYPE`.

1. Описание двумерного массива в разделе описания переменных – `VAR`

Описание двумерного массива в разделе `VAR` рассмотрим на примере.

Формула 18 Пример_18. Задан массив $R(5,4)$, состоящий из 5-ти строк и 4-х столбцов, элементы которого могут принимать любые вещественные значения.

```
VAR R:ARRAY[1..5,1..4] OF REAL; {задан двумерный массив
        R, состоящий из 5-ти строк и 4-х столбцов, значения эле-
        ментов которого вещественного типа}
```

Здесь `ARRAY` – переводится как «массив»;
`OF` – переводится как «из»;
1..5 и 1..4 – типы индексов, указывающие границы их изменения. Две горизонтальные точки между значением нижней границы индекса (в нашем случае 1) и значением верхней границы индекса (в нашем случае 5 и 4) ставить *обязательно*!

Если несколько двумерных массивов имеют одинаковый тип индексов и одинаковый тип данных, то в описании массивы можно объединить в список:

```
VAR F,D,R:ARRAY[1..5,1..4] OF REAL; {заданы списком три
                матрицы: F, D, R размерности 5*4}
```

2. Задание двумерного массива в разделе констант – `CONST`

Задание двумерного массива в разделе `CONST` рассмотрим на примере.

Формула 19 Пример 19. Задан массив $T(4,5)$, элементы которого могут принимать любые вещественные значения, например:

$$T=\begin{pmatrix} 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\ 2.1 & 2.2 & 2.3 & 2.4 & 2.5 \\ 3.1 & 3.2 & 3.3 & 3.4 & 3.5 \\ 4.1 & 4.2 & 4.3 & 4.4 & 4.5 \end{pmatrix}.$$

```
CONST T:ARRAY[1..4,1..5] OF
      REAL=((1.1,1.2,1.3,1.4,1.5),
            (2.1,2.2,2.3,2.4,2.5),
            (3.1,3.2,3.3,3.4,3.5),
            (4.1,4.2,4.3,4.4,4.5));
```

{задан двумерный массив T, состоящий из элементов вещественного типа с перечислением их значений по строкам, кото-

рые отделяются друг от друга запятыми; список значений по строкам заключается в круглые скобки, за которым ставится запятая, а весь список заключается в круглые скобки}

Обратите внимание, что в случае задания значений элементов массива в разделе CONST вводить значения элементов массива в программу не надо!

3. Описание одномерного массива в разделе типов - TYPE

Описание двумерного массива в разделе TYPE рассмотрим на примере:

```
TYPE MATRIZA:ARRAY[1..10,1..10] OF REAL;
  VAR F,D,R:MATRIZA;
```

{заданы списком три матрицы *F*, *D*, *R* действительных чисел, каждый из которых имеет по 10 строк и 10 столбцов как в описании массива *MATRIZA* раздела *TYPE*}

6.2.2. Ввод значений элементов двумерного массива на языке *Turbo Pascal*

Ввод значений элементов двумерного массива осуществляется только с помощью двух вложенных циклов: по строкам, по столбцам.

В случае *ввода значений элементов по строкам* в качестве переменной внешнего цикла выступает первый индекс элемента; в качестве переменной внутреннего цикла выступает второй индекс элемента.

В случае *ввода значений элементов по столбцам* в качестве переменной внешнего цикла выступает второй индекс элемента; в качестве переменной внутреннего цикла выступает первый индекс элемента.

В примерах приведены два варианта ввода:

первый: на строке экрана будем набирать сразу все значения элементов строки, разделяя их пробелом (в этом случае мы сможем вернуться к неверно набранному значению и изменить его);

второй: на строке экрана будем набирать только *одно* значение элемента строки и сразу будем нажимать клавишу <*ENTER*> (в этом случае нельзя вернуться к неверно набранному значению и изменить его).

{Программа ввода с клавиатуры значений элементов матрицы *R*(2,3)}

```
VAR R:ARRAY[1..2,1..3] OF REAL;
    I,J:INTEGER;{описаны переменные циклов, которые будут
                 использованы в качестве индексов элемента массива}
 BEGIN
                  {1-ый способ ввода – по строкам}
   {первый вариант – все значения строки матрицы на строке
                     монитора}
   FOR I:=1 TO 2 DO BEGIN  {цикл по строкам}
        WRITELN('вводи ',I,'-ю строку',
                ' по 3 значения через пробел');
   FOR J:=1 TO 3 DO READ(R[I,J]); {цикл по столбцам}
                     END; {окончание цикла по строкам}

   {второй вариант – элемент матрицы задаётся один на строке
                     монитора}
   WRITELN('вводи по строкам по одному элементу
            массива');
   FOR I:=1 TO 2 DO BEGIN {цикл по строкам}
    FOR J:=1 TO 3 DO BEGIN {цикл по столбцам}
              WRITE('вводи R[',I,',',J,']= ');
              READLN(R[I,J]);
                    END; {окончание цикла по столбцам}
                  END; {окончание цикла по строкам}
 END.
```

6.2.3. Вывод значений элементов матрицы

Вывод значений элементов двумерного массива осуществляется только с помощью двух вложенных циклов по строкам и по столбцам.

В случае *вывода значений элементов по строкам* в качестве переменной внешнего цикла выступает первый индекс элемента; в качестве переменной внутреннего цикла выступает второй индекс элемента.

В случае *вывода значений элементов по столбцам* в качестве переменной внешнего цикла выступает второй индекс элемента; в качестве переменной внутреннего цикла выступает первый индекс элемента.

{Программа вывода значений матрицы *R*(2,3)}

```
 CONST R:ARRAY[1..2,1..3] OF REAL=((1.1,1.2,1.3),
                                   (2.1,2.2,2.3));
 VAR I,J:INTEGER;
```

```
BEGIN
 FOR I:=1 TO 2 DO BEGIN {цикл по I - строкам}
                  WRITELN(I,'-я строка');
  FOR J:=1 TO 3 DO {цикл по столбцам}
           WRITE('R[',I,',',J,']=',R[I,J]:4:1);
                    END; {окончание цикла по I строкам}
END.
```

6.2.4. Алгоритмы обработки значений элементов двумерного массива

Матрица - это двумерный массив, элементами которого являются числа целого или вещественного типа.

Суммирование, произведение значений элементов матрицы

В математике *суммирование* значений элементов матрицы описывается выражением

$$S = \sum_{i=1}^{M} \sum_{j=1}^{N} X_{ij} \, .$$

При программировании сумма значений элементов матрицы будет вычисляться по формуле

$$S = S + X_{ij},$$

где S – переменная, которая используется для накопления суммы;

X_{ij} – значение i,j-го элемента матрицы $X(M,N)$.

По выражению $S = S + X_{ij}$ каждое новое значение S получается из предыдущего добавлением очередного значения элемента матрицы.

Суммирование значений элементов матрицы оформляется двумя вложенными циклами!

Перед началом внешнего цикла начальное значение суммы нужно обязательно обнулить (S=0), то есть очистить содержимое ячейки по адресу S.

В математике *произведение* значений элементов матрицы описывается выражением

$$P = \prod_{i=1}^{M} \prod_{j=1}^{N} X_{ij}.$$

При программировании произведение значений элементов матрицы будет вычисляться по формуле

$$P = P * X_{ij},$$

где P – переменная, которая используется для накопления произведения;

X_{ij} – значение *i,j*-го элемента матрицы *X*(*M,N*).

По выражению $P = P * X_{ij}$ каждое новое значение *P* получается из предыдущего умножением на очередной элемент матрицы.

Произведение значений элементов матрицы оформляется двумя вложенными циклами!

Перед началом цикла начальное значение произведения приравниваем единице, то есть *P*=1, чтобы не влиять на результаты перемножения.

На рис. 21 приведена блок-схема алгоритма суммирования и произведения значений всех элементов матрицы *X*(*M,N*).

{Программа суммирования и произведения значений всех элементов матрицы *X*(*M,N*) на языке *Turbo Pascal*}

```
CONST X:ARRAY[1..2,1..3] OF REAL=((1.1,1.2,1.3),
                                  (2.1,2.2,2.3));
 VAR J,I:INTEGER;
     S,P:REAL;
  BEGIN
   S:=0; {обнуление суммы}
   P:=1; {произведение приравниваем 1}
   FOR I:=1 TO 2 DO BEGIN
    FOR J=1 TO 3 DO BEGIN
                  S:=S+X[I,J];
                  IF P<>0 THEN P:=P*X[I,J];
                    END;
                    END;
   WRITELN('S=',S:7:2,' P=',P:7:2);
  END.
```

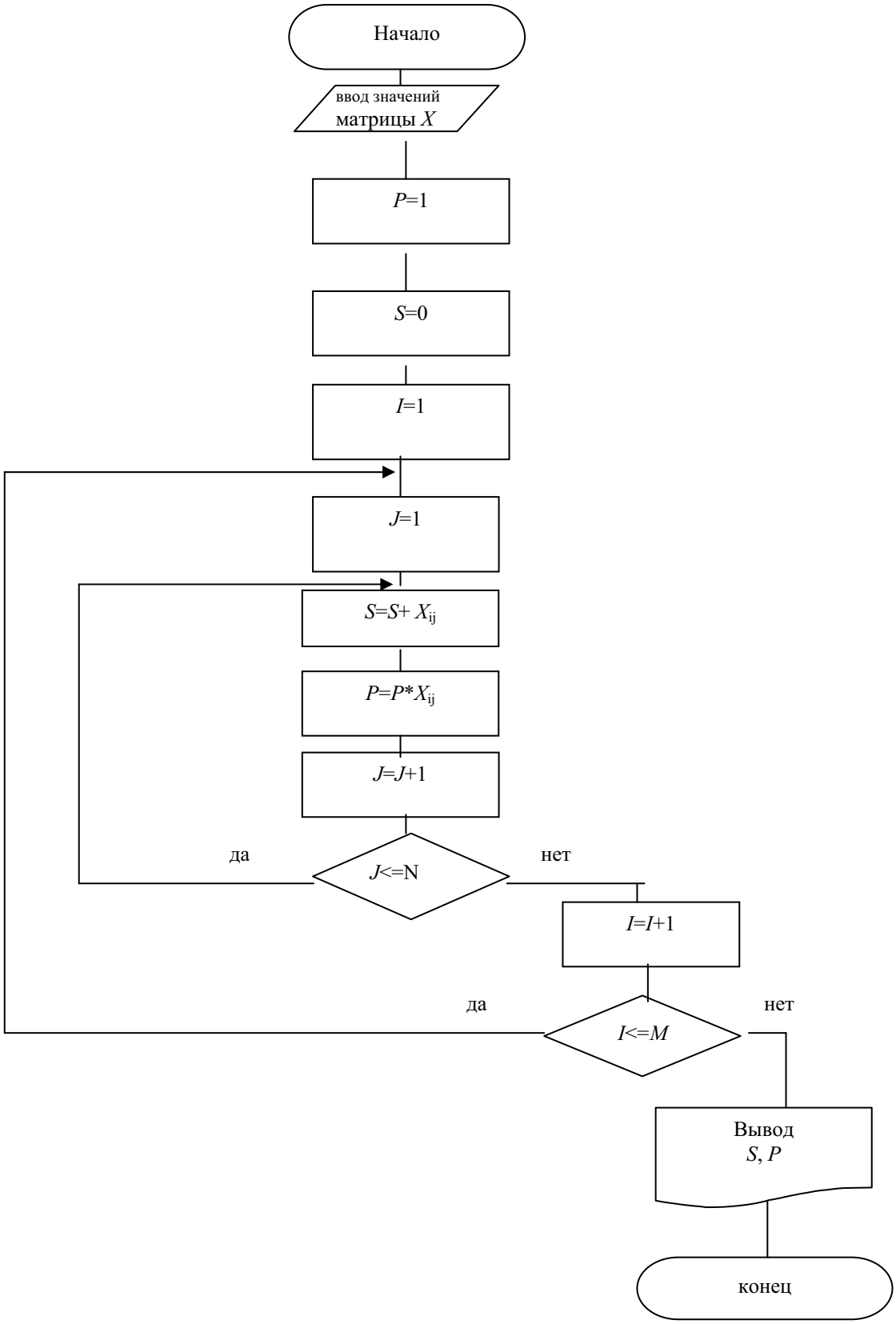


Рис. 21. Блок-схема алгоритма суммирования и произведения значений всех элементов матрицы

Поиск максимального, минимального значений элементов и их местоположения в матрице

Поиск наибольшего (максимального) или наименьшего (минимального) значения элемента в матрице, например *X*(*M*,*N*), осуществляется последовательным сравнением значений.

Для нахождения *наибольшего* значения элемента в матрице задаём во вспомогательной переменной, например *XMAX*, начальное значение, равное очень маленькому числу – (-*MAXINT*, где *MAXINT* – встроенная функция, возвращающая максимальное целое число, т. е. 32767). Если необходимо определить к тому же местоположение найденного максимального значения, то вводим две дополнительные переменные, например *IMAX* и *JMAX*, в которые записываем соответствующие значения строки и столбца.

Поиск максимального элемента будет происходить в двух вложенных циклах и будет описываться следующей формулой:

$$XMAX = \begin{cases} X_{ij}, & \text{если } X_{ij} > XMAX, \quad IMAX = i, \quad JMAX = j; \\ XMAX, & \text{если } X_{ij} \le XMAX. \end{cases}$$

Для нахождения *наименьшего* значения элемента в матрице задаём во вспомогательной переменной, например *XMIN*, начальное значение, равное очень большому числу – *MAXINT*. Если необходимо определить к тому же местоположение найденного минимального значения, то вводим две дополнительные переменные, например *IMIN* и *JMIN*, в которые записываем соответствующие значения строки и столбца.

Поиск минимального элемента будет происходить в двух вложенных циклах и будет описываться следующей формулой:

$$XMIN = \begin{cases} X_{ij}, & \text{если } X_{ij} < XMIN, \quad IMIN = i, \quad JMIN = j; \\ XMIN, & \text{если } X_{ij} \ge YMIN. \end{cases}$$

{Программа нахождения местоположения наименьшего и наибольшего значений элементов матрицы *R*(4,5) на языке *Turbo Pascal*}

```
CONST R:ARRAY[1..4,1..5] OF REAL=((1,1.2,1.3,4,5),
                                  (1.1,2.2,2.3,2.4,2.5),
                                  (3.1,3.2,3.3,3.4,3.5),
                                 (4.1,4.2,4.3,4.4,4.5));
  VAR I,J,IMAX,JMAX,IMIN,JMIN:INTEGER;
      RMAX,RMIN:REAL;
   BEGIN
```

```
        {поиск максимального значения среди элементов матрицы}
    RMAX:=R[1,1]; {задали эталонное значение}
    IMAX:=1; {запомнили номер строки}
    JMAX:=1; {запомнили номер столбца}
    FOR I:=1 TO 4 DO BEGIN
     FOR J=1 TO 5 DO BEGIN
                IF R[I,J]>RMAX THEN BEGIN
                  RMAX:=R[I,J]; {запомнили максимальное}
                  IMAX:=I; {запомнили номер строки}
                  JMAX:=J; {запомнили номер столбца}
                                        END; {окончание IF}
                    END;    {окончание цикла по J}
                  END;     {окончание цикла по I}
WRITELN('RMAX=',RMAX:7:2,'IMAX',IMAX,'JMAX',JMAX);
    {поиск минимального значения среди элементов матрицы}
    RMIN:=MAXINT; {задали очень большое значение}
    IMIN:=0; {запомнили номер строки}
    JMIN:=0; {запомнили номер столбца}
    FOR I:=1 TO 4 DO BEGIN
     FOR J=1 TO 5 DO BEGIN
                IF R[I,J]<RMIN THEN BEGIN
                   RMIN:=R[I,J]; {запомнили минимальное}
                   IMIN:=I; {запомнили номер строки}
                   JMIN:=J; {запомнили номер столбца}
                                        END; {окончание IF}
                    END;    {окончание цикла по J}
                  END;     {окончание цикла по I}
WRITELN('RMIN=',RMIN:7:2,'IMIN',IMIN,'JMIN',JMIN);
  END.
```

Перестановка элементов матрицы местами

Перестановка элементов в матрице или между матрицами осуществляется так же, как и в одномерном массиве, с использованием промежуточной переменной.

Формула 20 Пример 20. Значения элементов матриц $A(5,4)$ и $R(5,4)$ поменять местами.

{Программа обмена значениями элементов матрицы $A(5,4)$ и $R(5,4)$ на языке *Turbo Pascal*}

```
VAR I,J:INTEGER;
    R,A:ARRAY[1..5,1..4] OF REAL;
    P:REAL;
 BEGIN
  {ввод значений элементов матрицы A(5,4)}
  FOR I:=1 TO 5 DO BEGIN
    WRITELN('вводи ',I,'-ю строку массива A по 4
             значения через пробел');
      FOR J:=1 TO 4 DO READ(A[I,J]);
                    END;
  {ввод значений элементов матрицы R(5,4)}
  FOR I:=1 TO 5 DO BEGIN
    WRITELN('вводи ',I,'-ю строку массива R по 4
             значения через пробел');
     FOR J:=1 TO 4 DO READ(R[I,J]);
                    END;
   {перестановка элементов между матрицами}
  FOR I:=1 TO 5 DO BEGIN
   FOR J:=1 TO 4 DO BEGIN
     P:=R([I,J]);{запоминаем в переменной P значение R(i,j)}
     R([I,J]):=A([I,J]); {пересылаем в R(i,j) значение A(i,j)}
     A([I,J]):=P;{в A(i,j) пересылаем значение, хранящееся в
                  переменной P, то есть значение R(i,j)}
                   END;
                  END;
   {вывод значений элементов матрицы R(5,4)}
  FOR I:=1 TO 5 DO BEGIN
   WRITELN('значения',I,'-ой      строки','матрицы
           R[I,J]');
   FOR J:=1 TO 4 DO WRITE(R[I,J]:6:2);
                  END;
         {вывод значений элементов матрицы A(5,4)}
  FOR I:=1 TO 5 DO BEGIN
   WRITELN('значения  ',I,'-ой  строки','  матрицы
           A[I,J]');
    FOR J:=1 TO 4 DO WRITE(A[I,J]:6:2);
                  END;
  END.
```

Формула 21 Пример_21. В матрице $Q(3,4)$ определить сумму значений элементов по строкам и произведение значений элементов по столбцам (рис. 22).

Сумма значений элементов любой строки матрицы определяется следующей формулой:

$$S_i = \sum_{j=1}^{N} Q_{ij} .$$

Произведение значений элементов любого столбца матрицы определяется следующей формулой:

$$P_j = \prod_{i=1}^{M} Q_{ij} ,$$

где M – это количество строк матрицы;

N – это количество столбцов матрицы.

