MINISTERUL EDUCAŢIEI, CULTURII ŞI CERCETĂRII

ИНФОРМАТИКА

УЧЕБНИК ДЛЯ 10 КЛАССА

АНАТОЛ ГРЕМАЛСКИ • ЮРИЕ МОКАНУ • ИОН СПИНЕЙ
ЛУДМИЛА ГРЕМАЛСКИ

Ştiinţa, 2020

CZU 004(075.3)
И 741

Manualul şcolar a fost realizat în conformitate cu prevederile Curriculumului la disciplină, aprobat prin Ordinul Ministerului Educației, Culturii și Cercetării nr. 906 din 17 iulie 2019. Manualul a fost aprobat prin Ordinul Ministerului Educaţiei, Culturii şi Cercetării nr. 1048 din 28.09.2020, urmare a evaluării calităţii metodico-ştiinţifice.

Editat din sursele financiare ale Fondului Special pentru Manuale.

Comisia de evaluare:
Svetlana Brînză, profesor, grad didactic superior, Instituţia Publică Liceul Teoretic „N.M. Spătaru”, metodist DGETS, Chișinău (coordonator);
Gheorghe Chistruga, profesor, grad didactic superior, Instituţia Publică Liceul Teoretic „M. Eminescu”, Drochia;
Natalia Schițco, profesor, grad didactic superior, Instituţia Publică Liceul Teoretic „Socrate”, Chișinău;
Violina Bargan, profesor, grad didactic superior, Instituţia Publică Liceul Teoretic „Gh. Asachi”, Chișinău;
Ecaterina Adam, profesor, grad didactic unu, Liceul de Creativitate și Inventică „Prometeu-Prim”, Chișinău

Denumirea instituției de învățământ _____

Manualul a fost folosit: _____

Anul de folosire	Numele, prenumele elevului	Anul de studii	Aspectul manualului	
			la primire	la returnare

Dirigintele verifică dacă numele, prenumele elevului sunt scrise corect.
Elevii nu vor face niciun fel de însemnări în manual.
Aspectul manualului (la primire şi la returnare) se va aprecia cu unul dintre următorii termeni: *nou, bun, satisfăcător, nesatisfăcător.*

Responsabil de ediţie: Larisa Dohotaru
Redactor: Maria Volcovscaia
Corector: Aleftina Olari
Redactor tehnic: Nina Duduciuc
Machetare computerizată: Tudor Jalbă
Copertă: Romeo Șveț

ÎNTREPRINDEREA
EDITORIAL-POLIGRAFICĂ
ȘTIINȚA
str. Academiei, nr. 3; MD-2028, Chișinău, Republica Moldova
tel.: (+373 22) 73-96-16; fax: (+373 22) 73-96-27
e-mail: prini_stiinta@yahoo.com
www.editurastiinta.md

Descrierea CIP a Camerei Naţionale a Cărţii
Информатика: Учебник для 10 класса / Анатол Гремалски, Юрие Мокану, Ион Спиней, Лудмила Гремалски; comisia de evaluare: Svetlana Brînză (coordonator) [et al.]; Ministerul Educaţiei, Culturii şi Cercetării. – Ch. : Î.E.P. *Ştiinţa*, 2020 (Tipogr. „Bons Offices”). – 212 p.: fig., tab.

ISBN 978-9975-85-244-9

Tiparul executat la
Casa Editorial-Poligrafică „Bons Offices”
str. Feredeului, 4/6

СОДЕРЖАНИЕ

Повторение: Алгоритмы, программы и исполнители 6

Глава 1. Словарь и синтаксис языка ПАСКАЛЬ/С++

1.1. Знакомство с языком ПАСКАЛЬ/С++ 9
1.2. Метаязык БНФ 13
1.3. Синтаксические диаграммы 17
1.4. Алфавит языка 19
1.5. Словарь языка 19
1.5.1. Специальные символы и ключевые слова 19
1.5.2. Идентификаторы 23
1.5.3. Числа 25
1.5.4. Строки символов 28
1.5.5. Метки 31
1.5.6. Директивы 32
1.6. Разделители 33
Тест для самопроверки № 1 35

Глава 2. Простые типы данных

2.1. Концепция данных 39
2.2. Тип данных `integer`/**`int`** 43
2.3. Тип данных `real`/**`float`** 49
2.4. Тип данных `boolean`/**`bool`** 51
2.5. Тип данных `char` 54
2.6. *Перечисляемые* типы данных 57
2.7. *Интервальные* типы данных (ПАСКАЛЬ) 61
2.7.* Тип данных **`void`** (С++) 65
2.8. Порядковые типы данных 65
2.9. Объявление типов данных 70
2.10. Объявление переменных 78
2.11. Описание констант 81
Тест для самопроверки № 2 87

Глава 3. Операторы

3.1. Концепция действия 92
3.2. Выражения 94
3.3. Вычисление выражений 102
3.4. Тип выражений 105
3.5. Преобразование типов в языке C++ 111
3.6. Оператор присваивания 112
3.7. Оператор *вызова процедуры* в языке ПАСКАЛЬ 116
3.8. Вывод алфавитно-цифровой информации на экран 117
3.9. Ввод данных с клавиатуры 124
3.10. Пустой оператор 128
3.11. Оператор **`if`** 129
3.12. Оператор множественного выбора 133
3.13. Оператор **`for`** 138
3.14. Составной оператор 144
3.15. Оператор **`while`** 148
3.16. Оператор **`repeat`** 154
3.17. Оператор **`goto`** 159
3.18. Структура программы ПАСКАЛЬ / C++ 165
Тест для самопроверки № 3 168

Глава 4. Модули по выбору

4.1. Веб-дизайн 172
4.2. Компьютерная графика 179
4.3. Цифровая фотография 183

Ответы на задания из тестов для самопроверки 186

Приложения

Приложение 1. Словарь языка ПАСКАЛЬ 201
Приложение 2. Синтаксис языка ПАСКАЛЬ 202
Приложение 3. Компиляция и отладка ПАСКАЛЬ программ 205
Приложение 4. Словарь языка C++ 207
Приложение 5. Компиляция и отладка программ C++ 208

ДОРОГИЕ ДРУЗЬЯ!

Вам уже известно, что информатика является областью науки, изучающей способы хранения, передачи и обработки информации с помощью компьютеров.

В гимназических классах вы изучили основные понятия информатики – данные, информация, компьютер, исполнитель, алгоритм – и приобрели практические навыки работы на компьютере. Вы научились создавать и обрабатывать самые разнообразные документы с помощью текстового редактора, систематизировать и обрабатывать числовые и текстовые данные с помощью табличного процессора. Вы научились разрабатывать и отлаживать программы для управления исполнителями. В результате обучения вы убедились, что работой современных компьютеров можно управлять с помощью алгоритмов.

Данный учебник поможет вам усвоить знания, необходимые для развития алгоритмического мышления и формирования информационной культуры. Результатом обучения будет дальнейшее развитие способностей каждого ученика разрабатывать алгоритмы для решения возникающих в повседневной жизни задач с помощью компьютеров.

Известно, что алгоритмы очень точно описывают необходимые для обработки информации операции и порядок их следования. Обычно в процессе начальной разработки алгоритма пользователи применяют для его описания разговорный язык, например, румынский, русский или английский. Затем для более точного описания предполагаемых действий могут быть использованы, например блок-схемы. Но для того, чтобы алгоритм стал понятен компьютеру, он должен быть написан на одном из языков программирования.

В данном учебнике для дидактических целей используются языки программирования ПАСКАЛЬ и C++. Язык ПАСКАЛЬ, названный в честь великого математика и философа Блеза Паскаля, получил мировое распространение благодаря тому, что он наиболее полно подходит для преподавания и изучения информатики в школе. Будучи инструментом, предназначенным для развития алгоритмического мышления, язык ПАСКАЛЬ включает в себя все базовые концепции современных информационных систем: переменные, константы, типы данных, простые и составные команды.

Язык C++ предназначен для разработки программного обеспечения и очень популярен среди профессиональных программистов. Рекомендуем его изучение тем ученикам, которые мечтают стать специалистами именно в области информационных и коммуникационных технологий.

Выбор, какой именно из языков программирования изучать, остается на усмотрение каждого учебного заведения, причем учебник дает возможность изучения как одного, так и обоих языков.

За каждой частью теоретического материала, приведенного в учебнике, следуют примеры, упражнения и контрольные вопросы. Предполагается, что ученики введут и выполнят на компьютерах все приведенные в учебнике программы, ответят на поставленные вопросы, самостоятельно напишут и отладят программы, необходимые для решения предложенных задач. Все вошедшие в учебник программы были отлажены и протестированы в интегрированных средах для разработки программ PASCAL/C++.

Как и в случае других школьных предметов, при изучении информатики особая роль отводится формированию навыков самообучения, развитию творческого мышления. С этой целью в учебник включены модули по выбору. Ученики должны изучать указанные модули с помощью активных методов обучения, то есть самостоятельно, а еще лучше – в составе творческих групп, создавать веб-страницы, обрабатывать цифровые фотографии, разрабатывать оригинальные компьютерные программы.

Будучи неотделимой частью современной культуры, информатика играет значительную роль в развитии человечества и открывает многообещающие перспективы для каждого из нас. Реализация указанных перспектив зависит во многом от знания основных концепций языков программирования и способностей их применения на практике.

Успехов вам!
Авторы

Повторение

АЛГОРИТМЫ, ПРОГРАММЫ И ИСПОЛНИТЕЛИ

Известно, что алгоритм представляет собой конечную совокупность правил и предписаний, выполнение которых обеспечивает решение поставленной задачи. Процесс разработки алгоритмов называется алгоритмизацией.

В информатике понятие алгоритма неразрывно связано с понятием исполнителя. Исполнитель представляет собой объект, который может выполнять (исполнять) определенные команды. Множество таких команд образует систему команд исполнителя.

В гимназических классах мы изучили исполнителей **Кенгуренок** и **Муравей**, разработанных в учебных целях для школ нашей страны, различных исполнителей в графических интерактивных средах программирования типа *Logo, Scratch, Robo* и др. Напомним, что исполнитель **Кенгуренок** может исполнять команды ШАГ, ПОВОРОТ, ПРЫЖОК, а исполнитель **Муравей** – команды ВВЕРХ, ВНИЗ, ВПРАВО, ВЛЕВО.

Существуют два режима управления исполнителями: ручной и автоматический.

Режим ручного управления предполагает, что ввод каждой команды приводит к ее немедленному выполнению исполнителем. *Режим автоматического управления* предполагает выполнение последовательности команд без вмешательства пользователя. Перед использованием автоматического режима алгоритм необходимо сначала написать и ввести его в память центра управления исполнителем.

Последовательность команд, предназначенная для автоматического управления исполнителем, называется *программой*. Очевидно, что программа представляет собой алгоритм, записанный на языке исполнителя. Процесс разработки программ называется *программированием*.

В гимназических классах мы разработали большое число программ для управления исполнителями Кенгуренок и Муравей: рисование квадратов, орнаментов и спиралей, размещение символов в заданном порядке и др. Созданные нами программы были написаны на языке программирования и содержали простые команды ШАГ, ПОВОРОТ, ПРЫЖОК, ВВЕРХ, ВНИЗ, ВПРАВО, ВЛЕВО, вызовы процедур (подпрограмм) и составные команды ПОВТОРИ, ПОКА, ЕСЛИ.

Будучи разработанными в учебных целях, исполнители Кенгуренок и Муравей не могут обеспечить более сложную обработку информации. Очевидно, что для решения практических задач, возникающих в реальной жизни, нужны более мощные исполнители, а именно современные компьютеры.

Другими словами, компьютер является исполнителем, который выполняет в автоматическом режиме загруженные в его внутреннюю память программы. Напомним, что любая программа, загруженная во внутреннюю память компьютера, представляет собой последовательность двоичных слов, которая указывает компьютеру состав и порядок требуемых операций.

В качестве примера ниже приведен фрагмент программы, написанной на машинном языке:

```
10010101 10000011 00110100 01000100
01010010 01011101 00010010 10010101
11010010 01001100 00101001 01110100
00010101 01010100 11111010 10100011
```

Исторические ориентиры:
1955 – FORTAN (FORmula TRANslation)
1960 – ALGOL (ALGOritmic Language)
1960 – COBOL (COmmon Business Oriented Language)
1971 – PASCAL (Blaise PASCAL)
1972 – C
1980 – C++
1995 - Java

Так как разработка программ в двоичных кодах является трудоемкой и неэффективной работой, алгоритмы, предназначенные для решения задач с помощью компьютера, целесообразно писать на специальных языках, называемых языками высокого уровня. В процессе развития информатики были разработаны многие языки программирования высокого уровня. Их число в настоящее время достигает порядка восьми тысяч. Все же лишь небольшое их число широко используется, а в учебных целях самыми распространенными являются языки LOGO, BASIC, PASCAL, C, C++, Java. Эти языки содержат средства для написания и вызова подпрограмм, операторы для программирования линейных, разветвляющихся и циклических алгоритмов.

Для примера представляем программу, написанную на языке ПАСКАЛЬ/C++, которая вычисляет корни уравнения первой степени:

ПАСКАЛЬ

```
Program Exemplu;
var a, b, x : real;
begin
  readln(a, b);
  if a<>0 then
    begin
      x:=-b/a;
      writeln('Уравнение имеет
      один корень');
      writeln(x);
    end;
  if (a=0) and (b=0) then
      writeln('Уравнение
      имеет бесконечное
      множество корней');
  if (a=0) and (b<>0) then
      writeln('Уравнение не
      имеет смысла');
end.
```

C++

```
// Программа Exemplu
#include <iostream>
using namespace std;
int main()
{
float a, b, x;
cin>>a>>b;
if (a!=0)
{
x=-b/a;
cout<<"Уравнение имеет
       один корень"<<endl;
cout<<x<<endl;
}
if ((a==0)&&(b==0))
   cout<<"Уравнение
      имеет бесконечное
      множество корней"<<endl;
if ((a==0)&&(b!=0)) cout<<
 "Уравнение не имеет
 смысла"<<endl;
return 0;
}
```

Сравнив фрагмент программы, написанный на машинном языке, с программой Exemplu, можно убедиться, что применение языка высокого уровня, в данном примере – языка ПАСКАЛЬ/C++, значительно упрощает процесс разработки программ.

Обычно при использовании языков высокого уровня решение задачи с помощью компьютера включает следующие этапы:

1) описание алгоритма в общих чертах с помощью языка общения между людьми, например русского, румынского или английского;

2) если необходимо, более точное описание требуемых операций с помощью блок-схем;

3) написание алгоритма на одном из языков высокого уровня, например на ПАСКАЛЕ или на C++;

4) перевод программы с языка высокого уровня на машинный язык, то есть в последовательность двоичных слов;

5) обнаружение и исправление возможных ошибок и запуск программы на выполнение.

Перевод программ с языка высокого уровня на машинный язык называется компиляцией и осуществляется в автоматическом режиме с помощью специальных программ, называемых компиляторами. Для редактирования и отладки программ были разработаны специальные прикладные программы, называемые *средами разработки программ.*

Вопросы и упражнения

❶ Вспомните по крайней мере три алгоритма, которые вы изучали на уроках математики и информатики

❷ Какую информацию должно содержать полное описание определенного исполнителя?

❸ Каковы основные средства, предназначенные для представления алгоритмов?

❹ Чем отличается программный режим управления от ручного?

❺ В чем состоит отличие алгоритма от программы? Обоснуйте свой ответ.

❻ Назовите основные этапы решения задач с помощью компьютеров.

❼ Чем отличаются языки высокого уровня от машинных языков?

❽ **ИЗУЧИТЕ САМОСТОЯТЕЛЬНО ИЛИ ВМЕСТЕ С ОДНОКЛАССНИКАМИ!** Поищите в Интернете и узнайте:

- мнения выдающихся людей из мира бизнеса, из сектора информационных технологий и коммуникаций об изучении программирования в школах;
- какие языки программирования изучают в школах во всем мире;
- процентное соотношение различных языков программирования, изучаемых в школах, в странах с традициями в этой области;
- процентное соотношение различных языков программирования, используемых в индустрии программного обеспечения (*software*);
- популярность языков программирования по индексу TIOBE. Аббревиатура TIOBE состоит из первых букв знаменитой комедии Оскара Уайльда *The Importance Of Being Earnest* («Как важно быть серьезным»).

❾ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ (индивидуально или в команде).** Выберите два наиболее распространенных языка программирования: один используется в школах в учебных целях, а другой – в индустрии программного обеспечения. Сравните эти языки по следующим критериям: удобочитаемость, простота, информативность, интуитивность.

Глава 1

СЛОВАРЬ И СИНТАКСИС ЯЗЫКА ПАСКАЛЬ/C++

1.1. Знакомство с языком ПАСКАЛЬ/C++

Pascal

Рассмотрим следующую программу на языке ПАСКАЛЬ:

```
1  Program P1;
2   { Сумма целых чисел x, y, z }
3  var x, y, z, s : integer;
4  begin
5    writeln('Введите целые числа x, y, z:');
6    readln(x, y, z);
7    s:=x+y+z;
8    writeln('Сумма введенных чисел:');
9    writeln(s);
10 end.
```

Числа *1, 2, 3, …, 10* в левой части страницы не являются частью представленной программы. Они служат только для цифрового обозначения объяснения значения каждой строки.

Строка 1. Слово **program** является зарезервированным словом языка, а **P1** – словом, определяемым пользователем. Зарезервированные слова служат для составления программ, а слова, определяемые пользователем, – для имен переменных, констант, подпрограмм, программ и т. д.

Строка 2. Данная строка является пояснением, комментарием. Комментарий заключается в фигурные скобки: {, }. Комментарий не влияет на работу программы и предназначен исключительно для пользователя.

Строка 3. Зарезервированное слово **var** (*variable* – переменная) описывает переменные `x, y, z` и `s`, используемые в программе. Слово `integer` (целый) указывает тип соответствующих переменных. Таким образом, переменные `x, y, z` и `s` могут принимать только целые значения. Данная строка образует раздел описаний.

Строка 4. Зарезервированное слово **begin** (начало) означает начало выполняемой части программы – раздела операторов.

Строка 5. Вывод сообщения на стандартное устройство вывода, обычно на экран. Слово `writeln` (*write line* – запись и переход на новую строку) представляет собой обращение к стандартной процедуре, аргументом которой является текст сообщения:

```
Введите целые числа x, y, z:
```

Отметим, что апострофы не являются частью текста, отображаемого на экране.

Строка 6. Считывание трех чисел со стандартного устройства ввода, как правило, – с клавиатуры. Числа вводятся в одной строке и отделяются друг от друга одним или несколькими пробелами. После ввода последнего числа нажимается клавиша <ENTER>. Числа считываются в переменные `x, y, z`. Слово `readln` (*read line* – считывание и переход на новую строку) представляет собой обращение к стандартной процедуре. Аргументами данной процедуры являются имена переменных, в которых запоминаются введенные целые числа.

Строка 7. Оператор присваивания. Переменной `s` присваивается значение `x+y+z`.

Строка 8. Вывод сообщения

```
Сумма введенных чисел:
```

на стандартное устройство вывода.

Строка 9. Вывод значения переменной s на стандартное устройство вывода.

Строка 10. Зарезервированное слово **`end`** означает конец раздела операторов, а точка – конец программы.

Таким образом, программа на языке ПАСКАЛЬ состоит из следующих частей:

- **заголовок,** в котором указывается имя программы;
- **раздел объявлений,** где описываются используемые в программе переменные, функции, подпрограммы и т. д.;
- **раздел операторов,** содержащий операторы, которые компьютер должен выполнить в заданном порядке.

Для редактирования, компиляции и выполнения программ, написанных на языке ПАСКАЛЬ, были разработаны специальные приложения, называемые средами разработки программ. Как правило, в школьных компьютерных классах установлены среды разработки программ Turbo PASCAL или Free PASCAL.

Рассмотрим программу P1, написанную на C ++:

Для редактирования, компиляции и выполнения программ, написанных на языке ПАСКАЛЬ, были разработаны специальные приложения, называемые средами разработки программ. Как правило, в школьных компьютерных классах установлены среды разработки программ Turbo PASCAL или Free PASCAL.

Рассмотрим программу `P1`, написанную на C++:

```
1  // Программа P1
2  #include <iostream>
3  using namespace std;
4  // Сумма целых чисел x, y, z
5  int main()
6  {
7  int x, y, z, s;
8   cout<<"Введите целые числа x, y, z:"<<endl;
9  cin>>x>>y>>z;
10 s=x+y+z;
11 cout<<"Сумма введенных чисел: ";
12 cout<<s<<endl;
13 return 0;
14 }
```

Числа *1, 2, 3, …, 14* в левой части страницы не являются частью представленной программы. Они служат только для цифрового обозначения объяснения значения каждой строки.

Строка 1. Однострочный комментарий начинается с символа «//». Комментарий никак не влияет на выполнение программы и предназначен исключительно для пользователя.

Строка 2. Директива `#include`. Эта директива вставляет в текст программы P1 текст, содержащийся в файле `iostream`. Этот файл содержит описания подпрограмм, выполняющих ввод и вывод данных. Вообще говоря, программы на C++ могут использовать не только стандартные функции, но и функции, написанные другими программистами. Эти функции могут быть включены в разрабатываемую программу с помощью указания в директивах `#include` имен файлов, которые их содержат. Обычно в процессе разработки и отладки программ в среде разработки тексты, включенные в программу этими директивами, не отображаются.

Строка 3. Ключевые слова **`using namespace`** в этой строке указывают, что разрабатываемая программа должна использовать пространство имен `std`. Это пространство представляет собой описание идентификаторов, используемых в библиотеках, содержащих стандартные функции языка C++.

Строка 4. Пояснительный текст, комментарий.

Строка 5. Описание основной функции `main.` Эта функция должна появляться в любой программе на C++. Она является точкой, с которой программа запускается на выполнение, независимо оттого, где она находится в исходном коде.

Строка 6. Фигурная скобка { указывает на начало тела основной функции `main`.

Строка 7. Слово `int` (целое число) указывает тип соответствующих переменных. Следовательно, `x, y, z` и `s` могут иметь только целые числа в качестве значений.

Строка 8. Вывод на экран текста, заключенного в кавычках. Сами кавычки не являются частью текста, который будет отображаться. Слово `cout` (*console output* – вывод на консоль) – это имя потока данных (торрента), передаваемого программой стандартному устройству вывода. Символ `<<` – оператор передачи информации. Подпрограмма, реализующая вывод данных, описана в файле `iostream` в стандартной библиотеке C ++, а имя `cout` является частью пространства имен `std`. Любой оператор C++ заканчивается символом `;` (точка с запятой).

Слово `endl` (*end line* - конец строки) представляет собой манипулятор, который после отображения сообщения реализует переход на новую строку.

Строка 9. Считывание значений переменных `x, y, z` со стандартного устройства ввода, обычно – с клавиатуры. Слово `cin` (*console input* – консоль ввода) это имя потока данных (торрента), передаваемого в программу стандартным устройством ввода, а символ >> – оператор передачи информации. В процессе выполнения программы считываемые числа должны быть набраны в одной строке и разделены одним или несколькими пробелами. После ввода последнего числа нужно нажать клавишу <ENTER>.

Строка 10. Оператор присваивания. Переменная `s` получает значение `x+y+z`.

Строка 11. Отображение на стандартном устройстве вывода сообщения

```
Сумма введенных чисел:
```

Строка 12. Отображение значения переменной `s` на стандартном устройстве вывода.

Строка 13. Оператор **`return`** `0;` (возврат `0;`) имеет эффект завершения выполнения основной функции. Оператор **`return`** используется для передачи операционной

системе кода, характеризующего способ, которым завершилось выполнение программы, `0` означает, что выполнение программы завершилось без ошибок. Это наиболее распространенный способ завершить программу на C++.

Строка 14. Фигурная скобка `}` обозначает конец исполняемой части функции `main`.

Таким образом, программа на языке C++ состоит из следующих компонентов:

- ***директивы*** (опционально);
- ***глобальные объявления*** (опционально), описывающие типы пользователей, переменные, константы и т. д., используемые в программе;
- ***пользовательские функции*** (опционально);
- ***основная функция*** `main` (обязательно).

Специальные приложения, называемые средами разработки программ, например, такие как **Code :: Blocks** и **Dev-Cpp**, были разработаны для редактирования, компиляции и запуска программ C++.

Среды разработки программ предоставляют пользователям следующие возможности:

- ввод и редактирование программ;
- хранение программ в отдельных файлах;
- открытие, редактирование и сохранение файлов, содержащих разрабатываемые программы;
- обнаружение синтаксических ошибок;
- компиляция и запуск разрабатываемых программ;
- отладка программ.

Способы использования сред разработки программ на языках PASCAL и C++ описаны в приложениях.

Вопросы и упражнения

❶ Введите и запустите на выполнение программу `P1`.

❷ Из каких основных частей состоит программа на языке ПАСКАЛЬ/ C++?

❸ Введите и запустите на выполнение следующую программу:

ПАСКАЛЬ

```
Program P2;
{ Вывод предопределенной константы MaxInt }
begin
  writeln('MaxInt=', MaxInt);
end.
```

C++

```
// Программа P2
#include <iostream>
#include <limits>
using namespace std;
// Вывод предопределенной константы INT_MAX
int main()
{
cout<<"INT_MAX="<<INT_MAX<<endl;
return 0;
}
```

❹ Укажите заголовок, раздел объявлений и раздел операторов программы, представленной в качестве примера в этом параграфе. Объясните назначение каждой строки данной программы.

❺ **ПРИМЕНИТЕ!** Измените программу P1 (приведенную в качестве примера в данном параграфе) таким образом, чтобы она выводила на экран текст:

```
        По тонкому лучу скользни
        Из непомерной дали,
        Чтоб помыслы мои и дни
        Тобою засверкали.
  („Luceafărul”, Mihai Eminescu, Перевод Ю.Кожевникова)
```

❻ **ПРИМЕНИТЕ!** Измените программу P1 (приведенную в качестве примера в данном параграфе) таким образом, чтобы она вычислила сумму чисел *x*, *y*, введенных с клавиатуры.

❼ **ОБРАТИТЕ ВНИМАНИЕ!** Сохраните на своем компьютере программу, разработанную в задании 6. Обратите внимание, какое расширение имеет исходный файл.

❽ **ИЗУЧИТЕ САМОСТОЯТЕЛЬНО ИЛИ ВМЕСТЕ С ОДНОКЛАССНИКАМИ!** Поищите в Интернете и узнайте:

- какие языки программирования существуют и как они развивались. Заполните таблицу:

Поколение	Язык программирования

- от каких часто используемых языков программирования произошли новые языки?
- определите пять наиболее часто используемых языков программирования;
- откуда произошло название языка программирования, который вы изучаете?

1.2. Метаязык БНФ

Любой язык программирования определяется через синтаксис и семантику. Известно, что синтаксис – это совокупность правил, которые задают структуру предложений, а семантика – это совокупность правил, определяющих смысл и значение соответствующих предложений. В случае языков программирования эквивалентом предложения является программа.

Ясно, что синтаксис любого языка программирования может быть описан с помощью одного из языков общения между людьми, например русского, румынского, английского, французского и т.д. Однако такого рода описание будет объемным и может оказаться двусмысленным. Поэтому для лаконичного и точного описания синтаксиса языков программирования были разработаны специальные языки, называемые метаязыками. Самым распространенным метаязыком является *метаязык* БНФ – *формы Бэкуса-Наура.*

Метаязык БНФ использует следующие символы:

терминальные символы – символы, из которых состоит программа на языке ПАСКАЛЬ/С++;

нетерминальные символы – символы, которые обозначают грамматические единицы (конструкции) языка.

Нетерминальные символы записываются между знаками “<” и “>”.

Например, цифры `0`, `1`, `2` …, `9`, буквы `A`, `B`, `C`, …, `Z` являются терминальными символами, а <*Цифра*>, <*Буква*> являются нетерминальными символами.

Описание синтаксиса языка программирования состоит из совокупности металингвистических формул.

Под **металингвистической формулой** будем понимать конструкцию, состоящую из двух частей: левой и правой, разделенных символами: “::=”, что означает «является по определению». В левой части формулы находится нетерминальный символ.

В правой части металингвистической формулы с помощью символа “|”, означающего «или», описываются все возможные варианты определения нетерминального символа.

Например, формула

<*Цифра*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

определяет грамматическую единицу <Цифра> как один из терминальных символов: `0`, `1`, …, `9`.

Таким же образом интерпретируется металингвистическая формула:

<*Буква*> ::= `a | b | c | d | e | f | g | h | i | j | k | l |m |`
`n | o | p | q | r | s | t | u | v | w | x | y | z`

В правой части металингвистической формулы может быть указана последовательность из двух и более символов. Такая запись соответствует операции конкатенации (присоединения).

В частности, формула

<*Ид*> ::= <*Буква*> <*Цифр*а>

определяет грамматическую конструкцию <*Ид*> как букву, за которой следует цифра.

Пример:

1) `a9`

2) `a1`

3) `c3`

4) `x4`

5) `d0`

6) `e8`

В случае, если некоторая часть определения нетерминального символа может отсутствовать или повторяться произвольное число раз, она заключается в фигурные скобки {, }.

Например, формула

<*Целое без знака*> ::= <*Цифр*а> {<*Цифр*а>}

определяет нетерминальный символ <*Целое без знака*> как непустую последовательность цифр. Последовательности `0`, `0000`, `001`, `1900`, `35910` соответствуют данному определению, а последовательность `3a5910` – не соответствует.

Формула

<*Идентификатор*> ::= <*Буква*> {<*Буква*> | <*Цифр*а>}

указывает, что идентификатор всегда начинается с буквы, после которой может следовать конечная последовательность из букв и цифр. Например, `a`, `a1`, `a1b`, `a23x`, `a14bxz`, `ab5` соответствуют данной формуле, а `2a` – не соответствует.

В случае, когда некоторая часть определения нетерминального символа может отсутствовать или присутствовать ровно один раз, она заключаются в квадратные скобки: `[, ]`.

Например, формула

<*Масштабный множитель*> ::= [+ | –] <*Целое без знака*>

определяет масштабный множитель как целое число без знака, которому может предшествовать + или –. Например: `1`, `+1`, `-1`, `20`, `+20`, `-20`, `+003` соответствуют данной формуле, а `3-5` – не соответствуют.

Обратим внимание на то, что символы [,], {, } принадлежат метаязыку и их не следует путать с соответствующими символами, используемыми в программах на языке ПАСКАЛЬ.

Вопросы и упражнения

❶ Дайте объяснение терминам *синтаксис и семантика*.

❷ Для чего предназначен метаязык?

❸ Как задается синтаксис любого языка программирования с помощью метаязыка БНФ?

❹ **ПРИМЕНИТЕ!** Синтаксис некоторого простого языка программирования описан с помощью следующих металингвистических формул:

<*Цифра*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`
<*Число*> ::= <*Цифра*> {<*Цифра*>}
<*Знак*> ::= + | –
<*Арифметическое выражение*> ::= <*Число*> {<*Знак*><*Число*>}

Какие из приведенных ниже последовательностей соответствуют определению лексической единицы <*Число*>?

a)	`0`	*f)*	`0+0`	*k)*	`0000`
b)	`1`	*g)*	`11100+1`	*l)*	`0001`
c)	`11100`	*h)*	`11100–1`	*m)*	`–152`
d)	`00011`	*i)*	`931`	*n)*	`+351`
e)	`20013`	*j)*	`614`	*o)*	`412`

Какие из нижеприведенных последовательностей соответствуют определению лексической единицы <*Арифметическое выражение*>?

a)	`0+1`	*f)*	`–13`	*k)*	`21+00000`
b)	`1+0–3`	*g)*	`21+–16`	*l)*	`39+00001`
c)	`0+0+4`	*h)*	`–21–16`	*m)*	`00001–00001`
d)	`1+1–9`	*i)*	`68–13`	*n)*	`379–486`
e)	`6+6+21`	*j)*	`42+650`	*o)*	`31+12–51+861`

❺ Синтаксис некоторого языка общения компьютер – пользователь задается следующим образом:

<*Диск*> ::= A: | B: | C: | D: | E:
<*Список параметров* > ::= <*Диск*> {, <*Диск*>}
<*Название команды*> ::= Считывание | Копирование | Форматирование
<*Оператор*> ::= <*Название команды*><*Список параметров*>

Какие из нижеприведенных последовательностей соответствуют определению лексической единицы <*Оператор*>?

a) `Считывание`

b) `Считывание A:`

c) `Копирование F:`

d) `Копирование A:,`

e) `Форматирование D:, E:`

f) `Копирование A: B:`

g) `Считывание D`

h) `Форматирование D:, F:`

i) `Копирование E:, A:,`

j) `Копирование F:, A:`

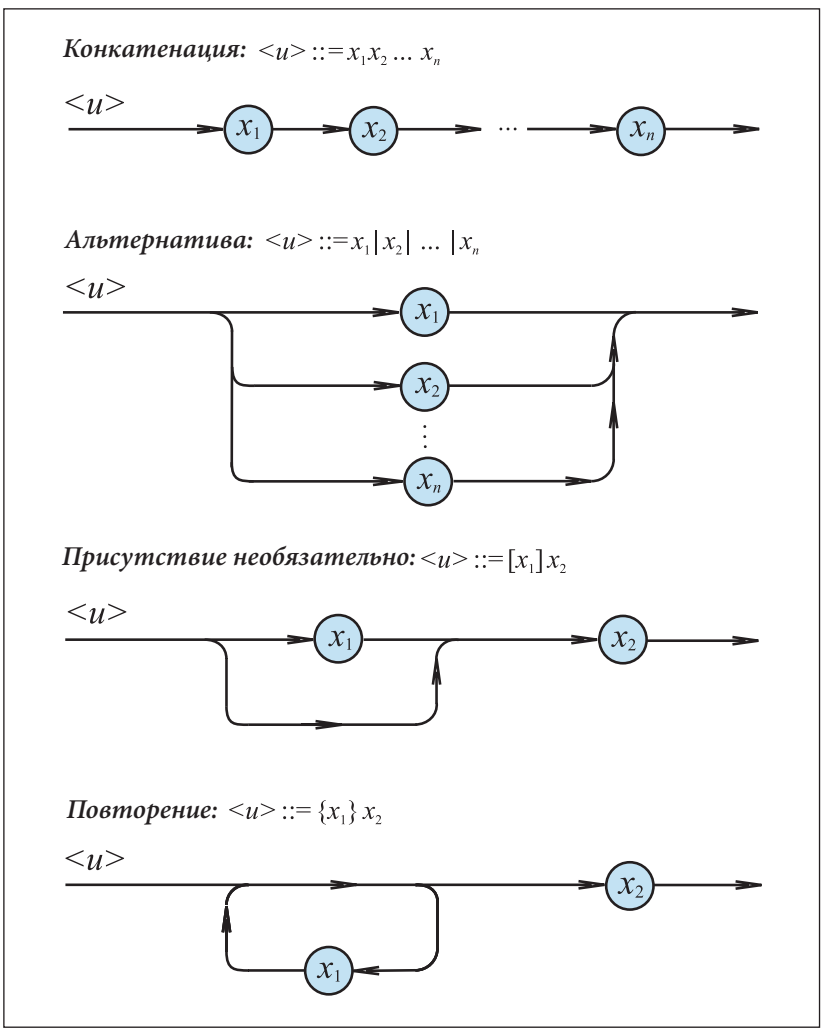


Рис. 1.1. Представление формул БНФ с помощью синтаксических диаграмм

1.3. Синтаксические диаграммы

Синтаксические диаграммы более наглядно описывают синтаксис языков программирования. Диаграммы составляются в соответствии с формулами БНФ.

Каждому терминальному символу на синтаксических диаграммах должен соответствовать круг или овал, в который вписывается соответствующий символ. Нетерминальные символы заключаются в прямоугольник. Овалы и прямоугольники объединяются согласно синтаксическим диаграммам, приведенным на *рис.1.1.*

На *рис. 1.2* показаны синтаксические диаграммы для грамматических единиц <*Целое без знака*>, <*Идентификатор*> и <*Масштабный множитель*>, определенных в предыдущем параграфе. Отметим, что каждому пути диаграммы соответствует синтаксически правильная последовательность терминальных символов.

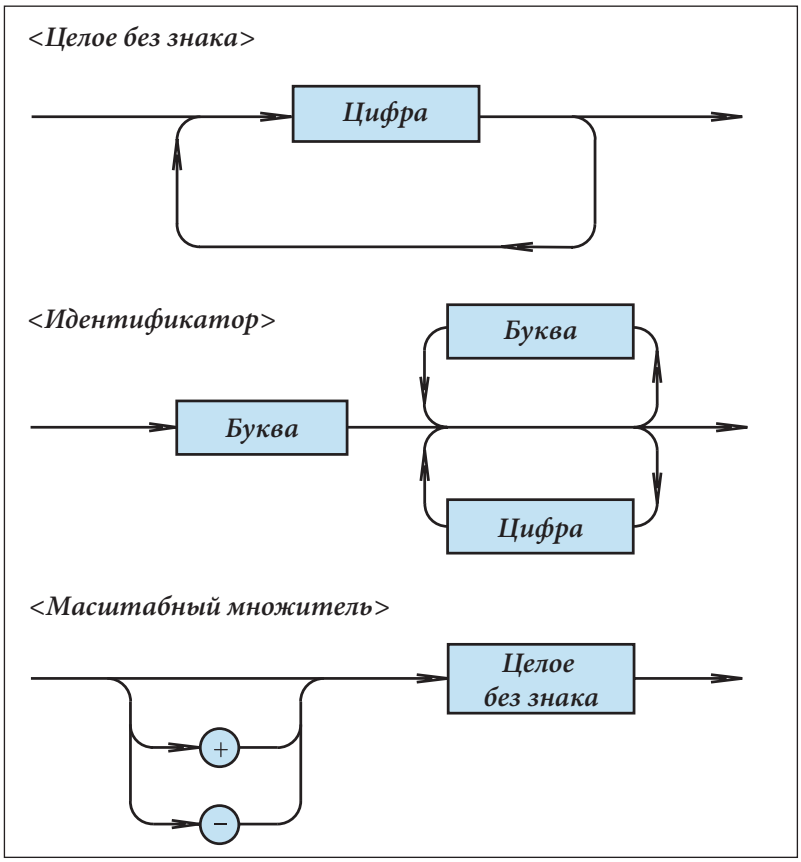


Рис. 1.2. Синтаксические диаграммы <*Целое без знака*>, <*Идентификатор*> и <*Масштабный множитель*>

Вопросы и упражнения

❶ Для чего предназначены синтаксические диаграммы?

❷ Как представляются терминальные и нетерминальные символы в синтаксических диаграммах?

❸ Как представляются формулы БНФ в виде синтаксических диаграмм?

❹ Представьте в виде синтаксических диаграмм:

<*Цифра*> :: = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<*Число*> :: = <*Цифра*> {<*Цифра*>}
<*Знак*> ::= + | –
<*Expresie aritmetică*> ::= <*Число*>{<*Знак*><Число>}

❺ Reprezentați cu ajutorul diagramelor sintactice:

<*Диск*> ::= A: | B: | C: | D: | E:
<*Список параметров* > ::= <*Диск*> {, <*Диск*>}
<*Название команды*> ::= Считывание | Копирование | Форматирование
<*Оператор*> ::= <*Название команды*><*Список параметров*>

❻ На *рис. 1.3* представлены синтаксические диаграммы, которые определяют грамматическую единицу <*Дробное число*>. Определите, какие из нижеприведенных примеров соответствуют данным диаграммам:

a)	0.1	*f)*	.538
b)	+0.1	*g)*	721.386.
c)	–0.0	*h)*	–421
d)	9.000	*i)*	247.532
e)	–538.	*j)*	+109.000

❼ Напишите формулы БНФ, соответствующие синтаксическим диаграммам *рис. 1.3.*

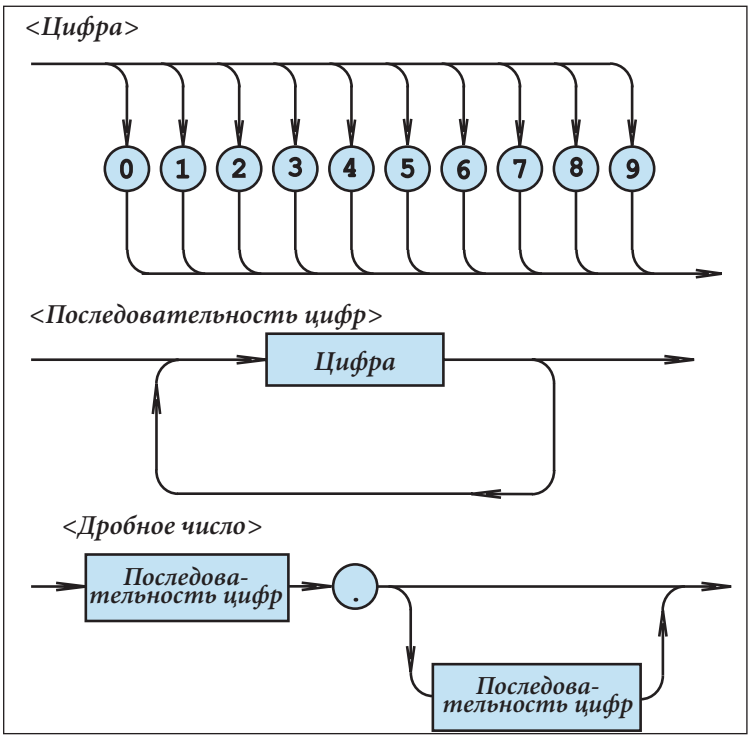


Рис. 1.3. Синтаксические диаграммы <*Цифра*>, <*Последовательность цифр*>, <*Дробное число*>

1.4. Алфавит языка

Алфавит языка ПАСКАЛЬ/ C++ состоит из следующих символов кода *ASCII* (*American Standard Code for Information Interchange*):

- десятичные цифры;
- прописные и строчные буквы латинского алфавита;
- знаки препинания;
- арифметические и логические операторы;
- управляющие символы (пробел, конец строки или возврат каретки и т. д.).

В некоторых конструкциях языка могут использоваться и буквы национальных алфавитов, например, буквы ă, â, î, ş, ţ румынского алфавита, буквы русского алфавита и др.

Язык C++, предназначенный для профессиональных программистов, использует некоторые символы, которые не встречаются в стандартном языке PASCAL, например: _ (*underscore*, подчеркивание, нижняя строка), ~ (тильда), ^ (каретка), \ (*backslash*, обратная косая черта) и др.

1.5. Словарь языка

Самые простые элементы языка программирования, составленные из символов и имеющие лингвистическое значение, называются лексемами или лексическими единицами. Они и составляют **словарь** языка.

Существуют следующие **лексические единицы:**

- специальные символы и ключевые слова;
- идентификаторы;
- числа;
- строки символов;
- метки;
- директивы.

1.5.1. Специальные символы и ключевые слова

Специальные символы состоят из одного или двух символов:

ПАСКАЛЬ			
+	плюс	<	меньше
–	минус	>	больше
*	звездочка	[	левая квадратная скобка
/	наклонная черта	]	правая квадратная скобка
=	равно	(	левая круглая скобка
,	запятая	)	правая круглая скобка
:	двоеточие	;	точка с запятой
.	точка	^	циркумфлекс
@	коммерческое at	$	доллар

{	левая фигурная скобка	<=	меньше либо равно
}	правая фигурная скобка	>=	больше либо равно
#	номер	:=	присваивание
...	многоточие	<>	не равно
(*	эквивалент фигурной скобки {	(.	эквивалент квадратной скобки [
*)	эквивалент фигурной скобки {	.)	эквивалент квадратной скобки [

C++			
+	плюс	<	меньше
–	минус	>	больше
*	звездочка	[	левая квадратная скобка
/	наклонная черта	]	правая квадратная скобка
==	равно	(	левая круглая скобка
,	запятая	)	правая круглая скобка
:	двоеточие	;	точка с запятой
.	точка	^	циркумфлекс
@	коммерческое at	$	доллар
{	левая фигурная скобка	<=	меньше либо равно
}	правая фигурная скобка	>=	больше либо равно
#	номер	=	присваивание
%	процент	!=	не равно
\n	новая строка	\b	*Backspace* (возврат на одну позицию)
\"	кавычки	\'	апостроф

Отметим, что специальные символы как в языке ПАСКАЛЬ, так и в C++, состоящие из двух знаков, например <= или :=, не допускают “вклинивания” промежуточных пробелов.

Ключевые слова состоят из двух или более букв. В нижеследующей таблице приведены примеры ключевых слов:

ПАСКАЛЬ			
and	и	**nil**	нуль
array	массив	**not**	нет
begin	начало	**of**	из
case	выбор	**or**	или
const	константа	**packed**	упакованный

div	целочисленное деление	**procedure**	процедура
do	выполнить	**program**	программа
down to	убывает до	**record**	запись
else	иначе	**repeat**	повторить
end	конец	**set**	множество
file	файл	**then**	тогда
for	для	**to**	до
function	функция	**type**	тип
goto	перейти на	**until**	до тех пор, пока
if	если	**var**	переменная
in	в	**while**	пока
label	метка	**with**	с
mod	остаток от деления	**delete**	удалить

C++			
auto	автоматически	**int**	целый
break	прыжок	**long**	длинный
case	выбор	**near**	возле
char	caractere	**new**	новый
const	константа	**private**	частный
continue	продолжать	**return**	возврат
default	по умолчанию	**short**	короткий
delete	удалить	**signed**	со знаком
do	делать	**sizeof**	размер
double	двойной	**static**	статический
else	иначе	**struct**	структура
enum	перечисление	**switch**	переключатель
extern	внешний	**typedef**	определение типа
float	плавающий	**union**	объединение
for	для	**unsigned**	беззнаковый
goto	перейти к	**void**	пустой
if	если	**volatile**	изменчивый
inline	в линию	**while**	пока

Как в языке ПАСКАЛЬ, так и в C++ ключевые слова являются зарезервированными и не могут использоваться с целью, отличной от той, которая предназначена им по определению языка.

Лексические единицы, рассматриваемые в данном параграфе, могут быть определены следующими формулами БНФ:

<*Специальный символ*> ::= + | – | * | / | = | < | > |] | [| , | (|)|
: | ; | ^ | . | @ | { | } | $ | # | <= | >= |
<> | := | .. |<*Ключевое слово*>|<*Эквивалентный символ*>

<*Эквивалентный символ*> ::= (* | *) | (. | .)

<*Ключевое слово*>::= **and** | **array** | **begin** | **case** | **const** | **div** | **do** | **downto** | **else** | **end** | **file** | **for** | **function** | **goto** | **if** | **in** | **label** | **mod** | **nil** | **not** | **of** | **or** | **packed** | **procedure** | **program** | **record** | **repeat** | **set** | **then** | **to** | **type** | **until** | **var** | **while** | **with**

Символы {, }, [и], используемые в формулах БНФ, являются одновременно и элементами языка ПАСКАЛЬ. Чтобы избежать двусмысленности, данные символы как элементы словаря могут быть представлены через эквивалентные символы (*, *) и соответственно (., .).

<*Специальный символ*> ::= + | – | * | / | = | < | > |] | [| , | (|)
| : | ; | ” | . | & | { | } | $ | # | <= | >= |
<> | != | % |<*Ключевое слово*>

<*Ключевое слово*>::= **auto** | **asm** | **bool** | **break** |**case** | **catch** | **char** | **const** | **continue** | **class** | **default** | **delete** | **do** | **double** | **elsa** | **enum** | **extern** | **float** | **for** | **friend** | **goto** | **if** | **inline** | **int** | **long** | **namespace** | **new** | **operator** | **private** | **public** | **protected** | **plate** | **register** | **return** | **short** | **signed** | **sizeof** | **static** | **struct** | **switch** | **typedef** | **union** | **unsigned** | **using** | **void** | **volatile** | **virtual**| **while**

Важно!

- Символы {, }, [и], используемые в нотации BNF, являются одновременно элементами словаря C++.
- В языке C++ все зарезервированные слова пишутся только строчными буквами.
- В зависимости от используемого компилятора этот может содержать другие специфические ключевые слова.

Вопросы и упражнения

❶ Запомните ключевые слова языка ПАСКАЛЬ/С++.

❷ В чем разница между символами и специальными символами?

❸ Постройте синтаксические диаграммы для лексических единиц: *<Специальный символ>, <Эквивалентный символ>, <Ключевое слово>.*

❹ **ИЗУЧИТЕ САМОСТОЯТЕЛЬНО ИЛИ ВМЕСТЕ С ОДНОКЛАССНИКАМИ!** Найдите в Интернете ответы на следующие вопросы :

- Сколько специальных символов содержит изучаемый язык программирования?
- Сколько ключевых слов содержит изучаемый язык программирования?
- Выберите любой другой язык программирования и определите различия между количеством специальных символов и ключевых слов. Как вы думаете, каким образом их количество влияет на эффективность процессов разработки и выполнения программ?

1.5.2. Идентификаторы

Идентификаторы – это лексические единицы, которые выступают в качестве имен переменных, констант, функций, программ и т. д.

Любой идентификатор начинается с буквы, за которой может следовать любая комбинация из букв и цифр. Длина идентификаторов не ограничена, но только первые 63 символа являются значимыми.

Напомним формулы БНФ, определяющие лексическую единицу *<Идентификатор>*:

<Цифра> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<Буква> ::= `a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z`

<Идентификатор> ::= *<Буква>* { *<Буква>* | *<Цифра>* }

Примеры Идентификаторов:

1)	`x`	*8)*	`listaelevilor`
2)	`y`	*9)*	`listatelefoanelor`
3)	`z`	*10)*	`registru`
4)	`x1`	*11)*	`adresa`
5)	`y10`	*12)*	`adresadomiciliu`
6)	`z01b`	*13)*	`casa10`
7)	`lista`	*14)*	`anul2020`

Использование прописных и строчных букв позволяет записывать идентификаторы в форме, удобной для чтения, например:

1)	`ListaElevilor`	*3)*	`AdresaDomiciliu`
2)	`ListaTelefoanelor`	*4)*	`BugetulAnului2020`

Отметим, что в основных конструкциях языка ПАСКАЛЬ и C++ не используются буквы, присущие русскому алфавиту, и буквы **ă, â, î, ş, ţ** румынского алфавита. Они заменяются на соответствующие буквы латинского алфавита:

Примеры:

1)	`Suprafata`	*4)*	`Patrat`
2)	`Numar`	*5)*	`SirDeCaractere`
3)	`NumarElevi`	*6)*	`NumarIncercari`

ПРИМЕЧАНИЯ

В грамматических конструкциях языка ПАСКАЛЬ, за исключением строк символов, прописные и строчные буквы считаются эквивалентными. Следовательно, в языке ПАСКАЛЬ идентификаторы `ANM`, `AnM`, `Anm`, `aNM` и `Anm` идентичны.

Язык C++ чувствителен к регистру, то есть он различает строчные и прописные буквы. Поэтому в C++ идентификаторы `ANM`, `AnM`, `Anm`, `aNM` и `AnM` различны.

В языке C++ и в некоторых реализациях языка ПАСКАЛЬ идентификаторы могут содержать и даже начинаться с символа «_» (*underscore*, подчеркивание, по́дчерк).

Вопросы и упражнения

❶ Нарисуйте синтаксические диаграммы для грамматических единиц <*Цифра*>, <*Буква*> и <*Идентификатор*>.

❷ **ПРОАНАЛИЗИРУЙТЕ!** Какие из нижеприведенных примеров соответствуют определению лексической единицы <*Идентификатор*>:

a)	`x1`	*h)*	`rădăcina`	*o)*	`abc`
b)	`X1`	*i)*	`radacina`	*p)*	`Dreptunghi`
c)	`1x`	*j)*	`R1`	*q)*	`iI`
d)	`1X`	*k)*	`A1x`	*r)*	`I1j`
e)	`xy`	*l)*	`ListaA`	*s)*	`Luni`
f)	`Suprafata`	*m)*	`Lista1`	*t)*	`Luna`
g)	`SUPRAFATA`	*n)*	`B-1`	*u)*	`20.07.2020`

Для последовательностей символов, соответствующих определению идентификатора, укажите соответствующий путь в синтаксической диаграмме <*Идентификатор*>.

❸ **ПРОАНАЛИЗИРУЙТЕ!** Найдите пары эквивалентных идентификаторов (в языке ПАСКАЛЬ)::

a)	`x101`	*d)*	`radacinaX2`
b)	`ya15`	*e)*	`triunghi`
c)	`radacinaX1`	*f)*	`cerc`

g)	`sirdecaractere`	*n)*	`RegistruClasa10`
h)	`registruclasa10`	*o)*	`zilE`
i)	`COTIDIAN`	*p)*	`X101`
j)	`ZILE`	*q)*	`RegistruClasa10`
k)	`CERCURI`	*r)*	`radaciniX1X2`
l)	`SirDeCaractere`	*s)*	`RADACINAX1`
m)	`Triunghiuri`	*t)*	`yA101`

❹ Для чего предназначены идентификаторы?

❺ **ПРИМЕНИТЕ!** Предложите идентификаторы для хранения данных из нижеследующей таблицы.

Данные	Идентификаторы
Рост человека	
Возраст человека в годах	
Длина радиуса круга	
Периметр прямоугольника	

❻ Для нахождения корней x_1, x_2 квадратного уравнения $ax^2 + bx + c = 0$ сначала находится дискриминант d. Предложите несколько вариантов представления коэффициентов a, b, c, дискриминанта d и решений x_1, x_2 посредством идентификаторов.

1.5.3. Числа

Числа могут быть целыми или вещественными. Обычно используется десятичная система счисления. На *рис. 1.4* показаны синтаксические диаграммы для лексических единиц <*Целое число*> и <*Вещественное число*>.

Примеры целых чисел:

`23`	`–23`	`0023`	`+023`
`318`	`00318`	`+0318`	`–318`
`1996`	`+1996`	`–1996`	`0001996`
`–0023`	`–0318`	`+001996`	`-000199`

В случае вещественных чисел дробная часть отделяется от целой точкой. Перед десятичной точкой должна стоять, по крайней мере, одна десятичная цифра.

Примеры вещественных чисел:

`3.1415`	`+3.04`	`0.0001`	`283.19`
`-3.04`	`-0.0001`	`-256.19`	`28.17`
`+0.0001`	`6.28`	`+3.12421`	`4563906.734`

При записи вещественных чисел можно использовать масштабный множитель. Масштабный множитель представляет собой целое число, перед которым ставится буква `e` (или `E`), и означает, что число, за которым он следует, умножается на 10 в соответствующей степени.

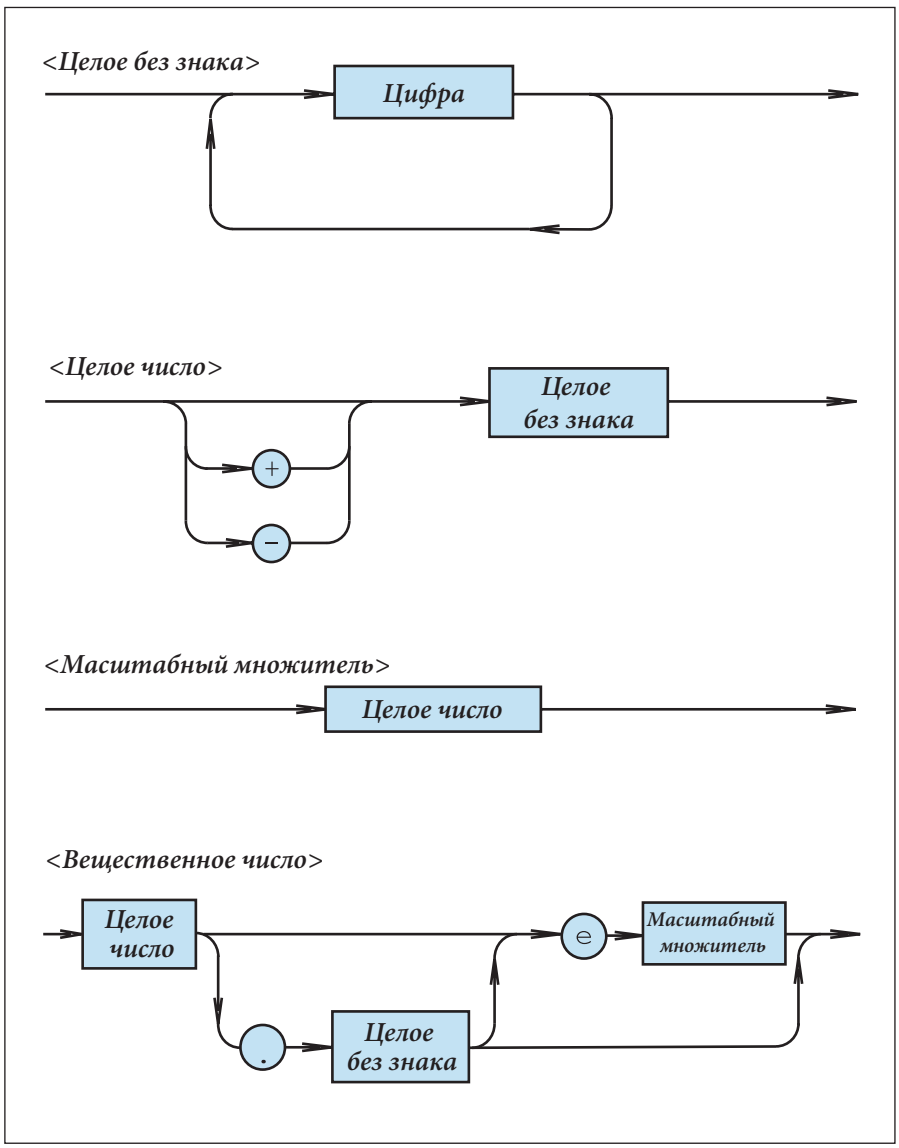


Рис. 1.4. Синтаксические диаграммы ***<Целое число>*** и ***<Вещественное число>***

Примеры:

	Обычная форма записи	*Запись на языке ПАСКАЛЬ/С++*
1)	$8{,}12 \cdot 10^{-5}$	`8.12e-5`
2)	$749{,}512 \cdot 10^{8}$	`749.512e+8`
3)	$-0{,}0823 \cdot 10^{-12}$	`-0.0823e-12`
4)	$3250{,}4 \cdot 10^{6}$	`3250.4e06`
5)	$3{,}421 \cdot 10^{16}$	`3.421e16`

Очевидно, что `8.12e-5`, `812e-7`, `0.812e-4`, `81.2e-6` представляют одно и то же значение $8{,}12 \cdot 10^{-5}$.

ПРИМЕЧАНИЕ

В языке С++ при записи действительных чисел целая либо дробная часть может отсутствовать, однако не одновременно. Например, «.021» и «318.» – это правильно записанные числа.

Вопросы и упражнения

❶ **ПРОАНАЛИЗИРУЙТЕ!** Какие из нижеприведенных примеров соответствуют определению лексической единицы **<***Целое число***>**?

a)	`-418`	*f)*	`0+2469`	*k)*	`32,014`
b)	`0-418`	*g)*	`32,14`	*l)*	`-719`
c)	`621+`	*h)*	`+00621`	*m)*	`+62.1`
d)	`2469`	*i)*	`24693.`	*n)*	`-00418`
e)	`-6210`	*j)*	`-621`	*o)*	`-00621`

Найдите целые числа с одним и тем же значением.

❷ С помощью синтаксических диаграмм *рис. 1.4* напишите формулы БНФ для определения лексической единицы <*Целое число*>.

❸ **ПРОАНАЛИЗИРУЙТЕ И ПРИМЕНИТЕ!** Какие из нижеприведенных примеров соответствуют определению лексической единицы **<***Вещественное число***>**?

a)	`3.14`	*h)*	`281.3`	*o)*	`0,618284e00`
b)	`2.514e+5`	*i)*	`591328`	*p)*	`1961.`
c)	`591328E+3`	*j)*	`2514e+2`	*q)*	`28130E-2`
d)	`.000382`	*k)*	`-464.597e+3`	*r)*	`591.328`
e)	`0.1961E+4`	*l)*	`+519.328e-4`	*s)*	`-658.14e-6`
f)	`+314629.`	*m)*	`591328e-3`	*t)*	`2514e+2`
g)	`0.000314E4`	*n)*	`28130e-2`	*u)*	`618.248e-3`

Найдите вещественные числа, которые имеют одно и то же значение. Запишите данные числа в обычном виде.

❹ С помощью синтаксических диаграмм *рис. 1.4* напишите формулы БНФ для определения лексической единицы <*Вещественное число*>.

❺ **ПРОАНАЛИЗИРУЙТЕ!** Укажите на синтаксических диаграммах *рис. 1.4* пути, которые соответствуют следующим числам:

a)	`-418`	*e)*	`2.514e+5`	*i)*	`+19.511e+2`
b)	`1961.0`	*f)*	`-1951.12`	*j)*	`+0001`
c)	`2514E+2`	*g)*	`32.014`	*k)*	`-614.85e-3`
d)	`281.3`	*h)*	`591.328`	*l)*	`2013e-4`

1.5.4. Строки символов

Pascal

Строка символов представляет собой последовательность печатных символов, заключенных в одиночные кавычки. Если в состав строки символов нужно включить сам символ одиночной кавычки, то этот символ печатается два раза. Отметим, что в строках символов строчные и прописные буквы являются различными символами.

Например:

1) `'Variabila x'`
2) `'Calculul aproximativ'`
3) `'Apostroful '' este dublat'`

В отличие от других лексических единиц языка ПАСКАЛЬ, в строках символов могут использоваться и буквы румынского и русского алфавитов. Для этого необходимо, чтобы на компьютере была установлена программа-драйвер, обеспечивающая ввод, вывод и печать таких букв.

Пример:

1) `'Şir de caractere'`
2) `'Английский язык'`
3) `'Поверхность'`
4) `'Число попыток'`

Лексическая единица <Строка> определяется с помощью следующих формул БНФ:

<*Строка*> :: = '<*Элемент строки*> {<*Элемент строки*>}'

<Элемент строки> ::= '' | <*Любой печатный символ*>

Синтаксическая диаграмма лексической единицы <*Строка*> показана на рис. 1.5.

C++

Строки символов – это последовательность печатных символов, заключенная в кавычки. Односимвольная строка может быть разграничена апострофом. Подчеркнем, что в случае строк символов прописные и строчные буквы отображаются как различные символы.

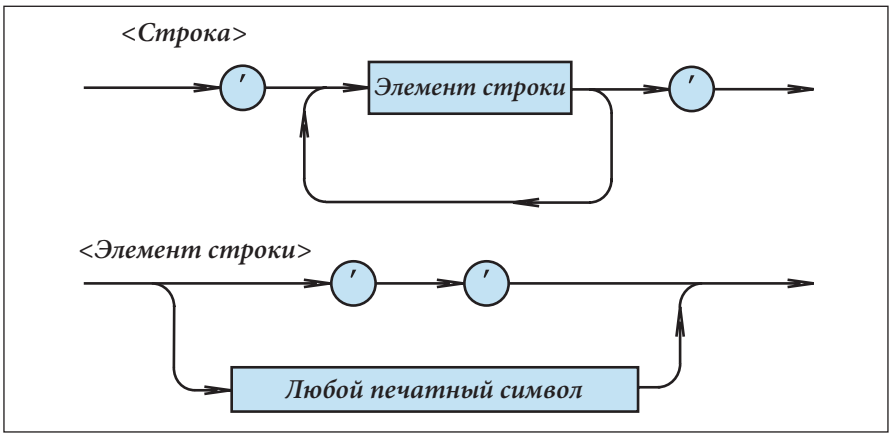


Рис. 1.5. Синтаксическая диаграмма **<*Строка*>**, язык ПАСКАЛЬ

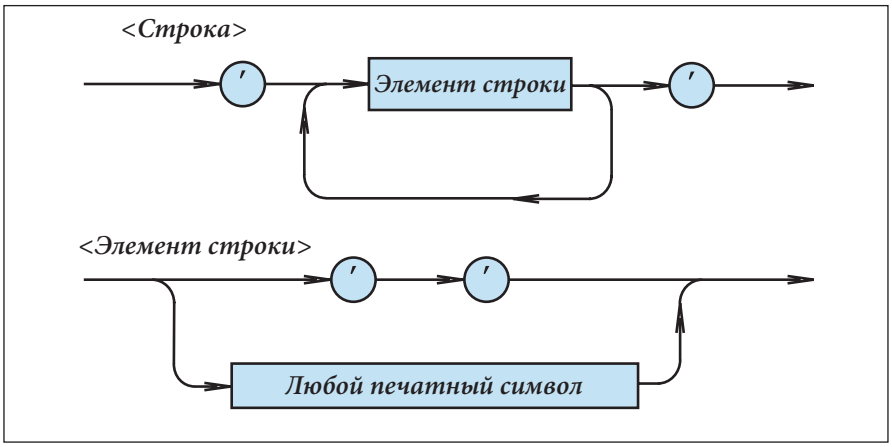


Рис. 1.5.* Синтаксическая диаграмма *<Строка символов>*, язык C++

Примеры:

1) `"Переменная x"`

2) `"Примерный расчет"`

3) `"Апостроф \' "`

4) `'B'`

В языке C++ буквы `ă, â, î, ş, ţ` румынского алфавита также могут использоваться в строках. Для этого необходимо будет использовать специальные библиотеки, обеспечивающие ввод, отображение и печать соответствующих букв.

Лексическая единица *<Строка символов>* определяется с помощью следующих формул BNF:

<Строка символов> ::= "*<Элемент строки>* {*<Элемент строки>*} "

<Элемент строки> ::= " "| *<Любой печатный символ>*

Синтаксическая диаграмма этой лексической единицы представлена на рисунке 1.5*.

Вопросы и упражнения

❶ **ПРОАНАЛИЗИРУЙТЕ!** Укажите на синтаксических диаграммах рис. 1.5 (Язык ПАСКАЛЬ) и соответственно *рис. 1.5** (Язык *C++*) пути, которые соответствуют следующим строкам:

	ПАСКАЛЬ		C++
a)	`'переменная z'`	*a)*	`"переменная z"`
b)	`''''`	*b)*	`"\'"`
c)	`'Символы "x", "y"'`	*c)*	`" Символы 'x', 'y' "`
d)	`'Лексические единицы'`	*d)*	`"ЛЕКСИЧЕСКИЕ ЕДИНИЦЫ"`

❷ **ПРОАНАЛИЗИРУЙТЕ!** Какие из нижеприведенных примеров соответствуют определению лексической единицы <*Строка символов*>:

	ПАСКАЛЬ		C++
a)	`'Целое число'`	*a)*	`"Целое число"`
b)	`'Конец программы'`	*b)*	`"Конец программы"`
c)	`'АПОСТРОФ'`	*c)*	`"АПОСТРОФ"`
d)	`"x"`	*d)*	`"'x'"`
e)	`'функция'`	*e)*	`"функция"`
f)	`'Год 1997'`	*f)*	`"Год 1997"`
g)	`'Квадратный корень'`	*g)*	`"Квадратный корень"`
h)	`'Год '97'`	*h)*	`"Год \'97"`
i)	`'Список телефонов'`	*i)*	`"Список телефонов"`
j)	`'''`	*j)*	`'''`

❸ **ПРОАНАЛИЗИРУЙТЕ!** Исследуйте нижеприведенный пример, запустите программу на выполнение, обратите внимание на полученные результаты.

ПАСКАЛЬ	C++

`Program P;`
`begin`
`  writeln ('Здравствуйте!');`
`  writeln ('Сегодня прекрасный день!');`
`end.`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`  cout<<"Здравствуйте!"<<endl;`
	`  cout<<"Сегодня прекрасный день!";`
	`return 0;`
	`}`

❹ **ПРИМЕНИТЕ!** Измените программу из задания 3 таким образом, чтобы на экран выводились следующие строки символов:

```
Это апостроф '
Мне нравится программировать!
```

1.5.5. Метки

Чтобы перейти к определенному оператору, его необходимо каким-то образом пометить. Делается это с помощью меток.

В языке ПАСКАЛЬ метка представляет собой целое число без знака в диапазоне от `0` до `9999`, которое используется для того, чтобы ссылаться на операторы языка ПАСКАЛЬ.

Примеры:

`1` `100` `999` `582` `1004`

Очевидно, что формула БНФ, которая определяет данную лексическую единицу, имеет вид:

<*Метка*> ::= <*Целое без знака*>

В C ++ метка состоит из идентификатора, за которым следует символ: (двоеточие). Метки имеют собственное пространство имен (один и тот же идентификатор может использоваться для метки и переменной, без пересечения). Метки имеют в качестве области видимости все тело функции, в которой они появляются, и распознаются только оператором перехода `goto`. В любом другом контексте помеченный оператор выполняется независимо от наличия метки.

`Suma1:` `Calcul:` `Afisare:` `Citire:` `Produs:`

Формула БНФ, которая определяет данную лексическую единицу, имеет вид:

<*Метка*> ::= <*Идентификатор*><:>

Пример:

```
#include <iostream>
using namespace std;
int main()
{
        int a;
        cout<<"a="; cin>>a;
        if (a>0) goto pozitiv;
        if (a<0) goto negativ;
        if (a==0) goto zero;
pozitiv:
       cout<<"положительное число";
       goto Final;
```

```
negativ:
        cout<<"отрицательное число";
        goto Final;
zero:
       cout<<"число, равное нулю";
Final:
       cout<<"\nпрограмма закончена";
return 0;
 }
```

1.5.6. Директивы

Директивы – это зарезервированные слова, которые имеют специальное назначение. Лексическая единица <*Директива*> определяется точно так же, как идентификатор:

<*Директива*> ::= <*Буква*> {<*Буква*> | <*Цифра*>}

Стандартный язык содержит одну-единственную директиву:

```
forward
```

Данная директива используется при описании некоторых процедур и функций, определенных пользователем.

Обработка исходного текста программы перед этапом компиляции называется **предварительной обработкой**, а программа, осуществляющая такую обработку, называется **препроцессором.** Обычно препроцессор является частью компилятора.

В языке C++ информация о том, как должен быть обработан текст программы перед ее компиляцией, передается с помощью директив предварительной обработки. Они распознаются препроцессором по наличию символа # (диез).

Включение в текст программы копий других исходных файлов осуществляется с помощью директивы `#include`, имеющей две формы:

`#include` <*имя_файла*>

или

`#include` “*имя_файла*”

Обе формы в качестве эффекта имеют включение файла, указанного в директиве, в файл, который содержит эту директиву. Различие между двумя формами состоит в месте, где выполняется поиск исходного файла. В первом случае файл ищется в стандартных каталогах (заданных параметрами или переменными среды,

в зависимости от компилятора). Во втором случае файл ищется сначала в текущем каталоге, а затем, если он не найден, в стандартных каталогах. Вторая форма также позволяет указать полный путь к подключаемому файлу, что исключает поиск в стандартных каталогах.

Примеры:

`#include <stdio>` – указывает препроцессору включить файл `stdio` из стандартного каталога.

`#include "Matrice"` – указывает препроцессору включить файл `Matrice`, который ищется сначала в текущем каталоге, а затем, если он не найден, в стандартных каталогах.

`#include "C:\ProgrameleMele\Vectori.cpp"` – указывает препроцессору включить файл `Vectori.cpp` из каталога `C:\Programe`. Если файл не найден, он не будет где-либо искаться и генерируется сообщение об ошибке.

В целом язык C++ содержит более десяти директив предварительной обработки. Эти директивы изучаются в углубленных курсах информатики.

1.6. Разделители

Любая программа на языке ПАСКАЛЬ или C++ состоит из лексических единиц и разделителей. В качестве разделителей используются: пробел, конец строки (возврат каретки) и комментарий.

Примеры:

	ПАСКАЛЬ		C++
1)	`x div y`	*1)*	`x/y`
2)	`not x`	*2)*	`!x`
3)	`a or b`	*3)*	`a || b`

4)	`begin`
	`  writeln(x);`
	`  writeln(y);`
	`end`
		4)	`{`
			`cout<<x<<endl;`
			`cout<<y<<endl;`
			`}`

При отсутствии разделителей, при последовательном написании идентификаторов, ключевых слов, чисел без знака и директив, начало новой лексической единицы может быть интерпретировано как продолжение предыдущей.

В частности, в языке ПАСКАЛЬ, в первом примере запись "`x div y`" сообщает компьютеру "раздели переменную `x` на переменную y". При отсутствии же разделительных пробелов запись "`xdivy`" будет воспринята компьютером как идентификатор.

Отметим, что как в языке ПАСКАЛЬ, так и в C++ используются специальные символы, состоящие из двух знаков: <=, >=, и т. д., ключевые слова, идентификаторы, числа

и т. д. являются лексическими единицами программы. Очевидно, что вставка пробелов или символов возврата каретки внутри составных лексических единиц недопустима.

Примеры:

	ПАСКАЛЬ		C++	
	Правильно	*Неправильно*	*Правильно*	*Неправильно*
1)	`CitireDisc`	`Citire Disc`	`CitireDisc`	`Citire Disc`
2)	`Program`	`Pro gram`	`typedef`	`type def`
3)	`:=`	`: =`	`==`	`= =`
4)	`..`	`. .`	`!=`	`! =`
5)	`345`	`3 45`	`345`	`3 45`
6)	`downto`	`down to`	`while`	`whi le`
7)	`begin`	`be gin`	`endl`	`end l`

Комментарии – это тексты, которые могут быть введены в исходную программу, но не рассматриваются компилятором и поэтому не оказывают никакого воздействия на выполнение программы. Комментарии полезны для лучшего понимания программы теми, кто ее читает.

В языке ПАСКАЛЬ комментарии можно использовать в программах в виде последовательности символов, заключенной внутри фигурных скобок { и } или внутри (* и *).

Примеры:

1)
```
{ Ввод исходных данных }
```

2)
```
(* Введенные с клавиатуры числа должны быть меньше
1000 **)
```

В языке C++ комментарии можно использовать в программах в виде последовательности символов, заключенной внутри пар символов /* и */ . Такой комментарий можно писать в нескольких строках.

Если нужно написать короткий комментарий, помещающийся в одной строке, тогда перед ним записываются символы `//`.

Примеры:

1)
```
// Ввод исходных данных
```

2)
```
/* Введенные с клавиатуры числа должны быть меньше
1000 **/
```

Отметим, что рациональное использование комментариев, пробелов и символов возврата каретки делает программы удобными для чтения.

Тест для самопроверки № 1

1. Синтаксис языка программирования исполнителя **Робот** описан с помощью следующих металингвистических формул:

<*Цифра*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<*Число*> ::= <*Цифра*> {<*Цифра*>}

<*Команда*> ::= `вверх | вниз | вправо | влево`

<*Оператор*> ::= <*Команда*>(<*Число*>)

<*Программа*> ::= **начало** {< *Оператор* >;} **конец**

Укажите синтаксически правильные программы:

a) **начало** `вверх(1); вправо(4); вниз(0); влево(00);` **конец**

b) **начало** `вверх(1); вправо(73); вниз(0); влево(00+23);` **конец**

c) **начало** `вниз(30); вправо(45); вверх(980);` **конец**

d) **начало** `влево(21); вниз(50); вправо(45); вверх(980);` **конец**

e) **начало** `влево(3); вниз(13); влево(21)` **конец; конец**

f) **начало** `вниз(73); вправо(5); вверх(71) влево(13);` **конец**

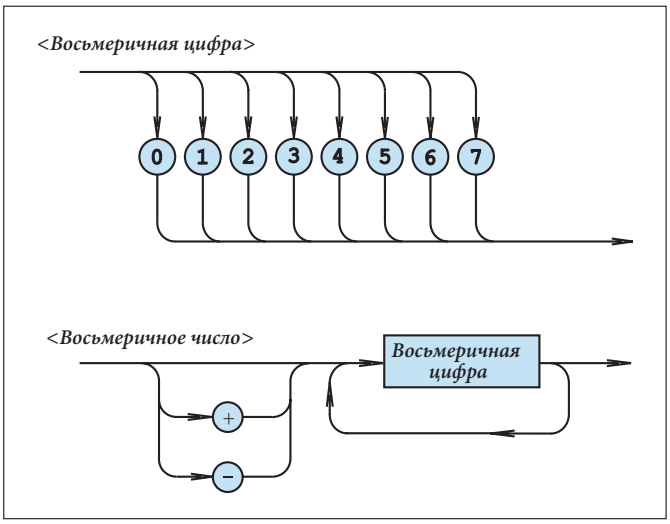
g) **начало** `вверх(1); вправо(-4); вниз(0); влево(10950);` **конец**

2. Нарисуйте синтаксические диаграммы, соответствующие металингвистическим формулам <*Команда*>, <*Оператор*> и <*Программа*> из задания 1.

3. На нижеследующем рисунке представлены синтаксические диаграммы, определяющие грамматическую единицу <*Восьмеричное число*>.

Определите, какие из нижеприведенных последовательностей символов записаны в соответствии с данными диаграммами:

a)	`+0`	*f)*	`-34637`	*k)*	`+123146482351`
b)	`18`	*g)*	`2347-523`	*l)*	`614,45`
c)	`-17250`	*h)*	`-0000007`	*m)*	`–152`
d)	`+6362,1`	*i)*	`527345372`	*n)*	`+35,1`
e)	`717424410571`	*j)*	`614.45`	*o)*	`-412`



4. Запишите металингвистические формулы, соответствующие синтаксическим диаграммам из задания 3.

5. В языках ПАСКАЛЬ и C++ идентификатор начинается с буквы, за которой может следовать любая комбинация из букв и цифр. Напишите металингвистические формулы, определяющие лексическую единицу <Идентификатор>.

6. Придумайте и запишите по крайней мере десять идентификаторов, отражающих специфику физических, математических, химических задач, а также задач по обработке текстов и изображений.

7. Запишите в соответствии с правилами языка ПАСКАЛЬ (или C++) следующие, представленные в обычном виде числа:

a)	3,14;	*f)*	–984,52;	*k)*	$-3628{,}297 \cdot 10^{12}$;
b)	265;	*g)*	–523;	*l)*	–38,00001;
c)	23,4635;	*h)*	+28;	*m)*	$35728{,}345452 \cdot 10^{-8}$;
d)	+0,000001;	*i)*	+28000000;	*n)*	24815;
e)	$6{,}1532 \cdot 10^{-5}$;	*j)*	$614{,}45 \cdot 10^{-12}$;	*o)*	–296,0020001.