Q_{11}		Q_{1j}		Q_{14}	$S_1 = \sum_{j=1}^{4} Q_{1j}$
Q_{i1}		Q_{ij}		Q_{i4}	$S_i = \sum_{j=1}^{4} Q_{ij}$
Q_{31}		Q_{3j}		Q_{34}	$S_3 = \sum_{j=1}^{4} Q_{3j}$
$P_1 = \prod_{i=1}^{3} Q_{i1}$		$P_j = \prod_{i=1}^{3} Q_{ij}$		$P_4 = \prod_{i=1}^{3} Q_{i4}$	

Рис. 22. Обработка элементов матрицы по строкам и столбцам

{Программа определения суммы значений элементов по строкам и произведения по столбцам значений элементов матрицы $Q(3,4)$ на языке *Turbo Pascal*}

```
CONST Q:ARRAY[1..3,1..4] OF REAL=((11,12,13,14),
                                  (21,22,23,24),
                                  (31,32,33,34));
  VAR S:ARRAY[1..3] OF REAL; {S – одномерный массив из 3-х элементов}
      P:ARRAY[1..4] OF REAL;{P – одномерный массив из 4-х элементов }
      I,J:INTEGER;
```

```
BEGIN       {определение сумм по строкам}
 FOR I:=1 TO 3 DO BEGIN
   S[I]:=0; {обнуление i-го элемента массива сумм S}
   FOR J:=1 TO 4 DO BEGIN
       S[I]:=S[I]*Q[I,J];
                    END; {окончание цикла по J}
                 END; {окончание цикла по I – по строкам}
       {определение произведений по столбцам}
 FOR J:=1 TO 4 DO BEGIN
       P[J]:=1; {j-ый элемент массива произведений =1}
   FOR I:=1 TO 3 DO BEGIN
       P[J]:=P[J]+Q[I,J];
                    END; {окончание цикла по I}
                 END; {окончание цикла по J – по столбцам}
{вывод полученных значений массива сумм и массива произведений}
   FOR I:=1 TO 3 DO WRITE('S[',I,']=', S[I]:6:2);
   WRITELN;
   FOR J:=1 TO 4 DO WRITE('P[',J,']=',P[J] ]:6:2);
END.
```

6.2.5. Матричная алгебра

В алгоритмических языках в отличие от обыкновенных арифметических операций не допускаются сложные выражения в операциях над матрицами. В операторе присваивания слева от знака должен стоять элемент матрицы, а справа от знака один или два элемента матриц в зависимости от типа операции. В операциях могут участвовать как элементы матриц, так и векторов.

Одномерный массив (вектор) можно интерпретировать как вектор-строку или как вектор-столбец. Конкретный способ интерпретации зависит от того, как этот вектор используется в операторе умножения матриц. В алгоритмических языках эти два типа векторов не различаются.

В табл. 13 приведены виды арифметических операций над матрицами. В таблице используются следующие обозначения:

- переменные *A*, *B* и *C* могут быть матрицами или векторами;

- *k* - скаляр, являющийся простой переменной, числовым выражением или константой.

Таблица 13 - Виды арифметических операций над матрицами

Операция	Действие	Комментарий
A=B	Копирование элементов *B* в *A*	
A=B+C	Поэлементное сложение *B* и *C* и копирование результатов в *A*	*B* и *C* должны иметь одинаковые размеры
A=B-C	Поэлементное вычитание *B* и *C* и копирование результатов в A	*B* и *C* должны иметь одинаковые размеры
*A=B*k*	Умножение каждого элемента *B* на скаляр *k* и копирование результатов в *A*	
*A=B*C*	Матричное умножение *B* и *C* в соответствии с правилами матричной алгебры	Число столбцов у *B* должно равняться числу строк у *C*

Матричная арифметика

Копирование элементов матрицы *B* в *A*.

{Программа копирования элементов матрицы *B* в *A* на языке *Turbo Pascal*}

```
CONST B:ARRAY[1..5,1..4] OF REAL=((1,2,3,4),
                                  (1,2,3,4),
                                  (1,2,3,4),
                                  (1,2,3,4),
                                  (1,2,3,4));
  VAR A:ARRAY[1..5,1..4] OF REAL;
      I,J:INTEGER;
     BEGIN
       FOR I:=1 TO 5 DO
       FOR J:=1 TO 5 DO A[I,J]:=B[I,J];
     END.
```

Сложение либо вычитание матриц

{Программа сложения либо вычитания матриц на языке *Turbo Pascal*}

```
VAR A,D,B,C:ARRAY[1..5,1..4] OF REAL;
    I,J:INTEGER;
  BEGIN
    FOR I:=1 TO 5 DO BEGIN
    FOR J:=1 TO 5 DO BEGIN
```

```
      READLN(B[I,J],C[I,J]);
      A[I,J]:=B[I,J]+C[I,J];
      D[I,J]:=B[I,J]-C[I,J];
                  END;
                END;
  END.
```

Умножение элементов матрицы на скаляр

{Программа умножения элементов матрицы на скаляр на языке *Turbo Pascal*}}

```
CONST B:ARRAY[1..3,1..2] OF REAL=((1,1),
                                  (2,2),(3,2));
       K=5; {скаляр, заданный константой}
  VAR A:ARRAY[1..3,1..2] OF REAL;
        I,J:INTEGER;
      BEGIN
       FOR I:=1 TO 5 DO
       FOR J:=1 TO 5 DO A[I,J]:=B[I,J]*K;
      END.
```

Операции *вычитания, сложения* и *деления* на скаляр элементов матрицы выполняются аналогично.

Матричное умножение

Для умножения матрицы *B* размером (L*M) на матрицу *C* размером (M*N) каждый элемент результирующей матрицы *A* равен выражению:

$$A(i,j)=\sum_{k=1}^{M}(B(i,k)\cdot C(k,j))\cdot$$

Размерность матриц для матричного умножения указана в табл. 14.

Таблица 14 – Размерность матриц для матричного умножения

Имя матрицы	*A*- итоговая матрица	B	C
Размеры	(L*N)	(L*M)	(M*N)

Из табл. 14 видно, что требуется совпадение числа столбцов у матрицы B с числом строк у матрицы C.

{Программа матричного умножения на языке *Turbo Pascal*}

```
CONST L=12; N=10; M=5;
  VAR A:ARRAY[1..L,1..N] OF REAL;
      B:ARRAY[1..L,1..M] OF REAL;
      C:ARRAY[1..M,1..N] OF REAL;
      I,J,K:INTEGER;
      S:REAL;
    BEGIN
        {ввод значений элементов массивов B и C}
      FOR I:=1 TO L DO    BEGIN
        FOR J:=1 TO N DO BEGIN
           S:=0;
           FOR K:=1 TO M DO BEGIN
            S:=S+B[I,K]*C[K,J];
                          END; {окончание цикла по K}
           A[I,J]:=S;
                       END;  {окончание цикла по J}
                        END;  {окончание цикла по L}
        END.
```

Транспонирование матрицы

Транспонированной матрицей называется квадратная матрица, у которой столбцы соответствуют строкам квадратной матрицы, причём, диагональные элементы у обеих матриц одни и те же. Операция транспонирования матрицы сводится к перестановке элементов симметрично относительно главной диагонали.

{Программа транспонирования матрицы на языке *Turbo Pascal*}

```
CONST B:ARRAY[1..4,1..4] OF REAL=((1,2,3,4),
                                  (1,2,3,4),
                                  (1,2,3,4),
                                  (1,2,3,4));
  VAR I,J:INTEGER;
      P:REAL;
    BEGIN
      FOR I:=1 TO 4 DO
        FOR J:=I+1 TO 4 DO BEGIN
          P:=A[I,J];
          A[I,J]:=A[J,I];
          A[J,I]:=P;
```

```
                END;
END.
```

6.3. Контрольные вопросы

1. Дайте определение массива в программировании.
2. Назовите возможные типы данных элементов массива.
3. Назовите способы задания одномерного массива на языке *Turbo Pascal.*
4. Как осуществляется ввод значений элементов одномерного массива?
5. Как осуществляется вывод значений элементов одномерного массива?
6. Как вычисляется сумма числовых значений элементов одномерного массива в математике и в программировании?
7. Как вычисляется произведение числовых значений элементов одномерного массива в математике и в программировании?
8. Как ведётся подсчёт различных значений элементов одномерного массива?
9. Как осуществляется перестановка местами элементов одномерного массива?
10. Опишите алгоритм поиска наибольшего числового значения элемента и его местоположение среди элементов одномерного массива.
11. Опишите алгоритм поиска наименьшего числового значения элемента и его местоположение среди элементов одномерного массива.
12. Назовите способы задания двумерного массива на языке *Turbo Pascal.*
13. Как осуществляется ввод значений элементов двумерного массива?
14. Как осуществляется вывод значений элементов двумерного массива?
15. Как вычисляется сумма числовых значений элементов двумерного массива в математике и в программировании?
16. Как вычисляется произведение числовых значений элементов двумерного массива в математике и в программировании?
17. Опишите алгоритм поиска наибольшего числового значения элемента и его местоположение среди элементов двумерного массива.
18. Опишите алгоритм поиска наименьшего числового значения элемента и его местоположение среди элементов двумерного массива.

19. Какие элементы называются главной диагональю квадратной матрицы?

20. Какие элементы называются побочной диагональю квадратной матрицы?

21. Как выполняется копирование значений элементов матрицы из одной в другую?

22. Как выполняется сложение (вычитание) значений элементов двух матриц?

23. Как выполняется умножение значений элементов матрицы на скаляр?

24. Как выполняется матричное умножение?

25. Как выполняется транспонирование элементов квадратной матрицы?

Глава 7.

ОБРАБОТКА СИМВОЛЬНЫХ И СТРОКОВЫХ ДАННЫХ

В этой главе рассказывается о двух типах данных: символьных и строковых, алгоритмах их обработки.

7.1. Символьный тип данных

Обработка символьных (знаковых, литерных) данных становится возможным благодаря привлечению значений и переменных типа `CHAR` и на языке *Turbo Pascal,* и на языке *C++*.

В программе на языке *Turbo Pascal* описание переменной задаётся следующим образом:

```
VAR  <список_имён_переменных>:CHAR;{символ занимает в
                                    памяти один байт}
```

В программе на языке *C++* описание переменной задаётся следующим образом:

```
char <список_имён_переменных>;
```

Операции над символьными данными

1. Операция присвоения осуществляется только между символьными переменными либо символьной переменной и символьной константой, причём символьная константа заключается в апострофы[14].

```
{Программа с использованием операции присвоения символьной
 переменной на языке Turbo Pascal}
VAR U,V:CHAR;  {U,V объявлены как символьные переменные}
  BEGIN
    U:='A';    {символьной переменной U присвоено значение
                символьной константы}
    V:=U;      {символьной переменной V присвоено значение
                символьной переменной U}
  END.
```

[14] В языке *C++* при наборе символьной или строковой константы набираются кавычки «“».

2. Операция ввода.

При наборе символа апострофы не набираются. И на отдельной строке набирается только один символ.

{Программа с использованием операции ввода символа на языке *Turbo Pascal*}

```
VAR U:CHAR; {U объявлена как символьная переменная}
  BEGIN
   WRITELN('вводи символьное значение');
   READLN(U);
  END.
```

3. Операция вывода.

При выводе символа апострофы не печатаются.

{Программа с использованием операции вывода символа на языке *Turbo Pascal*}

```
VAR V:CHAR; {V объявлена как символьная переменная}
  BEGIN
    V:='A'; {символьной переменной V присвоено значение символьной константы}
    WRITELN('символьное значение=',V);
  END.
```

4. В качестве условий после ключевых слов `IF` и `WHILE`, при этом допустимы операции отношения: равно, не равно, меньше и больше.

Формула 22 Пример 22. Ввести символы и распечатать их, заменяя при выводе восклицательные знаки на запятые. Ограничителем ввода принять символ «/».

{Программа с использованием символа в качестве условий в операторах `IF` и `WHILE` на языке *Turbo Pascal*}

```
VAR V:CHAR; {V объявлена как символьная переменная}
  BEGIN
    WRITELN('вводи символьное значение');
    READLN (V);
    WHILE V<>'/' DO BEGIN
       IF V='!' THEN WRITELN(',')
                ELSE WRITELN(V);
       WRITELN('вводи символьное значение');
```

```
      READLN (V);
                    END;
    END.
```

5. В операторе цикла FOR и операторе выбора CASE можно использовать переменную типа CHAR, так как каждому символу соответствует своё числовое значение.

```
      {Программа с использованием символьной переменной в
       операторе цикла FOR на языке Turbo Pascal}
VAR V:CHAR; {V объявлена как символьная переменная}
    R,C,D:REAL;
  BEGIN
    FOR V:='A' TO 'Z' DO
        WRITELN('символьное значение=',V);READLN;
    FOR V:='a' TO 'z' DO
        WRITELN('символьное значение=',V);READLN;
       {ввод двух чисел и выполнение над ними четырёх
        арифметических действий}
    WRITELN('вводи два числа');
    READLN(C,D);
    FOR V:='*' TO '/' DO BEGIN
     CASE V OF
     '+': BEGIN R:=C+D; WRITELN('V=',V,'R=',R:6:2);
          END;
     '-': BEGIN R:=C-D; WRITELN('V=',V,'R=',R:6:2);
          END;
     '*': BEGIN R:=C*D; WRITELN('V=',V,'R=',R:6:2);
          END;
     '/': BEGIN R:=C/D; WRITELN('V=',V,'R=',R:6:2);
          END;
     END; {конец CASE}
                                 END; {конец цикла FOR по V}
    READLN;
  END.
```

6. Функции, используемые при работе с символьными данными

Функции, используемые при работе с символьными данными на языке *Turbo Pascal*, приведены в табл. 15.

Таблица 15

Функции, используемые при работе с символьными данными на языке *Turbo Pascal*

Функция	Назначение
CHR(X)	- возвращает символ с заданным в *X* порядковым номером в коде *ASCII;*
ORD(X)	- возвращает порядковый номер для значения *X;*
PRED(X)	- возвращает значение, предшествующее аргументу *X;*
SUCC(X)¦	- возвращает значение, следующее аргументу *X.*

```
{Программа с использованием функций над символьными переменными на языке Turbo Pascal}
VAR V:CHAR; {V объявлена как символьная переменная}
    I,D:INTEGER;
  BEGIN
    FOR V:='A' TO 'Z' DO
       WRITELN('литерное=',V,'ЦИФРОВОЕ=',ORD(V));
    FOR I:=97 TO 122  DO
       WRITELN('ЦИФРОВОЕ=',I,'литерное=',CHR(I));
    FOR V:='B' TO 'Z' DO
       WRITELN('ПРЕДШЕСТВЕННИК',V,'=',PRED(V));
    FOR V:='A' TO 'Y' DO
       WRITELN('СЛЕДУЮЩИЙ ЗА',V,'=',SUCC(V));
    READLN;
  END.
```

7.2. Строковый тип данных

Строка – это массив символов.

Обработка строковых данных становится возможной благодаря привлечению значений и переменных типа STRING на языке *Turbo Pascal* и char[ ] в языке *C++*.

В программе на языке *Turbo Pascal* описание строковой переменной задаётся следующим образом:

VAR <имя_переменной>:STRING[N];

где N - длина строки в байтах;
если N отсутствует, то длина строки равна 256 байт.

Операции над строковыми данными

1. **Операция присвоения.**

Операция присвоения может осуществляться:

- между строковой переменной и строковой константой, которая заключается в апострофы;
- между строковыми переменными;
- между строковой переменной и символьной константой;
- между строковой переменной и символьной переменной.

```
{Программа с использованием операций присвоения строковой переменной на языке Turbo Pascal}
VAR STR1,STR2:STRING[56]; {STR1,STR2 объявлены как строковые переменные длиной в 56 символов}
    V:CHAR; {V объявлена как символьная переменная}
  BEGIN
    {между строковой переменной и строковой константой}
    STR1:='HELLO!';
    {между строковыми переменными}
    STR2:=STR1;
    {между строковой переменной и символьной константой}
    STR2:='A';
    {между строковой переменной и символьной переменной}
    V:='A';
    STR2:=V;
    END.
```

2. **Операция ввода**.

При вводе строки апострофы на языке *Turbo Pascal* и кавычки на языке *C++* не набираются. На отдельной строке набирается значение только одной строковой переменной!

```
{Программа с использованием операции ввода строки на языке Turbo Pascal}
VAR STR1:STRING[56]; {STR1 объявлены как строковая переменная длиной в 56 символов}
  BEGIN
    WRITELN('вводи строковое значение «HELLO!»');
    READLN(STR1);
  END.
```

3. **Операция вывода**.

При выводе строки апострофы на языке *Turbo Pascal* и кавычки на языке *C++* не печатаются.

```
{Программа с использованием операции вывода строки на языке
 Turbo Pascal}
VAR   STR1:STRING[56];   {STR1 объявлены как строковая
                          переменная длиной в 56 символов}
  BEGIN
    STR1:='КУ-КА-РЕ-КУ!';
    WRITELN('строковое значение=',STR1:20);
  END.
```

4. **В качестве условий** после служебных слов `IF` и `WHILE`, при этом допустимы операции отношения: равно, не равно, меньше и больше.

Сравнение строк производится *слева направо* до первого несовпадающего символа, и та строка считается бóльше, в которой первый несовпадающий символ имеет бóльший номер в кодовой таблице.

5. К отдельному символу строки можно обратиться по номеру (индексу) данного символа в строке. Индекс определяется выражением целочисленного типа, которое записывается в квадратных скобках сразу за именем строковой переменной или константы, например, выражения `STR1[5]` и `STR1[56]` обеспечат доступ соответственно к пятому и пятьдесят шестому символу строковой переменной `STR1`.