8. Запишите в обычном виде следующие числа, представленные в соответствии с правилами языка ПАСКАЛЬ/C++:

a)	`6124.485`	*f)*	`-0.03428e-08`	*k)*	`2005`
b)	`+18.315`	*g)*	`232847.5213`	*l)*	`+23.08e-5`
c)	`-218.034e-3`	*h)*	`-0000012e+2`	*m)*	`–17502`
d)	`193526`	*i)*	`18.45`	*n)*	`+1`
e)	`1000.01e+23`	*j)*	`623.495e-6`	*o)*	`-46341.2e-06`

9. Известно, что словарь языков ПАСКАЛЬ и С++ включает следующие лексические единицы: специальные символы и ключевые слова, идентификаторы, числа, строки символов, метки, директивы. Укажите лексические единицы, присутствующие в нижеприведенной программе:

ПАСКАЛЬ

```
Program TA1;
var a, b, x : real;
begin
  readln(a, b);
   if a<>0 then
     begin
       x:=-b/a;
       writeln('Уравнение имеет один корень');
       writeln(x);
     end;
   if (a=0) and (b=0) then
       writeln('Уравнение имеет бесконечное множество корней');
   if (a=0) and (b<>0) then
       writeln('Уравнение не имеет смысла');
end.
```

С++

```
#include <iostream>
using namespace std;
int main ()
{
float a, b, x;
cin>>a>>b;
if (a!=0)
 {
 x=-b/a;
 cout<<"Уравнение имеет один корень"<<endl;
 cout<<x;
 }
if ((a==0) && (b==0))
 {
  cout<<"Уравнение имеет бесконечное множество корней"<<endl;}
if ((a==0) && (b!=0))
 {
  cout<<"Уравнение не имеет смысла"<<endl;
 }
return 0;
}
```

Пример (ПАСКАЛЬ): **Program** – ключевое слово; TA1 – идентификатор; ; – специальный символ; **var** – ключевое слово и т.д.

Пример (*C++*): #include – директива; < и > – специальные символы; **iostream** – имя файла; **using namespace** std – оператор; ; – специальный символ; **int** – ключевое слово и т.д.

10. (ПАСКАЛЬ) Укажите заголовок, раздел описаний и раздел операторов программы TC1 из задания 9.

(C++) Укажите директивы, декларативную часть и основную функцию программы, представленной в задании 9.

11. Нижеследующая программа содержит ошибки. Выявите ошибки, исправьте их и запустите программу на выполнение.

ПАСКАЛЬ

```
(*Зима-Василе Александри, перевод В. Луговского}

Program P 11;
begin
write ("Пали с неба тучи снега,"),
write ('зимний ветер полон злобы');
write ln;
writeln ('Громоздятся над полями странствующие сугробы. ');
end
```

C++

```
*/ Зима-Василе Александри, перевод В. Луговского //
#include <stream>
int main
{
cout << "Пали с неба тучи снега",
cout >> " зимний ветер полон злобы, ";
cout << " endl;"
cout << 'Громоздятся над полями странствующие сугробы. ' <<
endl;
return 0;
)
```

Глава 2

ПРОСТЫЕ ТИПЫ ДАННЫХ

2.1. Концепция данных

Информация, подлежащая обработке, представлена в компьютере в виде данных. **Данные** состоят из цифр, букв, знаков, чисел, строк и др.

В машинном коде компьютера данные представляются как последовательности двоичных цифр. Например, на языке процессора натуральное число 1039 представляется в двоичной системе счисления как:

```
10000001111
```

Для того чтобы избавить пользователя от деталей внутреннего представления данных, в языках программирования используются различные *типы данных.*

Под **типом данных** понимается **множество значений** и **множество операций,** которые можно к ним применить.

Изучая языки программирования, вы узнаете и будете применять основные предопределенные типы данных (стандартные), а также научитесь создавать свои собственные типы.

Ниже приведены некоторые предопределенные (стандартные) типы данных:

ПАСКАЛЬ	C++
`char`	**`char`**
`integer`	**`int`**
`real`	**`float`**
`boolean`	**`bool`**

Тип данных `integer`/**`int`**. Для эффективного использования памяти и учета потребностей большого числа приложений существует несколько типов целых и вещественных чисел (расширений), которые различаются выделенной памятью и, следовательно, диапазоном значений.

Например, в версиях языков программирования, в которых компилятор поддерживает только двухбайтовые целые числа, т.е. 16 бит (например, Borland C ++, Turbo Pascal 7.0), тип данных `integer`/**`int`** будет включать множество целых чисел:

```
{-32768, -32767, ..., -2, -1, 0, 1, 2, ..., 32767}.
```

Отметим, что есть компиляторы, которые выделяют для представления целых чисел 4 байта, то есть 32 бита (например, DELPHI, OBJFPC, Linux). Аналогично для вещественных типов данных в разных компиляторах выделенное пространство может быть разным, соответственно их диапазоны значений будут разными.

С целыми числами можно осуществлять следующие операции:

Операция	ПАСКАЛЬ	C++
Сложение	+	+
Вычитание	–	–
Умножение	*	*
Остаток от деления	**mod**	%
Целочисленное деление	**div**	/

Тип данных `real/float` включает подмножество множества вещественных чисел, с которыми выполняются операции +, -, *, / (деление) и др.

Как и в случае с целыми числами, объем памяти, выделенной для вещественного значения, зависит от используемого компилятора. В языке программирования ПАСКАЛЬ/C++ объем памяти, занятой значением определенного типа в байтах, можно найти с помощью оператора `sizeof`.

Не все операции с данными одного типа допустимы для данных другого типа.

Например, операции с целыми числами **`div`** и **`mod`** языка ПАСКАЛЬ и % языка C++ недопустимы для типа `real/float`.

В программах данные представляются в виде величин: переменных и констант. Термин «величина» позаимствован из математики и физики, где величины используются для описания природных явлений. В качестве примера приведем некоторые величины, изучаемые на соответствующих уроках, это: масса m, длина l, площадь S, объем V, ускорение свободного падения $g \approx 9{,}8\ m/s^2$, иррациональное число $\pi \approx 3{,}14$ и др.

Переменная – это величина, значение которой может быть изменено в процессе выполнения программы. Любая переменная имеет имя, значение и тип. Имя переменной (например, `m, l, S, V, delta`) используется для ее обозначения в программе. В ходе выполнения программы каждая переменная, в текущий момент времени, имеет либо конкретное значение (например, 105 или -36), либо ее значение не определено.

Множество значений, которые может принимать переменная, и операции, допустимые для выполнения над ней, определяются посредством сопоставления имени переменной с необходимым типом данных.

Общая форма объявления переменной:

ПАСКАЛЬ

`var` <*Идентификаторы*> : <*Тип данных*>;

C++

<*Тип данных*> <*Идентификаторы*>**`;`**

В этих объявлениях <*Тип данных*> может быть любым стандартным или определяемым пользователем типом данных, а <*Идентификаторы*> – список, содержащий как минимум один идентификатор. В случае нескольких идентификаторов они разделяются запятыми.

Пример:

ПАСКАЛЬ

```
var x, y : integer;
    z : real;
```

C++

```
int x, y;
float z;
```

В процессе выполнения программы переменные x и y могут принимать любые целочисленные значения, а переменная z – любые вещественные значения. Множества значений типов данных в вышеприведенных объявлениях будут изучаться в следующих параграфах.

Константа – это величина, значение которой не может быть изменено в процессе выполнения программы. Тип константы объявляется неявно (по умолчанию) через форму ее записи. Например, 10 – это константа типа `integer/int`, а 10.0 – константа типа `real/float`.

Для большей наглядности константы могут иметь символические имена.

Общая форма объявления константы:

ПАСКАЛЬ

const <*Идентификатор*> = <*Значение*>;

C++

const <*Тип данных*> <*Идентификатор*>=<*Значение*>

Примеры:

ПАСКАЛЬ

```
const g = 9;
      pi = 3.14;
```

C++

```
const int g = 9;
const float pi = 3.14;
```

Очевидно, что значения констант g и pi не могут быть изменены в ходе выполнения программы.

Концепция данных, реализованная в языке ПАСКАЛЬ/C++, предполагает следующее:

1) каждая величина (переменная или константа), используемая в программе, должна быть ассоциирована с определенным типом данных;

2) тип переменной определяет множество значений, которые она может принимать, и операции, которые можно применять к данным значениям;

3) существуют предопределенные типы данных, которые считаются известными в любой программе, например, в языке ПАСКАЛЬ: `integer` и `real`, а в C++: **`int, float`**;

4) на основании известных типов данных программист может создавать новые типы, отражающие природу информации, подлежащей обработке.

Вопросы и упражнения

❶ Как представляются данные в машинном коде компьютера? Укажите преимущества и недостатки такого представления.

❷ Как представляются данные в программах на языке ПАСКАЛЬ? Укажите разницу между переменной и константой.

❸ Объясните значение термина *тип данных*. Приведите примеры.

❹ Как указывается тип переменных?

❺ **ОПРЕДЕЛИТЕ!** Установите тип переменных r, s, t, x, y и z из следующего описания:

ПАСКАЛЬ

```
var r, y : integer;
    s, z : real;
    t, x : boolean;
```

C++

```
int r,y;
float s,z;
bool t,x;
```

❻ **ПРИМЕНИТЕ!** Объявите переменные a, b и c как переменные целого типа, а p и q как переменные вещественного типа.

❼ **ОПРЕДЕЛИТЕ!** Установите тип следующих констант:

a)	`-301`	*d)*	`-61.00e+2`	*g)*	`314.0`
b)	`-301.0`	*e)*	`3.14`	*h)*	`0314`
c)	`+6100`	*f)*	`-0.0001`	*i)*	`-0.000672`

❽ **ПРОАНАЛИЗИРУЙТЕ И ПРИМЕНИТЕ!** Проанализируйте предложенную ниже программу. Определите используемые величины и их тип.

ПАСКАЛЬ

```
Program Ex8;
var z: real;
    a, b: integer;
begin
  a:=2;
  b:=17;
  z:=(a+b)*2;
  writeln('z=', z);
end.
```

C++

```
// Программа Ex8
#include <iostream>
using namespace std;
int main()
{
 float z;
 int a,b;
  a=2;
  b=17;
  z=(a+b)*2;
  cout<< "z="<<z<<endl;
  return 0;
}
```

Запустите программу на выполнение. Объясните полученные результаты.

Измените программу таким образом, чтобы она вычисляла площадь прямоугольника со сторонами a и b.

❾ **ИЗУЧИТЕ!**

a) Поищите в Интернете и узнайте, какие стандартные типы данных имеет язык программирования, который вы изучаете. Уточните множество значений каждого стандартного типа данных.

b) Поищите в Интернете и узнайте, какой тип данных в языке программирования, который вы изучаете, назван в честь известного ученого? Почему именно этот тип данных получил такое название?

c) Запустите на выполнение нижеследующую программу. Определите, сколько байтов компилятор выделяет для значений каждого типа данных.

ПАСКАЛЬ

```
Program Ex9c;
var   a: integer;
      b: real;
      c: boolean;
      d: char;
begin
  WriteLn (SizeOf (a));
  WriteLn (SizeOf (integer));
  WriteLn (SizeOf (real));
  WriteLn (SizeOf (char));
  WriteLn (SizeOf (boolean));
end.
```

C++

```
// Программа Ex9c
#include <iostream>
using namespace std;
int a;
float b;
bool c;
char d;
int main()
{
 cout<<sizeof(a)<<endl;
 cout<<sizeof(int)<<endl;
 cout<<sizeof(float)<<endl;
```

```
 cout<<sizeof(char)<<endl;
 cout<<sizeof(bool);
 return 0;
}
```

2.2. Тип данных integer/int

Pascal

Множество значений типа данных integer состоит из элементов подмножества целых чисел, определяемого при реализации. Максимальное значение определяется константой MaxInt, известной любой программе на языке ПАСКАЛЬ. Обычно минимальным значением в изучаемом типе данных является – MaxInt или – (MaxInt+1).

Приведенная ниже программа выводит на экран значение предопределенной константы MaxInt.

```
Program P2;
{ Вывод предопределенной константы MaxInt }
begin
  writeln('MaxInt=', MaxInt);
end.
```

В персональных компьютерах, работающих с версией Turbo PASCAL 7.0, константа MaxInt имеет значение 32767, а множеством значений типа integer является:

{-32768, -32767, ..., -2, -1, 0, 1, 2, ..., 32767}.

К целым значениям можно применить следующие операции: +, –, *, **mod** (остаток от деления), **div** (целая часть от деления) и др. Результаты данных операций можно увидеть с помощью программы P3.

```
Program P3;
 { Операции с данными типа integer }
var x, y, z : integer;
begin
  writeln('Введите целые числа x, y:');
  readln(x, y);
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x mod y; writeln('x mod y=', z);
  z:=x div y; writeln('x div y=', z);
end.
```

Очевидно, что результаты операций +, –, * над целыми числами также должны принадлежать множеству значений типа данных integer. Если программист нарушает

данное правило, появляются ошибки переполнения. Эти ошибки будут обнаружены в процессе компиляции или при выполнении соответствующей программы. Для пояснения приведем примеры программ P4 и P5:

```
Program P4;
 { Ошибка переполнения, обнаруженная в процессе компиляции }
var x : integer;
begin
  x:=MaxInt+1; { Ошибка, x>MaxInt }
  writeln(x);
end.
```

```
Program P5;
 { Ошибка переполнения, обнаруженная в процессе выполнения }
var x, y : integer;
begin
  x:=MaxInt;
  y:=x+1; { Ошибка, y>MaxInt }
  writeln(y);
end.
```

Приоритеты операций +, -, *, **mod**, **div** будут изучены позже (*таблица 3.2*).

В языке C++ множество значений базового типа данных **int** состоит из целых чисел, которые могут быть представлены на хост-компьютере. При этом отметим, что в языке C++ существуют так называемые квалификаторы или модификаторы, которые могут применяться к основным типам: `short, long, signed, unsigned.` Применение этих квалификаторов изменяет область значений соответствующего типа путем изменения количества байтов, необходимых для хранения этих величин и типа величин (со знаком или без знака).

Данные типа `signed` (со знаком) могут быть положительными или отрицательными, а данные типа `unsigned` (без знака) всегда положительными. По умолчанию все типы данных являются со знаком. Квалификатор long расширяет набор значений, а `short` – сокращает. В результате объем памяти, необходимый для хранения этих значений, будет увеличиваться или уменьшаться.

Чтобы указать целые числа разных типов, можно использовать суффиксы модификаторов:

Суффикс	Тип модификатора
u или U	unsigned
l или L	long
ll или LL	long long

Примеры:

```
48          // int
48u         // unsigned int
48l         // long
48ul        // unsigned long
48lu        // unsigned long
```

Максимальное значение типа данных **int** зависит от компилятора и операционной системы хост-компьютера. В случае компиляторов под MS-DOS, которые выделяют не более 2 байтов (16 бит), тип данных **int** имеет диапазон значений [-32768, 32767], а для компиляторов под UNIX (Linux) выделяются 4 байта (32 бита), поэтому **int** имеет диапазон значений [–2147483648, 2147483647].

Максимальное значение типа данных **int** является величиной предопределенной константы INT_MAX, а минимальное значение – величиной предопределенной константы INT_MIN. Эти символьные константы, а также другие, которые имеют в качестве значений нижний и верхний пределы диапазона значений для различных типов данных, определены в библиотеке (header - заголовок) <limits>.

Следующая программа отображает значение константы INT_MAX на экране.

```
// Программа P2
// Отображение константы INT_MAX
#include <iostream>
#include <limits>
using namespace std;
int main()
{
cout<<"INT_MAX="<<INT_MAX<<endl;
return 0;
}
```

Операции, которые можно осуществлять над целыми числами: +, -, *, %, / и др.

Операция % (modulo) имеет смысл, только если оба операнда – целые числа, а ее результат – это остаток от целочисленного деления. Приведем несколько примеров:

```
11 % 3 = 2
30 % 10 = 0
276 % 10 = 6
```

Операция /, примененная к целым числам, будет выполнять целочисленное деление, а результат операции / будет частным от целочисленного деления. Вот некоторые примеры:

```
4 / 2 = 2
18 / 4 = 4
0 / 4 = 0
```

Результаты этих операций можно вывести на экран с помощью следующей программы:

```
// Программа P3
// Операции с данными типа int
#include <iostream>
using namespace std;
int main()
{
int x, y, z;
cout<<"Введите целые числа x, y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<<z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x%y; cout<<"x%y="<<z<<endl;
z=x/y; cout<<"x/y="<<z<<endl;
return 0;
}
```

Очевидно, что результаты операций +, –, * с целочисленными значениями должны принадлежать множеству значений типа данных **int**. Если программист не обращает должного внимания на это правило, и значение, содержащееся в переменной, превышает пределы, налагаемые типом используемых данных, происходит так называемое переполнение (*overflow*), которое может вызвать, казалось бы, необъяснимые ошибки.

Приоритеты операций +, –, *, /, % будут изучены позже.

Отметим, что целые числа могут задаваться в системах счисления по основанию 10, 8 и 16:

- По основанию 10, например: `176, -540`. Могут содержать цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- По основанию 8, например: `015, 062.` Могут содержать цифры: 0, 1, 2, 3, 4, 5, 6, 7 и всегда начинаются с 0.
- По основанию 16, например: `0x15, 0x6f, 0xff`. Могут содержать цифры: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F и всегда начинаются с `0x`.

Вопросы и упражнения

❶ Укажите множество значений целочисленного типа данных. Какие операции можно применять к таким значениям?

❷ Когда появляются ошибки переполнения? Как обнаруживаются такие ошибки?

❸ **ПРОАНАЛИЗИРУЙТЕ!** Запустите на выполнение вышеприведенные программы. Объясните полученные на экране результаты.
Задания a) и b) можно раздать ученикам для индивидуального выполнения либо для работы в парах.

a)

ПАСКАЛЬ

```
Program Ex3a;
{ Операции с данными типа integer }
var x, y, z : integer;
    m : real;
begin
  writeln('Введите целые числа x, y:');
  readln(x, y);
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x mod y;
  writeln('x mod y=', z);
  z:=x div y;
  writeln('x div y=', z);
  end.
```

C++

```
// Программа Ex3a
// Операции с данными типа int
#include <iostream>
using namespace std;

int main()
{
int x, y, z;
float m;
cout<<"Введите целые числа x, y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<<z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x%y; cout<<"x%y="<<z<<endl;
z=x/y; cout<<"x/y="<<z<<endl;
return 0;
}
```

b)

ПАСКАЛЬ

```
Program Ex3b;
{ Программа с ошибкой }
var x, y, z : integer;
    m : real;
begin
  writeln('Введите целые числа x, y:');
  readln(x, y);
  writeln('Введите вещественное число m:');
  readln (m);
  z:=m mod y;
  writeln('m mod y=', z);
  z:=m div y;
  writeln('m div y=', z);
end.
```

C++

```
// Программа Ex3b
// Программа с ошибкой
#include <iostream>
using namespace std;
int main()
{
int x, y, z;
float m;
cout<<"Введите целые числа x, y:";
cin>>x>>y;
cout<<"Введите вещественное число m"; cin>>m;
z=m/y; cout<<"m/y="<<z<<endl;
z=m%y; cout<<"m%y="<<z<<endl;
return 0;
}
```

❹ **ИЗУЧИТЕ!** Работу можно организовать в группах, каждая из которых получит одно из нижеследующих заданий.

1) Напишите программу, которая выводит на экран значения констант:

ПАСКАЛЬ: `MaxInt, MaxLongInt`

C++: `INT_MAX, SHRT_MAX, UINT_MAX, LONG_MAX`

Заполните таблицу:

Константа	Значение константы
MaxInt	
...	

2) Изучив специальную литературу в Интернете, а также используя Sizeof, определите объем памяти, выделяемый для значений целочисленных типов и их расширений:

ПАСКАЛЬ: Byte, Word, Integer, ShortInt, LongInt.
C++: **int**, **unsigned int**, **long int**, **unsigned long int**, **short int**, **unsigned short int**.

Заполните таблицу:

Имя типа	Объем выделяемой памяти в битах	Область значений
Byte		
...		

❺ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ!** Даны программы:

ПАСКАЛЬ

```
Program P6;
 { Ошибка переполнения }
var x : integer;
begin
  x:=-2*MaxInt;
  writeln(x);
end.
```

C++

```
// Программа P6
// Ошибка переполнения
#include <iostream>
#include <limits>
using namespace std;
int main()
{
int x;
x=-2*INT_MAX;
cout<<x;
return 0;
}
```

ПАСКАЛЬ

```
Program P7;
 { Ошибка переполнения }
var x, y : integer;
begin
  x:=-MaxInt;
  y:=x-10;
  writeln(y);
end.
```

C++

```
// Программа P7
// Ошибка переполнения
#include <iostream>
#include <limits>
using namespace std;
int main()
{
int x,y;
x=-INT_MAX;
y=x-10;
cout<<y;
return 0;
}
```

Запустите программы на выполнение. Проанализируйте ошибки переполнения. Когда появляются сообщения об ошибках переполнения: в процессе компиляции или во время выполнения программы?
Приведите примеры значений переменных x и y, для которых ошибки переполнения не появляются.

2.3. Тип данных real/float

Множество значений изучаемого типа данных состоит из элементов вещественных чисел, которые могут быть представлены на хост-компьютере языка программирования.

В нижеследующих таблицах представлены наиболее часто применяемые типы вещественных данных и их расширений:

ПАСКАЛЬ			
Имя типа	**Байты**	**Область значений**	**Точность**
single	4	$-3{,}4\cdot10^{38}$ ... $3{,}4\cdot10^{38}$	6 десятичных цифр
real	6	$-1{,}7\cdot10^{38}$ ... $+1{,}7\cdot10^{38}$	11-12 десятичных цифр
double	8	$-1{,}7\cdot10^{308}$ ... $1{,}7\cdot10^{308}$	15-16 десятичных цифр
extended	10	$-3{,}4\cdot10^{4932}$ ... $3{,}4\cdot10^{4932}$	18-19 десятичных цифр

C++			
Имя типа	**Байты**	**Область значений**	**Точность**
float	4	$-3{,}4\cdot10^{38}$ ... $3{,}4\cdot10^{38}$	6-7 десятичных цифр
double	8	$-1{,}7\cdot10^{308}$ ... $+1{,}7\cdot10^{308}$	15-16 десятичных цифр
long double	10	$-3{,}4\cdot10^{4932}$ ... $3{,}4\cdot10^{4932}$	18-19 десятичных цифр

Напомним, что объем памяти, занимаемой значением определенного типа (т.е. на скольких байтах хранится значение), можно узнать с помощью оператора sizeof.

ПРИМЕЧАНИЕ

Как и в случае с целыми типами данных, объем памяти, выделяемой для хранения вещественных данных, зависит от компилятора и операционной системы хост-компьютера. Очевидно, что области значений этих типов данных также будут разными.

В следующей программе переменным x, y и z присваиваются соответственно значения 1.1, $-6.14\cdot10^{8}$ и $90.3\cdot10^{-29}$, которые затем выводятся на экран.

ПАСКАЛЬ

```
Program P8;
 { Данные типа real }
var x, y, z : real;
begin
  x:=1.1;
  y:=-6.14e8;
  z:=90.3e-29;
  writeln('x=', x);
```

C++

```
// Программа P8
// Данные типа float
#include <iostream>
using namespace std;
int main()
{
float x,y,z;
x=1.1;
```

```
  writeln('y=', y);
  writeln('z=', z);
end.
```

```
y=-6.14e8;
z=90.3e-29;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
cout<<"z="<<z;
return 0;
}
```

Напомним, что при записи вещественных чисел десятичная запятая заменяется точкой, а степень числа 10 представляется масштабным множителем.

Операции, которые можно применять к значениям вещественного типа: +, –, *,/ (деление) и др.

Операции над вещественными числами в общем случае являются приближенными из-за погрешностей округления. Естественно, результаты данных операций также должны принадлежать множеству значений вещественных типов данных. В противном случае возникают ошибки переполнения.

Свойства операций +, –, * и / могут быть изучены с помощью следующей программы:

ПАСКАЛЬ

```
Program P9;
 {Операции над данными типа real}
var x, y, z : real;
begin
 writeln('Введите вещественные числа x, y:');
  readln(x,y);
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x/y; writeln('x/y=', z);
end.
```

C++

```
// Программа P9
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, z;
cout<<"Введите вещественные числа x и y:";
cin>>x>>y;
cout<<"x="<<x<<endl;
cout<<"y="<<y<<endl;
cout<<scientific<<
      setprecision(10);
z=x+y; cout<<"x+y="<<z<<endl;
z=x-y; cout<<"x-y="<< z<<endl;
z=x*y; cout<<"x*y="<<z<<endl;
z=x/y; cout<<"x/y="<<z;
return 0;
}
```

В *таблице 2.1* представлены данные, которые выводятся на экран этими программами для некоторых значений переменных *x* и *y*. Отметим, что результаты операций *x+y* и *x–y* в первых двух строках *таблицы 2.1* являются точными. Для значений $x = 1{,}0$, $y = 1{,}0 \cdot 10^{-11}$ (третья строка таблицы) результат сложения – приближенный, а вычитания – точный. Оба результата из строки 4 являются приближенными. Для значений $x = y = 1{,}7 \cdot 10^{38}$ (строка 5), (версия Turbo PASCAL 7.0) при выполнении операции сложения имеет место переполнение. Для значений $x = 3{,}1 \cdot 10^{-39}$, $y = 3{,}0 \cdot 10^{-39}$ (строка 6) результат сложения – точный, а вычитания – приближенный.

Таблица 2.1

Результаты работы программы P9

№	x	y	Результаты, отображаемые для	
			$x + y$	$x - y$
1.	1,0	1,0	2.0000000000E+00	0.0000000000E+00
2.	1,0	$1{,}0 \cdot 10^{-10}$	1.0000000000E+00	9.9999999999E-01
3.	1,0	$1{,}0 \cdot 10^{-11}$	1.0000000000E+00	9.9999999999E-01
4.	1,0	$1{,}0 \cdot 10^{-12}$	1.0000000000E+00	1.0000000000E+00
5.	$1{,}7 \cdot 10^{38}$	$1{,}7 \cdot 10^{38}$	ПАСКАЛЬ: переполнение C++ :3.4000000000e+39	0.0000000000E+00
6.	$3{,}1 \cdot 10^{-39}$	$3{,}0 \cdot 10^{-39}$	6.1000000000E-39	0.0000000000E+00

Ошибки вычисления, характерные для типа данных `real/ float` и их расширений, могут нарушить ход выполнения программы. Оценка ошибок и, если необходимо, их устранение является задачей программиста.

Приоритеты операций +, -, *, / будут изучены в Главе 3.

Вопросы и упражнения

❶ Как записываются вещественные числа на языке ПАСКАЛЬ/C++?

❷ Определите множество значений типа данных real в версии языка ПАСКАЛЬ и соответственно данных типа double в версии языка C++, в которой вы работаете. Какова точность соответствующих чисел?

❸ Какие операции можно применять к данным типа `real` в языке ПАСКАЛЬ и соответственно данных типа double в языке C++? Являются ли эти операции точными?

❹ **ПРИМЕНИТЕ И ОБРАТИТЕ ВНИМАНИЕ!** Напишите и запустите на выполнение программу для нахождения суммы и разности для следующих значений переменных `x, y`:

a) $x = 2{,}0$; $y = -3{,}0$;
b) $x = 14{,}3 \cdot 10^{2}$; $y = 15{,}3 \cdot 10^{-3}$;
c) $x = 3{,}0$; $y = 2{,}0 \cdot 10^{12}$;
d) $x = 3{,}0$; $y = 2{,}0 \cdot 10^{-12}$;
e) $x = 2{,}9 \cdot 10^{-39}$; $y = 6{,}4 \cdot 10^{-3}$;
f) $x = 7{,}51 \cdot 10^{21}$; $y = -8{,}64 \cdot 10^{17}$;
g) $x = 1{,}0$; $y = 2{,}9 \cdot 10^{-39}$;
h) $x = 1{,}7 \cdot 10^{38}$; $y = 2{,}9 \cdot 10^{-39}$.

Проверьте результаты соответствующих операций. Объясните сообщения, выдаваемые на экран.

❺ Каковы причины ошибок вычисления при работе с данными типа `real / float`?

2.4. Тип данных `boolean/bool`

Значениями типа данных `boolean/bool` (логический) являются истинностные значения `false` (ложь) и `true` (истина). В нижеприведенной программе переменной x последовательно присваиваются значения `false` и `true`, выводимые впоследствии на экран.

ПАСКАЛЬ	C++

`Program P10;`
`{ Данные типа boolean }`
`var x : boolean;`
`begin`
`x:=false;`
`writeln(x);`
`x:=true;`
`writeln(x);`
`end.`
	`// Программа P10`
	`#include <iostream>`
	`using namespace std;`
	`// Данные типа boolean`
	`int main()`
	`{`
	`bool x;`
	`x=false;`
	`cout<<x<<endl;`
	`x=true;`
	`cout<<x;`
	`return 0;`
	`}`

Логические операции, которые можно применять к данным логического типа:

Операция	Обозначение	
	ПАСКАЛЬ	C++
Отрицание (логическая инверсия, логическая операция *НЕ*);	`not`	!
Конъюнкция (логическое произведение, логическая операция *И*)	`and`	&&
Дизъюнкция (логическая сумма, логическая операция *ИЛИ*)	`or`	||

Таблицы истинности данных операций представлены на *рис. 2.1.*

x	not x !x
false	true
true	false

x	y	x and y x&&y
false	false	false
false	true	false
true	false	false
true	true	true

x	y	x or y x||y
false	false	false
false	true	true
true	false	true
true	true	true

Рис. 2.1. Таблицы истинности логических операций отрицание, конъюнкция и дизъюнкция

Свойства логических операций отрицание, конъюнкция и дизъюнкция могут быть изучены с помощью следующих программ.:

ПАСКАЛЬ

```
Program P11;
{Операции с данными типа bo-
olean }
var x, y, z : boolean;

begin
  x:=false; y:=false;
   writeln('x=', x, 'y=', y);
  z:=not x;
  writeln('not x = ', z);
  z:=x and y;
  writeln('x and y = ', z);
  z:=x or y;
  writeln('x or y = ', z);
  writeln;
  x:=false; y:=true;
   writeln('x=', x, 'y=', y);
  z:=not x;
  writeln('not x = ', z);
  z:=x and y;
  writeln('x and y = ', z);
  z:=x or y;
  writeln('x or y = ', z);
  writeln;
  x:=true; y:=false;
   writeln('x=', x, 'y=', y);
  z:=not x;
  writeln('not x = ', z);
  z:=x and y;
  writeln('x and y = ', z);
  z:=x or y;
  writeln('x or y = ', z);
  writeln;
  x:=true; y:=true;
   writeln('x=', x, 'y=', y);
  z:=not x;
  writeln('not x = ', z);
  z:=x and y;
  writeln('x and y = ', z);
  z:=x or y;
  writeln('x or y = ', z);
  writeln;
end.
```

C++

```
// Программа P11
#include <iostream>
/* Операции с данными типа
   boolean */
using namespace std;
int main()
{
bool x, y, z;
x=false; y=false;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=false; y=true;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=true; y=false;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
cout<<endl;
x=true; y=true;
cout<<"x="<<x<<" y="<<y<<endl;
z=!x;
cout<<"!x="<<z<<endl;
z=x && y;
cout<<"x && y="<<z<<endl;
z=x || y;
cout<<"x || y="<<z<<endl;
return 0;
}
```

ПРИМЕЧАНИЯ

В языке **Паскаль**, в отличие от целочисленных или вещественных переменных, текущие значения логических переменных не могут быть прочитаны с клавиатуры с помощью стандартных процедур чтения. По этой причине в представленной выше программе текущие значения переменных x и y задаются путем присваивания.

В языке **C++** логические, символьные, целочисленные, вещественные и перечисляемые типы являются арифметическими типами, поскольку их значения можно интерпретировать как целые числа. Таким образом, в программе на C++ текущие значения логических переменных могут быть прочитаны с клавиатуры с помощью стандартных процедур чтения только как целочисленные значения, т.е. 0 вместо `false` и 1 вместо `true`. Кроме того, если логическое значение присваивается целочисленной переменной, `true` становится 1, а `false` становится 0. Если целочисленное значение присваивается логической переменной, 0 становится `false`, а любое ненулевое значение становится `true`.

Приоритеты операций отрицание, конъюнкция и дизъюнкция будут изучены в Главе 3.

Вопросы и упражнения

❶ Назовите множество значений данных логического типа и операции, применимые к ним.

❷ Выучите таблицы истинности логических операций.

❸ **ПРИМЕНИТЕ!** Напишите:

a) программу, которая выводит на экран таблицу истинности логической операции отрицание;

b) программу, которая вычисляет значения логической функции z = x & y для всех возможных значений аргументов x, y;

c) программу, которая выводит на экран значения логической функции $z = x \vee y$.

❹ **ОБРАТИТЕ ВНИМАНИЕ!** Запустите на выполнение программы, представленные в качестве примеров в этом параграфе, и посмотрите, как отображаются на экране значения логического типа данных. Какое значение ассоциируется со значением `true` и какое – со значением `false`

2.5. Тип данных char

Множеством значений данного типа является конечное упорядоченное множество символов. Значения рассматриваемого типа обозначаются символом, заключенным в одиночные кавычки (апострофы), например: `'A'`, `'B'`, `'C'` и т.д.

ПРИМЕЧАНИЯ

Напомним, что для того чтобы вывести на экран сам символ апострофа, используйте:
В ПАСКАЛЕ: дублирование апострофа, представляя его так: `''''`.
В C++: символ \ (*backslash*), за которым следует апостроф, представляя его так: `'\''`.

В следующей программе переменной x типа `char` последовательно присваиваются значения `'A'`, `'+'` и символ `'` (апостроф), выводимые на экран.

ПАСКАЛЬ	C++

`Program P12;`
`{ Данные типа char }`
`var x : char;`
	`// Программа P12`
	`#include <iostream>`
	`using namespace std;`

```
begin
  x:='A';
  writeln(x);
  x:='+';
  writeln(x);
  x:='''';
  writeln(x);
end.
```

```
// Данные типа char
int main()
{
  char x;
  x='A';
  cout<<x<<endl;
  x='+';
  cout<<x<<endl;
  x='\'';
  cout<<x<<endl;
return 0;
}
```

Текущие значения переменной типа `char` могут считываться с клавиатуры с помощью стандартных процедур чтения. Для пояснения представляем программы, которые считывают с клавиатуры и выводят на экран значения типа `char`.

ПАСКАЛЬ

```
Program P13;
  { Чтение и вывод на экран
символов }
var x : char;
begin
  readln(x); writeln(x);
  readln(x); writeln(x);
  readln(x); writeln(x);
end.
```

C++

```
// Программа P13
#include <iostream>
using namespace std;
/* Чтение и вывод на экран
символов */
int main()
{
  char x;
  cin>>x;  cout<<x<<endl;
  cin>>x;  cout<<x<<endl;
  cin>>x;  cout<<x<<endl;
  return 0;
}
```

Соответствующие символы вводятся с клавиатуры и выводятся на экран без апострофов, которые необходимы лишь для включения символьных значений в текст программы.

Как правило, символы языка программирования упорядочены согласно таблице кодов *ASCII* (см. параграф 1.4).

Ниже представляем вывод порядкового номера символа из множества значений типа `char`.

	ПАСКАЛЬ		**C++**
1)	`ord('A')= 65`	*1)*	`int('A')=65`
2)	`ord('B')= 66`	*2)*	`int('B')=66`
3)	`ord('C')= 67`	*3)*	`int('C')=67`

Следующие программы выводят на экран порядковые номера любых четырех символов, считываемых с клавиатуры.

ПАСКАЛЬ

```
Program P14;
 { Изучение функции ord }
var x : char;     { символ }
    i : integer; {порядковый
номер }
begin
  readln(x); i:=ord(x);
  writeln(i);
  readln(x); i:=ord(x);
  writeln(i);
  readln(x); i:=ord(x);
  writeln(i);
  readln(x); i:=ord(x);
  writeln(i);
end.
```

C++

```
// Программа P14
#include <iostream>
using namespace std;
int main()
{
char x;
int i;
cin>>x; i=int(x);
cout<<i<<endl;
cin>>x; i=int(x);
cout<<i<<endl;
cin>>x; i=int(x);
cout<<i<<endl;
cin>>x; i=int(x);
cout<<i;
return 0;
}
```

В языке ПАСКАЛЬ стандартная функция `chr` возвращает символ, который соответствует указанному порядковому номеру, а в C++ для этой цели используется преобразование типа `char`.

Примеры

ПАСКАЛЬ		C++	
1)	`chr(65)='A'`	*1)*	`char(65)='A'`
2)	`ord(66)='B'`	*2)*	`char(66)='B'`
3)	`ord(67)='C'`	*3)*	`char(67)='C'`

Следующие программы выводят на экран символы, которые соответствуют порядковым номерам, считанным с клавиатуры.

ПАСКАЛЬ

```
Program P15;
 { Изучение функции chr }
var i : integer;{порядковый
номер }
    x : char;    { символ }
begin
  readln(i); x:=chr(i);
  writeln(x);
  readln(i); x:=chr(i);
  writeln(x);
  readln(i); x:=chr(i);
  writeln(x);
  readln(i); x:=chr(i);
  writeln(x);
end.
```

C++

```
// Программа P15
#include <iostream>
using namespace std;
int main()
{
  int i;
  char x;
  cin>>i; x=char(i);
  cout<<x<<endl;
  cin>>i; x=char(i);
  cout<<x<<endl;
  cin>>i; x=char(i);
  cout<<x<<endl;
  cin>>i; x=char(i);
  cout<<x<<endl;
  return 0;
}
```

Напоминаем, что расширенное множество *ASCII* включает 256 пронумерованных символов: 0, 1, 2, …, 255.

Тип данных `char` используется для создания более сложных структур данных, в частности строк символов.

Вопросы и упражнения

❶ Укажите множество значений типа данных `char`?

❷ Как упорядочено множество значений типа `char`?

❸ **ПРИМЕНИТЕ!** Определите порядковые номера следующих символов:
- десятичных цифр;
- прописных букв латинского алфавита;
- знаков препинания;
- знаков арифметических и логических операций;
- управляющих символов;
- букв русского алфавита (если они установлены на вашем компьютере).

❹ **ПРИМЕНИТЕ!** Определите символы, которые соответствуют следующим порядковым номерам:

77	109	79	111	42	56	91	123

❺ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** Напишите программу, которая выводит на экран множество всех символов компьютера, на котором вы работаете.

2.6. Перечисляемые типы данных

Типы данных, изученные в параграфах 2.2, 2.3, 2.4 и 2.4, являются предопределенными типами данных, известными любой программе на языке ПАСКАЛЬ или C++. Кроме предопределенных типов данных существуют типы, создаваемые самим программистом, в частности *перечисляемые* типы.

Перечисляемый тип данных содержит упорядоченное множество значений, определяемых посредством идентификаторов.

В языке ПАСКАЛЬ перечисляемый тип данных и множество его значений определяются с помощью ключевого слова **`type`** (тип), а в языке C++ – с помощью ключевого слова **`enum`**.

Примеры:

ПАСКАЛЬ	C++

`type`
`Culoare = (Galben, Verde,`
`  Albastru, Violet);`
`Studii = (Elementare, Medii,`
`  Superioare);`
`Raspuns = (Nu, Da);`
	`enum Culoare {Galben, Verde,`
	`  Albastru, Violet};`
	`enum Studii {Elementare=1,`
	`  Medii, Superioare};`
	`enum Raspuns {Nu, Da};`

Первый идентификатор из списка элементов является наименьшим значением, и его порядковый номер равен нулю. Второй идентификатор имеет порядковый номер один, третий – два и т.д. В языке C++ порядковый номер идентификатора в списке перечисления может быть изменен, указав его начальное значение. Это использовалось для определения перечисляемого типа `Studii`. Таким образом, порядковый номер

идентификатора `Elementare` будет 1, порядковый номер идентификатора `Medii` будет 2, а порядковый номер идентификатора `Superioare` будет 3.

Проанализируйте следующие примеры, чтобы увидеть, как можно определить порядковый номер значения объявленного типа перечисления.

Примеры:

ПАСКАЛЬ	
1)	`ord(Galben)= 0`
2)	`ord(Verde)= 1`
3)	`ord(Albastru)= 2`
4)	`ord(Violet)= 3`
5)	`ord(Elementare)= 1`
6)	`ord(Medii)= 2`

C++		
1)	`cout<<Galben;`	`// 0`
2)	`cout<<Verde;`	`// 1`
3)	`cout<<Albastru;`	`// 2`
4)	`cout<<Violet;`	`// 3`
5)	`cout<<Elementare;`	`// 1`
6)	`cout<<Medii;`	`// 2`

ПРИМЕЧАНИЕ

В языке C++ перечисляемый тип является целочисленным, а идентификаторы перечисляемого типа могут использоваться как целочисленные переменные.

Следующая программа выводит на экран порядковые номера значений типов данных `Studii` и `Raspuns`.

ПАСКАЛЬ

```
Program P16;
 { Тип данных Studii и Raspuns}
type Studii = (Elementare=1,
Medii, Superioare);
Raspuns = (Nu, Da);
var i : integer; {numar de or-
dine}
begin
    i:=ord(Elementare); wri-
teln(i);
  i:=ord(Medii); writeln(i);
    i:=ord(Superioare); wri-
teln(i);
  i:=ord(Nu); writeln(i);
  i:=ord(Da); writeln(i);
end.
```

C++

```
// Программа P16
/*Тип данных Studii и Raspuns
*/
#include <iostream>
using namespace std;
int main()
{
  enum Studii {Elementare=1,
Medii, Superioare};
 enum Raspuns {Nu, Da};
 int i;
 i=Elementare; cout<<i<<endl;
 i=Medii; cout<<i<<endl;
 i=Superioare; cout<<i<<endl;
 i=Nu; cout<<i<<endl;
 i=Da; cout<<i<<endl;
 return 0;
 }
```

Переменные *перечисляемого* типа могут принимать только значения перечисляемых элементов из типа данных, к которому они относятся.

В следующей программе переменная x принимает значение `Albastru`, а переменная y принимает значение `Nu`. Порядковые номера этих значений выводятся на экран.

ПАСКАЛЬ	C++

Program P17;
  { Переменные перечисляемого типа }
type Culoare = (Galben, Verde, Albastru, Violet);
  Raspuns = (Nu, Da);
var x : Culoare; {variabila de tip Culoare}
  y : Raspuns; {variabila de tip Raspuns}
  i : integer; {numar de ordine}
begin
  x:=Albastru;
  i:=ord(x); writeln(i);
  y:=Nu; i:=ord(y); writeln(i);
end.
	// Программа P17
	// Переменные перечисляемого типа
	#include <iostream>
	using namespace std;
	int main()
	{
	enum Culoare {Galben, Verde, Albastru, Violet};
	enum Raspuns {Nu, Da};
	Culoare x; /*переменная типа Culoare */
	Raspuns y; /*переменная типа Raspuns */
	int i; /*порядковый номер*/
	x=Albastru;
	i=x; cout<<i<<endl;
	y=Nu;
	i=y; cout<<i<<endl;
	return 0;
	}

В случаях, когда в одной программе описываются несколько типов данных, соответствующие списки элементов не должны содержать одинаковые идентификаторы.

Например, объявления

ПАСКАЛЬ	C++
type Studii = (Elementare, Medii, Superioare); Grade = (Inferioare, Superioare)	**enum** Studii {Elementare, Medii, Superioare}; **enum** Grade {Inferioare, Superioare}

являются неправильными, так как идентификатор `Superioare` появляется в обоих списках.

Текущие значения переменных перечисляемых типов не могут считываться с клавиатуры и выводиться на экран с помощью стандартных процедур чтения и записи. Однако использование таких типов данных позволяет создавать простые и эффективные программы, удобные для чтения.

Вопросы и упражнения

❶ Как определяется *перечисляемый* тип данных? Укажите множество значений *перечисляемого* типа данных.

❷ Существен ли порядок идентификаторов в списке элементов перечисляемого типа данных?

❸ **ПРИМЕНИТЕ!** Напишите программу, которая выводит на экран порядковые номера значений следующих типов данных:

	ПАСКАЛЬ		C++
1)	`type Continente = (Europa, Asia, Africa, AmericaDeNord, AmericaDeSud, Australia, Antarctida);`	1)	`enum Continente {Europa, Asia, Africa, AmericaDeNord, AmericaDeSud, Australia, Antarctida};`
2)	`type Sex = (Masculin, Feminin);`	2)	`enum Sex {Masculin, Feminin};`
3)	`type PuncteCardinale = (Nord, Sud, Est, Vest);`	3)	`enum PuncteCardinale {Nord, Sud, Est, Vest};`
4)	`type Etaje = (Unu, Doi, Trei, Patru, Cinci);`	4)	`enum Etaje {Unu, Doi, Trei, Patru, Cinci};`

❹ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ!** Запустите на выполнение нижеследующую программу. Что эта программа отображает на экране? Подумайте, почему отображаются именно такие результаты. Назовите тип каждой переменной, объявленной в программе.

ПАСКАЛЬ	C++

`Program P18;`
`type Litere = (A, B, C, D, E, F, G);`
`var x : Litere; y : char; i : integer;`
`begin`
`  x:=A; i:=ord(x); writeln(i);`
`  y:='A'; i:=ord(y); writeln(i);`
`end.`
	`// Программа P18`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Litere {A, B, C, D, E, F, G};`
	`Litere x;`
	`char y;`
	`int i;`
	`x=A; i=x; cout<<i<<endl;`
	`y='A'; i=int(y); cout<<i;`
	`return 0;`
	`}`

❺ Даны объявления:

ПАСКАЛЬ	C++

`type Culoare = (Galben, Verde, Albastru, Violet);`
`Fundal = (Alb, Negru, Gri);`
`var x, y : Culoare;`
`z : Fundal;`
	`enum Culoare {Galben, Verde, Albastru, Violet};`
	`enum Fundal {Alb, Negru, Gri};`
	`Culoare x, y;`
	`Fundal z;`

Какие из следующих операторов являются правильными?

	ПАСКАЛЬ		C++
1)	`x:=Verde`	1)	`x=Verde`
2)	`y:=Negru`	2)	`y=Negru`
3)	`z:=Alb`	3)	`z=Alb`
4)	`x:=Gri`	4)	`x=Gri`

5)	`y:=Gri`	*5)*	`y=Gri`
6)	`z:=Violet`	*6)*	`z=Violet`
7)	`x:=Albastru`	*7)*	`x=Albastru`
8)	`y:=Azuriu`	*8)*	`y=Azuriu`

❻ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** В программе P17 перед ключевым словом **end** (соответственно перед **return** в программе C++) вставьте одну из следующих строк:

	ПАСКАЛЬ		C++
1)	`readln(x);`	*1)*	`cin>>x;`
2)	`writeln(x);`	*2)*	`cout<<x;`

Объясните сообщения, выводимые на экран в процессе компиляции полученной программы.

2.7. Интервальные типы данных (ПАСКАЛЬ)

ПРИМЕЧАНИЕ
Интервальный тип данных определен в языке ПАСКАЛЬ, но не имеет аналога в C++. Соответственно этот параграф будут изучать только ученики, обучающиеся программированию на языке ПАСКАЛЬ.

Интервальный тип данных включает подмножество значений уже известного типа данных, называемого базовым. *Базовым типом* может быть `integer`, `boolean`, `char` или любой перечисляемый тип.

Имя *интервального* типа данных, его наименьшее и наибольшее значения (в смысле порядкового номера) указываются в разделе описаний программы после ключевого слова **type**.

Пример:

1)
```
type Indice = 1..10;
     Litera = 'A'..'Z';
     Cifra = '0'..'9';
```

Множество значений типа **Indice** является подмножеством значений предопределенного типа **integer**. Множества значений типов `Litera` и `Cifra` являются подмножествами предопределенного типа **char**.

2)
```
type Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = L..V;
     ZiDeOdihna = S..D;
```

Множества значений типов `ZiDeLucru` и `ZiDeOdihna` являются подмножествами перечисляемого типа `Zi`, определенного пользователем.

3)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = A..F;
     T3 = C..H;
```

Множества значений типов T2 и T3 являются подмножествами перечисляемого типа T1. Из данных примеров следует, что базовыми типами интервальных типов данных являются:

	Интервальный тип	*Базовый тип*
1)	Indice	integer
2)	Litera	char
3)	Cifra	char
4)	ZiDeLucru	Zi
5)	ZiDeOdihna	Zi
6)	T2	T1
7)	T3	T1

Переменные *интервального* типа объявляются с помощью ключевого слова **var**. Переменная *интервального* типа обладает всеми свойствами переменных базового типа, но ее значения должны находиться в соответствующем диапазоне. В противном случае возникает ошибка и программа останавливается.

Пример:

```
Program P19;
 { Значения переменных интервального типа }
type Indice = 1..10;
     Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = L..V;
     ZiDeOdihna = S..D;
var  i : Indice;      { возможные значения: 1, 2, ..., 10 }
     z : Zi;          { возможные значения: L, Ma, ..., D }
     zl : ZiDeLucru; { возможные значения: L, Ma, ..., V }
     zo : ZiDeOdihna; { возможные значения: S, D }
begin
 i:=5; i:=11;           { Ошибка, i>10 }
 z:=L; zl:=J; zl:=S;    { Ошибка, zl>V }
 zo:=S; zo:=V;          { Ошибка, zo<S }
 writeln('Конец');
end.
```

Программа P20 показывает, как тип Pozitiv наследует свойства типа integer.

```
Program P20;
 { Тип Pozitiv наследует свойства типа integer }
type Pozitiv = 1..32767;
var x, y, z : Pozitiv;
begin
```

```
  writeln('Введите положительные числа x, y:');
  readln(x,y);
  writeln('x=', x);
  writeln('y=', y);
  z:=x+y; writeln('x+y=', z);
  z:=x-y; writeln('x-y=', z);
  z:=x*y; writeln('x*y=', z);
  z:=x mod y; writeln('x mod y=', z);
  z:=x div y; writeln('x div y=', z);
end.
```

Отметим, что операции +, -, *, **mod** и **div** базового типа `integer` применимы к *интервальному* типу `Pozitiv`. Но в отличие от переменных типа `integer` переменные типа `Pozitiv` не могут принимать отрицательные значения.

Использование *интервальных* типов данных улучшает наглядность программ и упрощает их проверку.

Следует подчеркнуть, что в языке ПАСКАЛЬ нельзя определять интервальные типы на базе типа `real`, так как его значения не имеют порядковых номеров.

Вопросы и упражнения

❶ Как определяется *интервальный* тип данных? Из чего состоит множество значений *интервального* типа данных?

❷ Назовите базовый тип каждого *интервального* типа:

```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = -60..60;
     T3 = 5..9;
     T4 = '5'..'9';
     T5 = A..E;
     T6 = 'A'..'E';
```

❸ Какие значения может принимать каждая переменная из следующих описаний:

```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = 1..9;
     T3 = 6..15;
     T4 = -100..100;
     T5 = 'A'..'Z';
     T6 = '0'..'9';
     T7 = C..F;
var  i : integer;
     j : T2;
     m : T4;
     p : T5;
     q : char;
     r : T6;
     s : T1;
     t : T7;
```

Назовите базовый тип каждого *интервального* типа. Укажите множество операций, унаследованных от базового типа.

❹ Какие из следующих определений являются правильными? Аргументируйте свой ответ.

a)
```
type Lungime = 1.0e-2..1.0;
     Latime = 1.0e-2..0.5;
```

b)
```
type Indice = 1..10;
     Abatere = +5..-5;
     Deviere = -10..+10;
```

c)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = C..H;
     T3 = F..B;
```

d)
```
type Luni = ( Ianuarie, Februarie, Martie, Aprilie, Mai,
              Iunie, Iulie, August, Septembrie, Octombrie,
              Noiembrie, Decembrie);
     LuniDeIarna = (Decembrie..Februarie);
     LuniDePrimavara = (Martie..Mai);
     LuniDeVara=(Iunie..August);
     LuniDeToamna=(Septembrie..Noiembrie);
```

❺ Дана следующая программа:

```
Program P21;
 type Indice=1..10;
 var  i, j, k, m : Indice;
begin
 writeln('Введите индексы i, j:');
 readln(i, j);
 k:=i+j; writeln('k=', k);
 m:=i-j; writeln('m=', m);
end.
```

При каких значениях переменных i, j появятся ошибки выполнения?

a) `i=3, j=2;`

b) `i=7, j=4;`

c) `i=4, j=7;`

d) `i=6, j=3;`

e) `i=2, j=2;`

f) `i=3, j=11;`

g) `i=8, j=4;`

h) `i=5, j=3.`

❻ **ТЕМАТИЧЕСКОЕ ИССЛЕДОВАНИЕ.** Рассмотрим программу `P20`. После запуска на выполнение пользователь вводит `x=1`, `y=2`. Очевидно, что `x-y=-1`. Так как значение `-1` не принадлежит типу данных `Pozitiv`, то при выполнении оператора

```
z:=x-y
```

возникнет ошибка.

Укажите операторы, при выполнении которых появятся ошибки, если:

a) `x=1000, y=1000;`

b) `x=1000, y=1001;`

c) `x=1001, y=1000;`

d) `x=30000, y=30000;`

e) `x=1, y=2;`

f) `x=1000, y=100;`

g) `x=0, y=1;`

h) `x=1, y=0.`

2.7.* Тип `void` (C++)

ПРИМЕЧАНИЕ
Тип данных `void` не определен в языке ПАСКАЛЬ.
Соответственно этот параграф будут изучать только ученики, обучающиеся языку программирования C++.

Тип `void` – это специальный тип, множество допустимых значений которого пусто.

Этот тип используется, когда необходимо указать отсутствие какого-либо значения. Например, его можно использовать для указания типа функции, которая не возвращает результата (результат типа `void`) или функции, которая не имеет формальных аргументов (список формальных аргументов пуст – `void`). Другие варианты использования типа `void` относятся, в частности, к переменным типа `pointer` (указателя) и будут уточнены в соответствующий момент.

Пример:

```
void mesaj()
{
  cout << "Я являюсь функцией!";
}
```

2.8. Общие сведения о порядковых типах данных

Типы данных `integer`, `boolean`, char, перечисляемые и интервальные типы в языке ПАСКАЛЬ, соответственно **`int`**, **`bool`**, **`char`** и **`enum`** в C++, являются порядковыми **типами данных.** Каждое значение порядкового типа данных имеет порядковый номер, определяемый следующим образом:

1) порядковым номером некоего числа типа `integer` в языке ПАСКАЛЬ, соответственно **`int`** в C++, является само это число;

2) порядковыми номерами значений истинности `false` и `true` типа данных `boolean` являются соответственно `0` и `1`;

3) порядковый номер некоторого символа (тип char) определяется его позицией в таблице кодов, обычно *ASCII*;

4) порядковый номер некоторого значения *перечисляемого* типа в языке ПАСКАЛЬ определяется его позицией в списке перечисляемых элементов. Отметим, что элементы списка, по умолчанию, нумеруются, начиная с нуля: 0, 1, 2, … и т.д.

5) порядковые номера значений *интервального* типа в языке ПАСКАЛЬ совпадают с их порядковыми номерами в базовом типе.

Порядковый номер любого значения порядкового типа можно определить и вывести на экран.

Нижеследующая программа выводит на экран порядковые номера значений: `-32`, `true`, `'A'`, A и B.

ПАСКАЛЬ

```
Program P22;
 { Порядковые номера значе-
ний порядковых типов }
type T1 = (A, B, C, D, E,
           F, G, H);
begin
  writeln(ord(-32));   { -32 }
  writeln(ord(true));  {   1 }
  writeln(ord('A'));   {  65 }
  writeln(ord(A));     {   0 }
  writeln(ord(B));     {   1 }
end.
```

C++

```
// Программа P22
#include <iostream>
/* Порядковые номера значе-
ний порядковых типов */
using namespace std;
int main()
{
enum T1 {A, B, C, D, E, F,
         G, H};
cout<<int(-32)<<endl;   // -32
cout<<int(true)<<endl;  // 1
cout<<int('A')<<endl;   // 65
cout<<A<<endl;           // 0
cout<<B<<endl;           // 1
return 0;
}
```

ПРИМЕЧАНИЕ

Оператор **`int`** в вышеприведенной программе C++ применяется для преобразования типа данных переменной из одного в другой. Такой оператор называется **оператором преобразования типов.**

К значениям любого порядкового типа можно применять операции отношения:

Операции отношения	ПАСКАЛЬ	C++
Меньше	<	<
Меньше или равно	<=	<=
Равно	=	==
Больше или равно	>=	>=
Больше	>	>
Не равно	<>	!=

Результатом операции отношения является значение типа `boolean`, то есть может получить одно из значений `true` или `false`. При выполнении таких операций сравниваются не сами значения, а их порядковые номера.

Примеры:

ПАСКАЛЬ

Пусть

```
type Culoare = (Galben, Verde, Albastru, Violet);
```

Результатом операции `Verde < Violet` является `true`, так как `ord(Verde)=1, ord(Violet)=3`, а 1 меньше, чем 3.

Результатом операции `Galben > Violet` является `false`, так как `ord(Galben)=0, ord(Violet)=3`, а 0 не больше, чем 3.

C++

Пусть

```
enum Culoare {Galben, Verde, Albastru, Violet};
```

Результат операции `Verde<Violet` равен `true`, так как порядковый номер значения `Verde` равен 1, а порядковый номер значения `Violet` равен 3 и 1 меньше 3.

Результат операции `Galben>Violet` равен `false`, так как порядковый номер значения `Galben` равен 0, порядковый номер значения `Violet` равен 3, а 0 не больше, чем 3.

Следующие программы выводят на экран результаты операций отношения для значений `Verde` и `Violet` типа данных `Culoare`.

ПАСКАЛЬ

```
Program P23;
  { Операции отношения над
значениями порядкового типа }
type Culoare = (Galben,
     Verde, Albastru, Violet);
begin
  writeln(Verde<Violet);
  {true}
  writeln(Verde<=Violet);
  {true}
  writeln(Verde=Violet);
  {false}
  writeln(Verde>=Violet);
  {false}
  writeln(Verde>Violet);
  {false}
  writeln(Verde<>Violet);
  {true}
end.
```

C++

```
// Программа P23
/*   Операции отношения над
значениями порядкового типа */
#include <iostream>
using namespace std;
int main()
{
enum Culoare {Galben, Verde,
     Albastru, Violet};
cout<<(Verde<Violet)<<endl;
//true
cout<<(Verde<=Violet)<<endl
//true
cout<<(Verde==Violet)<<endl;
//false
cout<<(Verde>=Violet)<<endl;
//false
cout<<(Verde>Violet)<<endl;
//false
cout<<(Verde!=Violet)<<endl;
//true
return 0;
}
```

Для значений порядковых типов данных можно определить предшествующее и последующее значения.

Значению с порядковым номером i предшествует значение с порядковым номером $i-1$. За значением с порядковым номером i следует значение с порядковым номером $i+1$.

ПРИМЕЧАНИЕ

В языке ПАСКАЛЬ для определения *предшествующих* и *последующих* значений применяются предопределенные функции `pred` и `succ`;
В языке C++ предопределенных функций для нахождения *предшествующих* и *последующих* значений не существует. Следовательно, с этой целью будут использоваться эквивалентные операции, применяя **операторы преобразования.** В языке C++, в арифметических выражениях, любое значение перечисляемого типа рассматривается как целое число, преобразование в **`int`** осуществляется по умолчанию, однако преобразование целого числа в перечисляемый тип должно быть запрошено явно.

Например, для значений порядкового типа данных `Culoare` получаем:

ПАСКАЛЬ		C++	
1)	`pred(Verde) = Galben`	*1)*	`Verde-1 == Galben`
2)	`succ(Verde) = Albastru`	*2)*	`Verde+1 == Albastru`
3)	`pred(Albastru) = Verde`	*3)*	`Albastru-1 == Verde`
4)	`succ(Albastru) = Violet`	*4)*	`Albastru+1 == Violet`

Очевидно, что наименьшее значение не имеет предшествующего, а наибольшее – последующего.

Следующие программы выводят на экран значения, предшествующие и следующие за `'B'`, `0` и `'0'`.

ПАСКАЛЬ

```
Program P24;
 { Предшествующие и последующие значения }
begin
  writeln(pred('B'));   { 'A' }
  writeln(succ('B'));   { 'C' }
  writeln(pred(0));     {  -1 }
  writeln(succ(0));     {   1 }
  writeln(pred('0'));   { '/' }
  writeln(succ('0'));   { '1' }
end.
```

C++

```
// Программа P24
/* Предшествующие и последующие значения */
 #include <iostream>
using namespace std;
int main()
{
cout<<char('B'-1)<<endl;   // 'A'
cout<<char('B'+1)<<endl;   // 'C'
cout<<0-1<<endl;           // -1
cout<<0+1<<endl;           // 1
cout<<char ('0'-1)<<endl;  // '/'
cout<<char('0'+1)<<endl;   // '1'
return 0;
}
```

Отметим, что значения, предшествующие (или следующие) за порядковыми значениями 0 (тип `integer/int`) и `'0'` (тип `char`) не совпадают, так как типы этих значений различны.

ПРИМЕЧАНИЕ

Тип данных `real/float` не является порядковым типом. Следовательно, для значений вещественных типов невозможно определить порядковый номер, предшествующее и последующее значения. Несоблюдение этого правила приводит к ошибкам.

Вопросы и упражнения

❶ Назовите порядковые типы данных. Какими общими свойствами они обладают?

❷ Как определяются порядковые номера значений любого порядкового типа данных?

❸ **ПРОАНАЛИЗИРУЙТЕ!** Что выведут на экран следующие программы?

ПАСКАЛЬ

```
Program P25;
type Zi = (L, Ma, Mi, J, V, S, D);
var z1, z2 : Zi;
begin
  z1:=Ma;
  writeln(ord(z1));
  z2:=pred(z1);
  writeln(ord(z2));
  z2:=succ(z1);
  writeln(ord(z2));
  z1:=Mi; z2:=V;
  writeln(z1<z2);
  writeln(z1>z2);
  writeln(z1<>z2);
end.
```

C++

```
// Программа P25
#include <iostream>
using namespace std;
int main()
{
enum Zi {L, Ma, Mi, J, V, S, D}
z1,z2;
z1=Ma;
cout<<z1<<endl;
cout<<z1-1<<endl;
cout<<z1+1<<endl;
z1=Mi; z2=V;
cout<<(z1<z2)<<endl;
cout<<(z1>z2)<<endl;
cout<<(z1!=z2)<<endl;
return 0;
}
```

❹ **ОБРАТИТЕ ВНИМАНИЕ!** Удалите из нижеследующих программ строку, содержащую ошибку:

ПАСКАЛЬ

```
Program P26;
 { Ошибка }
var i : integer;
begin
  i:=MaxInt;
  writeln(pred(i));
  writeln(succ(i));
end.
```

C++

```
// Программа P26
#include <iostream>
#include <limits>
using namespace std;
// Ошибка
int main()
{
int i;
i=INT_MAX;
cout<<i-1<<endl;
cout<<i+1<<endl;
return 0;
}
```

Что будет выведено на экран после запуска модифицированной программы на выполнение?

❺ Прокомментируйте следующую программу на языке ПАСКАЛЬ:

```
Program P27;
 { Ошибка }
var i : integer; x : real;
begin
 i:=1; x:=1.0;
 writeln(ord(i));
 writeln(ord(x));
 writeln(pred(i));
 writeln(pred(x));
 writeln(succ(i));
 writeln(succ(x));
end.
```

Удалите строки, содержащие ошибки. Что будет выведено на экран после запуска модифицированной программы?

2.9. Объявление типов данных

Языки программирования предоставляют пользователю предопределенные типы данных: целочисленные, вещественные, логические значения, символы и др. В случае необходимости пользователь может создавать собственные типы данных, например *перечисляемые.*

Pascal

Имя типа данных и множество его значений в языке ПАСКАЛЬ объявляются с помощью следующих грамматических единиц:

<*Типы* > ::= **type** <*Описание типа*>; { <*Описание типа*>;}

<*Описание типа*> ::= <*Идентификатор*> = <*Тип*>

<*Тип*> ::= <*Идентификатор*> | <*Перечисляемый тип*> | <*Интервальный тип*> |
<*Ссылочный тип*> |
<*Тип-массив*> | <*Тип-запись*> | <*Тип-множество*> | <*Файловый тип*>

<*Перечисляемый тип*> ::= (<*Идентификатор*> {, <*Идентификатор*>})

<*Интервальный тип*> ::= <*Константа*>..<*Константа*>

Соответствующие синтаксические диаграммы приведены на *рис. 2.2.*

Примеры:

1)
```
type T1 = (A, B, C, D, E, F, G, H);
     T2 = B..F;
     T3 = C..H;
```

2)
```
type Pozitiv = 1..MaxInt;
     Natural = 0..MaxInt;
     Negativ = -MaxInt..-1;
```

3)
```
type Abatere = -10...+10;
     Litera = 'A'..'Z';
     Cifra = '0'..'9';
```

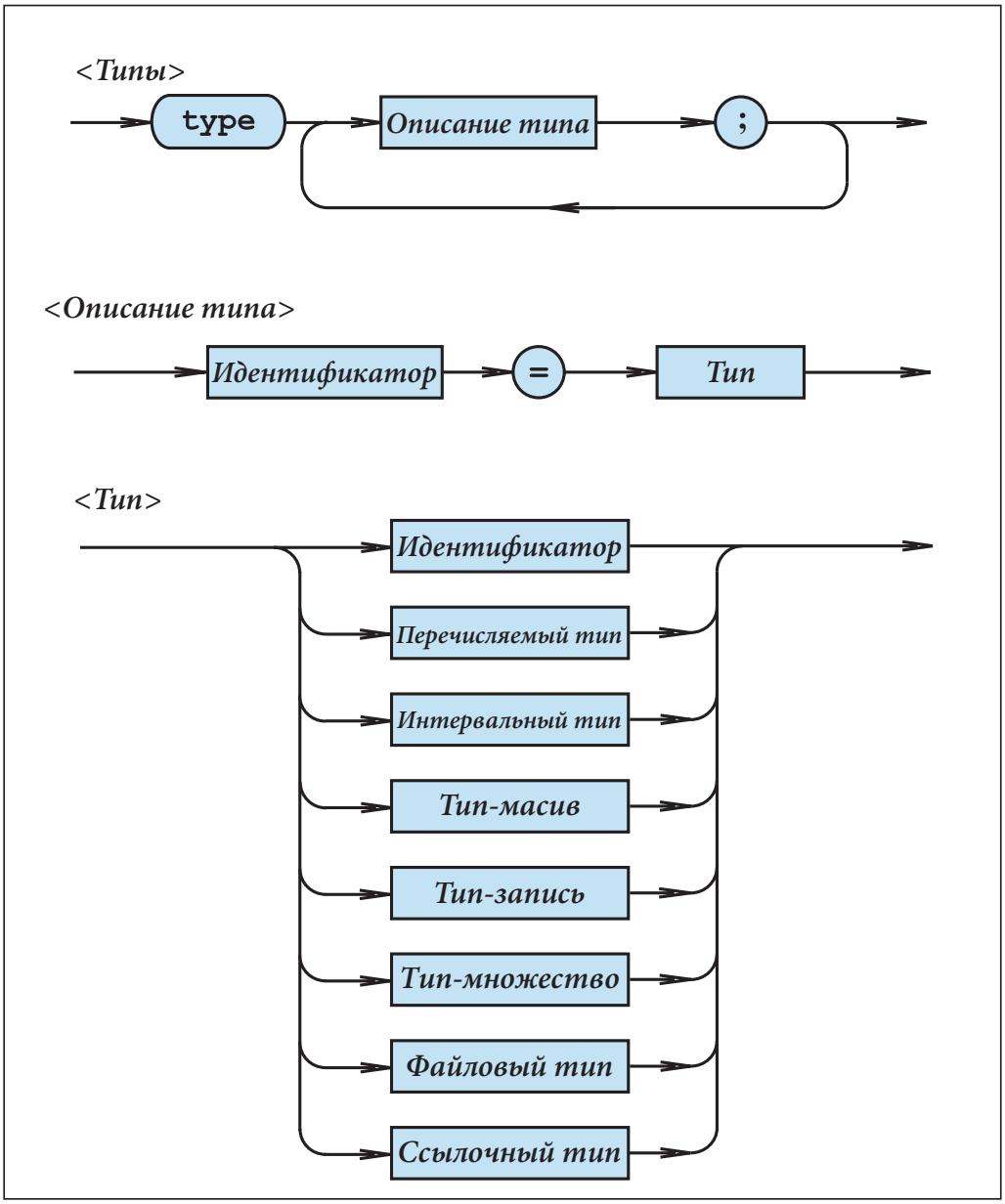


Рис. 2.2. Синтаксические диаграммы для описания типов данных

Классификация типов данных языка ПАСКАЛЬ представлена на *рис. 2.3.* Ранее изученные типы данных указаны на темном фоне.

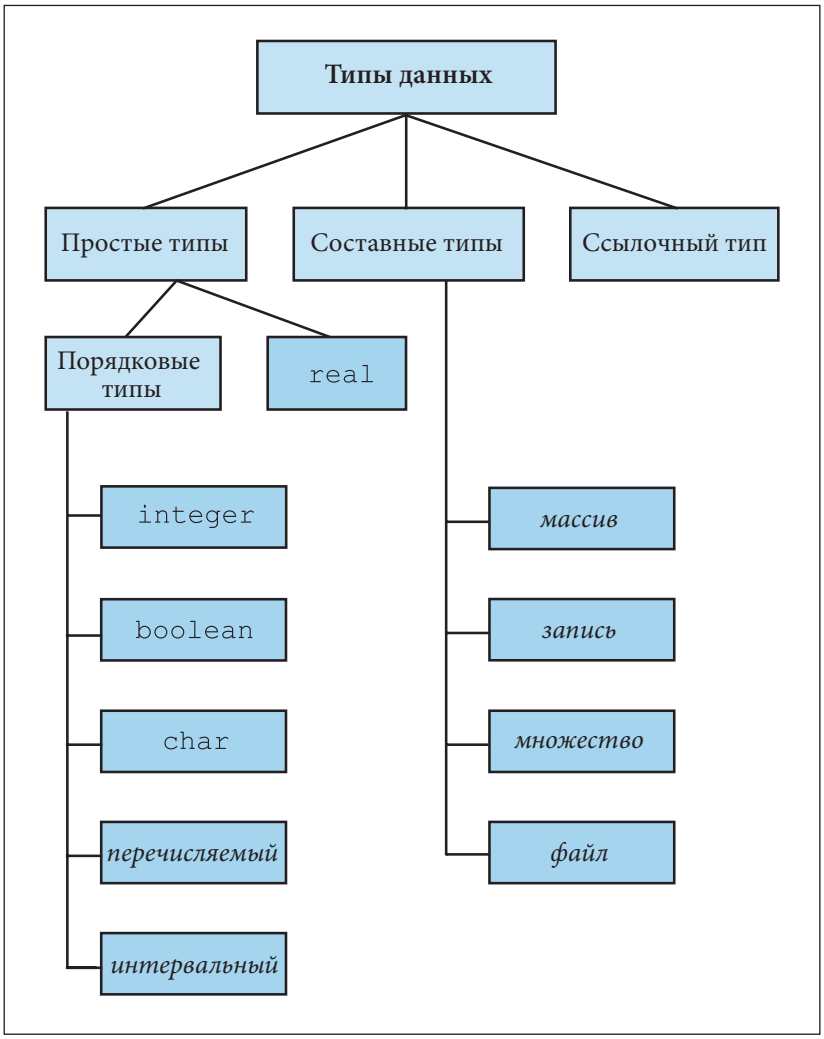


Рис. 2.3. Классификация типов данных, язык ПАСКАЛЬ

В некоторых конструкциях языка необходимо, чтобы переменные и константы принадлежали к идентичным или совместимым типам.

Два типа являются **идентичными,** если они описываются одним и тем же именем типа.

Например, пусть

```
type T4 = integer;
     T5 = integer;
```

Здесь типы `integer`, `T4`, `T5` являются идентичными.

Два типа могут быть идентичными и тогда, когда они описываются различными именами типа при условии, что эти имена эквивалентны по свойству транзитивности.

Например, пусть

```
type T6 = real;
     T7 = T6;
     T8 = T7;
```

Здесь типы `real`, `T6`, `T7` и `T8` являются идентичными.

Два типа называются **совместимыми** тогда, когда верно хотя бы одно из следующих утверждений:

1) рассматриваемые типы идентичны;
2) один тип является интервальным типом второго;
3) оба типа являются интервальными, с одним и тем же базовым типом.

Например, при следующем объявлении:

```
type Zi = (L, Ma, Mi, J, V, S, D);
     ZiDeLucru = (L, Ma, Mi, J, V);
     ZiDeOdihna = (S, D);
     Culoare = (Galben, Verde, Albastru, Violet);
```

типы `Zi`, `ZiDeLucru`, `ZiDeOdihna` являются совместимыми. Типы `Zi` и `Culoare` не совместимы. Отсюда следует, что допустимы следующие операции отношения:

```
L < D        Mi <> D        Verde <> Violet
```

и т.д., а операции типа

```
L < Violet        Verde = V        S <> Albastru
```

запрещены.

Кроме типов данных, определяемых пользователем явно с помощью ключевого слова **`type`**, в программе могут быть описаны анонимные типы (типы без имени).

Анонимный тип определяется неявно при описании переменных.

Пример:

```
var i : 1..20;
    s : (Alfa, Beta, Gama, Delta);
    t : Alfa..Gama;
```

Отметим, что интервальный тип `1..20`, перечисляемый тип (`Alfa`, `Beta`, `Gama`, `Delta`) и интервальный тип `Alfa..Gama` не имеют собственных имен.

Как правило, анонимные типы используются в программах с небольшим количеством переменных.

Язык С ++, разработанный для профессиональных целей, предлагает программистам больше возможностей использовать и создавать новые типы данных.

Таким образом, с помощью ключевого слова **`typedef`** с существующими типами могут быть ассоциированы новые имена (псевдонимы):

`typedef` <*Имя типа*> <*Имя нового типа*>;

Примеры:

1)
```
typedef int intreg;
intreg x1, i, t1, t2;
```

2)
```
typedef float real;
real a, b, c, x, delta, x1, x2;
```

Согласно вышеприведенным примерам, идентификатор `Integer` становится вторым именем типа данных **`int`**, а идентификатор `Real` – вторым именем типа данных **`float`**.

Следовательно, хотя переменные `x1`, `i`, `t1` и `t2` объявлены как имеющие тип `Integer`, на самом деле они будут иметь тип **`int`**. Аналогичным образом, хотя переменные `a, b, c, x, delta, x1` и `x2` объявлены как имеющие тип `Real`, на самом деле они будут иметь тип **`float`**.

В повседневной практике псевдонимы используются для облегчения чтения больших программ. Перечисляемые типы данных определяются с помощью ключевого слова **`enum`**:

`enum` <*Имя типа*> {<*Идентификатор*> {, <*Идентификатор*>};

В грамматической конструкции, приведенной выше, грамматическая единица {<*Идентификатор*> {, <*Идентификатор*>} представляет набор значений типа данных <*Имя типа*>.

Примеры:

1)
```
enum ZideLucru {Luni, Marti, Miercuri, Joi, Vineri};
ZideLucru Z;
```

2)
```
enum GenDeMuzica {Blues, Etno, Folk, Disco, HipHop,
                  Rock, Pop, Jazz};
GenDeMuzica G;
```

В вышеприведенных примерах определены два типа данных перечисления: `ZideLucru` и `GenDeMuzica`. Очевидно, что переменная Z может принимать значения `Luni`, ..., `Vineri`, а переменная `G - Blues`, ..., `Jazz`.

Отметим, что в языке C++ существует несколько способов классификации типов данных: простые и структурированные, предопределенные (принадлежащие языку) и определяемые пользователем, фундаментальные и производные.

Наиболее часто используемый способ классификации типов данных приведен на *рисунке 2.3**.

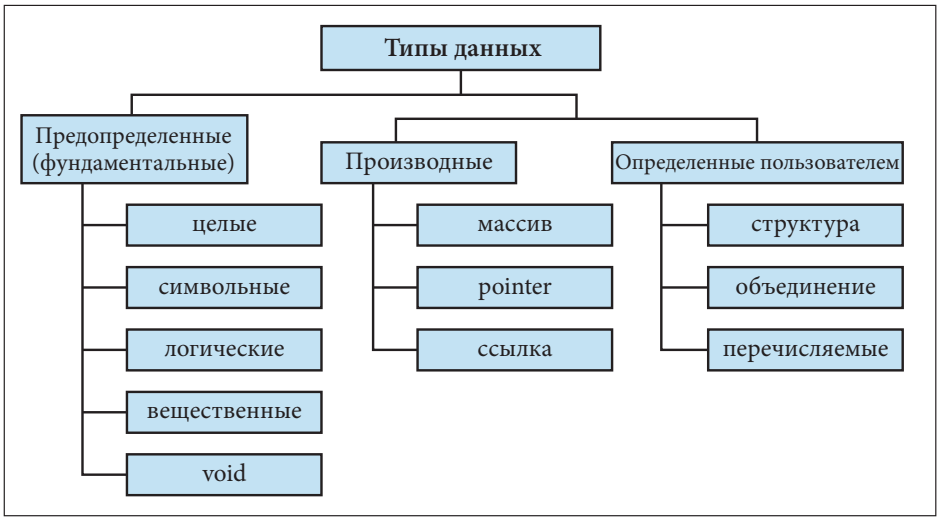


*Рис. 2.3.** Классификация типов данных, язык C++

Следующие предопределенные типы являются фундаментальными: тип **void**, логический тип **bool**, арифметические целочисленные типы **char** и **int**, а также арифметические вещественные типы с плавающей запятой **float** и **double**. Все остальные типы (массивы, функции, указатели, ссылки, структуры, объединения) являются производными типами, основанными на фундаментальных типах. Производные и определенные пользователем типы будут изучаться позже.

В некоторых конструкциях переменные и константы должны быть одинакового или **совместимого** типа.

Два типа **идентичны**, если они определены одним и тем же именем типа.

Например, пусть

```
typedef int T4;
typedef int T5;
```

или

```
typedef int T4, T5;
```

Здесь типы **int**, T4 и T5 идентичны.

Два типа являются идентичными и в случае, если они определены разными именами, однако эти имена эквивалентны через транзитивность.

Например, пусть

```
typedef double T6;
typedef T6 T7;
typedef T7 T8;
```

Здесь **double**, T6, T7 и T8 идентичные типы.

Вообще язык C++ предлагает программисту большую свободу в использовании типов данных, что объясняет его популярность в индустрии программных продуктов. Эта свобода в основном отражается в манипуляциях с типами (что недопустимо в Паскале). Вместе с тем язык ПАСКАЛЬ больше подходит для использования в учебных целях, а намеренное ограничение степени свободы при манипулировании типами данных способствует соблюдению принципа «от простого к сложному».

Наиболее очевидна эта «свобода» в случае типа **char**. Этот тип на самом деле является целочисленным типом, представленным на одном байте (аналогично типу byte в ПАСКАЛЕ). Таким образом, следующая последовательность кода является вполне допустимой, а подобный «фейерверк» в программировании является обычным явлением:

```
int i;
char c;

c='A';
i=c;
cout<<"Код ASCII символа "<<c<<" равен "<<i<<endl;
i='A'+1;
cout<<"и следующий символ имеет код "<<i;
```

в результате выполнения этой последовательности операторов нна экране отобазится:

```
Код ASCII символа A равен 65
и следующий символ имеет код 66
```

Компиляторы C++ проверяют совместимость типов в следующих случаях:
– при присваивании
– при передаче параметров
– при расчете значений выражений.