Формула 23 Пример 23. Ввести строку и подсчитать, сколько букв «*R*» встретится среди символов строки.

```
{Программа ввода строки и подсчёта количества букв «R» среди
 символов строки на языке Turbo Pascal}
VAR   STR1:STRING[56];   {STR1 объявлены как строковая
                          переменная длиной в 56 символов}
    I,CH:INTEGER;  {CH – это счетчик числа букв «R»}
  BEGIN
   CH:=0;
   WRITELN('вводи строковое значение <=56 символов');
   READLN(STR1);
   FOR I:=1 TO 56 DO BEGIN
           IF STR1[I]='R' THEN CH:=CH+1;
                     END;
```

```
   WRITELN('CH= ',CH);
  END.
```

6. **Операция сцепления.**

Операция сцепления – это операция «+», когда несколько строк объединяются в одну результирующую.

```
{Программа с использованием операции сцепления на языке
 Turbo Pascal}
VAR   STR1,STR2,STR3:STRING[56];   {объявлены строковые
                                    переменные}
  BEGIN
    STR1:='Petrow Iwan';
    STR2:='- programmist';
    STR3:=STR1+STR2; {Petrow Iwan- programmist}
    WRITELN('строковое значение 1=',STR3);
    STR3:='dom'+' nomer '+'21/2';
    WRITELN('строковое значение 2=',STR3);
  END.
```

7. Стандартные процедуры для обработки строковых переменных

Стандартные процедуры для обработки строковых переменных имеются только в языке *Turbo Pascal.*

7.1. `DELETE(STR1,POZ,N);` - это процедура удаления `N` символов строки `STR1`, начиная с позиции `POZ`.

```
VAR STR1:STRING[14]; {STR1 объявлена как строковая переменная}
  BEGIN
    STR1:='КУ-КА-РЕ-КУ';
    DELETE(STR1,4,6);{начиная с 4-ой позиции удаляются 6 сим-
                      волов строки STR1}
    WRITELN('строковое значение =',STR1);
  END.
```

7.2. `INSERT(STR1,STR2,POZ);` - это процедура вставки строки `STR1` в строку `STR2`, начиная с позиции `POZ`.

```
{Программа с использованием процедуры INSERT}
```

```
VAR STR1:STRING[6];
    STR2:STRING[12];
  BEGIN
    STR1:='КА-РЕ-';
    STR2:='КУ-КУ';
    INSERT(STR1,STR2,4);
    WRITELN('строковое значение=',STR2);
  END.
```

7.3. `STR(IBR,STR1);` - это процедура преобразования числового значения величины `IBR` и помещение результата в строковую переменную `STR1`.

Преобразование целочисленных и вещественных типов данных в строки удобно осуществлять пользовательскими функциями, назвав их, например: `INTST` и `REALST`.

```
      {Подпрограмма-функция преобразования целочисленного
      значения в строку на языке Turbo Pascal}
FUNCTION INTST(INT:INTEGER):STRING;
        {INT - целочисленное значение}
  VAR BUF:STRING[10];
    BEGIN
     STR(INT,BUF);
     INTST:=BUF;
    END;      {конец функции INTST}

      {Подпрограмма-функция преобразования вещественного
      значения в строку на языке Turbo Pascal}
FUNCTION REALST(R:REAL;DIG,DEC:INTEGER):STRING;
        {R - значение, DIG - количество символов}
        {DEC - количество символов после запятой}
  VAR BUF:STRING[20];
    BEGIN
     STR(R:DIG:DEC,BUF);
     REALST:=BUF;
    END;       {конец функции REALST}
```

7.4. `VAL(STR1,IBR,COD);` - это процедура преобразует строковое значение числа `STR1` в числовое значение и помещает результат в `IBR`. В переменной `COD` хранится информация о результате преобразо-

вания: 0 - если во время операции преобразования ошибки не обнаружено, в противном случае будет содержать номер позиции первого ошибочного символа.

8. Стандартные функции[15] для обработки строковых данных

8.1. `COPY(STR1,POZ,N)` – эта функция выделяет из `STR1` подстроку длиной `N` символов, начиная с позиции `POZ`.

```
     {Программа с использованием функции COPY}
VAR STR1:STRING[11];
    STR2,STR3:STRING[2];
    STR4:STRING[4];
  BEGIN
    STR1:='КУ-КА-РЕ-КУ';
    STR2:=COPY(STR1,7,2); {выделили 2 символа 'РЕ'}
    STR3:=COPY(STR1,4,2); {выделили 2 символа 'КА'}
    STR4:=STR2+STR3; {объединили все 4 символа, получили слово 'РЕКА'}
    WRITELN('строковое значение =',STR4);
  END.
```

8.2. `CONCAT(STR1,STR2,...,STRN)` - эта функция выполняет сцепление строковых переменных *STR1*,*STR2*,...,*STRN* в том порядке, в каком они указаны в списке параметров(функция по своему действию идентична операции сцепления, описанной в пункте 6).

```
     {Программа с использованием функции CONCAT}
VAR STR1:STRING[11];
    STR2,STR3:STRING[2];
    STR4:STRING[4];
  BEGIN
    STR1:='КУ-КА-РЕ-КУ';
    STR2:=COPY(STR1,7,2); {выделили 2 символа 'РЕ'}
    STR3:=COPY(STR1,4,2); {выделили 2 символа 'КА'}
    STR4:=CONCAT(STR2,STR3);{объединили все 4 символа, получили
                             слово 'РЕКА'}
    WRITELN('строковое значение=',STR4);
  END.
```

[15] В *C++* для работы со строками символов имеются функции, прототипы которых содержатся в заголовочном файле <string.h>.

8.3. POS(STR1,STR2) - эта функция обнаруживает первое появление в строковой переменной STR2 строковой переменной (подстроки) STR1. Результат имеет целочисленный тип и равен номеру той позиции, где находится первый символ подстроки STR1. Если в STR2 подстроки STR1 не найдено, то результат равен 0.

8.4. LENGTH(ST) - эта функция возвращает текущую длину строки.

{Программа с использованием функции POS и LENGTH}

```
VAR STR1:STRING[11];
    STR2,STR3:STRING[2];
    STR4:STRING[4];
    I:INTEGER;
  BEGIN
    STR1:='КУ-КА-РЕ-КУ';
    STR2:='РЕ';  {присвоили 2 символа 'РЕ'}
    I:=POS(STR2,STR1);  {обнаружили первое появление 'РЕ' в STR1}
    WRITE('I=',I);
    STR1:='КУ-КА';
    I:=LENGTH(STR1);  {получили текущую длину строки STR1=5}
    WRITELN('I=',I);
  END.
```

9. Массивы символьных и строковых значений

Символьные и строковые значения можно объединять в массивы.

{Программа с использованием массивов символьных и строковых переменных на языке *Turbo Pascal*}

```
CONST MASSTR1:ARRAY[1..12] OF STRING[6]=('МЫШЬ',
        'БЫК','ТИГР','КОТ','ДРАКОН','ЗМЕЯ','КОНЬ',
        'КОЗА','МАКАКА','ПЕТУХ','СОБАКА','КАБАН');
    MASSTR2:ARRAY[1..27] OF CHAR=('А','Б','В','Г',
        'Д','Е','Ж','З','И','К','Л','М','Н','О',
        'П','Р','С','Т','У', 'Ф','Х','Ч','Ш','Щ',
         'Э','Ю','Я');
 VAR MASSTR3:ARRAY[1..27] OF CHAR;
     MASSTR4:ARRAY[1..32] OF STRING[26];
```

10. Запись даты на документах в различных странах

Запись даты на документах в различных странах мира отличается порядком следования числа, месяца, года, разделителями между цифрами (см. табл. 16).

Таблица 16

Запись даты на документах в различных странах

Страна или стандарт	Обозначение даты
американская	ММ/ДД/ГГ
USA	ММ-ДД-ГГ
ANSI	ГГ-ММ-ДД
BRIT/FRANC	ДД/ММ/ГГ
GERMAN	ДД.ММ.ГГ
ITALIAN	ДД-ММ-ГГ
JAPAN	ГГ/ММ/ДД
MDY	ММ/ДД/ГГ
DMY	ДД/ММ/ГГ
YMD	ГГ/ММ/ДД

где ДД - цифры даты;
ММ - цифры месяца;
ГГ - год (две последние цифры).

Знайте, что сравнивать даты в строковом представлении бесполезно: такое сравнение будет неверным. Для сравнения дат, необходимо убрать разделители и преобразовать строковое значение даты в числовое значение с помощью процедуры `VAL(ST,IBR,COD)` (см. п.7.4.). После преобразований сравнивают численное значение дат.

7.3. **Контрольные вопросы**

1. Каким ключевым словом описывается символьный тип данных?
2. Между какими данными может осуществляться операция присвоения с участием символьной переменной?
3. Как осуществляется ввод и вывод значений символьной переменной?
4. Какие допустимы операции отношения в условиях с символьными переменными после ключевых слов `IF` и `WHILE`?

5. Почему символьную переменную можно использовать в операторе цикла `FOR` и операторе выбора?

6. Назовите функции, которые можно использовать при работе с символьными данными на языке *Turbo Pascal.*

7. Дайте определение термину «строка».

8. Между какими данными может осуществляться операция присвоения с участием строковой переменной на языке *Turbo Pascal*?

9. Как можно обратиться к отдельному символу строки на языке *Turbo Pascal*?

10. Как осуществляется ввод и вывод значений строковой переменной на языке *Turbo Pascal*?

11. Дайте определение операции сцепления над строковыми данными на языке *Turbo Pascal*.

12.Назовите стандартные процедуры для обработки строковых переменных на языке *Turbo Pascal*.

13. Назовите стандартные функции для обработки строковых переменных на языке *Turbo Pascal*.

11. Как описываются массивы символьных значений на языке *Turbo Pascal*?

12. Как описываются массивы строковых значений на языке *Turbo Pascal*?

13. В чём особенность обработки дат на документах в различных странах?

Глава 8.

ПОДПРОГРАММЫ: ПРОЦЕДУРЫ, ФУНКЦИИ. РЕКУРСИЯ

Подпрограммы

Часто в программе имеются однотипные участки, которые выполняют одни и те же действия или вычисления, но с различными, нерегулярными значениями данных. Такие части программы оформляются как *подпрограммы*. Подпрограмма – это логически самостоятельная именованная часть программы, в которую могут передаваться параметры, и которая может возвращать какое-то значение.

Использование подпрограмм позволяет:

– сделать основную программу более наглядной и компактной;

– уменьшить объём используемой памяти компьютера.

В *Turbo Pascal* имеются два вида подпрограмм:

- процедуры;
- функции.

На языке *C++* имеются только подпрограммы-функции.

На обоих языках <*имя_подпрограммы*> – это произвольно выбираемый идентификатор. Имя подпрограммы не должно совпадать:

- с ключевыми (служебными) словами языка;
- именами библиотечных функций;
- с другими именами переменных в головной программе.

8.1. Подпрограммы на языке *Turbo Pascal*

Структура процедур и функций в *Turbo Pascal* такая же, как и структура основной программы, то есть включает *заголовок* и *блок*. В свою очередь блок состоит из *раздела описаний* и *раздела операторов*.

Текст подпрограмм может быть размещён в основной программе одним из следующих способов:

– текст располагается непосредственно в разделе описаний (`VAR`) основной программы;

– текст оформляется в виде отдельного внешнего модуля.

8.1.1. Процедуры

Описание процедуры имеет следующий вид:

`PROCEDURE` <*имя_процедуры*>(<*формальные_параметры*>);

```
{раздел описаний}
  BEGIN
    {раздел операторов}
  END;
```

где <*формальные_параметры*> – это список переменных с указанием их типа, которые отделяются друг от друга точкой с запятой. Переменные из списка формальных параметров не надо описывать в разделе описаний процедуры.

Если процедуру необходимо расположить в основной программе, то она помещается в разделе описаний (VAR) основной программы.

Параметры процедур делятся на два вида:

– *входные*, так называемые *параметры-значения*;

– *выходные*, так называемые *параметры-переменные*.

Описание *входных* параметров процедуры в списке формальных параметров имеет следующий вид:

*список_переменных*1*:тип*1*; список_переменных*2*:тип*2*;...*

Входными параметрами, то есть теми, которые передаются в процедуру, могут быть константы, переменные и выражения. Значение входного параметра внутри процедуры может изменяться, однако никакого влияния на значение входного параметра это не оказывает. Поэтому входной параметр (параметр-значение) *не может быть результатом работы процедуры*!

Для сохранения результата работы процедуры необходимо использовать *выходные* параметры (*параметры-переменные*), которые могут быть только переменными.

Описание *выходных* параметров выполняется следующим образом:

VAR *список_переменных*11*:тип*11*; список_переменных*22*:тип*22*;...*

Вызов *процедуры* в основной программе производится оператором вида:

<*имя_процедуры*>(<*фактические_параметры*>);

где <*фактические_параметры*> – это список фактических параметров, перечисленных через запятую без указания их типа.

Между формальными и фактическими параметрами должно быть соответствие:

- по количеству параметров;
- по порядку следования параметров;
- по типу данных параметров.

Имена соответствующих параметров в процедуре и основной программе могут быть одинаковыми или разными.

Формула 24 Пример 24. Вычислить выражение $B = F(r, p) - F(15, a + c)$ при следующих значениях r и p: $r = a + 2;\ p = c.$

Вычисление функции $F(x, z)$ оформить процедурой по формуле

$$F(x, z) = x^2 + z / 2.$$

Блок-схема вычислений примера 24 с использованием подпрограммы приведена на рис. 23.

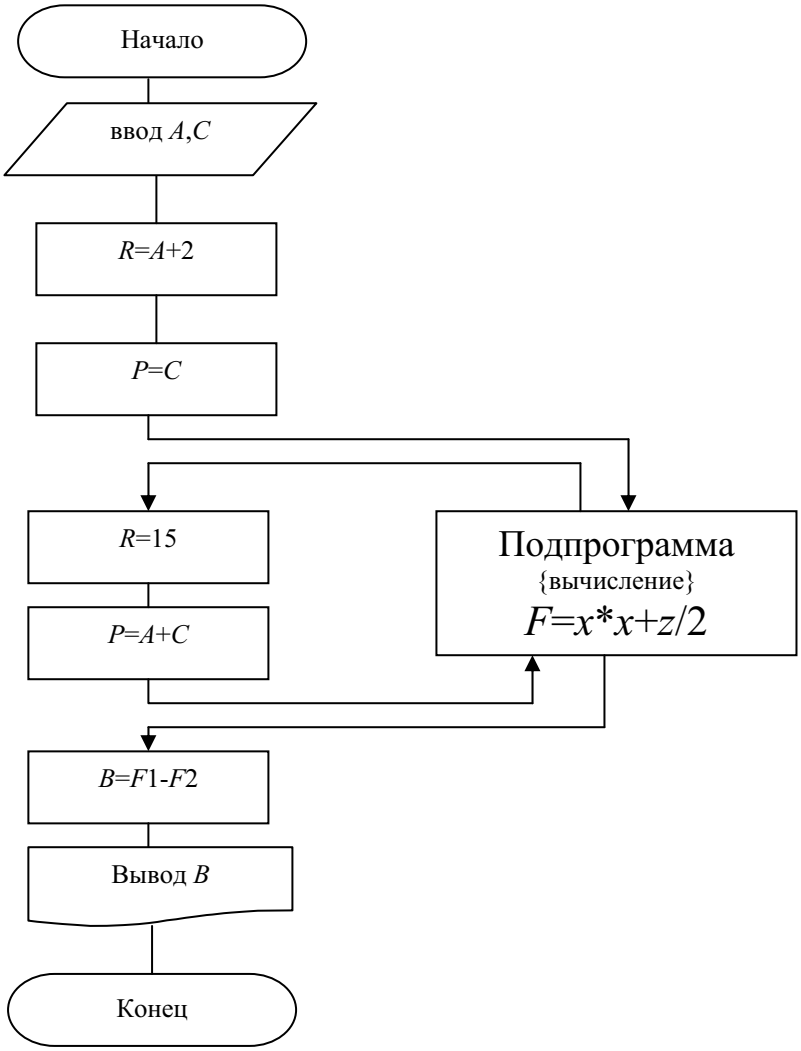


Рис. 23. Блок-схема вычислений с использованием подпрограммы

{Головная программа с подпрограммой-процедурой по блок-схеме рис. 23}

```
VAR A,C,R,P,F1,F2,B:REAL;
  PROCEDURE PROCF(X,Z:REAL;VAR F:REAL);
        {PROCF     – имя процедуры}
        {X, Z      – входные формальные параметры}
        {F         – выходной формальный параметр, перед описа-
                     нием которого обязательно необходим VAR}
        {раздел описаний VAR в данном примере отсутствует}
     BEGIN {вычислительный блок процедуры PROCF}
       F:=X*X+Z/2;
     END; {конец процедуры PROCF}
    BEGIN {начало головной программы}
     WRITELN('вводи значения A, C через пробел');
     READ(A,C);
     R:=A+2;
     P:=C;
     PROCF(R,P,F1);{первое обращение к процедуре PROCF, вы-
                    численный результат будет в переменной F1}
     PROCF(15,A+C,F2);{второе обращение к процедуре PROCF,
                   вычисленный результат будет в переменной F2}
     B:=F1+F2;
     WRITELN('B= ',B:7:2);
    END. {конец головной программы}
```

8.1.2. Функции

Функция оформляется аналогично процедуре и отличается от неё по структуре только заголовком, который имеет следующий вид

FUNCTION <*имя_функции*>(<*формальные_параметры*>):*тип*;

Функция имеет отличительные особенности от процедуры:

1. Функция всегда имеет только *один результат* выполнения, но может иметь несколько входных параметров.

2. Результат внутри функции обозначается именем функции. Поэтому в конце раздела операторов функции обязательно должен присутствовать оператор присваивания, в левой части которого стоит имя этой функции.

3. В заголовке функции обязательно должен быть указан тип функции.

4. Вызов функции в основной программе осуществляется непосредственно внутри выражения по её имени с указанием фактических параметров.

```
      {Головная программа с подпрограммой-функцией по блок-схеме
       рис. 23}
VAR A,C,R,P,F1,F2,B:REAL;
     FUNCTION FUNCF(X,Z:REAL):REAL;
        {FUNCF      – имя функции}
        {X, Z       – входные параметры}
       VAR F:REAL; {описана вспомогательная переменная F}
         BEGIN   {начало головной программы}
           F:=X*X+Z/2;
           FUNCF:=F;
         END;   {конец функции FUNCF}
   BEGIN {начало головной программы}
     WRITELN('вводи значения A,C через пробел');
     READ(A,C);
     R:=A+2;
     P:=C;
     F1:=FUNCF(R,P); {первое обращение к функции FUNCF; вычисленный результат будет в переменной
                      F1}
     F2:=FUNCF(15,A+C); {второе обращение к функции FUNCF;
                         вычисленный результат будет в переменной F2}
     B:=F1+F2;
     WRITELN('B= ',B:7:2);
   END. {конец головной программы}
```

Формула 25 Пример 25. Даны действительные числа a,d,c,y,x. Вычислить выражение:

$$B=\max(a,d,c)-\max(x+y,a-d,d+c)\cdot\max(y-x,d-a,c),$$

оформив вычисление функции $\max(f,g,h)$ в процедуре и в функции.

```
      {Программа с использованием процедуры и функции}
VAR A,D,C,Y,X:REAL;
     B,MAX1,MAX2,MAX3:REAL;

     PROCEDURE MAXI(F,G,H:REAL;VAR MAX:REAL);
```

```
    {здесь     MAXI   – имя процедуры;
               F,G,H  – входные параметры;
               MAX – выходной параметр, перед описанием ко-
      торого обязательно необходимо ключевое слово VAR}
         {раздел описаний в этом примере отсутствует}
 BEGIN {раздел операторов процедуры MAXI}
   IF F>G THEN MAX:=F
        ELSE IF G>F THEN MAX:=G;
   IF H> MAX THEN MAX:=H;
 END; {конец процедуры MAXI}

  FUNCTION MIXI(F,G,H:REAL):REAL;
    {здесь     MIXI   – имя функции;
               F,G,H  – входные параметры;
               MIXI   – результат вычисления функции}
        {раздел описаний функции MIXI}
    VAR MAX:REAL;
     BEGIN
       IF F>G THEN MAX:=F
              ELSE IF G>F THEN MAX:=G;
       IF H>MAX THEN MAX:=H;
       MIXI:=MAX;
     END; {конец функции MIXI}

BEGIN {начало головной программы}
 WRITELN('вводи значения a,d,c,x,y через пробел');
  READLN(A,D,C,X,Y);
  MAXI(A,D,C,MAX1); {первое обращение к процедуре MAXI,
       вычисленный результат будет в переменной MAX1}
  MAXI(X+Y,A-D,D+C,MAX2); {второе обращение к проце-
       дуре MAXI, вычисленный результат будет в переменной
       MAX2}
  MAXI(Y-X,D-A,C,MAX3); {третье обращение к процедуре
       MAXI, вычисленный результат будет в переменной MAX3}
  B:=MAX1+MAX2*MAX3;
  WRITELN('B= ',B:7:2);

  MAX1:=MIXI(A,D,C); {первое обращение к функции MIXI,
            вычисленный результат будет в переменной MAX1}
```

```
MAX2:=MIXI(X+Y,A-D,D+C); {второе обращение к функ-
     ции MIXI, вычисленный результат будет в переменной
     MAX2}
MAX3:=MIXI(Y-X,D-A,C); {третье обращение к функции
     MIXI, вычисленный результат будет в переменной
     MAX3}
B:= MAX1+MAX2*MAX3;
WRITELN('B= ',B:7:2);
END.
```

8.2. Подпрограммы-функции на языке *C++*

Данный раздел содержит краткое описание функций на языке *C++*, цель которого - это показать основные отличия в оформлении подпрограмм от языка *Turbo Pascal*.

Большинство языков программирования используют процедуры и функции. Язык *C++* не поддерживает подпрограммы-процедуры. Все подпрограммы языка *C++* являются функциями.

Описание функции имеет следующий вид:

<тип_результата> *<имя_функции>*(*формальные_параметры*)
{
<определение переменных>
<раздел операторов>
`return` [*<выражение>*];
}

Важным оператором тела функции является оператор возврата из функции в точку её вызова с помощью оператора

`return` [выражение]; .

Формула 26 Пример 26. Записать функцию вычисления *sin(x),* используя сумму ряда:

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$ до члена ряда с точностью $\varepsilon = 10^{-3}$.

```
/* Функция sinm вычисления суммы ряда для sin(x) */
double sinm(double x)
  { double eps=1e-3;
    double u=x, x2=x*x, s=x, n=2;
    do
```

```
    { u=(-1)*u*x2/(n*(n+1));
      s=s+u;
      n=n+2; }
   while (fabs(u)>eps);   /*fabs – возвращает абсолютное
                            значение аргумента типа double*/
   return s;
}
```

8.3. Рекурсия

Рекурсия - это очень мощный инструмент решения некоторых задач. Но пользоваться им следует достаточно и осторожно.

Использование идентификатора процедуры в теле самой процедуры означает *рекурсивное* обращение к процедуре.

Формула 27 Пример 27. Вычислить сумму $s = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{N}$.

{Обычная итеративная процедура на языке *Turbo Pascal*}

```
PROCEDURE EGSUM(VAR SUM:REAL;N:INTEGER);
  VAR I:INTEGER;
      S:REAL;
    BEGIN
     S:=0;
     FOR I:=1 TO N DO
         S:=S+1/I;
     SUM:=S;
    END;    {конец процедуры EGSUM}
```

Эти же вычисления можно описать и с использованием рекурсивной процедуры:

{Рекурсивная процедура на языке *Turbo Pascal*}

```
PROCEDURE SUMEG(VAR SUM:REAL;N:INTEGER);
  BEGIN
    IF N=1 THEN SUM:=1
          ELSE BEGIN
                 SUMEG(SUM,N-1);
                 SUM:=SUM+1/N;
               END;
  END;    {конец процедуры SUMEG}
```

Работу рекурсивной процедуры понять сложнее, чем итеративной. Приведём работу рекурсивной процедуры для значения N=5.

Оператор процедуры для вычисления суммы $s = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5}$ будет таким:

```
SUMEG(SUM,5);
```

При выполнении тела процедуры будет выполняться ELSE-часть условного оператора. В результате сформируются следующие два оператора:

```
SUMEG(SUM,4);
SUM:=SUM+1/5;
```

Появился новый оператор процедуры, но с другим вторым фактическим параметром. Он заставляет снова обратиться к выполнению этой же процедуры, что приводит к появлению новой пары операторов:

```
SUMEG(SUM,3);
SUM:=SUM+1/4;
SUM:=SUM+1/5;
```

После выполнения еще двух обращений ситуация окажется следующей:

```
SUMEG(SUM,1);
SUM:=SUM+1/2;
SUM:=SUM+1/3;
SUM:=SUM+1/4;
SUM:=SUM+1/5;
```

Теперь при очередном выполнении оператора процедуры сработает оператор присваивания SUM:=1; и рекурсивные обращения прекратятся. В результате формируется следующая последовательность операторов:

```
SUM:=1;
SUM:=SUM+1/2;
SUM:=SUM+1/3;
SUM:=SUM+1/4;
SUM:=SUM+1/5;
```

которая и даёт результат вычисления суммы.

Говорят, что решение некоторой задачи *рекурсивно*, если его можно выразить в терминах решения более простой версии этой же задачи. Естественно, что для прекращения вычислений должно существовать простое нерекурсивное решение простой версии исходной задачи.

В приведенном примере при *N*=1 решение очевидно. Однако если его не предусмотреть в процедуре, то она никогда не завершится.

8.4. Обработка массивов, строковых переменных в подпрограммах на языке *Turbo Pascal*

Может сложиться впечатление, что объявление переменных в списке *формальных* параметров подпрограмм (как процедур, так и функций) ничем не отличается от объявления их в разделе описания переменных. Действительно, в обоих случаях много общего, но есть одна существенная деталь: типом любого параметра в списке формальных параметров может быть только *стандартный* или *ранее объявленный тип* в разделе TYPE.

Обработка массивов в подпрограммах

Нельзя объявлять в заголовке процедуры массив в качестве формальных параметров:

```
PROCEDURE PROCS(C:ARRAY[1..10] OF REAL);
```

Если в подпрограммный блок передаётся весь массив, то следует первоначально описать его тип в разделе TYPE.

```
{Описание вектора или матрицы в разделе TYPE}
TYPE   MAS=ARRAY[1..10]   OF   REAL;{раздел   TYPE   в
                                    основной программе}
  VAR D:MAS;{переменная D объявлена типом MAS, то есть является
            одномерным массивом, состоящим из 10 элементов. В го-
            ловной программе все операции производятся над пере-
            менной D[I]}
    PROCEDURE PROCS(C:MAS);{формальный параметр C объяв-
            лен типом MAS, т.е. является одномерным массивом, со-
            стоящим из 10 элементов. В процедуре все операции про-
            изводятся над переменной C[I]}
          {операторы процедуры PROCS}
          END;
    FUNCTION FUNCS(B:MAS):REAL;{формальный параметр C
            объявлен типом MAS, то есть является одномерным мас-
            сивом, состоящим из 10 элементов. В функции все опера-
            ции производятся над переменной B[I]}
          {операторы функции FUNCS}
```

```
END;

{головная программа}
    {обращение к процедуре PROCS}
PROCS(D);{в списке фактических параметров указывается мас-
                сив D}
    {обращение к функции FUNCS}
S:=FUNCS(D);{в списке фактических параметров указывается
                    массив D}
```

Обработка строковых переменных в подпрограммах

Строковая переменная (тип STRING) является массивом символов, поэтому её передача в подпрограммы осуществляется аналогичным образом.

```
  {Описание строковой переменной разделе TYPE}
TYPE INTYP=STRING[15];
     OUTTYP=STRING[30];
 VAR STR1:INTYP;
     STR2:OUTTYP;
 PROCEDURE PROCIS(R:INTYP;VAR T:OUTTYP);
   {процедурный блок}
   END;
FUNCTION FUNCIS(R:INTYP):CHAR;
   {блок функции}
   END;
```

Формула 28 Пример 28. Составить программу с обращением:

- к процедуре, в которой оформить умножение матрицы на скаляр. Изменённую матрицу вернуть в основную программу;
- к функции, в которой оформить суммирование всех элементов матрицы.

Вычисления оформить для двух матриц: B(5,10) и C(12,6).

Сразу необходимо договориться, что при описании размерности матрицы в разделе типов (TYPE), значения верхних границ зададим по строкам 12 (максимальное из значений 5 и 12) и по столбцам 10 (максимальное из значений 10 и 6).

Значения элементов матриц B(5,10) и C(12,6) вводить с клавиатуры.

```
TYPE MAS=ARRAY[1..12,1..10] OF REAL;
  CONST K=5; {скаляр, заданный константой}
   VAR B,C:MAS; {массивы B и C имеют одинаковые размерности 12*10}
       M,N:INTEGER; {переменные для задания фактической размерности}
       I,J:INTEGER; {индексные переменные}
       S1,S2:REAL; {вспомогательные переменные для сумм}
     PROCEDURE PROCS(VAR D:MAS;M,N:INTEGER);
        {здесь     PROCS   - имя процедуры;
                   M,N     - входные параметры, задающие фактическую размерность обрабатываемой матрицы D в процедуре;
                   D       - выходной формальный параметр, перед описанием которого обязательно необходим VAR; описан типом MAS}
        VAR I,J:INTEGER; {описаны индексные переменные}
         BEGIN  {раздел операторов}
          FOR I:=1 TO M DO
           FOR J:=1 TO N DO
              D[I,J]:=D[I,J]*K; {K - скаляр, заданный константой в основной программе}
         END;   {конец процедуры PROCS}
     FUNCTION FUNCS(D:MAS;M,N:INTEGER):REAL;
        {здесь     FUNCS   - имя процедуры;
                   M,N     - входные параметры, задающие фактическую размерность обрабатываемой матрицы D в процедуре;
                   D       - матрица, описанная типом MAS}
       VAR S:REAL; {раздел описаний S-переменная для нахождения суммы}
           I,J:INTEGER;
        BEGIN {раздел операторов}
          S:=0; {обнуление суммы}
          FOR I:=1 TO M DO
            FOR J=1 TO N DO
                S:=S+D[I,J];
          FUNCS:=S;
        END;    {конец функции FUNCS}
  BEGIN      {начало головной программы}
       {ввод и обработка элементов матрицы B(5,10)}
```

```
WRITELN('вводи по строкам по одному элементу
            массива B(5,10)');
FOR I:=1 TO 5 DO BEGIN  {цикл по строкам}
 FOR J:=1 TO 10 DO BEGIN {цикл по столбцам}
   WRITE('вводи B ',I,',',J,' = ');
   READLN(B[I,J]);
                    END;
                  END;
   S1:=FUNCS(B,5,10);{обращение к функции FUNCS с
                     матрицей B(5,10)}
   PROCS(B,5,10);{обращение к процедуре PROCS с матри-
                 цей B(5,10)}
   {ввод и обработка элементов матрицы C(12,6)}
   WRITELN('вводи по строкам по одному элементу
           массива C(12,6)');
    FOR I:=1 TO 12 DO BEGIN {цикл по строкам}
     FOR J:=1 TO 6 DO  BEGIN  {цикл по столбцам}
      WRITE('вводи C ',I,',',J,' = ');
      READLN(C[I,J]);
                        END;
                      END;
   S2:=FUNCS(C,12,6); {обращение к функции FUNCS с
                       матрицей C(12,6)}
   PROCS(C,12,6);{обращение к процедуре PROCS с матри-
                 цей C(12,6)}
     {вывод элементов матрицы B(5,10) в цикле}
   FOR I:=1 TO 5 DO BEGIN  {цикл по строкам}
     WRITELN('строка',I);
     FOR J:=1 TO 10 DO BEGIN {цикл по столбцам}
       WRITE((B[I,J] :4:0,' ');
                   END;
    WRITELN;
                 END;
     {вывод элементов матрицы C(12,6) в цикле}
   FOR I:=1 TO 12 DO BEGIN  {цикл по строкам}
    WRITELN('строка',I);
    FOR J:=1 TO 6 DO    BEGIN {цикл по столбцам}
     WRITE(C[I,J]:4:0,' ');
                       END;
    WRITELN;
```

```
                    END;
  WRITELN(‘ S1=’,S1:7:2,’ S2=’,S2:7:2);
   END.
```

8.5. Контрольные вопросы

1. Дайте определение термину «подпрограмма».
2. Назовите виды подпрограмм, которые используются в языке *Turbo Pascal* (в языке *C++*).
3. Какие параметры называются формальными?
4. Какие параметры называются фактическими?
5. Какое соответствие должно быть между формальными и фактическими параметрами?
6. Назовите способы размещения текста подпрограмм в основной (головной) программе в языке *Turbo Pascal*.
7. Чем отличается подпрограмма-функция от подпрограммы-процедуры в языке *Turbo Pascal*?
8. Дайте определение термину «рекурсия».

Глава 9.
АЛГОРИТМЫ СОРТИРОВКИ И ПОИСКА ДАННЫХ

9.1. Алгоритмы сортировки элементов одномерного массива

Алгоритм сортировки – это алгоритм, который переставляет элементы в массиве таким образом, что массив становится упорядоченным.

Сортировка бывает двух видов:

1. Внутренняя сортировка, которая обрабатывает данные, хранимые в оперативной памяти.

2. Внешняя сортировка, которая работает с данными, расположенными на дисках.

Существует много алгоритмов внутренней сортировки; приведём некоторые из них:

- сортировка выбором;
- сортировка обменом (метод пузырька);
- сортировка подсчётом;
- сортировка вставками;
- сортировка бинарными вставками;
- быстрая сортировка (метод Ч. Хоара);
- сортировка Шелла (сортировка включениями с убывающим приращением);
- сортировка слияниями;
- пирамидальная сортировка.

Ниже рассмотрены наиболее популярные сортировки: выбором и методом пузырька.

9.1.1. Сортировка выбором

Сортировка выбором или упорядочение элементов массива, расположив их по возрастанию или убыванию в том же массиве, когда отыскивается максимальный (минимальный) элемент и переносится в начало массива; затем этот метод применяется ко всем элементам, кроме первого, так как он уже находится на своем окончательном месте, и т.д.

При решении воспользуемся приёмом нахождения наименьшего (минимального), если упорядочиваем по возрастанию, либо наибольшего (максимального), если упорядочиваем по убыванию. Далее пояснения проводим для случая упорядочения по возрастанию элементов массива $X_1, X_2, X_3, ..., X_N$.

Во внутреннем цикле (*K* - переменная внутреннего цикла) находим наименьший элемент *XMIN* и его порядковый номер *IMIN* массива *X* или части массива. Перед этим циклом задаём начальное значение наименьшего, равное текущему элементу массива ($XMIN = X_I, \quad где \quad I = \overline{1,(N-1)}$), а внутри цикла находим наименьший элемент *XMIN*. После окончания этого цикла наименьший элемент записывается на место первого (1-го) элемента рассматриваемой части массива, а первый (1-ый) записывается на его место. Затем отыскивается наименьший элемент массива, начиная со второго (2-го), далее с третьего (3-го) и т.д.

Все указанные действия надо повторять во внешнем цикле по переменной *I*. Внешний цикл должен повторяться для значения *I*, которое должно изменяться от 1 до (*N*-1), где *N* – это количество элементов в массиве *X* (когда останется последний элемент, находить наименьший не имеет смысла).

Блок-схема алгоритма приведена на блок-схеме рис. 24.

```
{Программа сортировки элементов массива методом выбора на
 языке Turbo Pascal}
CONST N=100;
 VAR X:ARRAY[1..N] OF REAL;
     K1,IMIN,I,K:INTEGER;
     XMIN:REAL;
  BEGIN
    WRITELN('вводи N=100 значений через пробел');
    FOR I:=1 TO N DO READ(X[I]);
    FOR I:=1 TO (N-1) DO BEGIN
       XMIN:=X[I]; {задаем начальное значение минимального}
       IMIN:=I; {запоминаем местоположение начального значения
                 минимального}
       K1=I+1; {задаем местоположение начала подмассива, в кото-
                ром отыскиваем минимальный элемент}
       FOR K:=K1 TO N DO BEGIN
         IF X[K]<XMIN THEN BEGIN
            XMIN:=X[K]; {заменяем значение минимального}
            IMIN:=K; {запоминаем местоположение минимального}
                                END;
          X[IMIN]:=X[I]; {на место наименьшего помещаем эта-
                          лонный элемент}
          X[I]:=XMIN; {на место эталонного помещаем наименьший}
                          END;
```

```
FOR I:=1 TO N DO WRITE(X[I]:6:2); {распечатываем
                                   упорядоченные элементы массива
                                   X}
END.
```