Последствия проверки зависят от языковой спецификации и, возможно, от особенностей операционной системы хост-компьютера.

Помимо определяемых пользователем типов данных, явно использующих ключевое слово **typedef** и **enum**, в программе на C++ также могут быть определены анонимные (безымянные) типы данных.

Анонимный тип данных определяется неявно при объявлении переменной.

Примеры:

1)
```
enum ZiDeOdihna {Sambata, Duminica}; // tip explicit
ZiDeOdihna Zi;
```

2)
```
enum {Sambata, Duminica} Zi; // tip anonim
```

В первом примере перечисляемый тип `ZileDeOdihna` был определен явно. Позже этот тип использован для объявления переменной `Zi`. Во втором примере тип переменной `Zi` не имеет имени и является анонимным, будучи объявлен по умолчанию.

Обычно анонимные типы данных используются в программах с небольшим количеством переменных.

Вопросы и упражнения

❶ **ПРОАНАЛИЗИРУЙТЕ!** Даны следующие программы:

ПАСКАЛЬ

```
Program P28;
type T1 = -100..100;
     T2 = 'A'..'H';
     T3 = (A, B, C, D, E, F, G, H);
     T4 = A..E;
     T5 = integer;
     T6 = real;
     T7 = char;
     T8 = boolean;
var i : T1;
    j : T5;
    k : T2;
    m : T3;
    n : T4;
    p : real;
    q : T6;
    r : char;
    s : T7;
    t : boolean;
```

C++

```
// Программа P28
#include <iostream>
using namespace std;
int main();
{
enum T3 {A, B, C, D, E, F, G, H};
typedef T3 T4;
typedef int T5;
typedef double T6;
typedef char T7;
typedef bool T8;
T5 j;
T3 m;
T4 n;
double p;
T6 q;
char r;
T7 s;
bool t;
```

```
    z : T8;
    y : real;
begin
  { расчеты с использованием }
  { объявленных переменных }
  writeln('Конец');
end.
```

```
T8 z;
double y;
// расчеты с использованием
// объявленных переменных
cout<<"Конец";
return 0;
}
```

Укажите типы данных, используемые в программе на языке ПАСКАЛЬ/C++. Какие значения может принимать каждая переменная, описанная в данной программе? Какие из указанных типов совместимы?

❷ (ПАСКАЛЬ) Укажите на синтаксических диаграммах *рис.2.2* пути, соответствующие описаниям типов данных из программы P28.

❸ **ПРОАНАЛИЗИРУЙТЕ!** Укажите анонимные типы данных, используемые в следующих программах:

ПАСКАЛЬ

```
Program P29;
type T1 = integer;
     T2 = -150..150;
     T3 = 1..5;
var i : T1;
    j : T2;
    k : T3;
    m : 1..5;
    n : (Unu, Doi, Trei, Patru,
Cinci);
begin
  { вычисления, в которых
используются }
  { объявленные переменные }
  writeln('Конец')
end.
```

C++

```
// Программа P29
#include <iostream>
using namespace std;
int main()
{
typedef int T1;
enum {Rosu, Galben,Verde}
Semafor;
enum {Iunie, Iulie, August} v;
enum Numere {Unu, Doi,Trei,
Patru, Cinci};
T1 i;
Numere n;
/* вычисления, в которых
используются  объявленные
переменные*
cout<<"Конец";
return 0;
}
```

Какие значения может принимать каждая переменная, описанная в данной программе на языке ПАСКАЛЬ/C++?

❹ Даны следующие объявления:

ПАСКАЛЬ

```
type T1 = boolean;
     T2 = T1;
     T3 = T2;
     T4 = T3;
var x : T4;
```

C++

```
typedef bool T1;
typedef T1 T2;
typedef T2 T3;
typedef T3 T4;
T4 x;
```

Какие значения может принимать переменная x? Назовите операции соответствующего типа данных.

❺ В каких случаях два типа данных являются идентичными? Приведите примеры.

❻ В каких случаях два типа данных являются совместимыми? Приведите примеры.

❼ Даны объявления:

ПАСКАЛЬ

```
type T1 = integer;
     T2 = T1;
     T3 = -5..+5;
     T4 = T3;
     T5 = -10..+10;
     T6 = (A, B, C, D, E, F,
G, H);
     T7 = A..D;
     T8 = E..H;
     T9 = 'A'..'D';
     T10 = 'E' ..'H';
```

C++

```
typedef int D1;
typedef D1 D2;
typedef double D3;
typedef D3 D4;
enum D6 {A, B, C, D, E, F,
G, H};
typedef D6 D5;
```

Найдите идентичные и совместимые типы данных.

2.10. Объявление переменных

Известно, что каждая встречающаяся в программе переменная должна быть ассоциирована с определенным типом данных. Для этого используются следующие грамматические конструкции:

ПАСКАЛЬ

<*Переменные*> ::= **var** <*Объявление переменных*>**;** {<*Объявление переменных*>**;**}

<*Объявление переменных*> ::= <*Идентификатор*> **{,** <*Идентификатор*>} **:** <*Тип*>

C++

<*Переменные*> ::= <*Объявление переменных*>**;** {<*Объявление переменных*>**;**}

<*Объявление переменных*> ::= <*Тип*> <*Идентификатор*> {**,**<*Идентификатор*>}**;**

Синтаксические диаграммы рассматриваемых грамматических единиц приведены на *рис. 2.4.*

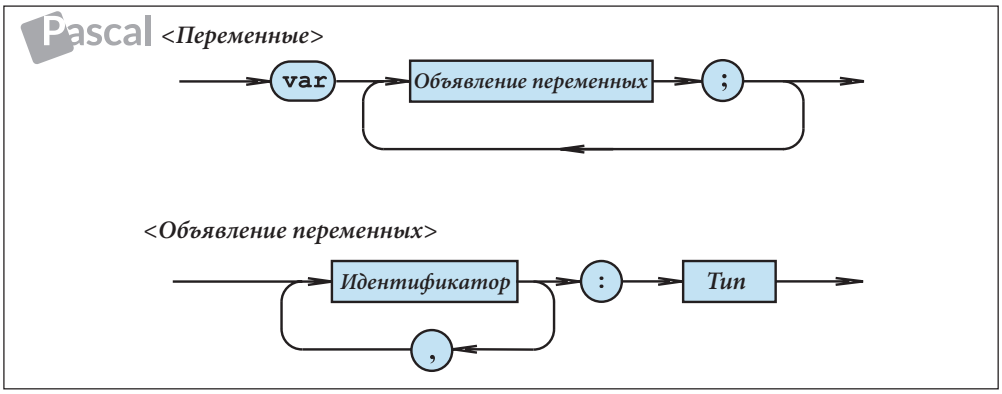


Рис. 2.4. Синтаксические диаграммы <*Переменные*> и <*Объявление переменных*>, ПАСКАЛЬ

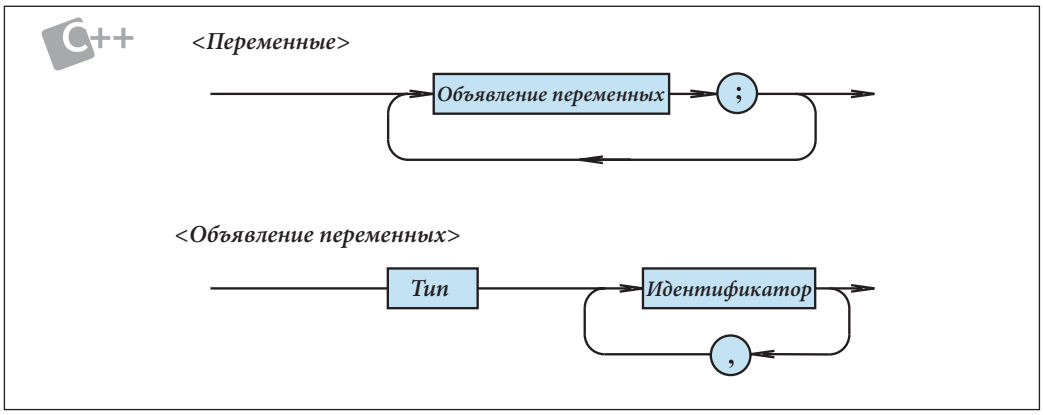


*Рис. 2.4**. Синтаксические диаграммы <*Переменные*> и <*Объявление переменных*>, C++

В объявлении переменных могут использоваться предопределенные типы данных (целочисленные, вещественные, символьные, логические и др.) и типы, определяемые пользователем (перечисляемые, массивы и т.д.).

Примеры:

	ПАСКАЛЬ		C++

1)	`var i, j : integer;`
	`x : real;`
	`p : boolean;`
	`r, s : char;`
		1)	`int i,j;`
			`double x;`
			`bool p;`
			`char r,s;`

2)	`type T1 = (A, B, C, D, E, F, G, H);`
	`T2 = C..F;`
	`T3 = 1..10;`
	`T4 = 'A'..'Z';`
	`var x, y, z : real;`
	`r, s : char;`
	`i, j : integer;`
	`k : T3;`
	`p : T1;`
	`q : T2;`
	`v : T4;`
		2)	`enum T1 {A, B, C, D, E, F, G, H};`
			`double x, y, z;`
			`char r, s;`
			`int i, j;`
			`T1 p;`

3)	`type Zi = (L, Ma, Mi, J, V, S, D);`
	`var x, y : real;`
	`z : Zi;`
	`zl : L..V;`
	`m, n : 1..10;`
		3)	`enum Zi {L, Ma, Mi, J, V, S, D};`
			`double x, y;`
			`Zi z;`
			`enum {A,B,C} z1;`
			`enum {True, False} m, n;`

Отметим, что в последнем примере тип переменных `z1`, `m` и `n` описывается непосредственно в разделе объявления переменных. Следовательно, эти переменные принадлежат анонимным типам данных.

Вопросы и упражнения

❶ **ПРОАНАЛИЗИРУЙТЕ!** Определите тип переменных, используемых в следующей программе:

ПАСКАЛЬ

```
Program P30;
type T1 = integer;
     Studii = (Elementare,
Medii, Superioare);
     T2 = real;
     Culoare = (Galben, Verde,
               Albastru, Violet);
var x : real;
    y : T1;
    i : integer;
    j : T2;
    p : boolean;
    c : Culoare;
    s : Studii;
    q : Galben..Albastru;
    r : 1..9;
begin
  { вычисления, в которых ис-
пользуются объявленные перемен-
ные}
  writeln('Конец');
end.
```

C++

```
// Программа P30
#include <iostream>
using namespace std;
int main()
{
typedef int T1;
enum Studii {Elementare,
Medii, Superioare};
typedef double T2;
enum Culoare {Galben, Verde,
Albastru, Violet};
float x;
T1 y;
int i;
T2 j;
bool p;
Culoare c,q;
Studii s;
int r;
//* вычисления, в которых
используются объявленные пе-
ременные*/
cout<<"Конец";
return 0;
}
```

Какие значения может принимать каждая переменная? Назовите операции соответствующего типа данных.

❷ Укажите на синтаксических диаграммах *рис. 2.4* пути, которые соответствуют объявлению переменных в программе из задания 1.

❸ **ПРИМЕНИТЕ!** Запустите следующие программы на выполнение и объясните сообщения, выведенные на экран в процессе компиляции программ. Измените программу таким образом, чтобы она выдавала на экран результат: 3

ПАСКАЛЬ

```
Program P31;
var i, j : integer;
begin
  i:=1; j:=2; k:=i+j;
  writeln('k=', k);
end.
```

C++

```
// Программа P31
#include <iostream>
using namespace std;
int main()
{
int i, j;
i=1; j=2; k=i+j;
cout<<"k="<<k<<endl;
return 0;
}
```

❹ Как объявляются переменные, принадлежащие анонимным типам данных?

2.11. Описание констант

Известно, что значения любого типа данных могут быть выражены через переменные или константы. Для того чтобы сделать программы более удобными для чтения и модификации, языки программирования ПАСКАЛЬ и C++ позволяют представлять константы в виде символических имен. Идентификатор, представляющий константе, называется именем константы или просто константой. Везде, где в программе встречается такое имя, оно заменяется на соответствующее значение.

Константы описываются с помощью следующих грамматических конструкций:

ПАСКАЛЬ

<*Константы*> ::= **const** <*Описание константы*>; { <*Описание константы*>;}
<*Описание константы*> ::= <*Идентификатор*> = <*Константа*>
<*Константа*> ::= [**+** | **-**] <*Число без знака*> | [**+** | **-**] <*Имя константы*> | <*Строка символов*>

C++

<*Константы*> ::= **const** <*Описание константы*>**;**
<*Описание константы*> ::= <*Тип*> <*Идентификатор*> = <*Константа*> {
<*Константа*> ::= [+ | -] <*Число без знака*> | [+ | -] <*Имя константы*> | <*Строка символов*>

Синтаксические диаграммы данных грамматических единиц приведены на рис. 2.5.

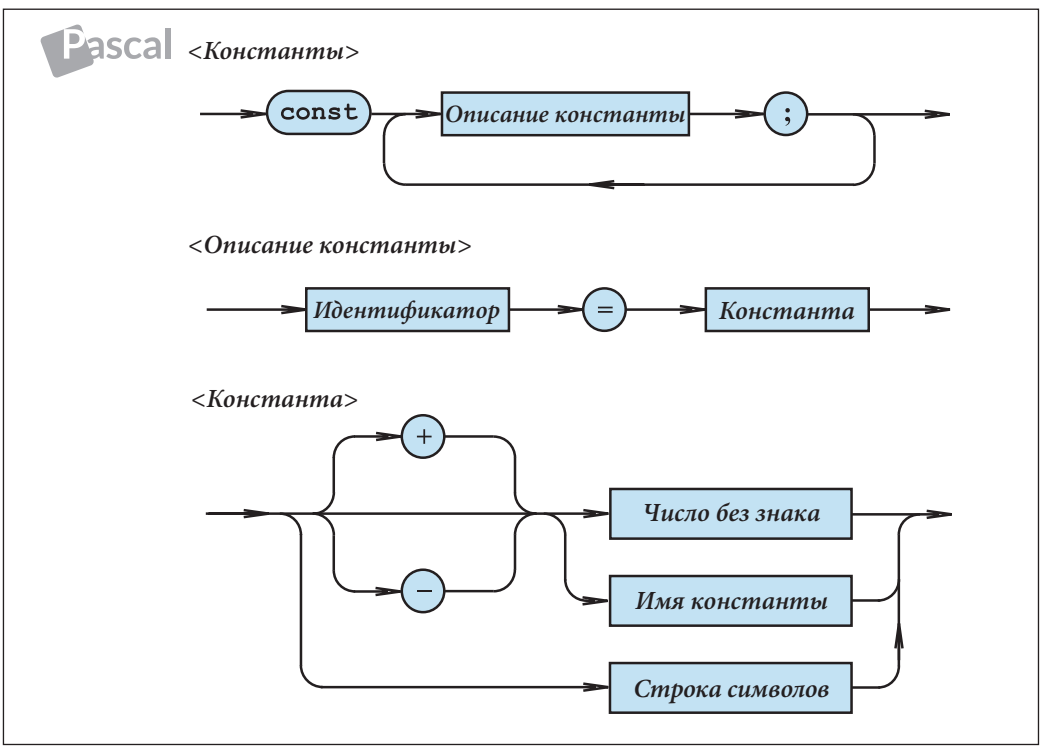


Рис. 2.5. Синтаксические диаграммы <*Константы*>, <*Описание константы*> и <*Константа*>, ПАСКАЛЬ

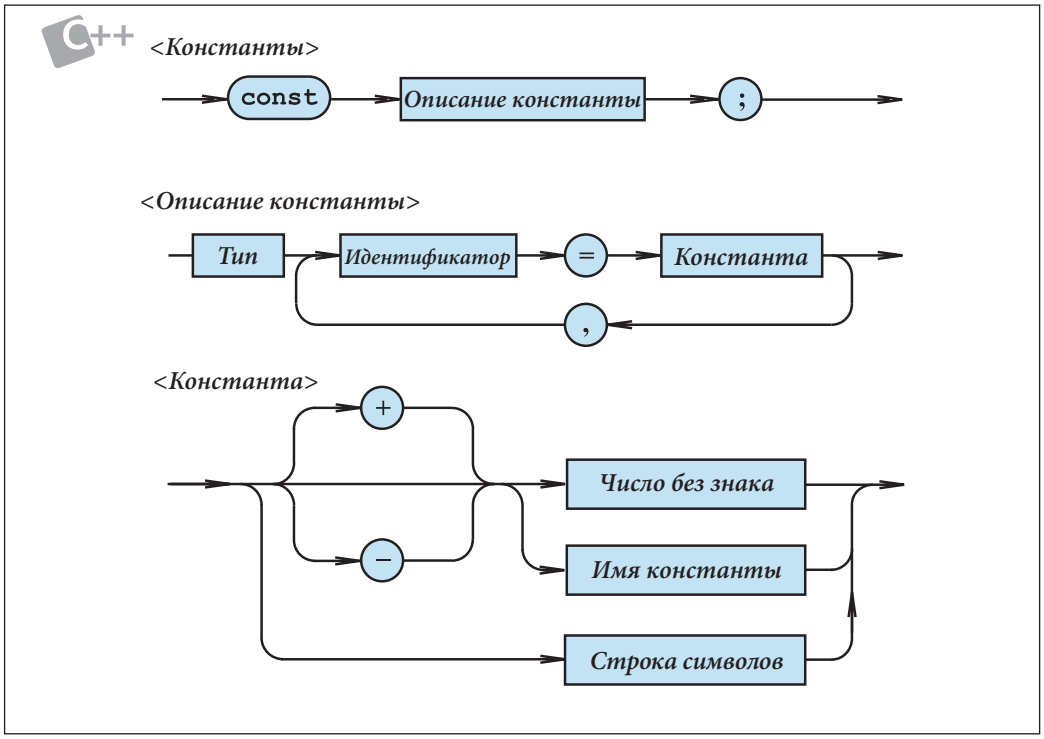


*Рис. 2.5**. Синтаксические диаграммы *<Константы>*, *<Описание константы>* и *<Константа>*, C++

Примеры:

	ПАСКАЛЬ		C++

1)	`const a = 10;`
	`b = 9.81;`
	`c = '*';`
	`t = 'TEXT';`
		1)	`const int a = 10;`
			`float b = 9.81;`
			`char c = '*';`
			`string t = "TEXT";`

2)	`const NumarCaractere = 60;`
	`LungimePagina = 40;`
		2)	`const`
			`int NumarCaractere = 60;`
			`int LungimePagina = 40;`

3)	`const n = 10;`
	`m = 20;`
	`Pmax = 2.15e+8;`
	`Pmin = -Pmax;`
	`S = 'STOP';`
		3)	`const int n = 10;`
			`int m = 20;`
			`double Pmax = 2.15e+8;`
			`double Pmin = -Pmax;`
			`string S ="STOP";`

В вышеприведенных примерах типами констант являются:

`a, NumarCaractere, LungimePagina, n, m` – *целочисленные;*
`b, Pmax, Pmin` – *вещественные;*
`c` – *char*;
`t, S` – *строки символов.*

В нижеследующих программах описываются константы Nmax, Nmin, Pi, Separator, Indicator и Mesaj. Значения этих констант выводятся на экран.

ПАСКАЛЬ

```
Program P32;
 { Описание констант }
const Nmax = 40; {Константа
типа integer}
   Nmin = -Nmax; {Константа
типа integer}
   Pi = 3.14; {Константа типа
real}
   Separator = '\';
   {Константа типа char}
   Indicator = TRUE;
   {Константа типа boolean}
   Mesaj =
   'Проверьте принтер';
        {Строка символов }
begin
  writeln(Nmax);
  writeln(Nmin);
  writeln(Pi);
  writeln(Separator);
  writeln(Indicator);
  writeln(Mesaj);
{ вычисления, в которых ис-
пользуются данные константы}
end.
```

C++

```
// Программа P32
#include <iostream>
using namespace std;
int main()
{
const int Nmax = 40;
const int Nmin = -Nmax;
const double Pi = 3.14;
const char Separator = '/';
const bool Indicator = true;
const string Mesaj = "
     Проверьте принтер";
cout<<Nmax<<endl;
cout<<Nmin<<endl;
cout<<Pi<<endl;
cout<<Separator<<endl;
cout<<Indicator<<endl;
cout<<Mesaj<<endl;
/*вычисления,в которых исполь-
зуются данные константы */
return 0;
}
```

Обычно данные, которые не изменяются в процессе выполнения программы, например, количество строк таблицы, число π, ускорение свободного падения g и т.д., описываются в виде констант. Это позволяет изменять постоянные значения, не изменяя всей программы в целом.

В нижеследующих программах длина L и площадь S круга вычисляются по формулам:

$$L=2\pi r;\ S=\pi r^2,$$

где r является радиусом окружности. Число π представлено в виде константы Pi = 3.14.

ПАСКАЛЬ

```
Program P33;
 { Длина и площадь круга }
const Pi = 3.14;
var L, S , r : real;
```

C++

```
// Программа P33
// Длина и площадь круга
#include <iostream>
using namespace std;
```

```
begin
  writeln('Введите радиус r=');
  readln(r);
  L:=2*Pi*r;
  writeln('Длина L=', L);
  S:=Pi*r*r;
  writeln('Площадь S=', S);
end.
```

```
int main()
{
const double Pi = 3.14;
float L, S , r;
cout<<"Введите радиус r=";
cin>>r;
L=2*Pi*r;
cout<<"Длина L="<<L<<endl;
S=Pi*r*r;
cout<<"Площадь S="<<S;
return 0;
}
```

Если пользователю необходимы более точные результаты, изменяется только строка программы, в которой указано значение константы:

ПАСКАЛЬ

```
const Pi = 3.141592654;
```

C++

```
const double Pi = 3.141592654;
```

Оставшаяся часть программы остается неизменной.

В отличие от переменных, значения констант не могут изменяться посредством операции присваивания или считывания. Несоблюдение данного правила приводит к ошибкам, которые обнаруживаются при компиляции программы.

Вопросы и упражнения

❶ Определите типы данных следующих констант:

	ПАСКАЛЬ		C++

1)	`const a = 29.1;`
	`b = TRUE;`
	`c = 18;`
		1)	`const double a = 29.1;`
			`const bool b = TRUE;`
			`const int c = 18;`

2)	`const d = -16.82e-14;`
	`f = -d;`
	`i = 15;`
	`j = '15';`
	`n = '-d';`
		2)	`const double d = -16.82e-14;`
			`const double f = -d;`
			`const int i = 15;`
			`const string j = "15";`
			`const string n = "-d";`

3)	`const t = 'F';`
	`q = '1';`
	`c = '18';`
		3)	`const char t = 'F';`
			`const char q = '1';`
			`const string c = "18";`

4)	`const x = 65;`
	`q = FALSE;`
	`y = -x;`
	`m = 'PAUZA';`
		4)	`const int x = 65;`
			`const bool q = FALSE;`
			`const int y = -x;`
			`const string m = "PAUZA";`

❷ **ПРИМЕНИТЕ!** Напишите программу, которая выводит на экран значения констант из задания 1.

❸ Покажите на синтаксических диаграммах *рис. 2.5* пути, которые соответствуют описаниям констант из программы P32.

❹ **ПРОАНАЛИЗИРУЙТЕ!** Приведенные ниже программы содержат ошибки. Выявите ошибки, объясните причину их возникновения, доработайте программу, чтобы она действовала без ошибок.

ПАСКАЛЬ

```
Program P34;
 { Erori }
const Nmax = 40;
   Nmin = -Nmax;
   Pi = 3.14;
   Separator = '\';
   Indicator = TRUE;
begin
  readln(Nmax);
  writeln(Nmax);
  writeln(Nmin);
  Nmax:=10;
  writeln(Nmax);
  writeln(Pi);
  writeln(Separator);
  writeln(Indicator);
end.
```

C++

```
// Программа P34
#include <iostream>
using namespace std;
// Erori
int main()
{
const int Nmax = 40;
const int Nmin = -Nmax;
const double Pi = 3.14;
const char Separator = '//';
const bool Indicator = true;
cin>>Nmax;
cout<<Nmax<<endl;
cout<<Nmin<<endl;
Nmax=10;
cout<<Nmax<<endl;
cout<<Pi<<endl;
cout<<Separator<<endl;
cout<<Indicator<<endl;
return 0;
}
```

Объясните сообщения, выводимые на экран в процессе компиляции модифицированных программ.

❺ Даны программы:

ПАСКАЛЬ

```
Program P35;
const t = '1';
      s = -t;
begin
  writeln(s);
end.
```

C++

```
// Программа P35
#include <iostream>
using namespace std;
int main()
{
const char t = '1';
const char s = -t;
cout<<s;
return 0;
}
```

Какие сообщения будут выведены на экран в процессе компиляции?

❻ **ПРОАНАЛИЗИРУЙТЕ!** Что появится на экране после запуска на выполнение следующих программ?

ПАСКАЛЬ

```
Program P36;
const i = 1;
      j = i;
      k = j;
var x, y, z : integer;
begin
  x:=i+1; writeln(x);
  y:=j+2; writeln(y);
  z:=k+3; writeln(z);
end.
```

C++

```
// Программа P36
#include <iostream>
using namespace std;
int main()
{
const char i = 1;
const int j = i;
const int k = j;
int x, y, z;
x=i+1; cout<<x<<endl;
y=j+2; cout<<y<<endl;
z=k+3; cout<<z;
return 0;
}
```

❼ Даны программы:

ПАСКАЛЬ

```
Program P37;
const a = 1;
      b = 2;
      c = 3;
      d = 4;
var i, j, k : integer;
begin
  i:=a+1; writeln(i);
  j:=b+1; writeln(j);
  k:=c+1; writeln(k);
  d:=a+1; writeln(d);
end.
```

C++

```
// Программа P37
#include <iostream>
using namespace std;
int main()
{
const char a = 1;
const int b = 2;
const int c = 3;
const int d = 4;
int i, j, k;
i=a+1; cout<<i<<endl;
j=b+1; cout<<j<<endl;
k=c+1; cout<<k<<endl;
d=a+1; cout<<d;
return 0;
}
```

Объясните сообщения, выводимые на экран.

❽ Исключите из следующих программ строку, в которой содержится ошибка.

ПАСКАЛЬ

```
Program P38;
const a = 1;
var i, j : integer;
begin
```

C++

```
// Программа P38
#include <iostream>
using namespace std;
```

```
   writeln('Введите i=');
  readln(i); j:=i+a;
  writeln(j);
   writeln('Введите a=');
  readln(a);
  j:=i+a;
  writeln(j);
end.
```

```
int main()
{
const int a = 1;
int i, j;
cout<<"Введите i=";
cin>>i; j=i+a;
cout<<j;
cout<<"Введите a=";
cin>>a;
j=i+a;
cout<<j;
return 0;
}
```

Что появится на экране после запуска модифицированных программ на выполнение?

Тест для самопроверки № 2

1. Объясните значение термина *тип данных*. Назовите по крайней мере два типа данных и приведите по нескольку примеров данных соответствующих типов.

2. Каким образом представляются данные в программе на языке ПАСКАЛЬ? Объясните значения терминов *величина*, *переменная* и *константа*. Какова разница между переменной и константой?

3. Укажите типы данных, объявленных в следующей программе:

ПАСКАЛЬ

```
Program TA2;
 { Простые типы данных }
var i, j : integer;
    a, b, c : real;
    s : char;
    p : boolean;
begin
  i:=5; j:=i+9;
  writeln(i); writeln(j);
  a:=1.0; b:=1.0e-01; c:=-2.001;
  writeln(a);
  writeln(b);
  writeln(c);
  s:='A'; writeln(s);
  p:=true; writeln(p);
end.
```

C++

```
// Программа TA2
#include <iostream>
using namespace std;
// Простые типы данных
int main()
{
int i, j;
double a, b, c;
char s;
bool p;
i=5; j=i+9;
cout<<i<<endl;
cout<<j<<endl;
a=1.0; b=1.0e-01; c=-2.001;
cout<<a<<endl;
cout<<b<<endl;
cout<<c<<endl;
s='A'; cout<<s<<endl;
p=true; cout<<p;
return 0;
}
```

Пример: `i` – переменная целого типа; a – переменная вещественного типа; 5 – целочисленная константа и т.д.

4. Как определяются целочисленные типы данных в языке программирования, который вы изучаете и каково множество их значений? Какие операции можно выполнять с этими значениями?

5. Даны следующие программы:

ПАСКАЛЬ

```
Program TA3;
 { Ошибки переполнения }
var x, y, z : integer;
begin
  writeln('Введите целые числа
x, y:');
  readln(x, y);
  z:=x*y; writeln('x*y=', z);
end.
```

C++

```
// Программа TA3
#include <iostream>
using namespace std;
int main()
{
int x, y, z;
cout<<"Введите целые числа
x, y:";
cin>>x>>y;
z=x*y; cout<<"x*y="<<z;
return 0;
}
```

Приведите примеры значений переменных x и y, при которых появляются ошибки переполнения.

6. Каково множество значений вещественных типов данных? Какие операции допустимы с этими значениями? Являются ли результаты этих операций точными?

7. Даны следующие программы:

ПАСКАЛЬ

```
Program TA4;
 { Ошибки переполнения }
var x, y, z : real;
begin
  writeln('Введите вещественные
числа x, y:');
  readln(x, y);
  z:=x*y; writeln('x*y=', z);
end.
```

C++

```
// Программа TA4
#include <iostream>
using namespace std;
int main()
{
double x, y, z;
cout<<"Введите вещественные
числа x, y:";
cin>>x>>y;
z=x*y; cout<<"x*y="<<z;
return 0;
}
```

Приведите примеры значений переменных x и y, при которых появляются ошибки переполнения.

8. Каково множество значений типа данных boolean/**bool**? Какие операции допустимы с этими значениями?

9. Напишите таблицы истинности для логических операций.

PASCAL: **not**, **and** и **or**;
C++: !, && şi ||.

10. Проанализируйте следующие программы. Какие сообщения будут выведены на экран в результате выполнения программ на компьютере?

ПАСКАЛЬ	C++

Program TA5;
var p, q, r : boolean;
begin
  writeln('*Введите логические значения p, q:*');
  readln(p, q);
  r:=p **and** q;
  writeln(q);
end.
	// Программа TA5
	#include <iostream>
	using namespace std;
	int main()
	{
	bool p, q, r;
	cout<<"Введите логические значения p, q:";
	cin>>p>>q;
	r=p && q;
	cout<<q;
	return 0;
	}

11. Каково множество значений типа данных char? Как упорядочено это множество? Какие операции можно выполнять со значениями типа char?

12. Напишите программу, которая выводит на экран порядковые номера десятичных цифр.

13. Каково множество значений *перечисляемого* типа данных? Как упорядочено это множество? Какие операции можно выполнять с этими значениями?

14. Напишите программу, которая выводит на экран порядковые номера значений следующих *перечисляемых* типов:

PASCAL

1)
```
type FunctiaOcupata = (Muncitor=10, SefDeEchipa, Maistru,
SefDeSantier, Director);
```

2)
```
type StareaCivila = (Casatorit, Necasatorit);
```

C++

1)
```
enum FunctiaOcupata {Muncitor=10, SefDeEchipa, Maistru,
SefDeSantier, Director};
```

2)
```
enum StareaCivila {Casatorit, Necasatorit};
```

15. (ПАСКАЛЬ) Каким образом определяется тип данных *интервал*? Каково множество значений типа данных *интервал*? Какие операции допустимы с этими значениями?

16. Какие значения может принимать каждая из объявленных ниже переменных?

ПАСКАЛЬ	C++

`type T1 = 'A'..'Z';`
`T2 = 1..9;`
`T3 = '0'..'9';`
`T4 = (Alfa, Beta, Gama, Delta);`
`var p : T1;`
`q : T2;`
`r : T3;`
`s : char;`
`t : integer;`
`u : T4;`
	`typedef char T1;`
	`typedef int T2;`
	`typedef char T3;`
	`enum T4 {Alfa, Beta, Gama, Delta};`
	`T1 p;`
	`T2 q;`
	`T3 r;`
	`char s;`
	`int t;`
	`T4 u;`

(ПАСКАЛЬ) Назовите базовый тип каждого из *интервальных* типов данных.

17. Назовите порядковые типы данных. Каковы их общие свойства?

18. Что выводят на экран следующие программы?

ПАСКАЛЬ	C++

`Program TA6;`
`type Culoare = (Galben, Verde, Albastru, Violet);`
`begin`
`writeln(pred('Z'));`
`writeln(succ('D'));`
`writeln(pred(-5));`
`writeln(succ(9));`
`writeln(ord(Verde));`
`writeln(Ord(Violet));`
`end.`
	`// Программа TA6`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Culoare {Galben, Verde, Albastru, Violet};`
	`cout<<char('Z'-1)<<endl;`
	`cout<<char('D'+1)<<endl;`
	`cout<<-5-1<<endl;`
	`cout<<9+1<<endl;`
	`cout<<Verde<<endl;`
	`cout<<Violet;`
	`return 0;`
	`}`

19. Что выводят на экран следующие программы?

ПАСКАЛЬ	C++

`Program TA7;`
`type Nivel = (A, B, C, D, E, F, G);`
`var n, m : Nivel;`
`begin`
`n:=B; writeln(ord(n));`
`m:=pred(n); writeln(ord(m));`
`m:=succ(n); writeln(ord(m));`
	`// Программа TA7`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Nivel {A, B, C, D, E, F, G};`

```
  n:=C; m:=E;
  writeln(n<m);
  writeln(n>m);
  writeln(n<>m);
end.
```

```
Nivel n, m;
n=B; cout<<n<<endl;
cout<<n-1<<endl;
cout<<n+1<<endl;
n=C; m=E;
cout<<(n<m)<<endl;
cout<<(n>m)<<endl;
cout<<(n!=m);
return 0;
}
```

20. Объясните следующие термины: идентичные типы, совместимые типы, анонимные типы.

21. Заданы объявления:

ПАСКАЛЬ	C++

`type T1 = integer;`
`T2 = T1;`
`T3 = -5..+5;`
`T4 = T3;`
`T5 = -10..+10;`
`T6 = (A, B, C, D, E, F, G, H);`
`T7 = A..D;`
`T8 = E..H;`
`T9 = 'A'..'D';`
`T10 = 'E' ..'H';`
`var x : 1..100;`
`y : (Alfa, Beta, Gama, Delta);`
	`typedef int T1;`
	`typedef T1 T2;`
	`typedef int T3;`
	`typedef T3 T4;`
	`typedef int T5;`
	`enum T6 {A, B, C, D, E, F, G, H};`
	`typedef T6 T7;`
	`typedef T6 T8;`
	`typedef char T9;`
	`typedef char T10;`
	`int x;`
	`enum {Alfa, Beta, Gama, Delta} y;`

Укажите идентичные типы, совместимые типы и анонимные типы данных.

22. Определите тип каждой из следующих констант:

ПАСКАЛЬ	C++

`const Alfa = 5;`
`Beta = 12.485;`
`Indicator = true;`
`Mesaj = 'Ошибка переполнения';`
`Semn = '+';`
`Inscris = '12.485';`
	`const int Alfa = 5;`
	`const float Beta = 12.485;`
	`const bool Indicator = true;`
	`const string Mesaj = "Ошибка переполнения";`
	`const char Semn = '+';`
	`const string Inscris = "12.485";`

23. В некотором алгоритме используются целые переменные i, j, вещественные переменные x, y, константы 3,14 и 9,8. Напишите на языке программирования, который вы изучаете, объявления для описания соответствующих переменных и констант.

Глава 3

ОПЕРАТОРЫ

3.1. Концепция действия

Согласно **концепции действия**, реализованной в изучаемом языке, компьютер является исполнителем, рабочая среда которого состоит из множества всех переменных и констант, объявленных в программе. В ходе выполнения программы исполнитель осуществляет над величинами из рабочей среды определенные действия, например, сложение или вычитание, считывание с клавиатуры или вывод на экран и т.п. Очевидно, что в результате этих действий значения переменных могут изменяться, в то время как значения констант – нет.

Действия, необходимые для обработки данных, и порядок их выполнения задаются с помощью **операторов**. Существуют две категории операторов:

1) простые операторы;

2) структурированные операторы.

Простые операторы не содержат других операторов. Простыми операторами являются:

– оператор присваивания;

– оператор вызова (процедуры);

– оператор перехода;

– пустой оператор.

Структурированные операторы составлены из других операторов. Структурированными операторами являются:

– составной оператор;

– операторы условного перехода (ветвления) (в языке ПАСКАЛЬ: **`if`**, **`case`**; в языке C++: **`if`**, **`switch`**)

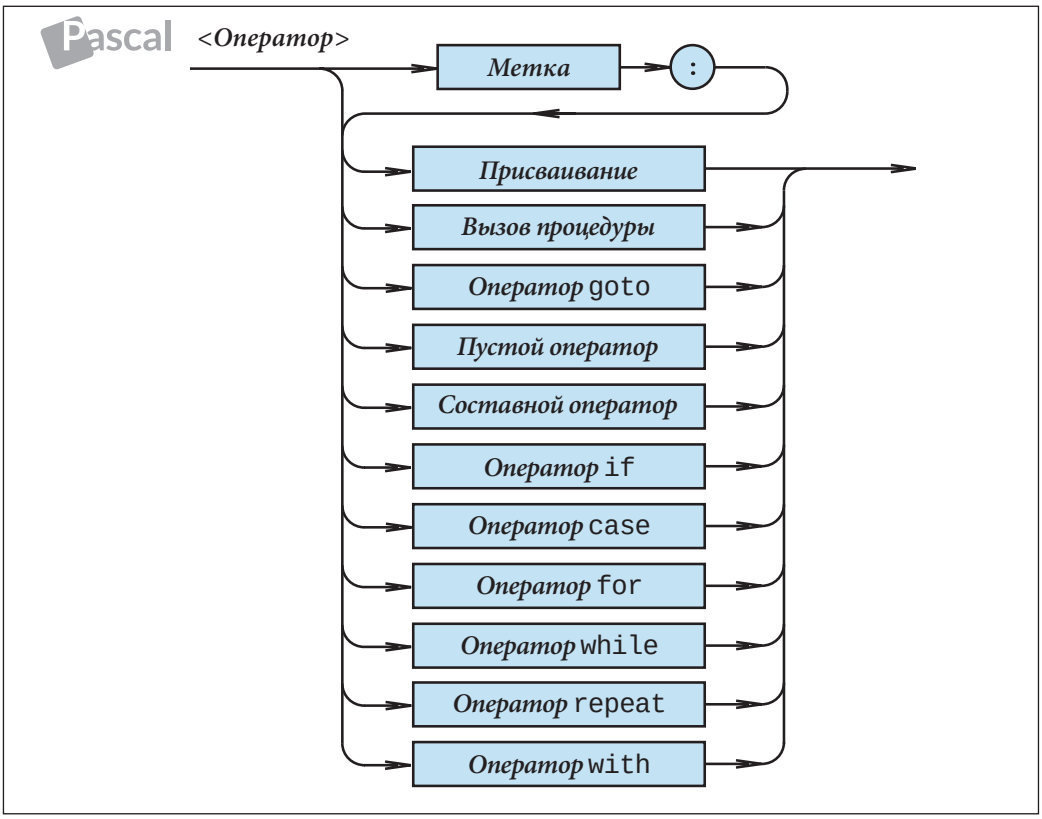
– операторы повторения (цикла) (в языке ПАСКАЛЬ: **`for`**, **`while`**, **`repeat`**; в языке C++: **`for`**, **`while`**, **`do while`**)

– оператор **`with`** (только в языке ПАСКАЛЬ).

Операторы условного перехода применяются для программирования алгоритмов с разветвлениями, а операторы повторения – для программирования циклических алгоритмов. Напомним, что циклические алгоритмы используются для описания многократно повторяющихся действий, а алгоритмы с разветвлениями – для выбора необходимых действий в зависимости от условий из рабочей среды исполнителя.

Каждому оператору может предшествовать метка. Ссылка на метку указывается в операторе перехода **`goto`**. Напомним, что метка – это целое число без знака (см. параграф 1.10).

Синтаксическая диаграмма <*Оператор*> показана на *рис. 3.1*. Напоминаем, что метка отделяется от оператора с помощью символа «:» (двоеточие).



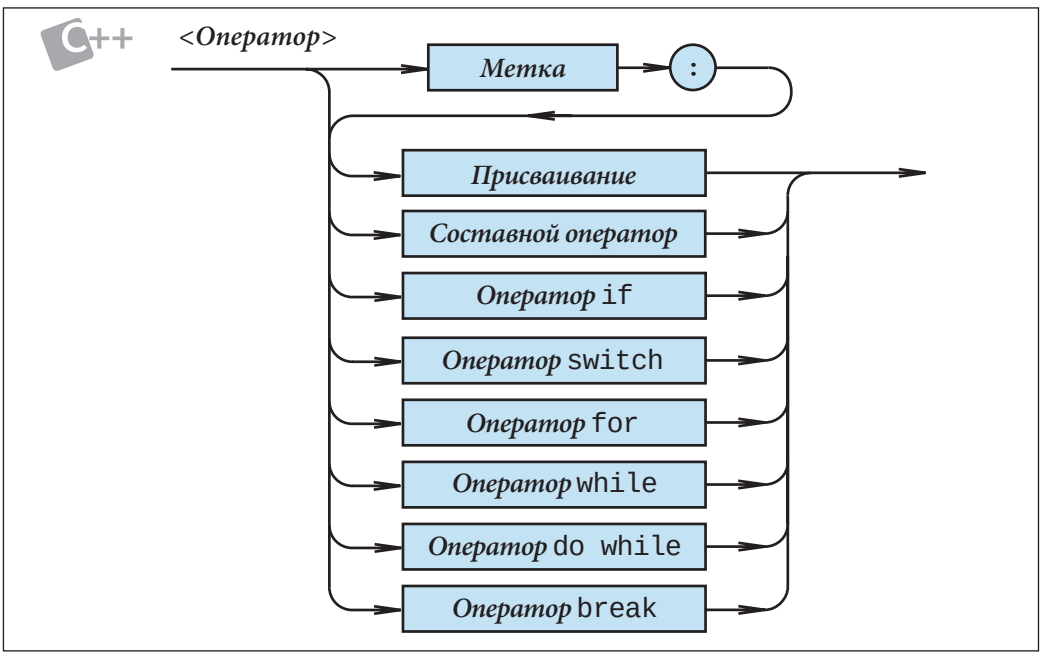


Рис. 3.1. Синтаксическая диаграмма <*Оператор*>

В программах на языке ПАСКАЛЬ / С++ количество операторов в строке не ограничено, один оператор может занимать одну или более строк, а в одной строке может быть несколько операторов. В качестве разделителя операторов используют символ «;» (точка с запятой).

3.2. Выражения

Формулы для вычисления значений представляются на языке ПАСКАЛЬ/ С++ в виде **выражений.** Выражения состоят из операндов (констант, переменных, ссылок на функции) и операций. Операции классифицируются следующим образом:

ПАСКАЛЬ

<*Мультипликативная операция*> ::= * | / | **div** | **mod** | **and**
<*Аддитивная операция*> ::= + | – | **or**
<*Операция отношения*> ::= < | <= | = | >= | > | <> | **in**

С++

<*Мультипликативная операция*> ::= * | / | % | &&
<*Аддитивная операция*> ::= + | – | ||
<*Операция отношения*> ::= < | <= | == | >= | > | !=

Выражения состоят из факторов (множителей), термов (слагаемых) и простых выражений.

Фактор состоит из отдельных переменных, констант без знака, обозначений функций и др.

<*Фактор*> ::= <*Переменная*> | <*Константа без знака*> | <*Вызов функции*> | <*Отрицание*><*Фактор*> | (<*Выражение*>) | <*Конструктор множества*>

Примеры:

	ПАСКАЛЬ		С++
1)	`15`	*1)*	`15`
2)	`x`	*2)*	`x`
3)	`sin(x)`	*3)*	`sin(x)`
4)	`not p`	*4)*	`!p`

Терм имеет вид:

<*Терм*> ::= <*Фактор*> {<*Мультипликативная операция*> <*Фактор*>}

Примеры:

	ПАСКАЛЬ		С++
1)	`15`	*1)*	`15`
2)	`x`	*2)*	`x`
3)	`x*y*z`	*3)*	`x*y*z`
4)	`p and q`	*4)*	`p && q`

Под **простым выражением** понимается:

<Простое выражение> ::= [+|–]*<Терм>*{*<Аддитивная операция>* *<Терм>*}

Примеры:

ПАСКАЛЬ		C++	
1)	`+15`	*1)*	`+15`
2)	`p or q`	*2)*	`p || q`
3)	`sin(x)/3+cos(x)`	*3)*	`sin(x)/3+cos(x)`

В свою очередь **выражение** имеет вид:

<Выражение> ::= *<Простое выражение>* {*<Операция отношения>* *<Простое выражение>*}

Примеры:

ПАСКАЛЬ		C++	
1)	`-15`	*1)*	`-15`
2)	`15*x+sin(x)/3<>11`	*2)*	`15*x+sin(x)/3 != 11`
3)	`x+6>z-3.0`	*3)*	`x+6>z-3.0`

Пусть переменные `a` и `b` имеют целочисленный тип, `c` и `d` – логический (boolean), а `x` и `y` – вещественный. Значения этих переменных: `a = 2`, `b = 8`, `c = true`, `d = false`, `x = -2,5` и `y = 1,4`.

Операции в языке ПАСКАЛЬ		
Операции	**Описание**	**Примеры**
Унарные операции		
+ (знак)	Обозначает положительное число	`+2`, `+0.25`, `+1999`
– (знак)	Обозначает отрицательное число	`-2`, `-0.25`, `-1999`
`not`	Отрицание (логический оператор)	**`not(c)`** → результат `false`
Бинарные операции		
+	Сложение	`a+b` → результат 10 `a+x` → результат –0,5
–	Вычитание	`a-b` → результат –6 `a-x` → результат 4,5
*	Умножение	`a*b` → результат 16 `a*y` → результат 2,8
/	Деление	`a/b` → результат 0,25 `b/a` → результат 4,0
`div`	Частное без остатка двух целых чисел	`a` **`div`** `b` → результат 0 `b` **`div`** `a` → результат 4
`mod`	Остаток от деления нацело двух целых чисел	`a` **`mod`** `b` → результат 8 `b` **`mod`** `a` → результат 0

<	Проверяет, меньше ли первый операнд второго.	a<b → результат true b<a → результат false
<=	Проверяет, меньше ли первый операнд второго или равен ему.	a<=b → результат true b<=a → результат false
>	Проверяет, больше ли первый операнд, чем второй.	a>b → результат false b>a → результат true
>=	Проверяет, больше ли первый операнд второго или равен ему.	a>=b → результат false b>=a → результат true
=	Проверяет, равен ли первый операнд второму.	a=b → результат false a=x → результат false
<>	Проверяет, отличается ли первый операнд от второго.	a<>b → результат true b<>y → результат true
and	Логическое произведение. Если оба операнда истинны, результат тоже истинен.	c and d → результат false c and c → результат true d and d → результат false
or	Логическая сумма. Если хотя бы один операнд истинен, результат будет истинным.	c or d → результат true c or c → результат true d or d → результат false

Операции в C++		
Операции	**Описание**	**Примеры**
Унарные операции		
+ (знак)	Обозначает положительное число	+2, +0, 25, +1999
– (знак)	Обозначает отрицательное число	–2, –0, 25, –1999
!	Логическое отрицание	!(c) → результат false
++	**Инкремент.** Есть два способа увеличить переменную x: • Постинкремент: x++. Приращение производится после вычисления выражения, в котором появляется ++. • Предварительный инкремент: ++x. Приращение выполняется до вычисления выражения, в котором появляется ++.	++a+b → результат 11. Значение a увеличилось до вычисления выражения. (a++)+b → результат 10. Значение a увеличилось после вычисления выражения.
––	**Декремент.** Операция уменьшения переменной x может быть: • Постдекремент: x--. Уменьшение выполняется после вычисления выражения, в котором появляется --. • Предварительный декремент: --x. Уменьшение выполняется до вычисления выражения, в котором появляется --.	--a+b → результат 9. Значение a уменьшилось до вычисления выражения. (a--)+b → результат 10. Значение a уменьшилось после вычисления выражения.

Бинарные операции		
+	Сложение	a+b → результат 10 a+x → результат –0,5
–	Вычитание	a–b → результат –6 a–x → результат 4,5
*	Умножение	a*b → результат 16 a*y → результат 2,8
/	Деление. Если хотя бы один операнд вещественный, то возвращаемый результат будет результатом деления «с запятой».	a/x → результат –0,8 y/x → результат –0,56
	Частное от деления. Если оба операнда являются целыми числами, то возвращаемый результат будет целым числом от их деления.	a/b → результат 0 b/a → результат 4
%	Modulo. Остаток от деления нацело двух целых чисел.	a % b → результат 8 b % a → результат 0
<	Проверяет, меньше ли первый операнд второго.	a<b → результат true b<a → результат false
<=	Проверяет, меньше ли первый операнд второго или равен ему.	a<=b → результат true b<=a → результат false
>	Проверяет, больше ли первый операнд, чем второй.	a>b → результат false b>a → результат true
>=	Проверяет, больше ли первый операнд второго или равен ему.	a>=b → результат false b>=a → результат true
==	Проверяет, равен ли первый операнд второму.	a==b → результат false a==x → результат false
!=	Проверяет, отличается ли первый операнд от второго.	a!=b → результат true b!=y → результат true
&&	Логическое произведение. Если оба операнда истинны, результат будет истинным.	c && d → результат false c && c → результат true d && d → результат false
||	Логическая сумма. Если хотя бы один операнд истинен, результат будет истинным.	c || d → результат true c || c → результат true d || d → результат false
Тернарные операции		
?	Операция ? имеет форму: <exp1> ? <exp2>:<exp3> где <exp1> логическое выражение. Когда <exp1> истина (true), результат операции ? равен <exp2>, а когда ложь(false), результат операции ? равен <exp3>. <exp1> и <exp1> должны иметь результаты одинакового типа либо совместимых типов.	a % 2 == 0? "par" : "impar" → результат "par" x < 0 ? x*2 : x-2 → результат x *2=–5 a < b ? b : a → результат 8 a < b ? a : b → результат 2

Тип результата вычисления выражения определяется типом операндов и соответствующими операциями. Эти правила будут изучены в следующих параграфах.

Синтаксические диаграммы рассмотренных грамматических единиц приведены на рис. 3.2 и *3.3*.

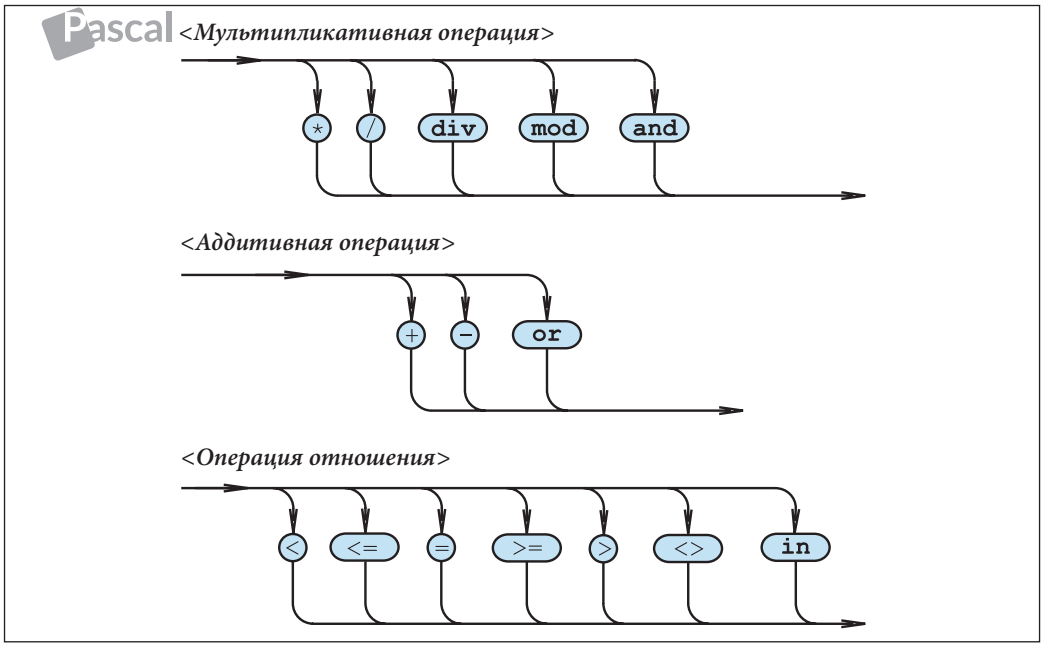


Рис. 3.2. Синтаксические диаграммы операций, язык ПАСКАЛЬ

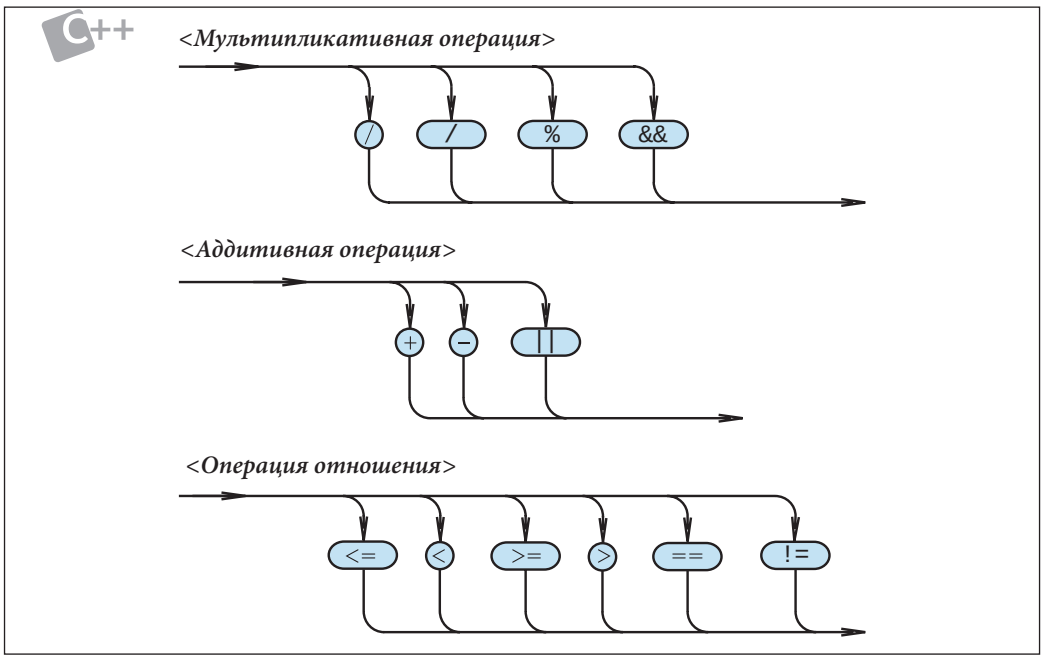


Рис. 3.2.* Синтаксические диаграммы операций, язык C++

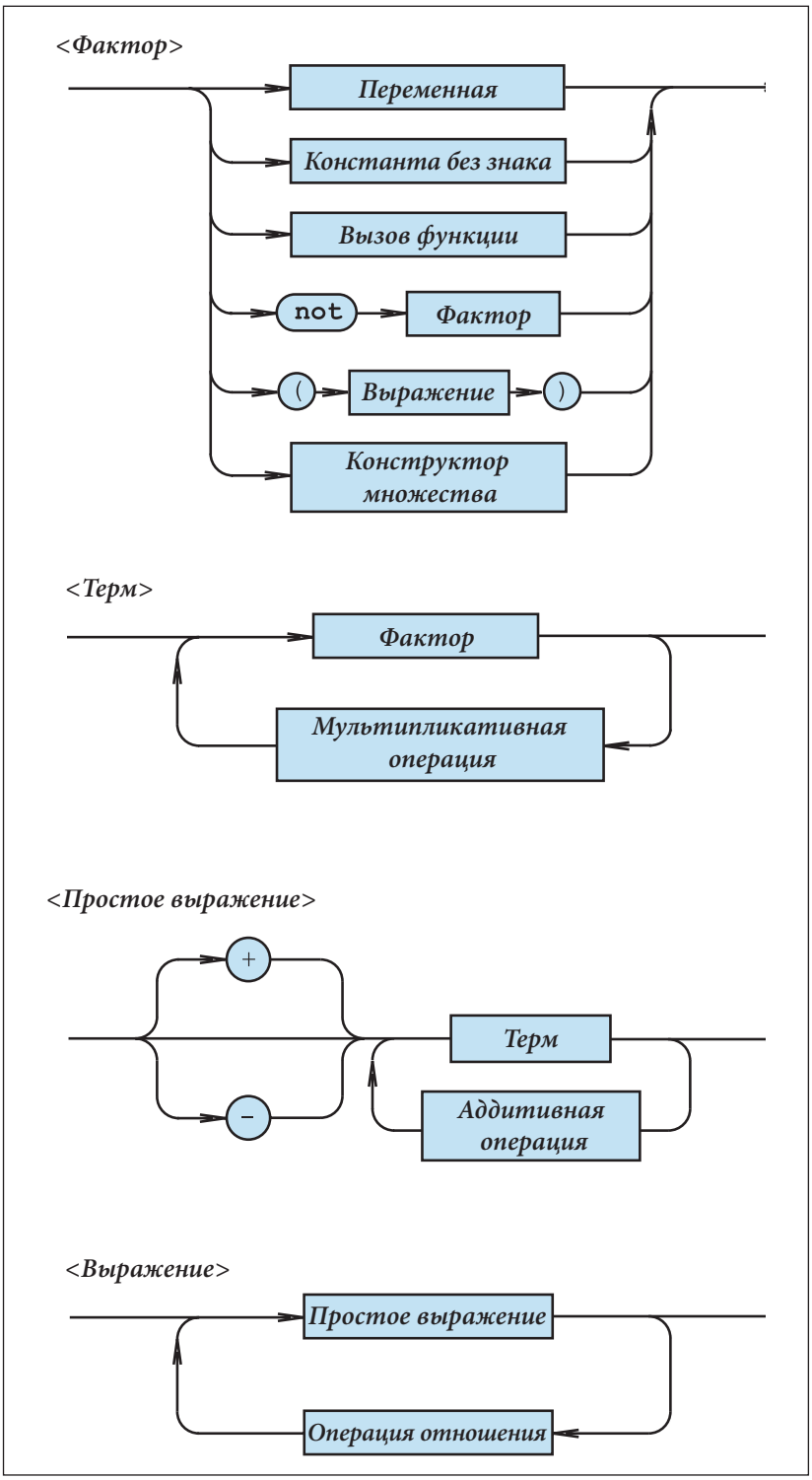


*Рис. 3.3**. Синтаксические диаграммы для определения выражений

Из синтаксических диаграмм для определения выражений видно, что они определены рекурсивно:

• выражение состоит из операнда или операции, примененной к операндам;

• операнд состоит из литерала (англ. *literal* - фиксированное значение), идентификатора переменной или записанного в круглых скобках выражения.

Например, выражение `a*(b+1)` состоит из операции умножения *, примененной к операндам a и (`b+1`), первый операнд – это идентификатор, второй – это выражение `b+1`. В свою очередь, это выражение состоит из операции сложения `+`, применяемой к идентификатору и литералу.

Выражение, взятое в скобки, превращается в фактор. Факторы могут образовывать новые термы, простые выражения, выражения и т.д.

Примеры:

	Математическая запись	Запись на языке ПАСКАЛЬ	Запись на языке С++
1)	$\frac{a+b}{c+d}$	`(a+b)/(c+d)`	`(a+b)/(c+d)`
2)	$\frac{-b+\sin(b-c)}{2a}$	`(-b+sin(b-c))/(2*a)`	`(-b+sin(b-c))/(2*a)`
3)	$-\frac{1}{xy}$	`-1/(x*y)`	`-1/(x*y)`
4)	$(p<q)\ \&\ (r>s)$	`(p<q)and(r>s)`	`(p<q)&&(r>s)`
5)	$\overline{x \vee y}$	`not (x or y)`	`!(x || y)`
6)	$\frac{1}{a+b}>\frac{1}{c+d}$	`1/(a+b)>1/(c+d)`	`1/(a+b)>1/(c+d)`

Вызов функции может появляться везде, где допустимо присутствие константы или переменной (*рис. 3.3*). Языки программирования содержат ряд **стандартных функций**, известных любой программе. Эти функции представлены в *таблице 3.1.*

Таблица 3.1

Предопределенные функции языков ПАСКАЛЬ и С++

Математическая запись	Запись на языке ПАСКАЛЬ	Запись на языке С++
Абсолютное значение $\|x\|$	`abs(x)`	`abs(x)` или `fabs(x)`
Синус sin *x*	`sin(x)`	`sin(x)`
Косинус cos *x*	`cos(x)`	`cos(x)`
Арктангенс arctg *x*	`arctan(x)`	`atan(x)`
Квадрат *x* x^2	`sqr(x)`	`pow(x,2)`
Степень *x* x^n	–	`pow(x,n)`

Степень 2 2^x	–	`exp2(x)`
Квадратный корень $\sqrt{x}$	`sqrt(x)`	`sqrt(x)`
Экспоненциальная функция e e^x	`exp(x)`	`exp(x)`
Натуральный логарифм $\ln x$	`ln(x)`	`log(x)`
Логарифм по основанию 10 числа x $\log_{10} x$	–	`log10(x)`
Округление вещественного числа x	`round(x)`	`round(x)`
Целая часть вещественного числа x	`trunc(x)`	`trunc(x)`
Округление x к ближайшему целому, большему, чем x	–	`ceil(x)`
Усечение x до ближайшего целого числа, меньшего, чем оно	–	`floor(x)`
Четность числа i (`false` для четного i и `true` в противном случае))	`odd(i)`	–
Порядковый номер элемента v	`ord(v)`	–
Предшествующий элемент	`pred(v)`	–
Следующий элемент	`succ(v)`	–
Символ с порядковым номером i	`chr(i)`	–
Признак конца файла	`eof(f)`	`EOF`
Признак конца строки	`eoln(f)`	–

ПРИМЕЧАНИЕ

Для работы математических функций в программу C++ должна быть включена библиотека `cmath`.

Вопросы и упражнения

❶ **ПРИМЕНИТЕ!** Перепишите следующие выражения согласно правилам изучаемого языка:

a) a^2+b^2;

b) $a^2 + 2ab + b^2$;

c) $(a + b)^2$;

d) $v_0 t + \frac{at^2}{2}$;

e) $\frac{-b+\sqrt{b^2-4ac}}{2a}$;

f) $\cos\alpha + \cos\beta$;

g) $\cos(\alpha + \beta)$;

h) $2\pi r$;

i) πr^2;

j) $x_1x_2 \vee x_3x_4$;

k) $\overline{x_1 \vee x_2}$;

l) $|x| < 3$;

m) $|z| < 6 \;\&\; |q| > 3{,}14$;

n) $x > 0 \;\&\; y > 8 \;\&\; R < 15$.

❷ **ОБРАТИТЕ ВНИМАНИЕ!** Укажите на синтаксических диаграммах *рис. 3.3* пути, которые соответствуют следующим выражениям:

a) `x`

b) `3.14`

c) `sin(x)`

d) **ПАСКАЛЬ:** `not q`
C++: `!q`

e) `(+3.14)`

f) `x>2.85`

g) **ПАСКАЛЬ:** `sqr(b)-4*a*c>0`
C++: `pow(b,2)-4*a*c>0`

h) **ПАСКАЛЬ:** `(a>b)and(c>d)`
C++: `(a>b)&&(c>d)`

❸ **ПРИМЕНИТЕ!** Переведите выражения в обычные:

a) **ПАСКАЛЬ:** `sqr(a)+sqr(b)`
C++: `pow(a,2)+pow(b,2)`

b) `2*a*(b+c)`

c) `sqrt((a+b)/(a-b))`

d) `exp(x+y)`

e) `cos(ALFA-BETA)`

f) **ПАСКАЛЬ:** `sqr(a+b)/(a-b)`
C++: `pow(a+b,2)/(a-b)`

g) **ПАСКАЛЬ:** `(x>0) or (q<p)`
C++: `(x>0) || (q<p)`

h) **ПАСКАЛЬ:** `not(x and y)`
C++: `!(x && y)`

❹ **ОБРАТИТЕ ВНИМАНИЕ!** Какие из следующих выражений на языке, который вы изучаете, содержат ошибки? Для ответа воспользуйтесь синтаксическими диаграммами *рис. 3.3*.

a) `((((+x))))`

b) `((((x))))`

c) `sinx+cosx`

d) `sqr(x)+sqr(y)`

e) `a<<b or c>d`

f) `not not not p`

g) `q and not p`

h) `a!=b || c>d`

i) `!!! p`

j) `Q && ! p`

k) `not q and p`

l) `a+-b`

m) `sin(-x)`

n) `sin-x`

o) `cos(x+y)`

p) `sin(abs(x)+abs(y))`

q) `sqrt(-y)`

r) `!q && p`

s) `x>0 or q<p`

t) `EXP(x)`

3.3. Вычисление выражений

Под вычислением некоторого выражения понимается нахождение его значения. Результат вычисления зависит от значений операндов и от операций, применяемых к ним. Правила вычисления выражений – такие же, как и в математике:

– операции выполняются в соответствии с их приоритетом;
– в случае одинаковых приоритетов операции выполняются слева направо;
– первыми вычисляются значения выражений, заключенных в скобках.

Приоритеты операций приведены в таблице 3.2.

Таблица 3.2

Приоритеты операций языка ПАСКАЛЬ и С++

Категория	Операции		Приоритет
	ПАСКАЛЬ	С++	
Унарные операции	+, -(знак), **not**, @	+, - (знак), !, &, - -, ++	Первый (наивысший)
Мультипликативные операции	*, /, **div**, **mod**, **and**	*, /, %	Второй
Аддитивные операции	+, -, **or**	+, -	Третий
Операции отношения	<, <=, =, >=, >, <>, **in**	*a*) <, <=, >=, > *b*) ==, !=	Четвертый и пятый
Логические операции	-	*a*) && *b*) ||	Шестой и седьмой (наименьший)

Пример:

Пусть x и y целые, а m и n вещественные числа, имеющие значения $x = 2$, $y = 6$, $m = 5$ и $n = 1{,}5$. Тогда:

1) `2*x+y = 2*2+6 = 4+6 = 10`.

2) `2*(x+y) = 2(2+6) = 2·8 = 16`.

3) `x+y/x–y = 2+6/2–6 = 2+3–6 = 5–6 = –1`.

4) `(x+y)/x–y = (2+6)/2–6 = 8/2–6 = 4–6 = –2`.

5) **ПАСКАЛЬ:** `x/y = 2/6 = 0.3333`.

*5)** **С++:** `x/y = 2/6 = 0`. В случае целочисленных операндов выполняется целочисленное деление, а результат операции / является частным от целочисленного деления.

6) **ПАСКАЛЬ:** `m/n = 5/1.5 = 3.3333`.

*6)** **С++:** `m/n = 5/1,5 = 3.3333`. В случае операндов вещественного типа (`float`, `double`, `long double`) выполняется десятичное деление, а результат операции / является результатом деления «с запятой».

7) **ПАСКАЛЬ:** `x+y/(x–y) = 2+6/(2–6) = 2+6/(–4) = 2+(–1.5) = 0.5;`

*7)** **С++:** `x+y/(x–y) = 2+6/(2–6) = 2+6/(–4) = 2+(–1) = 1;`

8) `n+x/(m–y) = 1,5+2/(5–6) = 1.5+2/(–1) = 1.5-2 = -0.5;`

9) `x+y<15 = 2+6<15 = 8<15 = true;`

10) **ПАСКАЛЬ:** `(x+y<15)`**`and`**`(x>3) = (2+6<15)`**`and`**`(2>3) = (8<15 `**`and`**` (2>3) = = true `**`and`**` false = false`.

*10)** **С++:** `(x+y<15)&&(x>3) =(2+6<15)&&(2>3) = (8<15)&&(2>3) = true && false = false`.

Отметим, что составные части выражения (*рис. 3.3*) вычисляются в следующем порядке:

1) факторы; 2) термы; 3) простые выражения; 4) само выражение.

Текущее значение выражения можно вывести на экран так:

ПАСКАЛЬ	C++
`writeln(`*<Выражение>*`);`	`cout << `*<Выражение>*` << endl;`

Приведенные ниже программы выводят на экран результаты вычисления выражений `x*y+z` и `x+y<z-1.0`. Текущие значения переменных `x`, `y` и `z` считываются с клавиатуры.

ПАСКАЛЬ

```
Program P39;
 { Вычисление выражений }
var x, y, z : real;
begin
  writeln('Введите
вещественные числа x, y, z:');
  readln(x, y, z);
  writeln(x*y+z);
  writeln(x+y<z-1.0);
end.
```

C++

```
// Программа P39
#include <iostream>
using namespace std;
// Вычисление выражений
int main()
{
double x, y, z;
cout<<"Введите вещественные
числа x, y, z:\n";
cin>>x>>y>>z;
cout<<x*y+z<<endl;
cout<<(x+y<z-1.0);
return 0;
}
```

Вопросы и упражнения

❶ **ПРИМЕНИТЕ!** Пусть `x=1`, `y=2` и `z=3`. Вычислите следующие выражения:

a) `x+y+2*z`

b) `(x+y+2)*z`

c) `x*y+y*z`

d) `x*(y+y)*z`

e) `(x*y+y)*z`

f) `x*(y+y)*z`

g) `x*y<y*z`

h) **ПАСКАЛЬ:** `(x>y) or (6*x>y+z)`
C++: `(x>y) || (6*x>y+z)`

i) **ПАСКАЛЬ:** `not(x+y+z>0)`
C++: `!(x+y+z>0)`

j) **ПАСКАЛЬ:** `not(x+y>0) and not(z<0)`
C++: `!(x+y>0) && !(z<0)`

❷ Сформулируйте правила вычисления выражений на языке, который вы изучаете.

❸ Назовите приоритеты операций языка, который вы изучаете.

❹ Укажите порядок вычисления компонент выражений в языке, который вы изучаете.

❺ **РАЗРАБОТАЙТЕ!** Напишите программу, которая вычисляет выражения c) и g) из задания 1. Текущие значения вещественных переменных x, y и z считываются с клавиатуры.

❻ Какие значения будут получены при вычислении выражений

ПАСКАЛЬ

```
a) sqr(2)+2*sgr(2)
b) (1+sqrt(4))/3
c) trunc(sqrt(20))
d) trunc(27%4) + round(27.3) / 4
e) trunc( -47%5 -
   sqrt(2))/round(19.3)/3
```

C++

```
a) pow(2,2)+pow(2,3)
b) 1+sqrt(4))/3
c) floor(sqrt(20))
d) floor(27%4) + ceil(27.3) / 4
e) floor( -47%5 -
   sqrt(2))/ceil(19.3)/3
```

3.4. Тип выражений

Каждому выражению, в зависимости от множества его значений, ставится в соответствие определенный тип данных. Согласно концепции данных, реализованной в языках ПАСКАЛЬ и С++, тип выражения определяется типами операндов и операциями, применяемыми к ним. Таким образом, тип выражения можно определить, не вычисляя его значения.

Типы результатов операций указаны в *таблице 3.3.*

Таблица 3.4. содержит тип результатов стандартных функций языка ПАСКАЛЬ/С++.

Таблица 3.3.

Типы результатов, предоставляемых операциями

ПАСКАЛЬ		
Операция	**Тип операндов**	**Тип результата**
+, –, *	`integer`	`integer`
	один `integer`, другой `real`	`real`
/	`integer` или `real`	`real`
div	`integer`	`integer`
mod	`integer`	`integer`
not, **and**, **or**	`boolean`	`boolean`
<, <=, =, >=, >, <>	Идентичные типы	`boolean`
	Совместимые типы	`boolean`
	один `integer`, другой `real`	`boolean`

С++		
Операция	**Тип операндов**	**Тип результата**
+, –, *	**int**	**int**
	один **int**, другой **double**	**double**
++, ––	Идентичные либо совместимые типы	Идентичный с типом операндов
/	**int**	**int**
	double	**double**
%	**int**	**int**
!, &&, ||	**bool**	**bool**
<, <=, ==, >=, >, !=	Идентичные типы	**bool**
	Совместимые типы	**bool**
	один **int**, другой **double**	**bool**

Таблица 3.4

Типы результатов, предоставляемых операциями

ПАСКАЛЬ		
Функция	**Тип аргумента**	**Тип результата**
`abs(x)`	`integer` или `real`	Совпадает с типом `x`
`sin(x)`	`integer` или `real`	`real`
`cos(x)`	`integer` или `real`	`real`
`arctan(x)`	`integer` или `real`	`real`
`sqr(x)`	`integer` или `real`	Совпадает с типом `x`
`sqrt(x)`	`integer` или `real`	`real`
`exp(x)`	`integer` или `real`	`real`
`ln(x)`	`integer` или `real`	`real`
`round(x)`	`real`	`integer`
`trunc(x)`	`real`	`integer`
`odd(i)`	`integer`	`boolean`
`ord(v)`	Порядковый	`integer`
`pred(v)`	Порядковый	Совпадает с типом `v`
`succ(v)`	Порядковый	Совпадает с типом `v`
`chr(i)`	`integer`	`char`
`eof(f)`	Файловый	`boolean`
`eoln(f)`	Файловый	`boolean`

C++		
Функция	**Тип аргумента**	**Тип результата**
`abs(x)` или `fabs(x)`	**`int`** или **`double`**	**`double`**
`sin(x)`	**`int`** или **`double`**	**`double`**
`cos(x)`	**`int`** или **`double`**	**`double`**
`atan(x)`	**`int`** или **`double`**	**`double`**
`pow(x,2)`	**`int`** или **`double`**	**`double`**
`pow(x,n)`	**`int`** или **`double`** если `x<0`, то `n` должно быть целым	**`double`**
`sqrt(x)`	**`int`** или **`double`**	**`double`**
`exp(x)`	**`int`** или **`double`**	**`double`**
`log(x)`	**`double`**, `x>0`	**`double`**

round(x)	**double**	**double**
trunc(x)	**double**	**double**
log10(x)	**double**	**double**
ceil(x)	**double**	**double**
floor(x)	**double**	**double**
EOF	Файловый	**bool**

В языке ПАСКАЛЬ независимо от типа операндов операция / (деление) возвращает результат типа real, а операции отношения возвращают результат типа boolean.

В языке C++ операция / предоставит целочисленный результат, если оба операнда целые числа, и результат вещественного типа, если хотя бы один операнд имеет вещественный тип, а операции отношения - только результаты типа **bool**.

Для того чтобы определить тип выражения, факторы, термы и простые выражения рассматриваются в порядке их вычисления, причем тип каждой составной части определяется с помощью *таблиц 3.3 и 3.4.*

Например, рассмотрим выражение:

```
6*i<sin(x/y)
```

в котором i – переменная типа integer/**int**, а x и y – переменные типа real/**double**.

Определим тип составных частей и всего выражения в порядке выполнения вычислений:

1) `6*i` — результат типа integer/**int**;

2) `x/y` — результат типа real/**double**;

3) `sin(x/y)` — результат типа real/**double**;

4) `6*i < sin(x/y)` — результат типа boolean/**bool**.

Таким образом, рассматриваемое выражение относится к типу boolean.

В зависимости от типа выражения разделяются на:

– арифметические (integer или real в ПАСКАЛЕ, **int** или **double** в C++);

– порядковые (integer, boolean, char, перечисляемые, в ПАСКАЛЕ, **int**, **bool**, **char**, перечисляемые в C++);

– логические (boolean в ПАСКАЛЕ, **bool** в C++).

Вопросы и упражнения

❶ Как определяется тип выражения?

❷ **ОБРАТИТЕ ВНИМАНИЕ!** При наличии объявлений:

ПАСКАЛЬ	C++
var x, y : real; i, j : integer; p, q : boolean; r : char; s : (A, B, C, D, E, F, G, H);	**double** x, y; **int** i, j; **bool** p, q; **char** r; **enum** s {A, B, C, D, E, F, G, H};

определите типы следующих выражений:

	ПАСКАЛЬ		C++
a)	`i mod 3`	*a)*	`i % 3`
b)	`i/3`	*b)*	`i/3`
c)	`i mod 3 > j div 4`	*c)*	`i % 3 > j / 4`
d)	`x+y/(x-y)`	*d)*	`x+y/(x-y)`
e)	`not(x<i)`	*e)*	`!(x<i)`
f)	`sin(abs(i)+abs(j))`	*f)*	`sin(abs(i)+abs(j))`
g)	`sin(abs(x)+abs(y))`	*g)*	`sin(abs(x)+abs(y))`
h)	`p and (cos(x)<=sin(y))`	*h)*	`p && (cos(x)<=sin(y))`
i)	`sqr(i)-sqr(j)`	*i)*	`pow(i,2)-pow(j,2)`
j)	`sqr(x)-sqr(y)`	*j)*	`pow(x,2)-pow(y,2)`
k)	`trunc(x)+trunc(y)`	*k)*	`trunc(x)+trunc(y)`
l)	`chr(i)`	*l)*	`char(i)`
m)	`ord(r)`	*m)*	`int(r)`
n)	`ord(s)>ord(r)`	*n)*	`ceil(x)>floor(y)`
o)	`pred(E)`	*o)*	`2*floor (x+y)`
p)	`(-x+sin(x-y))/(2*i)`	*p)*	`(-x+sin(x-y))/(2*i)`

❸ **ОБРАТИТЕ ВНИМАНИЕ И ПРИМЕНИТЕ!** Тип выражения можно узнать из текстовой формы результатов, выведенных на экран с помощью оператора

В языке ПАСКАЛЬ
`writeln(`<*Выражение*>`)`

В языке C++
`cout<<`*Выражение*`<<endl;`

Примеры::

	Выведенный на экран результат	Тип выражения в ПАСКАЛЕ	Тип выражения в C++
1)	`100`	`integer`	**`int`**
2)	`1.0000000000E+02`	`real`	**`double`**
3)	`A`	`char`	**`char`**

Напишите соответствующие программы и на основании текстовой формы результатов, выведенных на экран, определите типы следующих выражений:

	ПАСКАЛЬ		C++
a)	`1+1.0`	*a)*	`1+1.0`
b)	`1/1+1`	*b)*	`1/1+1`
c)	`9*3 mod 4`	*c)*	`9*3 % 4`
d)	`4*x>9*y`	*d)*	`4*x>9*y`
e)	`chr(65)`	*e)*	`char(65)`

f)	`not(x>y)`	*f)*	`!(x>y)`
g)	`pred(9)>succ(7)`	*g)*	`9 > 7`
h)	`15 div ord(3)`	*h)*	`15 / 3`
i)	`trunc(x)+round(6*y)`	*i)*	`trunc(x)+round(6*y)`
j)	`sqr(3)-sqrt(16)`	*j)*	`pow(3,2)-sqrt(16)`

где переменные x и y являются переменными вещественного типа.