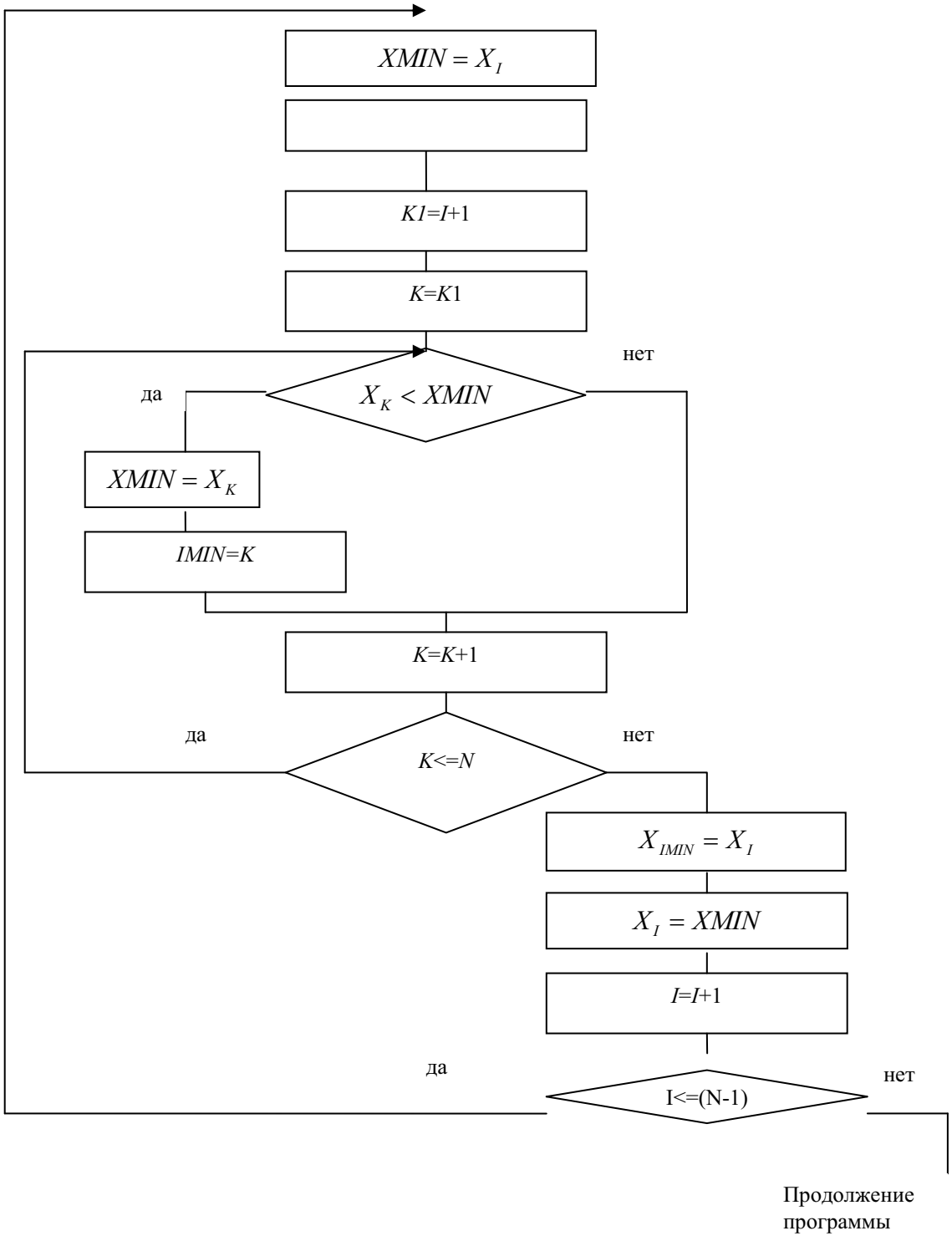


Рис. 24. Блок-схема алгоритма сортировки методом выбора

```
/*Фрагмент программы сортировки элементов массива методом
  выбора на языке C++, в котором описание массива отсутствует*/
/* в программе используются int X[],int n*/
  int imin,xmin;
  int k1,i,k;
  for (i=0;i<n-1;i++)
    {
     xmin=X[i];
     imin=i;
     k1=i+1;
     for (k=k1;k<n;k++)
     {
     if (X[k]<xmin)
       {
        xmin=X[k];
        imin=k;
       }
     }
     X[imin]=X[i];
     X[i]=xmin;
    }
```

9.1.2. Сортировка обменом (метод пузырька)

В методе сортировки обменом (методе пузырька) последовательно сравниваются пары соседних элементов X_K *и* $\mathrm{X_{K+1}}$ $(K=\overline{1,(N-1)})$, где N - количество просматриваемых элементов, и, если $X_K > \mathrm{X_{K+1}}$, то они переставляются; тем самым наибольший элемент окажется на своем месте в конце массива (сортировка по возрастанию); затем этот метод применяется ко всем элементам, кроме последнего, и т. д.

Таким образом, чтобы упорядочить все элементы массива из N элементов, необходимо организовать внешний цикл по I, которое должно изменяться от N до 2. Блок-схема алгоритма методом пузырька приведена на рис. 25.

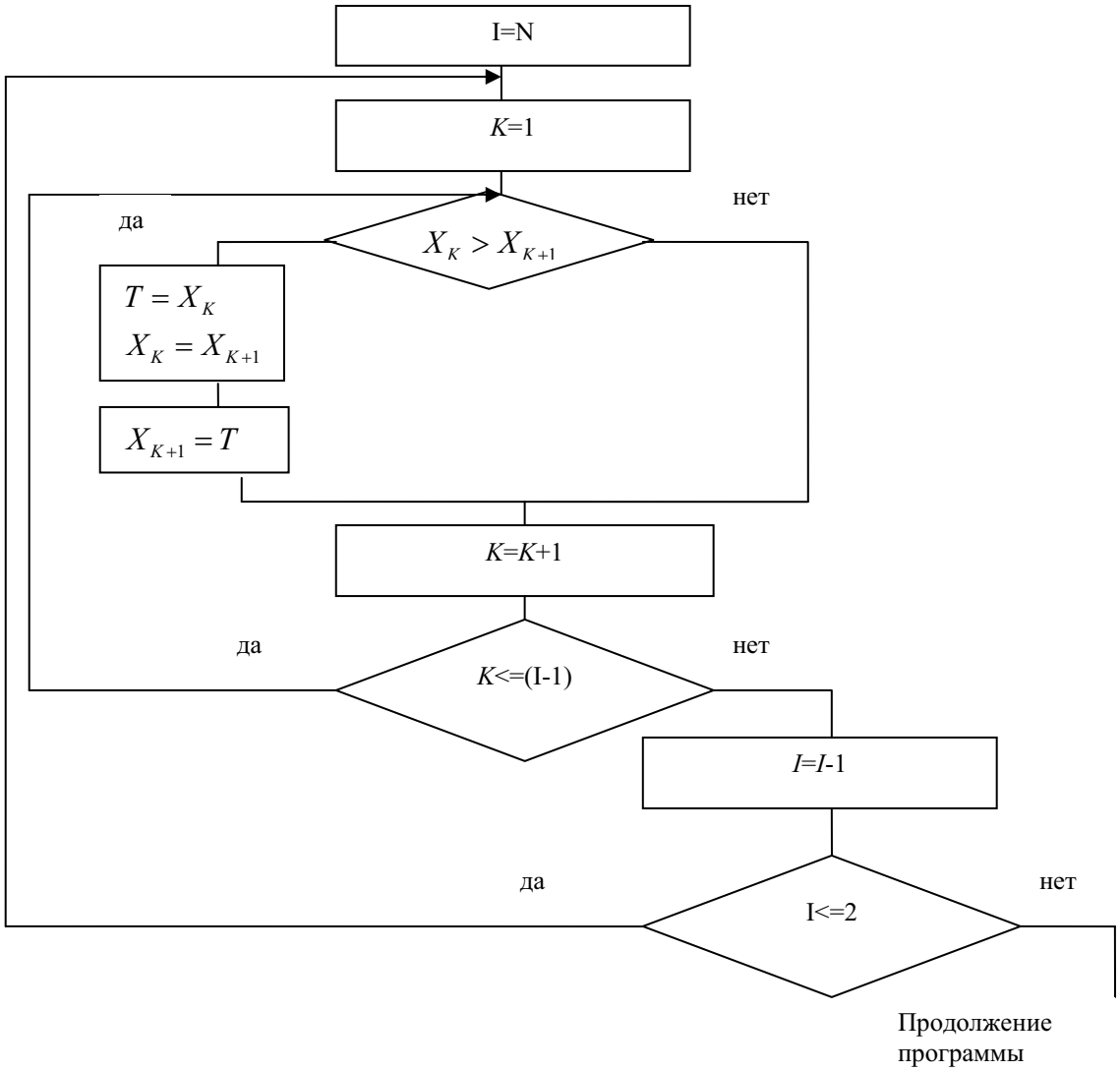


Рис. 25. Блок-схема алгоритма обменом (метод пузырька)

{Программа сортировки элементов массива методом пузырька на языке *Turbo Pascal*}

```
CONST N=100;
 VAR X:ARRAY[1..N] OF REAL;
     I,K:INTEGER;
     T:REAL;  {вспомогательная переменная для перестановки элементов}
  BEGIN
   WRITELN('вводи n=100 значений через пробел');
   FOR I:=1 TO N DO READ(X[I]);
   FOR I:=N DOWNTO 2 DO BEGIN {внешний цикл I -количество просмотров}
    FOR K:=1 TO (I-1) DO BEGIN
     IF X[K]<X[K+1] THEN BEGIN
```

```
          {перестановка элементов местами}
          T:=X[K];
          X[K]:=X[K+1];
          X[K+1]:=T;
                                  END;  {окончание цикла по K}
                                END;    {окончание цикла по I}
   {вывод на экран упорядоченных элементов массива X}
    FOR I:=1 TO N DO WRITE(X[I]:6:2);
  END.
```

```
      /*Фрагмент программы сортировки элементов массива методом
        пузырька на языке C++*/
/*в программе используются int X[],int n*/
for (i=n;i>=2;i--)
  {
  for (k=1;k<=(i-1);k++)
    {
     if (X[k]<X[k+1])
       /*перестановка элементов местами*/
       {  t:=X[k];
          X[k]:=X[k+1];
          X[k+1]:=t;
       }
    }
  }
```

9.2. Поиск данных

Поиск элемента с заданным значением z в целочисленном массиве $Q(N)$ сводится к определению такого номера i элемента массива, для которого $Q(i) = z$, или к определению, что такого номера нет. Способов решения этой задачи два:

1. Способ последовательного поиска.
2. Бинарный (дихотомический) способ.

Первый способ – это последовательный поиск. Просмотрим все элементы массива в цикле до первого совпадения элемента $Q(i)$ с z.Так как после нахождения искомого значения просматривать массив не надо, то в программе будет использоваться цикл с предусловием *WHILE*. В этом случае цикл завершится, если будет достигнут конец массива или найдено искомое значение. Если параметр цикла будет больше длины

массива, то это означает, что искомого значения в массиве нет. В противном случае элемент со значением *z* находится на месте *i*.

{Фрагмент программы последовательного поиска элемента *z* в массиве *Q(N)* на языке *Turbo Pascal*}

```
I:=1;
WHILE (I<=N) AND (Q[I]<>Z) DO I:=I+1;
  IF I>N THEN WRITELN('элемент не найден')
         ELSE  WRITELN('элемент  со  значением  ',Z,
                       'стоит на месте ',I);
```

/*Функция поиска элемента *z* в массиве *Q(N)* на языке *C++*. Функция в качестве формальных параметров использует массив, количество элементов, значение искомого элемента и возвращает, либо индекс найденного элемент, либо (-1) при неудачном поиске*/

```
int poisk(int Q[],int n,int z)
{
  for (i=0;i<n;i++)
     if (Q[i]==z) return i;
  return -i;
 }
```

Второй способ, в котором используется свойство упорядоченности массива, - это *бинарный* (*дихотомический*) способ, то есть деление пополам. В алгоритме быстрого поиска выделяется область элементов, в которой искомый элемент мог бы находиться. Каждое сравнение выполняется так, чтобы область элементов уменьшалась вдвое. Поэтому на каждом шаге цикла надо сравнивать *z* с элементом *Q(C)*, который должен располагаться в середине выделенной области. При этом возможны два случая:

1. *Q(C)<z*. В этом случае не учитываются элементы, которые расположены от начала и до номера *C* включительно.

2. *Q(C)>=z*. В этом случае не учитываются элементы, которые расположены от номера (*C*+1) и до конца области.

{Фрагмент программы бинарного поиска элемента *z* в упорядоченном массиве *Q(N)* на языке *Turbo Pascal.* В примере границы интервала области поиска элемента обозначены как [*C1,C2*]}

```
  C1:=1; C2:=N;
  WHILE (C1<C2) DO BEGIN
```

```
    C:=TRUNC((C2+C1)/2);  {TRUNC – встроенная функция вычисления
                           целой части выражения}
    IF Q[C]<Z THEN C1:=C+1
              ELSE C2:=C;
                   END;
  IF Q[C1]=Z THEN I:=C1
             ELSE I:=0;
```

/*Функция бинарного поиска элемента *z* в упорядоченном массиве *Q(N)* на языке *C++*. Функция в качестве параметров использует массив, значение `c1`, значение `c2`, значение искомого элемента `z` и возвращает, либо индекс найденного элемент, либо (-1) при неудачном поиске */

```
int poisk_bin(int Q[],int c1,int c2,int z)
  { int c;
    While (c1<c2)
     { c=(c1+c2)/2;
       if (Q[c]==z) return c;
                    else if ((Q[c]<z) c1=c+1;
                                      else c2=c;
     }
   return -i;
  }
```

9.3. Контрольные вопросы

1. Дайте определение термину «алгоритм сортировки».
2. Назовите виды сортировок.
3. Назовите алгоритмы сортировок.
4. Опишите алгоритм сортировки выбором.
5. Опишите алгоритм сортировки обменом (метод пузырька).
6. Назовите способы поиска элемента с заданным значением в целочисленном массиве.
7. Опишите алгоритм последовательного поиска элемента с заданным значением в целочисленном массиве.
8. Опишите алгоритм бинарного поиска элемента с заданным значением в целочисленном массиве.

Глава 10.

МНОЖЕСТВА

Множество - это структурированный тип данных, который представляет набор взаимосвязанных по какому-либо признаку или группе признаков объектов, которые рассматриваются как единое целое. Каждый объект в множестве называется *элементом множества*. Все элементы множества должны принадлежать одному из скалярных типов, кроме вещественного типа. Выбранный скалярный тип называется *базовым типом* множества. Базовый тип задаётся диапазоном или перечислением.

Множественные типы данных используются на языке *Turbo Pascal* с целью сокращения, наглядности и эффективности программы за счёт уменьшения числа различных проверок.

10.1. Описание множеств

Область значений типа множество – это набор различных подмножеств, которые составлены из элементов базового типа. Если базовый тип принимает *N* значений, то тип множество для него будет иметь 2^N вариантов различных значений. В выражениях на языке *Turbo Pascal* значения элементов множества указываются в квадратных скобках, например: [1,2,3,4], [*'a','b','c'*], [*'a'..'z'*].

Если множество не имеет элементов, то оно называется *пустым* и обозначается как [].

Для описания множественного типа используется словосочетание `SET OF` (множество из …).

Способы описания множественного типа

Множественный тип можно задать двумя способами:

- первый способ – это способ с использованием раздела `TYPE`:

```
TYPE
     <имя типа>=SET OF <элемент1,…,элементN>;
 VAR
     <идентификатор,…>:<имя типа>;
```

- второй способ – это способ без использования раздела TYPE сразу в разделе VAR:

```
VAR
    <идентификатор,…>:SET OF <элемент1,…,элементN>;
```

Формула 29 Пример 29. Описать переменные множественных типов данных.

```
TYPE ALPHAWIT=SET OF ['A'..'Z','a'..'z'];
     NOMER=SET OF 1..31;
VAR AL:ALPHAWIT;
     NO:NOMER;
     BUKWA:SET OF ['a','e','i','j'];
```

В приведённом примере 29 переменная AL может принимать значение в диапазоне от прописной *'A'* до *'Z'*, от строчной *'a'* до *'z'*; переменная NO может принимать значение в диапазоне чисел от 1 до 31; переменная BUKWA может принимать любое значение из *'a'*, *'e'*, *'i'*, *'j'*.

10.2. Операции над множествами

В *Turbo Pascal* имеются следующие операции над множествами:

- проверка принадлежности – это операция IN;
- сравнения множеств – это операции отношения: = (равно), <> (не равно), >= (больше или равно), <= (меньше или равно);
- объединение множеств – это операция +;
- пересечение множеств– это операция *;
- разность множеств– это операция -.

Результатом выражений с применением перечисленных операций является значение TRUE (ИСТИНА) или FALSE (ЛОЖЬ), в зависимости от того, истинно выражение или ложно.

Операция IN. Данная операция используется для проверки принадлежности какого-либо значения указанному множеству. Операция IN применяется в условном операторе IF.

При использовании операции IN проверяемое на принадлежность значение и множество в квадратных скобках не обязательно предварительно описывать в разделе описаний (см. табл. 17).

Таблица 17

Использование операции IN

Значение переменной *a*	Условный оператор *IF*	Результат
2	IF a IN [1,2,3,4,6,7] THEN …	TRUE
´v´	IF a IN [´a´…´n´] THEN …	FALSE
X1	IF a IN [X1,X2,X3] THEN …	TRUE

Операция IN позволяет просто и эффективно производить сложные проверки условий в операторе IF.

```
{Фрагмент программы с использованием операции IN}
VAR ALPHAWIT=SET OF [´A´..´Z´,´a´..´z´];
    STR1:CHAR;
BEGIN
READLN(STR1);
IF STR1 IN ALPHAWIT THEN …
```

В приведённом фрагменте программы условный оператор

```
IF STR1 IN ALPHAWIT THEN …
```

будет лучше, чем операции сравнения вида:

```
IF ((CH>=´A´) AND (CH<=´Z´)) OR
    ((CH>=´a´) AND (CH<=´z´)) THEN …
```

Операция «равно» (=). Два множества *A* и *B* считаются равными, если они состоят из одних и тех же элементов. Порядок следования элементов в сравниваемых множествах может быть любой (см. табл. 18).

Таблица 18

Использование операции «равно»

Значение *A*	Значение *B*	Выражение в операторе IF	Результат
[5,6,8,9]	[9,6,8,5]	IF A=B THEN …	TRUE
[´d´,´f´,´g´]	[´g´,´f´]	IF A=B THEN …	FALSE
[´z´..´a´]	[´a´..´z ´]	IF A=B THEN …	TRUE

Операция «не равно» (<>). Два множества *A* и *B* считаются не равными, если они различаются по значению элементов хотя бы одним (см. табл. 19).

Таблица 19

Использование операции «не равно»

Значение *A*	Значение *B*	Выражение в операторе IF	Результат
[5,7,9]	[9,3,7,5]	IF A<>B THEN …	TRUE
[´b´..´z´]	[´d´..´z´]	IF A<>B THEN …	TRUE
[´c´..´z´]	[´z´..´c´]	IF A<>B THEN …	FALSE

Операция «больше или равно» (>=). Данная операция используется для определения принадлежности множеств. Результат операции A>=B равен TRUE, если все элементы множества *B* содержатся во множестве *A*. В противном случае результат равен FALSE (см. табл. 20).

Таблица 20

Использование операции «больше или равно»

Значение *A*	Значение *B*	Выражение в операторе IF	Результат
[3,5,7,9]	[9,7,5]	IF A>=B THEN …	TRUE
[´d´,´e´,´f´]	[´f´,´d´]	IF A>=B THEN …	TRUE
[´a´..´z´]	[´o´..´t´]	IF A>=B THEN …	TRUE

Операция «меньше или равно» (<=). Данная операция также используется для определения принадлежности множеств. Результат операции A<=B равен TRUE, если все элементы множества *A* содержатся в множестве *B*. В противном случае результат равен FALSE (см. табл. 21).