❹ **ОБРАТИТЕ ВНИМАНИЕ!** Даны описания:

ПАСКАЛЬ	C++

`type T1=1..10;`
`T2=11..20;`
`T3='A'..'Z';`
`T4=(A, B, C, D, E, F, G, H);`
`var i : T1;`
`j : T2;`
`k : T3;`
`m : 'C'..'G';`
`n : T4;`
`p : C..G;`
`q : boolean;`
	`typedef int T1;`
	`typedef int T2;`
	`typedef char T3;`
	`enum T4 {A, B, C, D, E, F, G, H};`
	`T1 i;`
	`T2 j;`
	`T3 k;`
	`char m;`
	`T4 n;`
	`T4 p;`
	`bool q;`

Найдите типы следующих выражений:

	ПАСКАЛЬ		C++
a)	`i-j`	*a)*	`i-j`
b)	`i div j`	*b)*	`i / j`
c)	`6.3*i`	*c)*	`6.3*i`
d)	`cos(3*i-6*j)`	*d)*	`cos(3*i-6*j)`
e)	`4*i>5*j`	*e)*	`4*i>5*j`
f)	`k<m`	*f)*	`k<m`
g)	`k<>m`	*g)*	`k!=m`
h)	`chr(i)`	*h)*	`char(i)`
i)	`ord(k)`	*i)*	`int(k)`
j)	`ord(m)`	*j)*	`int(m)`
k)	`n>p`	*k)*	`n>p`
l)	`ord(n)`	*l)*	`int(n)`
m)	`succ(n)`	*m)*	`n+1`
n)	`pred(p)`	*n)*	`p-1`
o)	`ord(p)`	*o)*	`int(p)`
p)	`ord(k)>ord(m)`	*p)*	`int(k)> int(m)`
q)	`(i>j) and q`	*q)*	`(i>j) && q`

	ПАСКАЛЬ		C++
r)	`not(i+j>0) or q`	*r)*	`!(i+j>0) || q`

❺ Какого типа будет результат выражений?

	ПАСКАЛЬ		C++
a)	`sqrt(2)+2*sgr(2)`	*a)*	`pow(2.5,2)`
b)	`(1+sqr(4))*3`	*b)*	`(1+pow(4,2))*3`
c)	`trunc(sqrt(20))`	*c)*	`floor(sqrt(20))`
d)	`trunc(27%4) + round(27.8)`	*d)*	`floor(27%4) + ceil(27.8)`
e)	`(trunc(-47%3) - round(19.5))*2`	*e)*	`(floor(-47%5) - ceil(19.5))*2`

❻ **ПРОАНАЛИЗИРУЙТЕ!** Какое из следующих выражений имеет значение true (1) тогда и только тогда, когда вещественное число, хранящееся в переменной x, находится в диапазоне (-2,2)? Укажите порядок, в котором выполняются операции.

	ПАСКАЛЬ		C++
a)	`x*x-4<=0`	*a)*	`x*x-4<=0`
b)	`4-x*x>0`	*b)*	`4-x*x>0`
c)	`(2<x) and (x<-2)`	*c)*	`(2<x)&&(x<-2)`
d)	`(x-2)*(x+2)>0`	*d)*	`(x-2)*(x+2)>0`

❼ Пусть $x = 8$ и $y = 6$. Какое из следующих выражений имеет значение `false`? Укажите порядок, в котором выполняются операции.

	ПАСКАЛЬ		C++
a)	`3*x-4*y= 0`	*a)*	`3*x-4*y==0`
b)	`(x+y)/2 > x mod y+1`	*b)*	`(x+y)/2 > x%y+1`
c)	`not (x/2+2 = y)`	*c)*	`!(x/2+2 == y)`
d)	`x-y+3 <> 0`	*d)*	`x-y+3 != 0`

❽ Пусть переменная `x` вещественного типа. Какое из следующих выражений имеет значение `true` тогда и только тогда, когда вещественное число, хранящееся в переменной `x`, принадлежит диапазону (5, 8)?

	ПАСКАЛЬ		C++
a)	`(x<=8) or (x>5)`	*a)*	`(x<=8) || (x>5)`
b)	`(x>8) or (x<=5)`	*b)*	`(x>8) || (x<=5)`
c)	`(x<=8) and (x>5)`	*c)*	`(x<=8) && (x>5)`
d)	`(x<8) and (x>=5)`	*d)*	`(x<8) && (x>=5)`

3.5. Преобразования типов в языке C++

При вычислении выражений в языке C++ могут происходить неявные или явные преобразования типов.

Преобразование типов (конверсия) по умолчанию выполняется автоматически. При присваивании (*v* = *Выражение*) тип правой части присваивания преобразуется в тип левой части, который и является типом результата.

При изучении параграфа 2.9 мы столкнулись с некоторыми ситуациями преобразования типов.

Примеры преобразований типов по умолчанию:

```
int i;
char c='c';
float f;
char c1, c2;
c1=100;    /* 100 автоматически преобразуется в char, c1
              получит значение символа с кодом 100, т. е. 'd'*/
i=c;       /*  i будет равно 99 (код ASCII символа 'c')  */
i=2.9;     /* 2.9 по умолчанию преобразуется к целому int, значит i
           будет равно 2 */
c2=c1;     /* c2 будет равно 'd'  */
f='A';     /* f будет равно 65.0 ( код ASCII символа 'A')  */
i=30000+c;/* выражение вычисляется на области значений
           int, i будет равно 30099  */
```

ВНИМАНИЕ! Возможны следующие моменты:

- потеря точности (**double** -> **float** -> **long int**);
- потеря значимых битов (**long** ->**int**);
- неопределенности.

Оператор явного преобразования типа используется в случаях, когда требуется, чтобы тип операнда (выражения) был другим, чем тип по умолчанию. Оператор имеет вид:

(<*Тип*>) <*Выражение*> или <*Тип*> (<*Выражение*>)

Такие конструкции называются выражениями *cast* (*cast* – нечто).

Первая форма – традиционная, вторая – специфическая. В специфической форме имя предопределенного или определяемого пользователем типа данных используется как «функция преобразования» с указанием в качестве параметра типа преобразования.

Поскольку в обоих режимах все типы данных обрабатываются одинаково, рекомендуется использовать традиционную форму.

Примеры явных преобразований типов:

```
float r;
long l;
int i, x=7, y=3, z=-2;
float a=2.1,b=3.0, c=-1.5, m;
double d;
r=5/2;          /* деление производится в области int, значит.
                   r будет равно 2 */
r=(float)5/2;   /* r будет равен 2.5, т.к. первый операнд
                конвертируется к типу float, значит вычисления
                производятся в области вещественных чисел */
```

```
r=(float)(5/2);          /* r будет равен 2, т.к. результат
                         целочисленного деления преобразуется к
                         типу float, значит, вычисления
                         производятся в области int */
l=(long)('A' + 1.0);      // l=66
i=(int)(b*b-4*a*c);       // i=21
d=(double)(x+y)/z;        // d=-5.0
m=x*y/z;                  // m=-10
m=(float)x*y/z;          // m=-10.5
m=x/(float)2;            // m=3.5
```

3.6. Оператор присваивания

Pascal

Оператор присваивания имеет вид:

<*Переменная*> := <*Выражение*>

При выполнении оператора присваивания происходит следующее:
а) вычисляется выражение, стоящее в правой части;
б) полученное значение присваивается переменной, стоящей в левой части.

Примеры:

1) `x:=1`

2) `y:=x+3`

3) `z:=sin(x)+cos(y)`

4) `p:=not q`

5) `q:=(a<b) or (x<y)`

6) `c:='A'`

Отметим, что символ ":=" (читается «присвоить») означает присваивание и не следует путать его с операцией отношения "=" (равно).

Присваивание возможно только тогда, когда переменная и результат вычисления выражения совместимы с точки зрения присваивания. В противном случае возникает ошибка.

Переменная и результат вычисления выражения являются совместимыми с точки зрения присваивания, если справедливо одно из следующих утверждений:

1) тип переменной и тип результата идентичны;
2) тип результата является интервалом типа переменной;
3) оба типа являются интервалами одного и того же типа, а тип результата принадлежит интервальному типу переменной;
4) тип переменной – `real`, а тип результата – `integer` или его интервал.

В качестве примера рассмотрим следующую программу:

```
Program P40;
 { Совместимость с точки зрения присваивания }
type T1=1..10; { интервал типа integer }
     T2=5..15; { интервал типа integer }
```

```
var i : T1;
    j : T2;
    k, m, n : integer;
    x : real;
begin
  write('k=');
  readln(k);
  i:=k;          { верно для 1<=k<=10 }
  write('m=');
  readln(m);
  j:=m;          { верно для 5<=m<=15 }
  write('n=');
  readln(n);
  i:=n+5;        { верно для -4<=n<=5 }
  j:=n+2;        { верно для 3<=n<=13 }
  x:=i+j;
  writeln('x=', x);
end.
```

Программа будет работать без ошибок только при следующих значениях на входе:

$$1 \le k \le 10;\ 5 \le m \le 1\ 5;\ 3 \le n \le 5.$$

Очевидно, что в программе `P39` операторы вида

```
k:=x        i:=x+1        j:=sin(i)
```

являются ошибочными, так как тип выражений `x`, `x+1`, `sin(i)` – `real`, а тип переменных `k`, `i`, `j` – `integer` или его интервал.

Изучаемый оператор имеет вид:

<Переменная> = <Выражение>

Выполнение оператора присваивания предполагает:
a) вычисление выражения из правой части;
b) присвоение полученного значения переменной из левой части оператора.

Примеры:

1) `x=1`

2) `y=x+3`

3) `z=sin(x)+cos(y)`

4) `p= !q`

5) `q=(a<b) || (x<y)`

6) `c='A'`

Запомните, что символ "=" (читается "присвоить") означает присваивание и не следует путать его с операцией отношения "==" (равно).

Присваивание происходит, если переменная и результат вычисления выражения совместимы с точки зрения присваивания. В противном случае будет выдано сообщение об ошибке.

Примеры:
Даны объявления:

```
int x, y;
float m, n;
```

и присваивания:

`x=2;`	Значение 2 присваивается переменной `x`.
`y=x+2;`	Переменной y присваивается значение 2 + 2 = 4.
`m=x/y;`	Переменной m присваивается значение 0, поскольку в процессе вычисления выражения в правой части присваивания происходит целочисленное деление 2/4.
`n=m*2+x-5;`	Переменной n присваивается значение 0 × 2 + 2 - 5 = -3.
`x=y+12;`	Значение 4 + 12 = 16 присваивается переменной x.
`x=(m>n);`	Переменной x присваивается значение (0> -3), т.е. `true`.

В языке C++ используются и операции составного присваивания +=, -=, /=, *=, %=, позволяющие более компактное написание программ. Форма операторов присваивания с использованием таких операций следующая:

<Переменная> <Операция составного присваивания> = <Выражение>

Примеры:
Даны объявления:

```
int x=6, y=3;
```

и составные присваивания:

`x+=12;`	Эквивалентно `x = x + 12`. Переменной `x` присваивается 18.
`y/=5;`	Эквивалентно `y = y / 5`. Поскольку операнды имеют тип **`int`**, выполняется деление нацело. Переменной `x` присваивается значение 0.
`x%=y;`	Эквивалентно `x = x% y`. Переменной x присваивается значение 0.
`y*=2;`	Эквивалентно `y = y * 2`. Переменной y присваивается значение 3.

Отметим, что выражение в правой части оператора присваивания также может быть оператор присваивания, то есть можно записывать связанные операторы присваивания.

Примеры:
Даны объявления:

```
int a=8, b=2, c=7, j=9, k=11;
```

и составные присваивания:

`a=b*=2;`	Эквивалентно `a=b*2`. Переменным `a` и `b` присваивается значение 4.
`a=b=a*b;`	Эквивалентно `a=b=8*2`. Переменным `a` и `b` присваивается значение 16.
`k%=j-=4;`	Сначала выполняется присваивание `j- = 4`. Следовательно, переменной `j` будет присвоено значение 5. Это значение используется для выполнения присваивания `k% = j`, поэтому переменной `k` будет присвоено значение 1.
`a=b=c=10;`	Каждой из переменных `a, b, c` будет присвоено значение 10.

В эту же категорию входят унарные операции инкремента (++) и декремента (--).

Вопросы и упражнения

❶ Как выполняется оператор присваивания?

❷ Объясните понятие “совместимость с точки зрения присваивания”.

❸ Даны следующие объявления:

ПАСКАЛЬ

```
type Zi = (L, Ma, Mi, J,
V,
     S, D);
     Culoare = (Galben,
     Verde, Albastru, Vio-
let);
var i, j, k : integer;
    z : Zi;
    c : Culoare;
    x : real;
```

C++

```
enum Zi {L, Ma, Mi, J, V, S, D};
enum Culoare {Galben, Verde,
     Albastru, Violet};
int i, j, k;
Zi z;
Culoare c;
double x;
```

Какие из следующих операторов являются правильными?

ПАСКАЛЬ		C++	
a)	`i:=12`	*a)*	`i=12`
b)	`j:=ord(i)`	*b)*	`j=int(i)`
c)	`x:=ord(z)+1`	*c)*	`x=int(z)+1`
d)	`k:=ord(x)+2`	*d)*	`k=int(x)+2`
e)	`c:=i+4`	*e)*	`c=i+4`
f)	`c:=Verde`	*f)*	`c=Verde`
g)	`z:=D`	*g)*	`z=D`
h)	`c:=Pred(Galben)`	*h)*	`c=Galben-1`
i)	`x:=Succ(z)`	*i)*	`x=z+1`
j)	`i:=Succ(c)`	*j)*	`i=c+1`

❹ Уточните, для каких значений переменной `j` нижеследующие программы будут выполняться без ошибок.

ПАСКАЛЬ

```
Program P41;
var i : -10..+10;
    j : integer;
begin
  write('j=');
  readln(j);
  i:=j+15;
  writeln('i=', i);
end.
```

C++

```
// Программа P41
#include <iostream>
using namespace std;
int main()
{
short int i;
int j;
cout<<"j=";  cin>>j;
i=j+15;   cout<<"i="<<i;
return 0;
}
```

❺ **ПРОАНАЛИЗИРУЙТЕ!** Дана следующая последовательность операторов программы на языке PASCAL/C++, в которой переменные x, y и z представляют целочисленные величины.

ПАСКАЛЬ	C++
`x:=y+z; z:=x-z; y:=z; z:=x-y;`	`x=y+z; z=x-z; y=z; z=x-y;`

После выполнения этого фрагмента программы некоторые из переменных `x, y, z` могли изменить свои значения. Выберите из списка, приведенного ниже, вариант, содержащий все переменные, значения которых останутся неизменными:

a) x; *d)* x и y;
b) y; *e)* x и z;
c) z; *f)* x, y и z.

3.7. Оператор вызова процедуры в языке ПАСКАЛЬ

ПРИМЕЧАНИЕ
В языке C++ существуют только подпрограммы типа функции. Для процедур используется специальная форма функций, которая будет изучаться в следующем классе. Следовательно, данный параграф предусмотрен только для учеников, изучающих язык ПАСКАЛЬ.

Процедура представляет собой подпрограмму, к которой в процессе выполнения программы можно обращаться произвольное число раз. Каждая процедура имеет свое имя, например: `readln, writeln, CitireDate, A15` и т. д. Язык ПАСКАЛЬ содержит ряд стандартных процедур, известных любой программе: `read, readln, write, writeln, get, put, new` и т. д. В дополнение к ним программист может создавать собственные процедуры.

Оператор вызова процедуры вызывает процедуру с соответствующим именем. Оператор имеет вид:

<*Вызов процедуры*> ::= <*Имя процедуры*> [<*Список фактических параметров*>]

<*Имя процедуры*> ::= <*Идентификатор*>

<*Список фактических параметров*> ::= (<*Фактический параметр*>
{, <*Фактический параметр*>})

Синтаксические диаграммы указанных металингвистических формул представлены на *рис. 3.4.*

Как правило, <*Фактический параметр*> является выражением.

Примеры:

1)
```
Exit
```

2)
```
writeln(x+y, sin(x))
```

3)
```
writeln(2*x)
```

Тип каждого фактического параметра и порядок его появления в списке указываются в разделе описаний соответствующих процедур. Правила составления списка фактических параметров будут изучены в следующих главах.

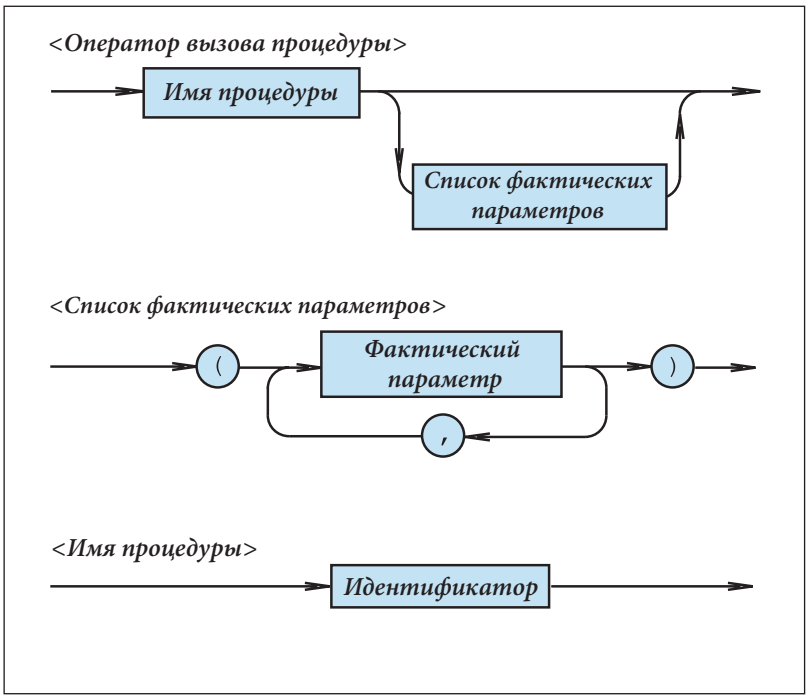


Рис. 3.4. Синтаксические диаграммы оператора *вызова процедуры*

Вопросы и упражнения

❶ Для чего необходим оператор *вызова процедуры?*

❷ **ПРОАНАЛИЗИРУЙТЕ!** Даны следующие операторы:

a) `readln(x, y, z, q)`

b) `CitireDate(ff, tt)`

c) `Halt`

d) `writeln('x=', x, 'y=', y)`

e) `writeln('x+y=', x+y, 'sin(x)=', sin(x))`

Укажите имена вызываемых процедур, число фактических параметров каждой процедуры, а также назовите эти фактические параметры.

❸ Укажите на синтаксических диаграммах *рис. 3.4* пути, которые соответствуют операторам из задания 2.

3.8.Вывод алфавитно-цифровой информации

Pascal

В большинстве версий языков ПАСКАЛЬ и C++ экран монитора является стандартным устройством вывода.

В языке ПАСКАЛЬ, как правило, экран разделяется на условные зоны, называемые символьными. Указанные зоны образуют 25 строк по 80 символов в каждой. Зона, в которой будет выведен текущий символ, обозначается курсором.

Данные выводятся на экран с помощью процедуры `write(x)` или `writeln(x)`.

Вызов процедуры

`write(`x_1`, `x_2`, …, `x_n`)`

эквивалентен последовательности

`write(`x_1`); write(`x_2`); …; write(`x_n`)`

Фактические параметры процедур write или writeln называются параметрами вывода. Они могут иметь следующий вид:

e $\quad$ $e : w$ $\quad$ $e : w : f$

где e – выражение типа `integer`, `real`, `boolean`, `char` или строкового типа, значение которого нужно вывести на экран; w и f являются выражениями типа `integer` и называются **спецификаторами формата**. Значение выражения w указывает минимальное число символов, используемых для вывода значения e; если для представления значения e требуется меньше, чем w символов, то ему предшествует такое число пробелов, чтобы было записано точно w символов (*рис. 3.5*).

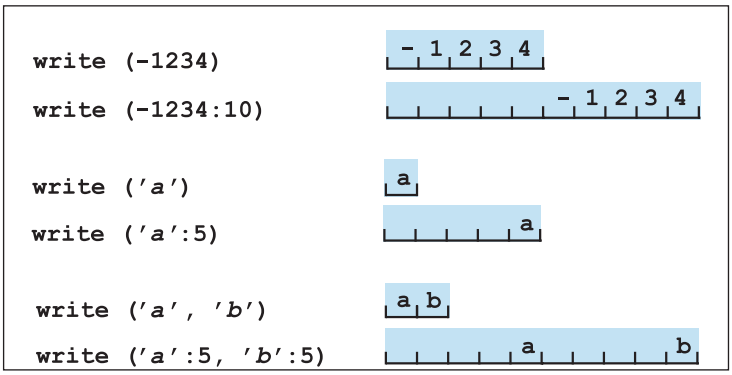


Рис. 3.5. Значение спецификатора формата *w*

Спецификатор формата *f* может присутствовать только в том случае, когда e является выражением типа `real`. Данный параметр указывает количество цифр после запятой в представлении значения выражения *e* с фиксированной точкой, без масштабного множителя. При отсутствии *f* значение e записывается с плавающей точкой, с масштабным множителем (*рис. 3.6*).

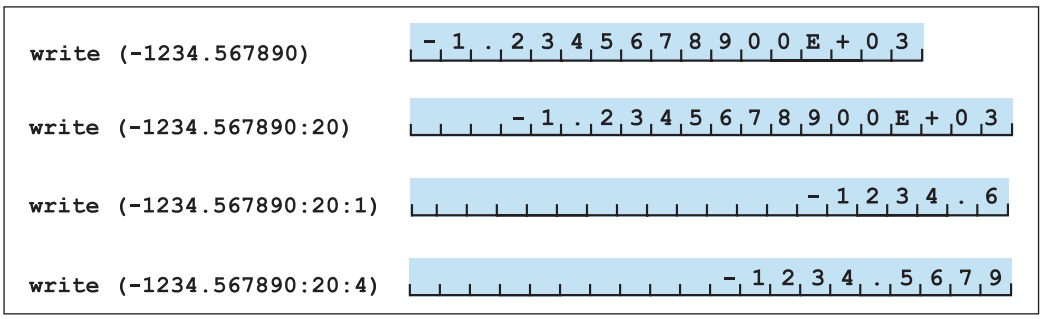


Рис. 3.6. Значение спецификатора формата *f*

Разница между процедурами `write` и `writeln` состоит в том, что после вывода данных write оставляет курсор в текущей строке, тогда как writeln переводит курсор на начало следующей строки. Рациональное использование процедур `write`, `writeln` и параметров размера поля обеспечивает вывод данных в форме, удобной для чтения. При выводе на экран нескольких значений рекомендуется указывать соответствующие им идентификаторы или сопровождать их комментариями.

Примеры:

```
1) write('Сумма введенных чисел')
2) writeln(s:20)
3) writeln('Сумма=', s)
4) writeln('s=', s)
5) writeln('x=', x, 'y=':5, y, 'z=':5, z)
```

В языке C++ вывод чисел и строк символов на экран (консоль) осуществляется с помощью оператора:

`cout` << <*Выражение*>

Примеры:

```
1) cout<<x;
2) cout<<x<<y<<z;
3) cout<<"Aria cercului="<<(Pi*r*r);
4) cout<<x<<endl;
5) cout<<"Введите целое число \n";
```

Чтобы вывести алфавитно-цифровую информацию на определенном количестве символьных позиций и установить размер каждого отображаемого значения, используется оператор `setw`, который также называется манипулятором:

```
cout<<setw(n);
```

Количество позиций (столбцов), установленных для отображения, называется полем. Отображаемое значение будет выровнено по правому краю, а позиции полей, оставленные свободными, будут заполнены пробелами. Параметр n – это целочисленное выражение, называемое *спецификатором размера поля.*

Чтобы использовать манипулятор `setw`, – в программу C++ должен быть включен `iomanip`, что выполняется с помощью директивы `#include`.

Примеры:

В присутствии объявления

```
double x=3;
```

оператор

```
cout<<setw(10)<<x;
```

выведет значение переменной x на 10 позициях.

Так как значение переменной x равно 3, для его вывода нужна лишь одна позиция. Следовательно, остальные девять позиций будут заполнены пробелами.

Смысл спецификатора *n* манипулятора `setw` представлен на *рис. 3.5**.

При отображении вещественных данных могут использоваться специальные функции

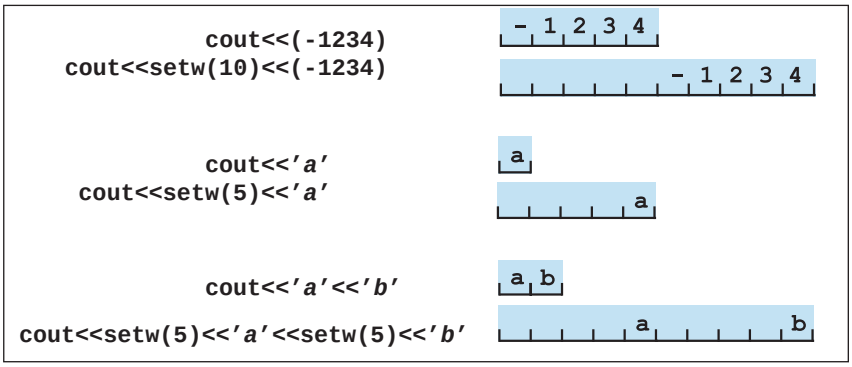


*Fig. 3.5**. Смысл спецификатора поля манипулятора `setw`

форматирования, устанавливающие определенную точность отображения, например, количество цифр после запятой числа типа double. Для этого в программу могут быть включены следующие операторы:

`cout.setf(ios::fixed);`	Действительные числа должны отображаться в десятичной форме.
`cout.setf(ios::showpoint);`	Десятичная точка отображается всегда, даже для целых чисел.
`cout.precision(n);`	Указывает количество отображаемых десятичных знаков *n*.

Первый оператор позволяет использовать функцию precision только для дробной части (после точки); в противном случае учитывалось бы все число.

Второй оператор обязательно отображает десятичную точку, даже для целых чисел.

Третий оператор устанавливает точность числа до *n* десятичных знаков. Выводимое значение округляется! Аргумент `n` должен быть положительным целым числом или выражением типа **`int`**.

Выполнив эти операторы, функция `cout` отобразит вещественные числа в новом формате.

При указании размера поля для вещественных чисел необходимо учитывать, что десятичная точка также занимает позицию. Например, значение 3.14 занимает не 3, а 4 позиции.

Примеры:

```
double pi = 3.14159;
cout<<pi<<endl;           // будет выведено 3.14159
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
cout<<pi;                  // будет выведено 3.14
```

Операторы форматирования, использованные в представленном выше примере, могут быть записаны в одной строке:

```
cout<<setiosflags(ios::fixed)<<setiosflags(ios::showpoint)<<
setprecision(2)<<pi;
```

или в короткой форме:

```
Cout<<fixed<<showpoint<<setprecision(2)<<pi;
```

Для отображения числа в экспоненциальном представлении (с масштабным коэффициентом) будут использоваться следующие операторы:

```
cout.setf(ios::scientific); /*     Обеспечивает отображение
                                   числа в экспоненциальном
                                   представлении (с масштабным
                                   коэффициентом) */
 cout.setf(ios::showpoint);
 cout.precision(2);
 cout<<pi;                  //     будет выведено 3.14e+000
```

или

```
cout<<setiosflags(ios::scientific)<<setiosflags(ios::showpoint)<<
setprecision(2)<<pi;
```

или в еще более короткой форме:

```
Cout<<scientific<<showpoint<<setprecision(2)<<pi;
```

Примеры:

```
 double x = 0.0000123;
 cout.setf(ios::scientific);
 cout.setf(ios::showpoint);
 cout.precision(3);
 cout << x;                // будет выведено 1.230e-005
```

```
      cout<<setprecision(10)
              <<-1234.567890      -1234.56789

              cout<<setw(20)
              <<-1234.567890                -1234.56789

     cout<<setiosflags(ios::
   scientific)<<-1234.567890      -1.2345678900E+003

     cout<<setiosflags(ios::
scientific)<<setprecision(1)       -1.2E+003
              <<-1234.567890

  cout.setf(ios::fixed);
  cout.setf(ios::showpoint);       -1234.56789
  cout.precision(10):
  cout<<-1234.567890
```

*Рис. 3.6**. Значение спецификаторов формата для вещественных чисел

Другой часто используемый манипулятор – `endl`, который обеспечивает перемещение курсора на новую строку:

Примеры:

```
cout<<"Введите два числа"<<endl;
```

Этот оператор отобразит текст `Введите два числа` на экране, а затем последующий вывод переместится на новую строку.

Для перехода на новую строку можем также использовать спецификатор «\ n» (*escape*). Это эквивалент `endl`. Управляющий символ *escape* должен быть введен в текст в кавычках. Он не будет отображаться, но заставит курсор переместиться на следующую строку:

Примеры:

1)
```
cout<<"Сумма введенных чисел равна \n";
```

2)
```
cout<<s<<endl;
```

3)
```
cout<<"Давным \n давно \n";
```

4)
```
cout<<"s="<<s<<endl;
```

5)
```
cout<<"\n x="<<x<<"\n y="<<y<<" \n z="<<z<<endl;
```

Вопросы и упражнения

❶ Для чего нужны спецификаторы формата?

❷ (ПАСКАЛЬ) Как называются фактические параметры процедур `write` и `writeln`?
(C++) Как называются параметры манипулятора `setw`?

❸ **ПРИМЕНИТЕ!** Определите форматы данных, выводимых на экран следующими программами:

ПАСКАЛЬ

```
Program P42;
 { Вывод данных типа inte-
ger }
var i : integer;
begin
  i:=-1234;
  writeln(i);
  writeln(i:1);
  writeln(i:8);
  writeln(i, i);
  writeln(i:8, i:8);
  writeln(i, i, i);
  writeln(i:8, i:8, i:8);
end.
```

C++

```
// Программа P42
#include <iostream>
#include <iomanip>
using namespace std;
//Вывод данных типа int
int main()
{
int i;
i=-1234;
cout<<i<<endl;
cout<<setw(1)<<i<<endl;
cout<<setw(8)<<i<<endl;
cout<<i<<i<<endl;
cout<<setw(8)<<i<<setw(8)<<i;
cout<<i<<i<<i<<endl;
cout<<setw(8)<<i<<setw(8)<<i;
return 0;
}
```

ПАСКАЛЬ	C++

Program P43;
{Вывод данных типа real }
var x : real;
begin
  x:=-1234.567890;
  writeln(x);
  writeln(x:20);
  writeln(x:20:1);
  writeln(x:20:2);
  writeln(x:20:4);
  writeln(x, x, x);
  writeln(x:20, x:20, x:20);
  writeln(x:20:4, x:20:4, x:20:4);
end.
	// Программа P43
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Вывод данных вещественных типов
	int main()
	{
	double x;
	x=-1234.567890;
	cout<<x<<endl;
	cout<<setw(20)<<x<<endl;
	cout<<setw(20)<<setprecision(1)<<x
	  <<endl;
	cout<<setw(20)<<setprecision(2)<<x
	  <<endl;
	cout<<setw(20)<<setprecision(4)<<x
	  <<endl;
	cout<<x<<x<<x<<endl;
	cout<<setw(20)<<x<<setw(20)<<x<<
	  setw(20)<<x<<endl;
	cout<<setw(20)<<setprecision(4)<<x
	  <<setw(20)<<setprecision(4)<<x
	  <<endl;
	cout<<fixed<<showpoint<<setprecision(2)<<x<<endl;
	cout<<scientific<<showpoint<<
	  setprecision(2)<<x<<endl;
	return 0;
	}

Program P44;
 { Вывод данных типа boolean }
var p : boolean;
begin
  p:=false;
  writeln(p);
  writeln(p:10);
  writeln(p, p);
  writeln(p:10, p:10);
end.
	// Программа P44
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Вывод данных типа bool
	int main()
	{
	bool p;
	p=false;
	cout<<p<<endl;
	cout<<setw(10)<<p<<endl;
	cout<<p<<p<<endl;
	cout<<setw(10)<<p<<setw(10)
	  <<p<<endl;
	return 0;
	}

```
Program P45;
  {Вывод строк символов }
begin
  writeln('abc');
  writeln('abc':10);
  writeln('abc', 'abc');
  writeln('abc':10,
    'abc':10);
end.
```

```
// Программа P45
#include <iostream>
#include <iomanip>
using namespace std;
// Вывод строк символов
int main()
{
cout<<"abc"<<endl;
cout<<setw(10)<<"abc \n";
cout<<"abc"<<"abc \n";
cout<<setw(10)<<"abc"<<setw(10)
  <<"abc \n";
return 0;
}
```

❹ **ПРИМЕНИТЕ!** Напишите программу, которая выводит на экран значения 1234567890, 123, 123.0 и true следующим образом:

```
  1234567890
  123
  123.0
  true
  1234567890
                    123
                    123.000
                    true
```

3.9. Ввод данных с клавиатуры

Как правило, **стандартным устройством** ввода является клавиатура.

Pascal

В ПАСКАЛЕ ввод данных с клавиатуры выполняется с помощью стандартных процедур read или readln. Список фактических параметров процедуры read или readln может включать переменные типа integer, real, char и строкового типа.

Вызов

```
read(x)
```

выполняет следующие действия: если переменная x является переменной типа integer или real, тогда считывается вся строка символов, представляющая целое или вещественное значение; если переменная *x* является переменной типа char, то процедура считывает только один символ.

Вызов

```
read(x1, x2, ..., xn)
```

эквивалентен последовательности

```
read(x1); read(x2); ...; read(xn)
```

Числа, вводимые с клавиатуры, должны разделяться пробелами или символами конца строки. Пробелы, стоящие перед самим числом, игнорируются. Строка символов, представляющая собой число, должна соответствовать синтаксису числовых констант соответствующего типа. В противном случае возникает ошибка ввода-вывода.

Например, рассмотрим программу:

```
Program P46;
 { Считывание чисел с клавиатуры }
var  i, j : integer;
     x, y : real;
begin
  read(i, j, x, y);
  writeln('Были введены:');
  writeln('i=', i);
  writeln('j=', j);
  writeln('x=', x);
  writeln('y=', y);
end.
```

в которой считываются с клавиатуры значения переменных `i, j, x, y`. После запуска программы на выполнение пользователь набирает:

```
1<ENTER>
2<ENTER>
3.0<ENTER>
4.0<ENTER>
```

На экран будет выведено:

```
Были введены:
i=1
j=2
x=3.0000000000E+00
y=4.0000000000E+00
```

Если данные вводить одной строкой, то результат не изменится:

```
1 2 3.0 4.0<ENTER>
```

В случае необходимости целые числа, введенные пользователем, переводятся в вещественные значения. Например, в программе `P46` пользователь может набрать на клавиатуре

```
1 2 3 4<ENTER>
```

Процедура `readln` считывает данные точно так же, как и процедура `read`. Однако после считывания последнего значения оставшиеся символы текущей строки игнорируются. В качестве примера рассмотрим программу `P47`:

```
Program P47;
 { Использование процедуры readln }
var  i, j : integer;
     x, y : real;
```

```
begin
  writeln('Использование процедуры read'');
  read(i, j);
  read(x, y);
  writeln('Были введены::');
  writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
  writeln('Использование процедуры readln');
  readln(i, j);
  readln(x, y);
  writeln('Были введены');
  writeln('i=', i, ' j=', j, ' x=', x, ' y=', y);
end.
```

При выполнении операторов

```
read(i, j);
read(x, y);
```

числовые значения из строки

```
1 2 3 4<ENTER>
```

введенной пользователем, будут присвоены соответственно переменным i, j, x, y. При выполнении оператора

```
readln(i, j)
```

числовые значения 1 и 2 из строки

```
1 2 3 4<ENTER>
```

будут присвоены переменным i и j. Числа 3 и 4 игнорируются. Затем компьютер выполняет оператор:

```
readln(x, y)
```

т. е. ожидается набор значений для x и y. Отметим, что при вызове процедуры readln без параметров компьютер ожидает нажатия клавиши <ENTER>. Такой вызов процедуры используется с целью приостановки выполнения программы, предоставляя тем самым пользователю возможность прочитать результаты, выведенные до этого на экран.

Для того чтобы сообщить пользователю, какие данные нужно вводить, рекомендуется выводить на экран соответствующие наводящие сообщения.

Примеры:

1) `write('Введите два числа:'); readln(x, y);`
2) `write('Введите целое число:'); readln(i);`
3) `write('x='); readln(x);`
4) `write('Ответьте да/D или нет/N:'); readln(c);`

В C++ ввод данных с клавиатуры выполняется с помощью оператора

`cin >>` <*Переменная*> `;`

где <Переменная> может указывать переменные целых, вещественных типов, символы или строки.

Вызов

`cin >>` *x*;

состоит во вводе некоторого значения с клавиатуры и автоматическом присваивании введенного значения переменной *x*.

Если переменная *x* имеет тип **`int`** или **`double`**, то считывается вся строка, представляющая целое или вещественное значение. Если *x* имеет тип **`char`**, оператор считывает один символ.

Оператор

`cin >>` x_1 `>>` x_2 `>> ... >>` x_n `;`

Эквивалентна следующим операторам:

`cin >>` x_1`; cin >>` x_2`; ...; cin >>` x_n`;`

Числовые данные, вводимые с клавиатуры, должны быть разделены пробелами или символами конца строки. Пробелы перед числовым значением игнорируются. При чтении символьных переменных пробелы игнорируются.

Для облегчения ввода данных рекомендуется, чтобы операторам ввода предшествовал вывод наводящих сообщений.

Примеры:

1)
```
cout<<"Введите два числа: ";  cin>>x>>y;
```

2)
```
cout<<"Введите целое число: "; cin>>i;
```

3)
```
cout<<"x= "; cin>>x;
```

4)
```
cout<<"Ответьте да/D или нет/N: "; cin>>c;
```

Вопросы и упражнения

❶ Как разделяются числовые данные, вводимые с клавиатуры?

❷ (ПАСКАЛЬ) Каковы различия между процедурами `read` и `readln`?

❸ **ПРОАНАЛИЗИРУЙТЕ!** Даны следующие программы:

ПАСКАЛЬ	C++

`Program P48;`
`var i : integer;`
`  c : char;`
`  x : real;`
`begin`
`  readln(i);`
`  readln(c);`
	`// Программа P48`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`int i;`
	`char c;`
	`double x;`
	`cin>>i;`

```
  readln(x);
  writeln('i=', i);
  writeln('c=', c);
  writeln('x=', x);
  readln;
end.
```

```
cin>>c;
cin>>x;
cout<<"i="<<i<<endl;
cout<<"c="<<c<<endl;
cout<<"x="<<x<<endl;
return 0;
}
```

Определите результаты, которые будут выведены на экран при вводе следующих данных:

a)
```
1
2
3
```

b)
```
1 2 3
5 6 7
8 9 0
```

c)
```
123
456
789
```

d)
```
123  456  789
abc  def  ghi
890  abc  def
```

❹ **ПРИМЕНИТЕ!** Напишите программу, которая считывает символ или число с клавиатуры и отображает на экране следующее:

a)
```
*
**
***
****
*****
```

b)
```
############
########
####
#
```

c)
```
 00      00
  0    0  0    0
0       0       0
0               0
  0           0
    0       0
      0   0
        0
```

❺ С клавиатуры считываются две меры углов, выраженные в градусах, минутах и секундах: g1, m1, s1; g2, m2, s2. Напишите программу, которая вычислит и отобразит на экране сумму измерений двух углов, выраженную также в градусах, минутах и секундах.

❻ С клавиатуры считываются координаты двух точек: x1, y1; x2, y2. Напишите программу, которая отобразит на экране длину отрезка, образованного этими двумя точками, и координаты его середины.

❼ Товар имеет цену x леев за единицу, к которой при продаже применяется НДС в размере t процентов. Значения x и t считываются с клавиатуры. Напишите программу, которая отобразит на экране цену продажи.

❽ Мария и Петр хотят испечь яблочный пирог. Для этого им нужно x граммов муки, y граммов сахара, p яиц, m кг яблок, z мл молока. Известно, что цена за килограмм муки равна px леев, за килограмм сахара py леев, килограмм яблок стоит pm леев, литр молока стоит pz леев, а яйца стоят pp леев / штука. Напишите программу, которая определит и отобразит на экране цену пирога.

3.10. Пустой оператор

Выполнение данного оператора никак не влияет на значения переменных, используемых в программе. Синтаксис пустого оператора:

<Пустой оператор> ::=

Таким образом, в тексте программы пустой оператор ничем не представляется. Так как операторы программы разделяются с помощью «`;`», присутствие пустого оператора отмечается появлением этого символа.

Например, в тексте:

ПАСКАЛЬ	C++
`x:=4;;;; y:=x+1`	`x=4;;;; y=x+1`

есть 5 операторов, 3 из которых – пустые.

Обычно пустой оператор используется на этапах разработки и отладки сложных программ. Хотя пустой оператор не выполняет никаких действий, ее (точнее, символа «`;`») присутствие или отсутствие может повлиять на ход программы.

3.11. Оператор if

Оператор ветвления **`if`** выполняет одно из двух возможных действий в зависимости от значения некоторого условия – логического выражения. Синтаксис данного оператора:

ПАСКАЛЬ

<*Оператор* **`if`**> ::= **`if`** <*Логическое выражение*> **`then`** <*Оператор1*>
[**`else`** <*Оператор2*>]

C++

<*Оператор* **`if`**> ::= **`if`** (<*Логическое выражение*>) <*Оператор1*>
[**`else`** <*Оператор2*>]

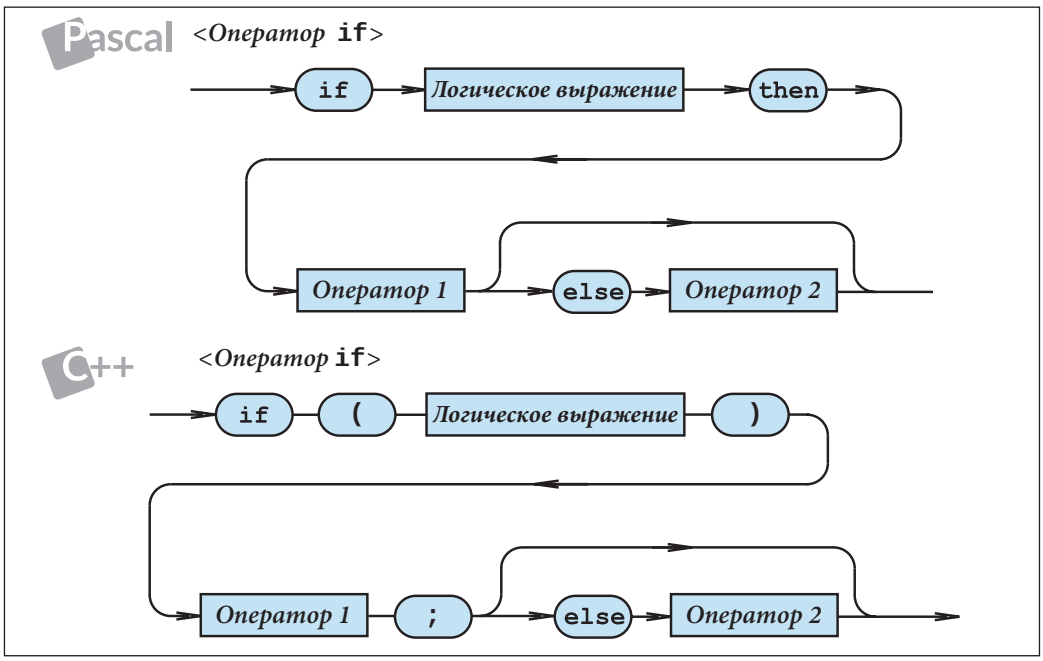


Рис. 3.7. Синтаксическая диаграмма <*Оператор* `if`>

Синтаксическая диаграмма рассматриваемого оператора представлена на *рис. 3.7*. Логическое выражение, входящее в состав оператора **if**, называется **условием**.

Выполнение оператора **if** начинается с проверки условия. Если результатом проверки является true, то выполняется *Оператор1*. Если условие принимает значение false, то либо выполняется *Оператор2*, стоящая после ключевого слова **else** (если оно есть), либо управление передается оператору, следующему непосредственно за оператором **if**.

В следующих программах оператор **if** используется для определения максимального из двух чисел x и y, считываемых с клавиатуры.

ПАСКАЛЬ

```
Program P49;
{ Определение большего из двух
чисел }
var x, y, max : real;
begin
  writeln('Введите два числа:');
  write('x='); readln(x);
  write('y='); readln(y);
  if x>=y then max:=x
          else max:=y;
  writeln('max=', max);
  readln;
end.
```

C++

```
// Программа P49
#include <iostream>
using namespace std;
/* Определение большего из двух
чисел  */
int main()
{
float x, y, max;
cout<<"Введите два числа: \n";
cout<<"x="; cin>>x;
cout<<"y="; cin>>y;
if (x>=y) max=x; else max=y;
cout<<"max="<<max;
return 0;
}
```

Следующая программа переводит римские цифры: *I* (один), *V* (пять), *X* (десять), *L* (пятьдесят), *C* (сто), *D* (пятьсот), *M* (тысяча), считываемые с клавиатуры, в соответствующие им числа в десятичной системе счисления.

ПАСКАЛЬ

```
Program P50;
 { Конверсия римских цифр }
var i : integer; c : char;
begin
  i:=0;
  writeln(Введите одну из
римских цифр');
  writeln(' I, V, X, L, C,
D, M');
  readln(c);
  if c='I' then i:=1;
  if c='V' then i:=5;
  if c='X' then i:=10;
  if c='L' then i:=50;
  if c='C' then i:=100;
```

C++

```
// Программа P50
#include <iostream>
using namespace std;
// Конверсия римских цифр
int main()
{
int i; char c;
i=0;
cout<<"Введите одну из
римских цифр\n";
cout<<"I, V, X, L, C, D, M
\n";
cin>>c;
if (c=='I') i=1;
if (c=='V') i=5;
```

```
  if c='D' then i:=500;
  if c='M' then i:=1000;
  if i=0 then writeln(c,
  ' - не является римской
цифрой')
         else writeln(i);
  readln;
end.
```

```
if (c=='X') i=10;
if (c=='L') i=50;
if (c=='C') i=100;
if (c=='D') i=500;
if (c=='M') i=1000;
if (i==0) cout<<c<<" не
является римской цифрой"; else
cout<<i;
return 0;
}
```

Вопросы и упражнения

❶ Для чего необходим оператор **if**?

❷ **ПРОАНАЛИЗИРУЙТЕ И ПРИМЕНИТЕ!** Какие значения будет принимать переменная x после выполнения каждого из следующих операторов? Подразумевается, что `a = 18`, `b = -15`, `p = true`.

	ПАСКАЛЬ		C++
a)	`if a>b then x:=1 else x:=4`	*a)*	`if (a>b) x=1; else x=4`
b)	`if a<b then x:=15 else x:=-21`	*b)*	`if (a<b) x=15; else x=-21`
c)	`if p then x:=32 else x:=638`	*c)*	`if (p) x=32; else x=638`
d)	`if not p then x:=0 else x:=1`	*d)*	`if (!p) x=0; else x=1`
e)	`if (a<b) and p then x:=-1 else x:=1`	*e)*	`if ((a<b)&& p) x=-1; else x=1`
f)	`if (a>b) or p then x:=-6 else x:=-5`	*f)*	`if ((a>b)|| p) x=-6; else x=-5`
g)	`if not (a>b) then x:=19 else x:=-2`	*g)*	`if (!(a>b)) x=19; else x=-2`
h)	`if (a=b) or p then x:=89 else x:=-15`	*h)*	`if ((a==b)||p) x=89; else x=-15`

❸ **ПРИМЕНИТЕ!** Напишите программу, которая вычисляет значение одной из следующих функций:

a) $y=\begin{cases}2x, x\ge 0;\\ \dfrac{x}{2}, x<0;\end{cases}$ *b)* $y=\begin{cases}x+3, x>5;\\ x-3, x\le 5;\end{cases}$ *c)* $z=\begin{cases}\sqrt{x+y}, \text{ если } c>0;\\ x\cdot y, \text{ если } c=0;\\ \dfrac{1}{x-y}, \text{ если } c<0;\end{cases}$

d) $y=\begin{cases}x, x\ge 3;\\ x+4, x<3;\end{cases}$ *e)* $y=\begin{cases}x, |x|>5;\\ 2x, |x|\le 5;\end{cases}$ *f)* $y=\begin{cases}x^3-6x, \text{ если } x<-12;\\ \sqrt{x^4+12}, \text{ если } -12\le x<-5;\\ 2x+12, \text{ если } -5\le x<2;\\ 14, \text{ если } x\ge 2.\end{cases}$

Пример, для $y=\begin{cases}x+6, x>4;\\ x-3, x\le 4;\end{cases}$

ПАСКАЛЬ	C++

Program P51;
var x, y : real;
begin
  write('x='); readln(x);
  if x>4 **then** y:=x+6
  else y:=x-3;
  writeln('y=', y);
  readln;
end.
	// Программа P51
	#include <iostream>
	using namespace std;
	int main()
	{
	float x, y;
	cout<<"x="; cin>>x;
	if (x>4) y=x+6; **else** y=x-3;
	cout<<"y="<<y<<endl;
	return 0;
	}

❹ Какие результаты выведет на экран следующая программа?

ПАСКАЛЬ	C++

Program P52;
var x, y : real;
begin
  write('x='); readln(x);
  y:=x;
  if x>0 **then**; y:=2*x;
  writeln('y=', y);
  readln;
end.
	// Программа P52
	#include <iostream>
	using namespace std;
	int main()
	{
	float x, y;
	cout<<"x="; cin>>x;
	y=x;
	if (x>0); y=2*x;
	cout<<"y="<<y;
	return 0;
	}

❺ Прокомментируйте сообщения, выводимые на экран в процессе компиляции программы P52:

ПАСКАЛЬ	C++

Program P53;
var x, y : real;
begin
  write('x=');
  readln(x);
  if x>4 **then** y:=2*sqr(x)+6;
  else y:=x*x*x-3;
  writeln('y=', y); readln;
end.
	// Программа P53
	#include <iostream>
	using namespace std;
	int main()
	{
	float x, y;
	cout<<"x="; cin>>x;
	if (x>4) y=2*pow(x,2)+6;
	else y=x*x*x-3;
	cout<<"y="<<y;
	return 0;
	}

❻ Напишите программу, которая переводит десятичные числа 1, 5, 10, 50, 100, 500 и 1000, считываемые с клавиатуры, в римские.

❼ Дано ненулевое натуральное число N, представляющее отметку ученика. Выведите на экран сообщение «*Переведен*», если $N \geq 5$, и «*Не переведен*» в противном случае.

❽ Дни недели закодированы следующим образом: 1 - *Понедельник*, 2 - *Вторник*, ..., 7 - *Воскресенье*. Напишите программу, которая считывает с

клавиатуры ненулевое натуральное число n, $1 \le n \le 7$ и отображает на экране название дня, которому соответствует это число (*Понедельник*, *Вторник*, ..., *Воскресенье*).

❾ Напишите программу, которая решала бы уравнение второй степени, $ax^2+bx+c = 0$. где a, b и c – вещественные числа, считываемые с клавиатуры.

❿ Даны три отрезка длиной a, b и c соответственно. Напишите программу, которая проверяет, можно ли построить треугольник из этих отрезков. Если такое построение возможно, программа отобразит тип треугольника (равносторонний, равнобедренный или разносторонний) и его площадь. Для вычисления площади треугольника воспользуйтесь формулой *Герона*. Действительные числа a, b и с читаются с клавиатуры.

3.12. Оператор множественного ветвления

В предыдущем параграфе мы познакомились с одним из операторов ветвления и заметили, что оператор **if** позволяет нам создавать потоки выполнения максимум с двумя ветвями. В случаях, когда необходимо создать потоки выполнения с более чем двумя ветвями, используются операторы множественного ветвления.

Оператор множественного ветвления обязательно включает выражение, называемое переключателем (селектором), и список операторов. Каждому оператору соответствует префикс в виде одной или нескольких констант выбора. Синтаксис рассматриваемого оператора:

ПАСКАЛЬ

<*Оператор* **case**> ::= **case** <*выражение*> **of** [<*Вариант*>{**;** <*Вариант*>}] [**;**] **end**
<*Вариант*> ::= <*Константа*> {**,** <*Константа*>} **:** <*Оператор*>

C++

<*Оператор* **switch**> ::= **switch** (<*выражение*>) { <*Вариант*>[{**;** <*Вариант*>}]**;**
[**default**: <*Оператор*> }
<*Вариант*> ::= **case** <*Константа*> **:**{**case** <*Константа*>:} <*Оператор*>**; break;**

Соответствующие синтаксические диаграммы представлены на *рис. 3.8.*

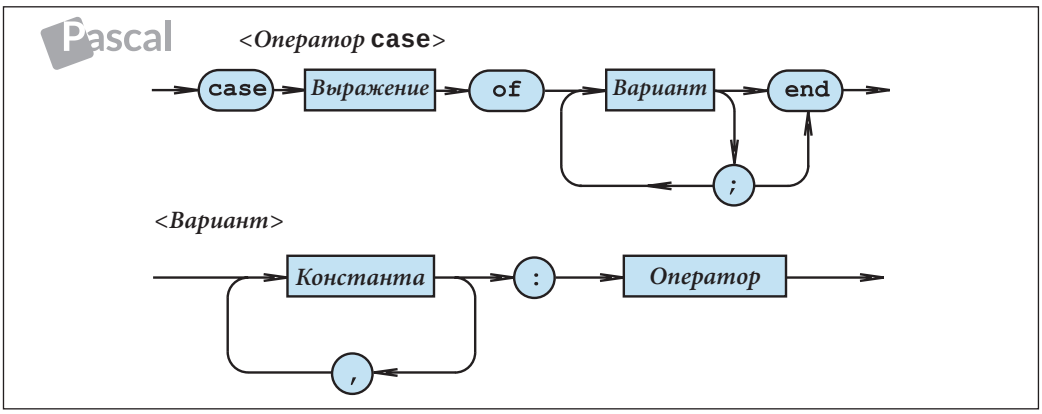


Рис. 3.8. Синтаксическая диаграмма <*Оператор* case>, ПАСКАЛЬ

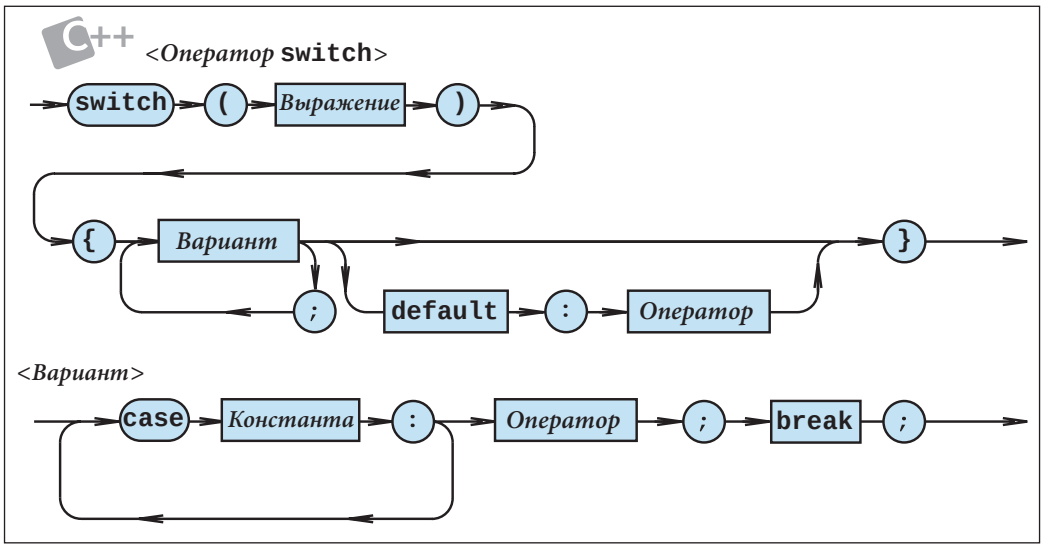


*Рис. 3.8**. Синтаксическая диаграмма <*Оператор* switch>, C++

Селектор должен относиться к порядковому типу. Константы выбора должны быть совместимыми с типом селектора и не могут повторяться.

Примеры:

	ПАСКАЛЬ		C++
	`var i : integer; c : char; a, b, y : real;`		`int i; char c; float a, b, y;`

1)	`case i of`
	`  0, 2, 4, 6, 8 : writeln('Четная цифра');`
	`  1, 3, 5, 7, 9 : writeln('Нечетная цифра');`
	`end;`
		1)	`switch (i)`
			`{`
			`case 0: case 2: case 4: case 6:case 8: cout<<"Четная цифра"; break;`
			`case 1: case 3: case 5: case 7: case 9: cout<<"Нечетная цифра";`
			`}`

2)	`case c of`
	`  '+' : y:=a+b;`
	`  '-' : y:=a-b;`
	`  '*' : y:=a*b;`
	`  '/' : y:=a/b;`
	`end;`
		2)	`switch (c)`
			`{`
			`case '+' : y=a+b; break;`
			`case '-' : y=a-b; break;`
			`case '*' : y=a*b; break;`
			`case '/' : y=a/b; break;`
			`}`

Выполнение оператора множественного ветвления начинается с проверки селектора. Если селектор принимает одно из значений констант выбора, то выполняется оператор, соответствующий этой константе.

В следующей программе оператор множественного ветвления используется для перевода римских цифр в десятичные числа.

ПАСКАЛЬ	C++

Program P54;
{ Перевод римских цифр в десятичные }
var i : integer; c : char;
begin
i:=0;
writeln('*Введите одну из римских цифр*');
writeln('*I, V, X, L, C, D, M*');
readln(c);
case c **of**
'I' : i:=1;
'V' : i:=5;
'X' : i:=10;
'L' : i:=50;
'C' : i:=100;
'D' : i:=500;
'M' : i:=1000;
end;
if i=0 **then** writeln(c, ' – *не является римской цифрой*')
else writeln(i);
readln;
end.
	// Программа P54
	#include <iostream>
	using namespace std;
	// Перевод римских цифр в десятичные
	int main()
	{
	int i; **char** c;
	i=0;
	cout<<"Введите одну из римских цифр ";
	cout<<" I, V, X, L,
	C, D, M \n";
	cin>>c;
	switch (c)
	{
	case 'I' : i=1; **break**;
	case 'V' : i=5; **break**;
	case 'X' : i=10; **break**;
	case 'L' : i=50; **break**;
	case 'C' : i=100; **break**;
	case 'D' : i=500; **break**;
	case 'M' : i=1000; **break**;
	}
	if (i==0) cout<<c<<" не является римской цифрой"; **else** cout<<i;
	return 0;
	}

Отметим, что в некоторых версиях языка ПАСКАЛЬ/C++ синтаксис и семантика оператора **case/switch** были изменены. Список вариантов может включать оператор, которому предшествует ключевое слово **else / default** (в некоторых версиях **otherwise**).

Пример:

ПАСКАЛЬ	C++

Program P55;
{ Модель карманного калькулятора }
var a, b : real;
c : char;
begin
write('*a=*'); readln(a);
write('*b=*'); readln(b);
write('*Код операции* '); readln(c);
	// Программа P55
	/* Модель карманного калькулятора */
	#include <iostream>
	using namespace std;
	int main()
	{
	double a, b;
	char c;
	cout<<"a="; cin>>a;
	cout<<"b="; cin>>b;

```
  case c of
    '+' : writeln('a+b=', a+b);
    '-' : writeln('a-b=', a-b);
    '*' : writeln('a*b=', a*b);
    '/' : writeln('a/b=', a/b);
    else writeln ('Недопусти-
мый код операции');
  end;
  readln;
end.
```

```
cout<<"Код операции "; cin>>c;
switch (c)
{
case '+' : cout<<"a+b="<<a+b;
break;
case '-' : cout<<"a-b="<<a-b;
  break;
case '*' : cout<<"a*b="<<a*b;
  break;
case '/' : cout<<"a/b="<<a/b;
  break;
default: cout<<"Недопустимый
код операции";
}
return 0;
}
```

Вопросы и упражнения

❶ Укажите на синтаксических диаграммах *рис. 3.8* (соответственно, *рис.3.8**) пути, которые соответствуют операторам множественного ветвления из программ, представленных в этом параграфе.

❷ Как выполняется оператор множественного ветвления? Каким должен быть тип селектора?

❸ Какие константы можно использовать в качестве констант выбора?

❹ Замените операторы множественного ветвления в программах из этого параграфа последовательностью эквивалентных ему операторов **if**.

❺ **ПРИМЕНИТЕ!** Используя оператор множественного ветвления, напишите программу, которая переводит десятичные числа 1, 5, 10, 50, 100, 500, 1000, считываемые с клавиатуры, в римские цифры.

❻ **ПРОАНАЛИЗИРУЙТЕ**! Что появится на экране в процессе выполнения следующих программ?

ПАСКАЛЬ

```
Program P56;
type Semnal=(Rosu, Galben,
Verde);
var s : Semnal;
begin
  s:=Verde;
  s:=pred(s);
  case s of
   Rosu   : writeln('СТОП');
   Galben : writeln('ВНИМАНИЕ');
   Verde  : writeln('СТАРТ');
  end;
  readln;
end.
```

C++

```
// Программа P56
#include <iostream>
using namespace std;
int main()
{
enum Semnal {Rosu, Galben, Verde};
Semnal s;
int x;
s=Verde; x=s-1;
switch ((Semnal)x)
{
case Rosu  : cout<<"СТОП"; break;
case Galben : cout<<"ВНИМАНИЕ";
  break;
case Verde : cout<<"СТАРТ"; break;
}
return 0;
}
```

❼ **ПРОАНАЛИЗИРУЙТЕ!** Прокомментируйте следующие программы:

ПАСКАЛЬ	C++

Program P57;
{ Ошибка }
var x : real;
begin
writeln('x='); readln(x);
case x **of**
0,2,4,6,8 : writeln
('*Четная цифра*');
1,3,5,7,9 : writeln
('*Нечетная цифра*');
end;
readln;
end.
	// Программа P57
	#include <iostream>
	using namespace std;
	// Ошибка
	int main()
	{
	double x;
	cout<<"x="; cin>>x;
	switch (x)
	{
	case 0: **case** 2: **case** 4: **case** 6:
	case 8 : cout<<"Четная цифра";
	break;
	case 1: **case** 3: **case** 5: **case** 7:
	case 9 : cout<<"Нечетная цифра";
	break;
	}
	return 0;
	}

Program P58;
{ Ошибка }
var i : 1..4;
begin
write('i='); readln(i);
case i **of**
1 : writeln('*один*');
2 : writeln('*два*');
3 : writeln('*три*');
4 : writeln('*четыре*');
5 : writeln('*пять*');
end;
readln;
end.
	// Программа P58
	#include <iostream>
	using namespace std;
	// Ошибка
	int main()
	{
	enum Dis {A,B,C,D} x;
	int i;
	cout<<"i="; cin>>i;
	x=(Dis)i;
	switch (x)
	{
	case A : cout<<"один"; **break;**
	case B : cout<<"два"; **break;**
	case C : cout<<"три"; **break;**
	case D : cout<<"четыре"; **break;**
	case E : cout<<"пять"; **break;**
	}
	return 0;
	}

Program P59;
{ Ошибка }
type Semnal = (Rosu, Galben,
Verde);
Culoare = (Albastru,
Portocaliu);
var s : Semnal;
c : Culoare;
	// Программа P59
	#include <iostream>
	using namespace std;
	// Ошибка
	int main()
	{
	enum Semnal {Rosu, Galben, Verde};
	enum Culoare {Albastru,
	Portocaliu};

```
begin
  { ... }
  case s of
   Rosu     : writeln
('СТОП');
   Galben   : writeln
('ВНИМАНИЕ');
   Verde    : writeln
('СТАРТ');
   Albastru : writeln
('ПАУЗА');
  end;
  { ... }
end.
```

```
Semnal s;  Culoare c;
// ...
switch (s)
{
case Rosu : cout<<"СТОП"; break;
case Galben : cout<<"ВНИМАНИЕ";
  break;
case Verde: cout<<"СТАРТ"; break;
case Albastru : cout<<"ПАУЗА";
}
  // ...
return 0;
}
```

❽ **ПРОГРАММИРУЙТЕ!** Дни недели *понедельник, вторник, ..., воскресенье* пронумерованы числами *1, 2, 3*, ...,7. Напишите программу, которая считывает с клавиатуры число x и отображает на экране:
a) название дня, которому соответствует считанное число;
b) сообщение «Учебный день», если число соответствует учебным дням, и сообщение «Свободный день», если оно соответствует выходным дням.

❾ **СОЗДАЙТЕ!** Напишите программу, которая многократно выполняет следующие операции:
1) Считывает с клавиатуры действительные числа *x* и *y*.
2) Отображает на экране меню, содержащее команды, определяющие операции, которые необходимо выполнить над числами *x* и *y*.
3) Считывает с экрана целое число c, обозначающее выбранную пользователем команду.
4) В зависимости от команды c вычисляет и отображает на экране действительное число *r* – результат выполнения соответствующей операции над числами *x* и *y*.
5) Повторяющийся процесс продолжается до тех пор, пока пользователь не выберет из меню команду выхода из программы.

Меню, которое будет отображаться на экране, имеет вид:

```
ВЫЧИСЛИ:
1. Сумму
2. Разность
3. Произведение
4. Соотношение
5. Выход
```

Очевидно, целое число c, введенное пользователем, может принимать только значения 1,2, 3, ..., 5. Если пользователь вводит неправильное значение, будет отображено сообщение «Неверная команда» и повторяющийся процесс возобновится..

3.13. Оператор `for`

Оператор **for** предназначен для повторного выполнения другого оператора, в зависимости от значения управляющей переменной. Синтаксис рассматриваемого оператора:

ПАСКАЛЬ

<*Оператор* **for**> ::= **for** <*Переменная*> **:=** <*Выражение1*> <*Шаг*> <*Выражение2*>
do <*Оператор*>

<*Шаг*> ::= **to** | **downto**

C++

<*Операция* **for**> ::= **for** (<*Переменная*>=<*Выражение1*>; <*Переменная*>
<*Операция*> <*Выражение2*> ; <*Выражение3*>) <*Оператор*>

<*Операция*>::= < | <= | > | >=

Соответствующие синтаксические диаграммы представлены на *рис. 3.9.*

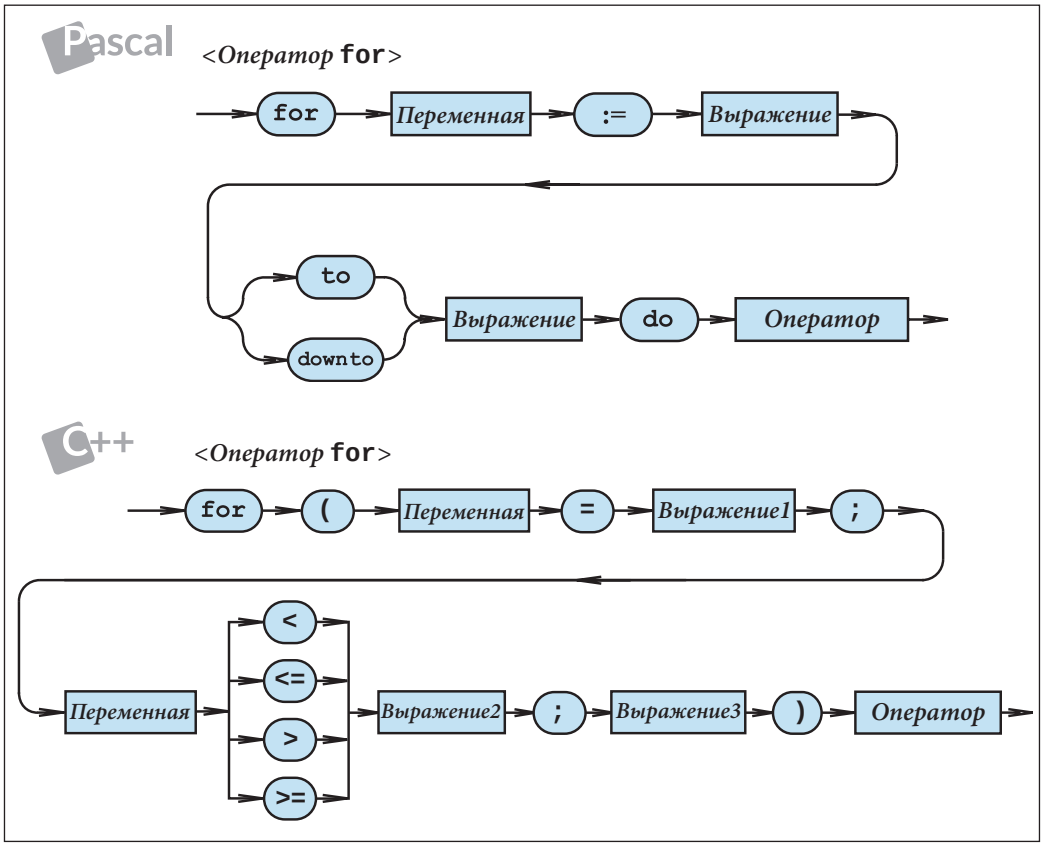


Рис. 3.9. Синтаксическая диаграмма <*Оператор* **for**>

Переменная, находящаяся после ключевого слова **for**, называется **управляющей переменной, параметром цикла** или **счетчиком**. Эта переменная должна принадлежать некоторому порядковому типу.

Значения выражений, входящих в состав оператора **for**, должны быть совместимыми, с точки зрения присваивания, с параметром цикла. *Выражение1* указывает исходное значение параметра цикла, а *Выражение2* – конечное значение.

Оператор, встроенный в цикл **for**, выполняется для каждого значения из диапазона, определяемого начальным и конечным значениями.

Процесс выполнения оператора **for** можно описать так:

Шаг 1. Вычисляются значения *выражений 1* и *2*. Эти значения вычисляются только один раз, в начале выполнения цикла.

Шаг 2. Сравнивается начальное (*Выражение1*) и конечное (*Выражение2*) значения диапазона повторения. Если конечное значение больше для случая **to** и меньше для случая **downto**, чем начальное значение, *оператор*, встроенный в цикл **for**, никогда не выполняется, и процесс выполнения оператора **for** завершается.

Шаг 3. Счетчику присваивается начальное значение (значение *Выражения1*).

Шаг 4. Выполняется *оператор*, встроенный в цикл **for**. Это может быть простой или составной оператор.

Шаг 5. Значение счетчика изменяется (в случае **to** производится переход к последующему значению, а в случае **downto** – к предшествующему значению текущего значения счетчика).

Шаг 6. Если измененное значение счетчика больше для случая **to** и меньше для случая **downto** конечного значения (*Выражения2*), выполнение оператора **for** заканчивается, в противном случае происходит переход к *Шагу 4*.

Шаг 1. Вычисляется значение *Выражения1*, которое в качестве начального значения присваивается *Переменной*-счетчику.

Шаг 2. Вычисляется значение *Выражения2*.

Шаг 3. Вычисляется значение логического *выражения*, указывающего на условие повторения.

Шаг 4. Если текущее значение логического выражения ложно (`false`), выполнение оператора **for** завершается.

Шаг 5. Если же значение логического выражения истинно (`true`), выполняется оператор, встроенный в цикл **for**. Это может быть простой или составной оператор.

Шаг 6. Вычисляется *выражение 3*, которое показывает, насколько текущее значение счетчика увеличится или уменьшится и осуществляется переход к *Шагу 2*.

Примечание. Любое из трех *выражений* в заголовке оператора **for** может отсутствовать, обязательным является только наличие двух символов «;» (точка с запятой). Если *Выражение2* отсутствует, условие возобновления итерации считается неявно удовлетворенным. В этом случае, если принудительный выход (скачком) из цикла не предусмотрен, выполнение оператора **for** будет продолжаться бесконечно.

Пример:

ПАСКАЛЬ	C++

Program P60;
{ Оператор **for** }
var i : integer;
c : char;
begin
for i:=0 **to** 9 **do** write(i:2);
writeln;
	// Программа P60
	#include <iostream>
	#include <iomanip>
	using namespace std;
	// Оператор for
	int main()
	{
	int i; **char** c;

```
for i:=9 downto 0 do
write(i:2);
  writeln;
  for c:='A' to 'Z' do
write(c:2);
  writeln;
  for c:='Z' downto 'A' do
write(c:2);
  writeln;
  readln;
end.
```

```
for (i=0; i<=9; i++)
cout<<setw(2)<<i;
cout<<endl;
for (i=9; i>=0; i--)
cout<<setw(2)<<i;
cout<<endl;
for (c='A'; c<='Z'; c++)
cout<<setw(2)<<c;
cout<<endl;
for (c='Z'; c>='A'; c--)
cout<<setw(2)<<c;
cout<<endl;
return 0;
}
```

Результаты, выводимые на экран:

```
 0 1 2 3 4 5 6 7 8 9
 9 8 7 6 5 4 3 2 1 0
 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
 Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

Значения параметра цикла не могут быть изменены внутри цикла, т. е.

1) параметру цикла не присваиваются никакие значения;

2) параметр цикла не может быть параметром цикла другого вложенного оператора **for;**

3) нельзя использовать операторы считывания, в которых указывается параметр цикла.

После выхода из оператора **for** значение параметра цикла не определено, за исключением случая, когда выход из цикла осуществляется принудительно, через оператор безусловного перехода **goto**.

Оператор **for** используется для программирования итеративных алгоритмов, в которых число повторений известно. В качестве примера рассмотрим программы, которые вычисляют соответственно $n!$, x^n, и сумму $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+...+\frac{1}{n}$.

ПАСКАЛЬ

```
Program P61;
 { Вычисление факториала }
var n, i, f : 0..MaxInt;
begin
  write('n='); readln(n);
  f:=1;
  for i:=1 to n do f:=f*i;
  writeln('n!=', f);
  readln;
end.
```

C++

```
// Программа P61
// Вычисление факториала
#include <iostream>
using namespace std;
int main()
{
unsigned short int n, i, f;
cout<<"n="; cin>>n;
f=1;
for (i=1; i<=n; i++) f=f*i;
cout<<"n!="<<f<<endl;
return 0;
}
```

```
Program P62;
 { Вычисление x в степени n }
var x, y : real;
    n, i : 0..MaxInt;
begin
  write('x='); readln(x);
  write('n='); readln(n);
  y:=1;
  for i:=1 to n do y:=y*x;
  writeln('y=', y);
  readln;
end.
```

```
// Программа P62
 /*  Вычисление x в степени n */
#include <iostream>
using namespace std;
int main()
{
float x, y;
unsigned short int n, i;
cout<<"x="; cin>>x;
cout<<"n="; cin>>n;
y=1;
for (i=1; i<=n; i++) y=y*x;
cout<<"y="<<y<<endl;
return 0;
}
```

```
Program P63;
 { Вычисление суммы 1 + 1/2 +
1/3 + ... + 1/n }
var n, i : 1..MaxInt;
    s : real;
begin
  write('n=');
  readln(n);
 s:=0;
  for i:=1 to n do s:=s+1/i;
  writeln('s=', s);
  readln;
end.
```

```
// Программа P63
/* Вычисление суммы 1 + 1/2 +
1/3 + ... + 1/n  */
#include <iostream>
using namespace std;
int main()
{
unsigned short int n, i;
double s;
cout<<"n="; cin>>n;
s=0;
for (i=1; i<=n; i++)
s=s+(double)1/i;
cout<<"s="<<s<<endl;
return 0;
}
```

Вопросы и упражнения

❶ Укажите на синтаксической диаграмме *рис. 3.9* пути, которые соответствуют операторам `for` из программы `P60`, для языка ПАСКАЛЬ или C++ (соответственно выбору языка).

❷ Как выполняется оператор **`for`**?

❸ **ПРОАНАЛИЗИРУЙТЕ!** Что выведут на экран следующие программы?

ПАСКАЛЬ	C++

`Program P64;`
`type Zi = (L, Ma, Mi, J, V, S, D);`
`var z : Zi;`
`begin`
`  for z:=L to S do`
	`Программа P64`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`enum Zi {L,Ma,Mi,J,V,S,D};`

```
writeln(ord(z));
  readln;
  for z:=D downto Ma do
writeln(ord(z));
  readln;
end.
```

```
int z;
for (z=(int)L; z<=(int)V; z++)
cout<<z; cout<<endl;
for (z=(int)D; z>=(int)Ma; z--)
cout<<z; cout<<endl;
return 0;
}
```

❹ Даны объявления:

ПАСКАЛЬ	C++

`var i, j, n : integer;`
`x, y : real;`
`c : char;`
	`int i,j,n;`
	`double x,y;`
	`char c;`

Какие из следующих операторов являются синтаксически правильными?

	ПАСКАЛЬ		C++
a)	`for i:=-5 to 5 do j:=i+3`	*a)*	`for (i=-5; i<=5; i++) j=i+3`
b)	`for i:=-5 to 5 do i:=j+3`	*b)*	`for (i=-5; i<=5; i++) i=j+3`
c)	`for j:=-5 to 5 do i:=j+3`	*c)*	`for (j=-5; i<=5; i++) i=j+3`
d)	`for i:=1 to n do y:=y/i`	*d)*	`for (i=1; i<=n; i++) y=y/i`
e)	`for x:=1 to n do y:=y/x`	*e)*	`for (x=1; x<=n; x++) y=y/x`
f)	`for c:='A' to 'Z' do writeln(ord(c))`	*f)*	`for (c='A'; c<='Z'; c++) cout<<(int)c`
g)	`for c:='Z' downto 'A' do writeln(ord(c))`	*g)*	`if (c='Z'; c>='A'; c--) cout<<c`
h)	`for i:=-5 downto -10 do readln(i)`	*h)*	`if (i=-5; i>=-10; i--) cin>>i`
i)	`for i:=ord('A') to ord('A')+ 9 do writeln(i)`	*i)*	`for (i=(int)'A'; i<=(int) 'A'+9; i++) cout<<i`
j)	`for c:='0' to '9' do writeln(c, ord(c):3)`	*j)*	`for (c='0'; c<='9'; c++) cout<<c<<setw(3)<<(int)c`
k)	`for j:=i/2 to i/2+10 do writeln(j)`	*k)*	`for (j=i/2; j<=i/2+10; j++) cout<<j`

❺ **ПРОАНАЛИЗИРУЙТЕ!** Даны объявления:

ПАСКАЛЬ	C++
`var i, m, n : integer;`	`int i, m, n;`

Сколько раз будут выполнены операторы вывода, входящие в состав операторов `for`?

ПАСКАЛЬ	C++
`for i:=m to n do writeln(i);` `for i:=m to n do writeln(2*i);`	`for (i=m; i<=n; i++) cout<<i;` `for (i=m; i<=n; i++) cout<<2*i;`

если:

a) `m=1, n=5;`

b) `m=3