Таблица 21

Использование операции «меньше или равно»

Значение *A*	Значение *B*	Выражение в операторе *IF*	Результат
[5,7,9]	[9,5,7,3]	IF A<=B THEN …	TRUE
[´z´,´y´]	[´y´,´x´,´z´]	IF A<=B THEN …	TRUE
[´c´..´g´]	[´z´..´c´]	IF A<=B THEN …	TRUE

Объединение множеств (операция **+)**. Объединением двух множеств является третье множество, которое будет содержать элементы обоих множеств (см. табл. 22).

Таблица 22

Использование операции «объединение множеств»

Значение *A*	Значение *B*	Выражение	Результат
[4,5,7]	[6,7,8]	A+B	[4,5,6,7,8]
[´D´,´e´]	[´f´]	A+B	[´D´,´e´,´f´]
[´o´..´s´]	[´a´..´t´]	A+B	[´a´..´t´]

Пересечение множеств (операция ***)**. Пересечением двух множеств является третье множество, которое будет содержать элементы, входящие одновременно в оба множества (см. табл. 23).

Таблица 23

Использование операции «Пересечение множеств»

Значение *A*	Значение *B*	Выражение	Результат
[4,5,9]	[1,5,9,7]	A*B	[5,9]
[´z´,´t´]	[´t´,´o´]	A*B	[´t´]
[´d´..´h´]	[´b´..´t´]	A*B	[´d´..´h´]

Разность множеств (операция **-)**. Разностью двух множеств является третье множество, которое будет содержать элементы первого множества, не входящие во второе множество (см. табл. 24).

Таблица 24

Использование операции «Разность множеств»

Значение *A*	Значение *B*	Выражение	Результат
[9,8,7,6]	[6,8,7]	A-B	[9]
[´c´,´d´]	[´d´]	A-B	[´c´]
[´z´..´a´]	[´e´..´z´]	A-B	[´a´..´d´]

10.3. Использование множеств

Использование множественных типов данных в программе даёт несколько преимуществ:

- упрощаются сложные операторы `IF`;
- увеличивается наглядность программы;
- экономится оперативная память, время компиляции и выполнения.

Формула 30 Пример_30. Подсчитать количество цифр в ведённой символьной строке и напечатать все символы, кроме пробелов, знаков операций и знаков препинания.

{Программа подсчёта количества цифр в ведённой символьной строке и печати всех символов, кроме пробелов, знаков операций и знаков препинания на языке *Turbo Pascal*}

```
VAR STR1:STRING[50];
    CH,I,N:INTEGER;
  BEGIN
   WRITELN('ввод строки'); READLN(STR1);
   N:=LEHGTH(STR1);  {определена фактическая длина введённой
                      строки}
   FOR I:=1 TO N DO BEGIN {анализ текущего символа}
    IF NOT (STR1[I] IN [' ','.',';',':',',','+',
            '-','*','/']) THEN BEGIN
            IF STR1(I) IN ['0'..'9'] THEN CH:=CH+1;
            WRITE(STR1[I]);
                              END;
                    END;
            WRITELN;
            WRITELN('число цифр в строке=',CH);
    END.
```

П р о т о к о л р а б о т ы п р о г р а м м ы:

```
ввод строки12345,.;: 6+7-8*9ABCD
123456789ABCD
число цифр в строке=9
```

В приведённой программе запись в явном виде полного набора проверок по всем символам заняло бы много места и увеличило бы время выполнения программы, так как вместо одной операции определения принадлежности элемента множеству, которая реализована одной логической операцией, пришлось бы выполнить много операций отношения.

10.4. Контрольные вопросы

1. Дайте определение термину «множество».
2. В каких случаях рекомендуется использовать множественные типы данных?
3. Назовите операции над данными множественного типа.
4. Для чего используется операция «объединение множеств»?
5. Для чего используется операция «пересечение множеств»?
6. Для чего используется операция «разность множеств»?

Глава 11.

ЗАПИСИ

Во многих экономических и информационных задачах обрабатываются ведомости, документы, списки. При этом появляется необходимость объединять данные различного типа в одну группу. Для работы с группой данных введено понятие *записи*.

Запись – это иерархическая структура ограниченного числа данных различного типа.

Понятие записи рассмотрим на примере «Ведомости оценок студентов» из табл. 25.

Таблица 25

Ведомость оценок студентов ИМОЯК

Номер п/п	Фамилия, имя, отчество (Ф.И.О.)	№ группы	Оценки		
			Математика	Информатика	Физика
1.	Ле Ши Хоанг	158Б80	5	5	5
2.	То Тхи Уиэн	152А80	5	5	4
3.	Карим Пешунг	152Б80	4	4	4
4.	Ван Чао	151Б80	5	5	5

Данные табл.25 будут соответствовать следующей иерархической структуре рис. 26.

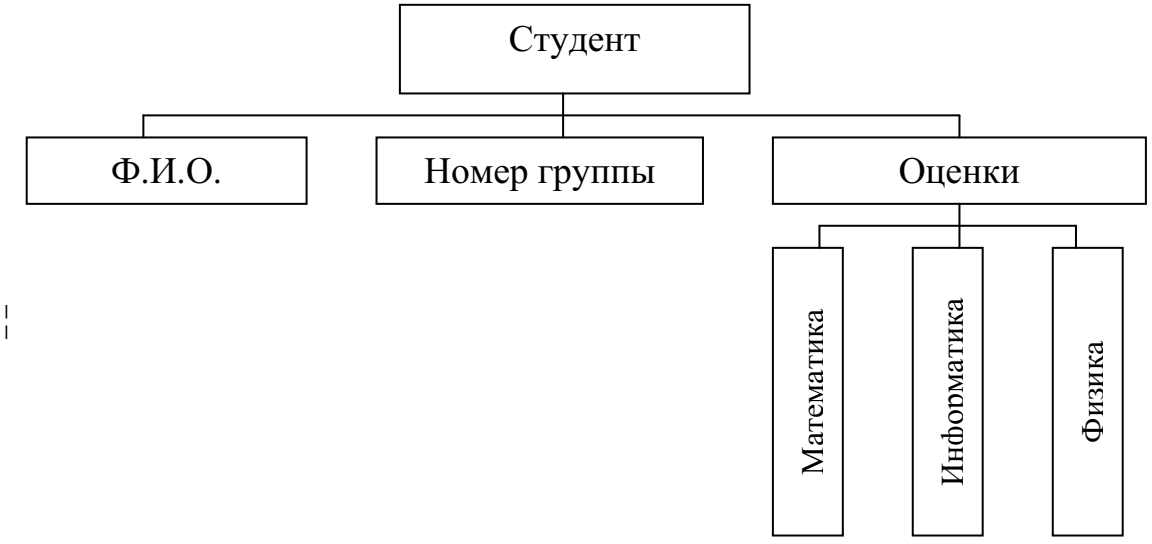


Рис. 26. Иерархическая структура данных по табл. 25

Каждая строка табл.25 состоит из отдельных элементов – это поля, которые имеют различный тип:

а) фамилия, имя, отчество (Ф.И.О.) – это строковое данное;

б) номер группы – это строковое данное;

в) оценки по математике, по информатике, по физике – это целые числа.

Обратите внимание, что номер записи по порядку (номер п/п) из таблицы отсутствует в иерархической структуре.

Данные, приведённые в табл.25, объединим в одну группу, которую будем считать *записью*[16].

Запись в целом и отдельные элементы записи (поля) обозначаются именами. Для нашего примера введём следующие обозначения полей записи:

`B`	- имя всей записи;
`FIO`	- фамилия, имя, отчество;
`NGR`	- номер группы;
`MAT`	- оценка по математике;
`INF`	- оценка по информатике;
`FYZ`	- оценка по физике.

11.1. Описание записей на языке *Turbo Pascal*

Обычно структура записи задаётся в разделе типов `TYPE`. В этом случае описание записи имеет следующий вид:

```
TYPE <имя-типа-записи>=RECORD
       <имя-поля1>:<тип-поля1>;
       <имя-поля2>:<тип-поля2>;
                  . . .
       <имя-поляN>:<тип-поляN>;
                          END;
```

Чтобы работать с отдельными элементами записи, необходимо объявление записи сделать в разделе переменных `VAR`. В этом случае объявление будет иметь вид:

```
VAR <имя-записи>:<имя-типа-записи>;
      <массив-записей>:ARRAY[1..N] OF <имя-типа-записи>;
```

[16] На языке *C++* вместо термина «запись» используется термин «структура».

{Описание записи и массива записей на языке *Turbo Pascal* по данным табл. 25.}

```
TYPE STUDENT=RECORD
     FIO:STRING[18];
     NGR:STRING[4];
     MAT,INF,FYZ:INTEGER;
              END; {окончание описания записи}
  VAR B:STUDENT;
      BMAS:ARRAY[1..50] OF STUDENT; {массив записей}
```

Обращение к полю (элементу) записи в программе выполняется с помощью квалифицированного имени, которое содержит имя записи, точку, имя поля, например, для полей записи B:

`B.FIO` - обращение к переменной, содержащей фамилию, имя, отчество;

`B.NGR` - обращение к переменной, содержащей номер группы;

`B.MAT` - обращение к переменной, содержащей оценку по математике;

`B.INF` - обращение к переменной, содержащей оценку по информатике;

`B.FYZ` - обращение к переменной, содержащей оценку по физике.

Поля массива записей `BMAS` необходимо квалифицировать, используя индекс, следующим образом:

`BMAS[I].FIO` - обращение к *I*-ой переменной массива `BMAS`, содержащей фамилию, имя, отчество;

`BMAS[I].NGR` - обращение к *I*-ой переменной массива `BMAS`, содержащей номер группы;

`BMAS[I].MAT` - обращение к *I*-ой переменной массива `BMAS`, содержащей оценку по математике;

`BMAS[I].INF` - обращение к *I*-ой переменной массива `BMAS`, содержащей оценку по информатике;

`BMAS[I].FYZ` - обращение к *I*-ой переменной массива `BMAS`, содержащей оценку по физике.

Оператор присоединения *`with`*

Чтобы упростить доступ к полям записи, используется оператор присоединения `WITH`. Формат оператора `WITH`:

`WITH` <переменная> `DO` <оператор>

Здесь `WITH, DO` - ключевые слова (с, делать);

<переменная> - это имя переменной типа запись, за которым может следовать список вложенных полей;

<оператор> - любой оператор языка *Turbo Pascal*.

Поля записи используются в программе в том же самом смысле, как и обычная переменная. Таким образом, поле записи можно указывать как в левой части оператора присваивания, так и в выражениях правой части. Над полями записи можно выполнять действия, допустимые для данных его типа.

```
{Фрагмент программы определения суммы трёх оценок студента
 на языке Turbo Pascal}
S:=B.MAT+B.INF+B.FYZ;
{либо}
WITH B DO
S:=MAT+INF+FYZ;
                {для элементов массива BMAS}
S:=BMAS[I].MAT+BMAS[I].INF+BMAS[I].FYZ;
                              {либо}
WITH BMAS[I] DO
S:=MAT+INF+FYZ;
```

Обращение к записи в целом, а не только к её элементам, допускается лишь в операторе присваивания (:=). Слева и справа от знака присваивания при этом должны использоваться имена записей одинакового типа, то есть имеющие одно и то же описание из раздела типов `TYPE`.

11.2. Описание структурной переменной на языке *C++*

На языке *C++* описание структурной переменной состоит из двух этапов:

1. Задание шаблона структуры.
2. Описание структурной переменной.

Формат задания шаблона:

```
struct <имя_структуры>{
 <тип_поля1> <имя_поля1>;
 <тип_поля2> <имя_поля2>; …
 <тип_поляN> <имя_поляN>;  }
```

Формат описания структурной переменной:
`struct` <имя_структуры> <имя_структурной_переменной>;

Обращение к полю структурной переменной в программе выполняется с помощью квалифицированного имени, которое содержит <*имя_структурной_переменной*>, точку, <*имя поля*>.

```
/*Задание шаблона и описание структурной переменной*/
struct STUDENT{
char[18] fio;
char[6] ngr;
int mat,inf,fyz;
} b;
/*Обращение к полям структурной переменной b*/
b.fio=”Ле Ши Хоанг”;
b.ngr=”158Б80”;
b.mat =5;
b.inf =5;
b.fyz =5;
```

11.3. Контрольные вопросы

1. Дайте определению термину «запись».
2. Как описываются поля записи в программе?
3. Как необходимо обращаться к полям записи в программе?
4. Для чего используется оператор присоединения `with`?

Глава 12.

ФАЙЛЫ

В практике программирования часто встречаются задачи, решение которых необходимо выполнять постоянно с большим количеством данных, мало изменяемых во времени. Ввод данных с экрана заново при каждом счёте для этих задач очень трудоёмок, а часто просто невозможен из-за временны́х ограничений.

Для решения этой проблемы в языках программирования была реализована концепция файлов, что позволяет после набора информации на клавиатуре запоминать её на магнитном носителе и обращаться к ней непосредственно из обрабатывающих программ при каждом счёте.

Файл - это абстрактная модель физического набора данных, который состоит из последовательности записей.

В отличие от массива длина файла, то есть количество записей (структур), не задаётся, место записи не определяется индексом и каждая запись становится доступной только после перебора всех предыдущих элементов.

Физические наборы данных размещаются на внешних запоминающих устройствах компьютера: на жёстком магнитном диске, на гибком магнитном диске (дискете), на флэш-устройстве.

Файлы в программе бывают двух типов:

- стандартные;
- пользовательские (файлы пользователя).

Стандартные файлы определены разработчиками систем *Turbo Pascal* и *C++,* и программисты могут их только использовать.

Файлы пользователя задаются программистом по необходимости в программах.

В программе файлы принято различать:

- *входные*, то есть уже существующие (старые);
- *выходные*, то есть вновь создаваемые (новые).

На языке *Turbo Pascal* в зависимости от формы представления информации в файле на диске (во *внутреннем* представлении или во *внешнем* представлении данных) при описании файлов в программе их различают по следующим трём видам:

- текстовые файлы;
- *типизированные* файлы;
- *нетипизированные* файлы.

При обработке записей входного файла необходимо использовать цикл. Для этих целей используется оператор WHILE. Условием продолжения цикла должна стать проверка, которая выясняет, не достигнут ли конец во входной последовательности данных.

Когда данные поступают из внешнего файла, его окончание обнаруживается при помощи стандартной функции EOF (End Of File – конец файла).

Обращение к стандартной функции имеет вид:

EOF(<имя_файла>) ,

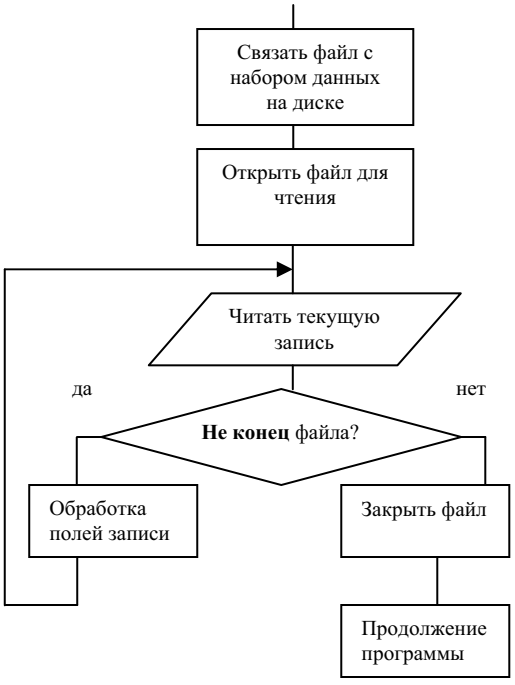
которое возвращает одно из двух булевских значений:

- TRUE – достигнут конец указанного файла;
- FALSE - не был достигнут конец указанного файла.

Чтобы цикл WHILE выполнялся по булевскому значению TRUE (ИСТИНА), в программе запись должна быть следующей:

WHILE **NOT EOF(**<*имя_файла*>**)** DO

Оператор с предупреждающим чтением WHILE записывается по блок-схеме, приведённой на рис. 27.




```
ASSIGN(F,NAME);

RESET(F);


READLN(F,STR.X1,STR.XN);

WHILE NOT EOF(F) DO BEGIN

        {обработка полей записи}
READLN(F,STR.X1,STR.XN);
                        END;
CLOSE(F);
        {продолжение программы}
```

Рис. 27. Блок-схема и фрагмент программы с использованием оператора с предупреждающим чтением WHILE

12.1. Текстовые файлы на языке *Turbo Pascal*

Текстовый файл - это файл во внешнем представлении; данные в нём хранятся как символы. Символы при считывании файла в оперативную память преобразуются в тот тип данных, который был объявлен в программе. Компонентами текстового файла являются строки переменной длины, каждая из которых завершается маркером конца строки.

При считывании текстового файла в оперативную память компьютера символы файла преобразуются в тот тип данных, который объявлен в программе; *при записи* в текстовый файл происходит обратное преобразование из внутреннего представления в символьный тип.

Если программа взаимодействует с набором данных, данные которого хранятся во внешнем представлении, то файл должен быть описан в программе явно в разделе переменных VAR:

VAR <*имя-файла*>:TEXT;

где TEXT - ключевое слово файлового типа при описании файла, поля записей которого хранятся во внешнем представлении, то есть в виде символов (при вызове на экран читаются).

```
{Описание файла, который хранит информацию об аттестации
 студентов (см. табл. 25). Информация об аттестации занесена в
 файл на диске набором символов с клавиатуры}
TYPE STUDENT=RECORD
     FIO:STRING[18];
     NGR:STRING[6];
     MAT,INF,FYZ:INTEGER;
              END;
  VAR B:STUDENT;
      BTEXT:TEXT; {описан текстовый файл, данные которого хра-
                   нятся в символьном виде. Обратите внимание, что вид
                   структуры не влияет на описание текстового файла}
```

Стандартные процедуры для работы с файлами во внешнем представлении

Стандартные процедуры для работы с файлами во внешнем представлении приведены в табл. 26. При записи стандартных процедур используются следующие обозначения:

F - файловая переменная, имеющая описатель в программе типа TEXT;

NAME - строковая переменная (типа STRING) вида: 'диск:\имя_катологa\имя-подкаталога\имя_файла';

STR1 - имя-записи;

STR.X1,STR.X2,...,STR.XN - квалифицированные поля записи *STR*.

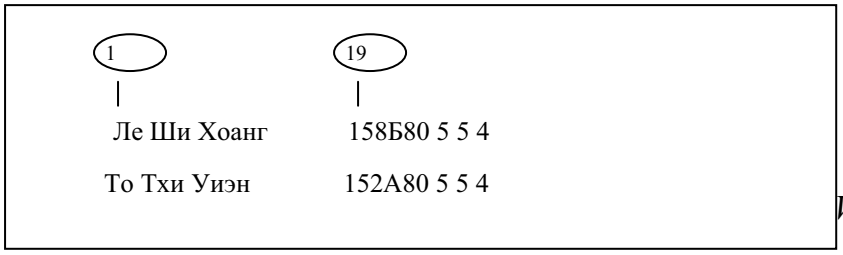
Таблица 26

Стандартные процедуры для работы с файлами во внешнем представлении

`ASSIGN(F,NAME);`	- связывает имя файловой переменной *F* программы с именем физического набора данных *NAME* на диске. Процедура ASSIGN используется *до начала* работы с файлами. После выполнения процедуры ASSIGN все действия над файловой переменной будут эквивалентны действиям над набором данных, определяемым спецификацией *NAME*;
`CLOSE(F);`	- закрывает файл *F*;
`RESET(F);`	- открывает файл *F* для чтения;
`READLN(F,STR1.X1,STR1.X2, …,STR.XN);`	- считывает в переменные *STR1.X1,..., STR1.XN* данные полей одной строки записи текстового файла *F*, начиная чтение с элемента, на который указывает текущий указатель;
`REWRITE(F);`	- открывает файл F для записи;
`WRITELN(F,STR1.X1,STR1.X2, ,STR1.XN);`	- записывает одну компоненту во внешнем представлении из переменных *STR.X1, ..., STR.XN* в файл *F*, начиная с той позиции, на которую установлен указатель;
`APPEND(F);`	- открывает текстовый файл и устанавливает указатель на маркер конца файла. Используется *только для добавления* новых компонентов в конец текстовых файлов.

```
      {Ввод данных из текстового файла (структура записей приведена
       в табл. 25)}
TYPE STUDENT=RECORD
     FIO:STRING[18];
     NGR:STRING[6];
     MAT,INF,FYZ:INTEGER;
                 END;
  VAR B:STUDENT;
      BTEXT:TEXT;{BTEXT - файл во внешнем представлении}
      BMAS:ARRAY[1..50] OF STUDENT;{массив записей}
    BEGIN
      ASSIGN(BTEXT,'DAV.TXT');{связать имя файловой переменной BTEXT программы с именем физического набора данных DAV.TXT в текущем подкаталоге}
      RESET(BTEXT);{открыть файл BTEXT для чтения, как существующий}
      READLN(BTEXT,B.FIO,B.NGR,B.MAT,B.INF,B.FYZ);{читать в переменные B.FIO, B.NGR, B.MAT, B.INF,
                B.FYZ данные из первой записи файла BTEXT}
      WHILE NOT EOF(BTEXT) DO BEGIN {пока не конец файла выполнить }
          {операторы обработки полей записи}
        READLN(BTEXT,B.FIO,B.NGR,B.MAT,B.INF,B.FYZ);
                {читать в переменные B.FIO, B.NGR, B.MAT, B.INF,
                B.FYZ данные из текущей записи файла BTEXT}
                                  END;{конец работы оператора цикла WHILE}
       CLOSE(BTEXT);{закрыть файл BTEXT}
      END.
```

Файл *DAV.TXT* должен содержать данные, набранные согласно описанию полей записи (рис. 28).



V.TXT

Формула 31 Пример 31. Ввести значения элементов двух матриц *Q*(2,3) и *R*(2,3) из текстового файла *NEW.TXT*, подготовленного в формате, который представлен на рис 29.

NEW.TXT

```
матрица Q(2,3)
1.1     1.2     1.3
2.1     2.2     2.3
матрица R(2,3)
3.1     3.2     -3.3
-4.1    4.2     -4.3
```

Рис. 29. Формат данных файла *NEW.TXT*

```
    {Ввод значений элементов двух матриц Q(2,3) и R(2,3)}
VAR Q,R:ARRAY[1..2,1..3] OF REAL;
    J,I:INTEGER;{индексные переменные}
    BTEXT:TEXT;{файл во внешнем представлении}
  BEGIN
    ASSIGN(BTEXT,'NEW.TXT');{связали имя файловой переменной BTEXT программы с именем физического набора данных NEW.TXT текущего подкаталога}
    RESET(BTEXT);{файл BTEXT открыт для чтения, как существующий}
    READLN(BTEXT);{пропуск текущей строки с данными «матрица Q(2,3)»}
    FOR I:=1 TO 2 DO BEGIN {цикл по строкам}
       FOR J:=1 TO 3 DO READ(BTEXT,Q[I,J]);{цикл по столбцам}
       READLN(BTEXT);{переход на следующую строку}
                     END;
    READLN(BTEXT);{пропуск текущей строки с данными «матрица R(2,3)»}
    FOR I:=1 TO 2 DO BEGIN {цикл по строкам}
       FOR J:=1 TO 3 DO READ(BTEXT,R[I,J]);{цикл по столбцам}
       READLN(BTEXT); {переход на следующую строку}
                     END;
    CLOSE(BTEXT); {закрыть файл BTEXT}
  END.
```

12.2. Типизированные файлы на языке *Turbo Pascal*

Записи типизированных файлов имеют жёсткую структуру либо типа RECORD, либо любой простой тип. Длина записей в типизированном файле фиксирована и определяется суммой длин полей во внутреннем представлении согласно типу переменных.

Типизированные файлы должны быть описаны в программе явно в разделе переменных VAR следующим образом:

VAR <*имя-файла*>:FILE OF <*тип-компонента-файла*>;

где FILE - ключевое слово файлового типа при описании файла, компоненты которого хранятся либо будут храниться во внутреннем представлении (при вызове на экран монитора записи такого файла нельзя прочитать);

<*тип-компонента-файла*> - может быть либо любым простым типом, либо типа <*имя-типа-записи*>.

```
      {Описание типизированных файлов}
TYPE STUDENT=RECORD
      FIO:STRING[18]; {занимает 18 байт}
      NGR:STRING[6]; { занимает 6 байт}
      MAT,INF,FYZ:INTEGER; {2 байта * 3=6 байт}
                END; {длина записи 18+6+6=30 байт}
  VAR B:STUDENT;
       BFILE1:FILE OF STUDENT; {типизированный файл, то есть
                файл во внутреннем представлении, состоящий из
                записей типа STUDENT}
       BFILE2:FILE OF INTEGER; {типизированный файл из дан-
                                ных целого типа}
       BFILE3:FILE OF REAL; {типизированный файл из данных
                                вещественного типа}
```

Стандартные процедуры и функции для работы с типизированными файлами

Стандартные процедуры приведены в табл. 27, стандартные функции приведены в табл. 28.

При записи стандартных процедур и функций используются следующие обозначения:

F - файловая переменная, имеющая описатель в программе типа `FILE OF ;`

NAME - строковая переменная (типа `STRING`) вида: 'диск:\имя_катологa\имя-подкаталога\имя_файла';

STR1 - имя-записи;

N - целое число.

Таблица 27

Процедуры для работы с типизированными файлами

`ASSIGN(F,NAME);`	- связывает имя файловой переменной *F* программы с именем физического набора данных *NAME* на диске. Процедура `ASSIGN` используется *до начала* работы с файлами. После выполнения процедуры `ASSIGN` все действия над файловой переменной будут эквивалентны действиям над набором данных, определяемым спецификацией *NAME*;
`CLOSE(F);`	- закрывает файл *F*;
`READ(F,STR1);`	- считывает в переменные *STR1* одну компоненту файла *F* во внутреннем представлении, начиная чтение с элемента, на который указывает текущий указатель;
`RESET(F);`	- открывает файл *F* для чтения;
`REWRITE(F);`	- открывает файл F для записи;
`SEEK(F,N);`	- осуществляет перемещение указателя к записи с номером *N*, где *N* – целая положительная константа, соответствующая порядковому номеру записи в файле, причём, первая запись файла имеет номер *N*=0, второй N=1 и т.д.
`WRITE(F,STR1);`	- записывает одну компоненту во внутреннем представлении из переменной *STR1* в файл *F*, начиная с той позиции, на которую установлен указатель.

Таблица 28

Функции для работы с типизированными файлами

`FILEPOS(F);`	- определяет текущее положение указателя файла;
`FILESIZE(F);`	- определяет размер файла, то есть количество записей.

```
{Программа чтения записей, структура которых определена в
  табл. 25, из файла во внешнем представлении, подготовленном в
  текущем подкаталоге (рис. 28). Фамилии студентов, у которых нет
  троек, переписать в новый типизированный файл во внутреннем
  представлении. Вывести записи из вновь созданного файла на эк-
  ран монитора }
TYPE STUDENT=RECORD
     FIO:STRING[18];
     NGR:STRING[6];
     MAT,INF,FYZ:INTEGER;
              END;
 VAR B:STUDENT;
     BTEXT:TEXT; {BTEXT - файл во внешнем представлении}
     BFILE:FILE OF STUDENT; {BFILE - файл во внутреннем
                             представлении}
  BEGIN
    ASSIGN(BTEXT,'DAV.TXT');{связать имя файловой перемен-
            ной BTEXT программы с именем физического набора
            данных DAV.TXT в текущем подкаталоге}
    RESET(BTEXT);{открыть файл BTEXT для чтения как сущест-
                  вующий}
    ASSIGN(BFILE,'BFILE.TXT');{связать имя файловой пере-
            менной BFILE программы с именем физического на-
            бора данных BFILE.TXT в текущем подкаталоге}
    REWRITE(BFILE);{открыть файл BFILE для записи как новый}
    READLN(BTEXT,B.FIO,B.NGR,B.MAT,B.INF,B.FYZ); {чи-
            тать в переменные B.FIO,B.NGR,B.MAT,B.INF,B.FYZ
            данные из первой записи файла BTEXT}
    WHILE NOT EOF(BTEXT) DO BEGIN {пока не конец файла
                                   выполнить}
       {операторы обработки полей записи}
       WITH B DO BEGIN
         IF (MAT>=4) AND (INF>=4) AND (FYZ>=4) THEN
            WRITE(BFILE,B); {записать запись во внутреннем
            представлении из переменной B в файл BFILE}
                 END;
       READLN(BTEXT,B.FIO,B.NGR,B.MAT,B.INF,B.FYZ);
            {читать в переменные B.FIO, B.NGR, B.MAT,
            B.INF,B.FYZ данные из текущей записи файла BTEXT}
            END;{конец работы оператора цикла WHILE}
```

```
CLOSE(BTEXT);{закрыть файл BTEXT}
CLOSE(BFILE);{закрыть файл BFILE}

RESET(BFILE);{открыть файл BFILE для чтения как существующий}
WHILE NOT EOF(BFILE) DO BEGIN {пока не конец файла BFILE выполнить}
   READ(BFILE,B); {читать запись во внутреннем представлении из файла BFILE в переменную B}
   {операторы обработки полей записи}
   WITH B DO BEGIN
     WRITELN(FIO:18,NGR:6,MAT:2,INF:2,FYZ:2);
     {вывести на экран монитора текущую запись из файла BFILE}
                END;
                          END;
    CLOSE(BFILE);{закрыть файл BFILE}
END.
```

12.3. Контрольные вопросы

1. Дайте определение термину «файл».
2. Приведите различные виды классификации файлов.
3. Приведите имя стандартной функции обработки конца файла.
4. Как записывается в программе стандартная функция обработки конца файла?
5. Дайте определение термину «текстовый файл».
6. Какой описатель имеет текстовый файл в программе?
7. Назовите стандартные функции обработки текстового файла.
8. Дайте определение термину «типизированный файл».
9. Какой описатель имеет в программе типизированный файл?
10. Назовите стандартные функции обработки типизированого файла.

Глава 13.

ПОСТРОЕНИЕ ГРАФИКОВ

В персональных компьютерах используется растровый способ вывода изображений. Это означает, что изображение формируется из ряда светящихся точек, которые называются *пикселями*. Каждый пиксель задаётся парой координат. Начало координат всегда находится в левом верхнем углу экрана; то есть координата левого верхнего угла – это точка с координатами (0,0). Состояние каждого пикселя (точнее его цвет) кодируется несколькими байтами, которые хранятся в видеопамяти.

13.1. Графический режим на языке *Turbo Pascal*

Для работы в графическом режиме на языке *Turbo Pascal* необходимы два файла: драйвер *egavga.bgi* и файл *graph.tpu* . Драйверы - это специальные программы, которые находятся в так называемых *bgi*-файлах и управляют работой видеокарты.

Для формирования графических изображений на языке *Turbo Pascal* предназначен стандартный библиотечный модуль *GRAPH*, который хранится на диске в файле *graph.tpu* . До работы в графическом режиме надо, чтобы файл *graph.tpu* находился либо в подкаталоге пользователя, либо задать его местоположение с помощью опции "*Options/Directories.../EXE & TPU directory*" (рис. 30 и рис. 31).

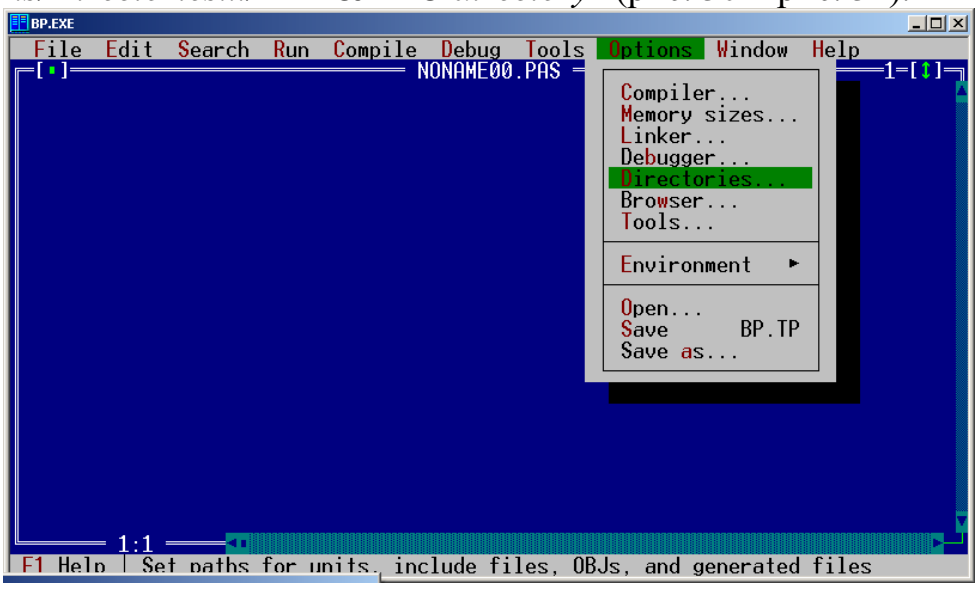


Рис. 30. Вызов диалогового *Directories* окна с помощью команды *Options/Directories...*

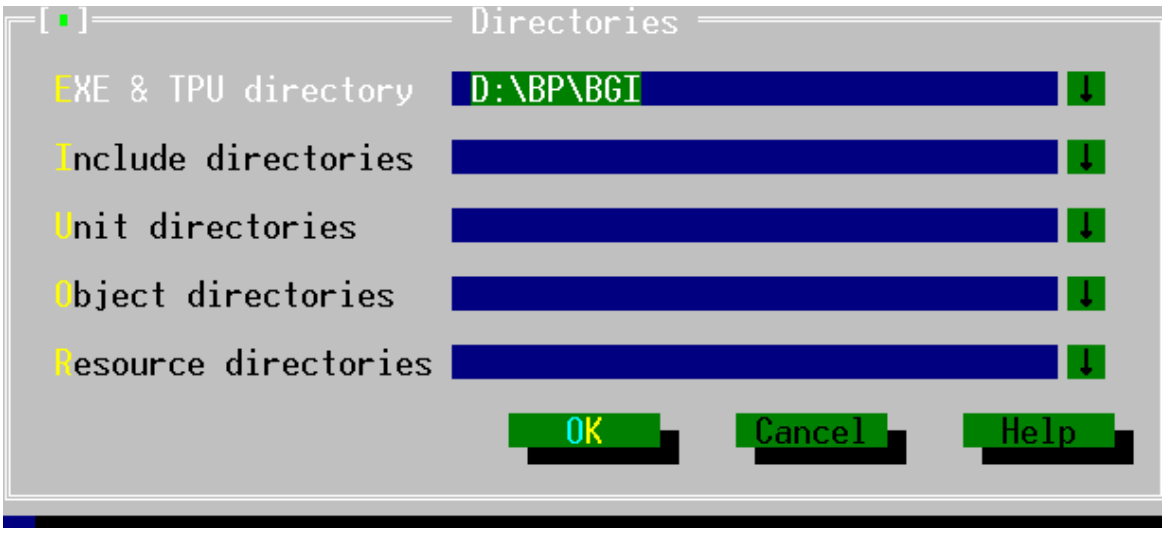


Рис. 31. Задание спецификации файла *graph.tpu* в диалоговом окне *Directories...* в строке *EXE&TPU directory*

Подключение модуля *GRAPH* к пользовательской программе осуществляется с помощью ключевого слова `USES` в самом начале программы:

```
USES GRAPH;
```

После подключения модуля *GRAPH* к пользовательской программе все графические средства можно использовать.

Инициализация графического режима

Для инициализации (запуска) графического режима необходимо вызвать процедуру `InitGraph`, которая устанавливает один из возможных режимов.

Формат процедуры:

`INITGRAPH`(*GraphDriver,GraphMode,PathToDriver*);

где *GraphDriver* – это переменная типа `INTEGER`, которая задаёт номер графического драйвера в диапазоне от 1 до 10;

GraphMode – это переменная типа `INTEGER`, которая задаёт режим работы графического адаптера в диапазоне от 0 до 2;

PathToDriver – это строковая переменная типа `STRING`, которая содержит спецификацию файла *egavga.bgi*. Если файл *egavga.bgi* находится в том же подкаталоге, где и программа пользователя, то в качестве третьего параметра надо поставить два апострофа (' ').

Перед инициализацией графического режима удобнее использовать стандартную константу DETECT, которая позволит процедуре INITGRAPH автоматически подключить нужный драйвер и автоматически установить наиболее подходящий для дисплея режим, то есть значение переменной *GraphMode.*

Стандартная константа DETECT присваивается параметру *GraphDriver* до инициализации графического режима:

```
GraphDriver:=Detect;
```

```
{Программный блок с инициализацией графического режима}
USES GRAPH;
 VAR DR:INTEGER; {переменная задаёт номер графического драйвера}
     MD:INTEGER; {переменная задаёт режим работы графического адаптера}
     ERR:INTEGER; {переменная для хранения кода ошибки}
  BEGIN
   DR:=DETECT;
   INITGRAPH(DR,MD,'');
   ERR:=GRAPHRESULT;
   IF ERR<>GROK THEN BEGIN
    WRITELN('Граф. ошибка: ',GRAPHERRORMSG(ERR));
    HALT(1);
                        END;
    {программный блок построения графика}
    CLOSEGRAPH; {процедура завершает работу адаптера в графическом режиме и восстанавливает текстовый режим работы экрана}
  END.
```

Этапы построения графика

Этапы построения графика должны быть следующими:

- определение размера экрана в пикселях по горизонтали и по вертикали, то есть определение текущего разрешения;
- построение осей координат по оси *X* и по оси *Y*;
- построение графика либо точками, либо линиями;
- вывод на графике строковых и численных данных.

Рассмотрим подробнее каждый этап построения графика.

1. Определение разрешения монитора.

После инициализации графического режима необходимо определить значение двух переменных, например *MX* и *MY* (обе переменные должны быть описаны типом INTEGER), как:

```
MX:=GETMAXX;   {размер экрана в пикселях по горизонтали}
MY:=GETMAXY;   {размер экрана в пикселях по вертикали}
```

где GETMAXX – это функция, возвращающая значение максимальной координаты (количество пикселей) по горизонтали монитора;

GETMAXY – это функция, возвращающая значение максимальной координаты (количество пикселей) по вертикали монитора.

2. Построение осей координат.

Построение осей координат выполняется с помощью процедуры

```
LINE(X1,Y1,X2,Y2);
```

Процедура LINE проводит прямую из точки с координатами (*X1,Y1*) в точку с координатами (*X2,Y2*) текущим цветом (переменные *X1, Y1, X2, Y2* должны быть описаны как INTEGER).

```
{Фрагмент программы: построение осей координат с нулевой
 точкой в середине экрана (рис. 32)}
MX:=GETMAXX;   {размер экрана в пикселях по горизонтали}
MY:=GETMAXY;   {размер экрана в пикселях по вертикали}
X1:=MX div 2;  {смещение по горизонтали для переменной X}
Y1:=10;
X2:=MX div 2;
Y2:=MY-10;
LINE(X1,Y1,X2,Y2);
X1:=10;
Y1:=MY div 2;  {смещение по вертикали для переменной Y}
X2:=MX-10 ;
Y2:=MY div 2;
LINE(X1,Y1,X2,Y2);
```

3. Построение графика.

Для построения графика используют процедуры:

- *PUTPIXEL* – при построении графика точками;
- *LINE* – при построении графика линиями.

3.1. Построение графика точками.

При построении графика точками используют процедуру

```
PUTPIXEL(X,Y,PIXEL);
```

Процедура *PUTPIXEL* выдаёт на экран точку с координатами *X* и *Y* цветом *PIXEL*.

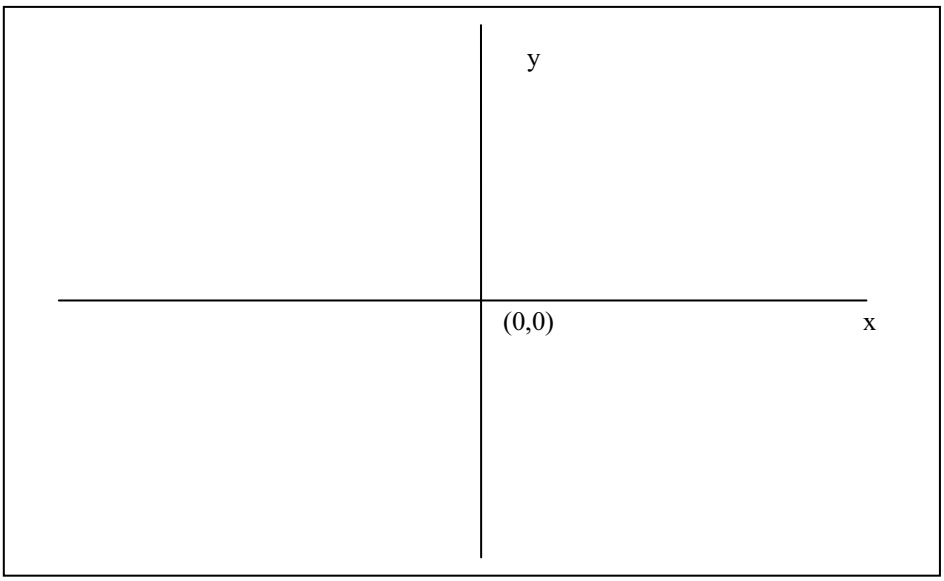


Рис. 32. Оси координат с нулевой точкой в центре экрана

PIXEL – это переменная типа *WORD*, поэтому её можно задавать либо цифрой, либо значением цвета по-английски без апострофов. Номера и названия цветов приведены в табл. 26.

Таблица 29 – Название и номера цветов в графическом режиме

Название цвета по-англий-ски	Цифровое обозначе-ние цвета	Название цвета по-русски	Название цвета по-английски	Цифровое обозначение цвета	Название цвета по-русски
Black	0	черный	*DarkGray*	8	темно-серый
Blue	1	синий	*LightBlue*	9	ярко-синий
Green	2	зеленый	*LightGreen*	10	ярко-зеленый
Cyan	3	голубой	*LightCyan*	11	ярко-голубой
Red	4	красный	*LightRed*	12	розовый
Magenta	5	фиолетовый	*LightMagenta*	13	малиновый
Brown	6	коричневый	*Yellow*	14	желтый
LightGray	7	светло-серый	*White*	15	белый

{Фрагмент программы: вывод точки белым цветом в графическом режиме}

(*Первый способ: цвет обозначается названием по-английски*)

```
PUTPIXEL(X1+15*I,Y1-10*I,WHITE);
```

(*Второй способ: цвет обозначается цифрой*)

```
PUTPIXEL(X2-15*I,Y2-10*I,15);
```

3.2. Построение графика отрезками линий.

График строится с использованием отрезков прямых, которые разбиваю график на мелкие участки. Для построения графика отрезками прямых линий используется процедура

```
LINE(X1,Y1,X2,Y2);
```

При этом началом каждого следующего отрезка служит конец предыдущего, поэтому в конце цикла необходимо выполнить присвоение координатам *X1* и *Y1* значений *X2* и *Y2*:

```
X1:=X2;
Y1:=Y2;
```

{Фрагмент программы вывода графика отрезками прямых линий. Вычисленные значения переменных *X1, Y1, X2, Y2* и процедура *LINE* с параметрами приведены в табл. 23}

```
X1:=20;
Y1:=100;
FOR I:=1 TO 5 DO BEGIN
   X2:=20+40*I;
   Y2:=100+40*I;
   LINE(X1,Y1,X2,Y2);
   X1:=X2;
   Y1:=Y2;
END;
```

Таблица 30 - Значения переменных *X1, Y1, X2, Y2* и процедура *LINE* с параметрами

Значение *I*	Значение *X1*	Значение *Y1*	Значение *X2*	Значение *Y2*	Процедура *LINE(X1,Y1,X2,Y2);*
1	20	100	60	140	*LINE(20,100,60,140);*
2	60	140	100	180	*LINE(60,140,100,180);*
3	100	180	140	220	*LINE(100,180,140,220);*
4	140	220	180	260	*LINE(140,220,180,260);*
5	180	260	220	300	*LINE(180,260,180,260);*

3.3. Построение графика линиями на заданном интервале X с шагом h

Чтобы начертить кривую $y = F(x)$ в интервале [*A,B*], который переменная *X* будет проходить с шагом *DELTAX=h*, а две соседние точки будут соединяться прямой, то блок построения графика будет оформлен следующим образом:

```
{Фрагмент программы построения графика y = F(x) в интервале
 [A,B] с шагом DELTAX=h}
X1:=A;
Y1:=F(X1);
WHILE X1<=B DO BEGIN
    X2:=X1+DELTAX;
    Y2:=F(X2);
    LINE(ROUND(X1),ROUND(Y1),ROUND(X2),ROUND(Y2));
              {ROUND(X) – встроенная функция округления X в
               сторону ближайшего целого}
   X1:=X2;
   Y1:=Y2;
                END;
```

3.4. Построение графика линиями на заданном интервале X с шагом h

Чтобы построить график функции *Y=F(X)* в заданном интервале [*A,B*], необходимо убедиться в том, что значения переменной *X* не выходят за пределы экрана, то есть 0<=*X*<=*GETMAXX* и значения переменной *Y* также не выходят за пределы экрана, то есть. 0<=*Y*<=*GETMAXY*. Значения *X* и *Y* нельзя (!) использовать в качестве параметров процедур *PUTPIXEL* и *LINE,* так как они в общем случае – переменные вещественного типа.

Для этого *до вызова* процедуры *InitGraph* необходимо вывести соответствующие значения *X* и *Y* на экран монитора для того, чтобы подобрать нормирующие коэффициенты *Kx* и *Ky* для обеих переменных.

Эти действия необходимо выполнить и в случае построения графика по значениям из массивов.

Пример_ . Пусть значения *X* и *Y* представлены значениями в табл. 24.

Таблица 31 – Табличные значения X и Y

X	-1.056	0	1.0056	2.0056	3.01	4.02	5.9	7
Y	-12	-10	-3	0	12	15	17	18

Тогда для вычисления *X1* и *X2* нормирующий коэффициент *Kx* будет равен 10; для вычисления Y1 и Y2 нормирующий коэффициент *Ky* будет равен 10.

4. Вывод строковых и численных данных.

4.1. Вывод строковых данных.

Чтобы сопроводить изображения пояснительным текстом, используются процедуры *OUTTEXT* и *OUTTEXTXY*. Пусть переменная *TEXT* – это переменная типа *STRING*.

Процедура

```
OUTTEXT(TEXT);
```

выдаёт строку текста *TEXT*, начиная с текущей позиции курсора.

{Фрагмент блока построения графика с процедурой *OUTTEXT*}

```
OUTTEXT('вводите данные');
```

Процедура

```
OUTTEXTXY(X,Y,TEXT);
```

выдаёт строку текста *TEXT*, начиная с точки (*X,Y*).

{Фрагмент блока построения графика с процедурой *OUTTEXTXY*}

```
OUTTEXTXY(60,100,'вводите данные');
```

4.2. Вывод численных данных.

В модуле *GRAPH* нет процедур для вывода численных данных. Поэтому сначала необходимо преобразовать число в строку с помощью процедуры *STR*, а затем знаком '+' подключить её к выводимой строке в процедурах *OUTTEXT* или *OUTTEXTXY*.

Преобразование целочисленных и вещественных типов данных в строки удобно выполнять созданными пользовательскими функциями, которые уже рассматривались в главе 8.

{Подпрограмма-функция преобразования целочисленного
значения в строковую переменную на языке *Turbo Pascal*}

```
FUNCTION INTST(INT:INTEGER):STRING;
```

{INT - целочисленное значение}

```
  VAR BUF:STRING[10];
    BEGIN
     STR(INT,BUF);
     INTST:=BUF;
    END;      {конец функции INTST}

      {Подпрограмма-функция преобразования вещественного
      значения в строковую переменную на языке Turbo Pascal}
FUNCTION REALST(R:REAL;DIG,DEC:INTEGER):STRING;
        {R - значение, DIG - количество символов}
        {DEC - количество символов после запятой}
  VAR BUF:STRING[20];
    BEGIN
     STR(R:DIG:DEC,BUF);
     REALST:=BUF;
    END;       {конец функции REALST}
```

Функции *INTST* и *REALST* указываются как параметры в процедурах *OUTTEXT* и OUTTEXTXY и снимают все проблемы вывода численных данных в графическом режиме.

```
VAR X:REAL;
    I:INTEGER;
    {…}
     X:=5.295643871;
     OUTTEXT('X='+REALST(X,11,9));
     I:=400;
     MOVETO(150,150);  {перемещение курсора в точку с
                        координатами(150,150)}
     OUTTEXT('I='+INTST(I));
```

П р о т о к о л р а б о т ы п р о г р а м м ы:

```
X=5.295643871
I=400
```

Пример_. Построить график функции $y = \sin(x)$ на интервале [1,10] с шагом изменения h=0.01 .

```
{Программа построения графика функции y = sin(x) на интервале
 [1,10] с шагом изменения h=0.01 }
USES GRAPH;
  VAR X,Y:REAL;
      X1,Y1,X2,Y2,MX,MY,YM,XM,DR,MD:INTEGER;
      KX,KY:INTEGER;
    FUNCTION FUN(X:REAL):REAL;
      VAR Y: REAL;
       BEGIN
        Y:=SIN(X);
        FUN:=Y;
       END;
    FUNCTION REALST(R:REAL;DIG,DEC:INTEGER):STRING;
                {R - значение, DIG - количество символов}
                {DEC - количество символов после запятой}
      VAR BUF:STRING[20];
       BEGIN
        STR(R:DIG:DEC,BUF);
        REALST:=BUF;
       END;
   BEGIN
     {вывод на экран монитора значений X и Y}
     X:=1;
     WHILE (X<=10) DO BEGIN
          Y:=FUN(X);
          WRITELN(X:8:2,'    ',Y:8:2);
          X:=X+0.01;
                      END;
     READLN;
    {инициализация графического режима}
     DR:=DETECT;
     INITGRAPH(DR,MD,'C:\BP\BGI');
        MX:=GETMAXX;
        XM:=TRUNC(MX/2);
        MY:= GETMAXY;
        YM:=TRUNC(MY/2);
        LINE(0,YM,MX,YM);
        LINE(XM,0,XM,MY);
        OUTTEXTXY(MX-10,YM+5,'x');
    {вывод графика точками}
        X:=1;
```

```
      KX:=10;
      KY:=100;
      WHILE (X<=10) DO BEGIN
        Y:=FUN(X);
        X1:=XM+TRUNC(KX*X);
        Y1:=YM-TRUNC(KY*Y);
        PUTPIXEL(X1,Y1,5);
        X:=X+0.01;
                        END;
      READLN;
      CLEARDEVICE;  {процедура очистки экрана}
  {вывод графика отрезками линий}
      LINE(0,YM,MX,YM);
      LINE(XM,0,XM,MY);
      X:=1;
      Y:=FUN(X);
      X1:=XM+TRUNC(KX*X);
      Y1:=YM-TRUNC(KY*Y);
      WHILE (X<=10) DO BEGIN
        X:=X+0.01;
        Y:=FUN(X);
        X2:=XM+TRUNC(KX*X);
        Y2:=YM-TRUNC(KY*Y);
        LINE(X1,Y1,X2,Y2);
        X1:=X2;
        Y1:=Y2;
                        END;
      READLN;
      CLOSEGRAPH;
END.
```

13.2. Контрольные вопросы

1. Какой способ вывода изображений используется в персональных компьютерах?

2. Где находится точка на экране с координатами (0,0)?

3. Какие два файла необходимы для работы в графическом режиме на языке *Turbo Pascal*?

4. Как инициализируется графический режим?

5. Назовите этапы построения графика.

Заключение

В пособии рассмотрены простые – это численные, логический, символьный и сложные – это массивы, строки и записи типы данных. На примерах рассмотрены основные алгоритмы обработки данных: линейные, ветвления, циклические.

При составлении простейших программ изученного материала достаточно, но для создания сложных программ материала пособия мало. В пособие не вошли важные разделы: применение теории графов в программировании, модульная организация программ, объединение в одну общую программу программ, которые написаны на разных языках программирования. В раздел «Алгоритмы сортировки и поиска данных» вошли не все известные алгоритмы сортировки данных.

Материал в пособии приведён по степени возрастания трудности. Тема «Подпрограммы» сложная, поэтому в отдельной главе рассказано о способах оформления подпрограмм. Тема «Файлы» является трудной, но её освоение поможет студентам в дальнейшем освоить дисциплину «Базы данных».

В целом материал пособия поможет студентам в дальнейшем усовершенствовать свои знания в программировании, в изучении инженерных прикладных программ: *Mathlab*, *Mathcad*, *Mathematica.*

ЛИТЕРАТУРА

По программированию на алгоритмическом языке *Turbo Pascal*

1. Ахо А.В., Хопкрофт Д., Ульман Д.Д. Структуры данных и алгоритмы.: Пер. с англ. – М., Издательский дом «Вильямс», 2001. – 384 с.: ил.

2. Вальвачев А.Н., Крисевич В.С. Программирование на языке ПАСКАЛЬ для персональных ЭВМ: Справ. пособие. – Минск: Вышэйшая школа, 1989. – 223 с.: ил.

3. Васюкова Н.Д., Тюляева В.В. Практикум по основам программирования. Язык ПАСКАЛЬ:Учебное пособие. – М.: Высш. шк.,2001. – 160 с.: ил.

4. Костюк Ю.Л., Фукс А.Л., Фукс И.Л. Основы разработки алгоритмов. – Томск: Изд-во ТГУ, 2004. – 60 с.

5. Немнюгин С.А. Turbo Pascal. – СПб.: Питер, 2002. – 496 с.: ил.

6. Немнюгин С.А. Turbo Pascal: практикум – СПб.: Питер, 2000. – 256 с.:ил.

7. Окулов С.М., Ашихмина Т.В., Бушмелёва Н.А. и др. Задачи по программированию. Под ред. С.М. Окулова. – М., БИНОМ. Лаборатория знаний, 2006. - 820с.: ил.

8. Офицеров Д.В., Старых В.А. Программирование в интегрированной среде Турбо-Паскаль: Справ. Пособие. - Минск: Беларусь, 2002. – 240 с.: ил.

9. Степанова И.П., Чердынцев Е.С., Моторина В.И. Информатика: Учебное пособие. – Томск, Изд-во ТПУ, 2003. -

По программированию на алгоритмическом языке *C++*

10. Дмитриева Е.А. Алгоритмические языки программирования. Учебное пособие для иностранных студентов. – Томск: Изд-во ТПУ, 2003. – 107 с.

11. Ишкова Э.А. C++. Начала программирования. Изд. 2-е, перераб. и доп. – М., ЗАО «Издательство БИНОМ», 2001. - 352 с.: ил.

12. Рейзлин В.И. Программирование на языке C++. Учебное пособие. – Томск: Изд-во ТПУ, 2003. – 179 с.

13. Триханова Н.В. Программирование на языке C++. Часть I: Учебное пособие. – Томск: Изд-во ТПУ, 1999. – 96 с.

14.Цапко И.В. Структуры и алгоритмы обработки данных: Учебное пособие. – Томск: Изд-во ТПУ, 2004. – 136 с.

Учебное издание

СТЕПАНОВА Ирина Павловна
ЧЕРДЫНЦЕВ Евгений Сергеевич

ИНФОРМАТИКА

Часть 2

Учебное пособие

Научный редактор
кандидат технических наук,
доцент О.Б. Фофанов
Редактор *В.И. Моторина*
Дизайн обложки *А.И. Сидоренко*

Подписано к печати 19.11.2010. Формат 60x84/16. Бумага «Снегурочка».
Печать XEROX. Усл. печ. л. 9,13. Уч.-изд. л. 8,26.
Заказ 1972-10. Тираж 100 экз.

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Томского политехнического университета сертифицирована
NATIONAL QUALITY ASSURANCE по стандарту ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru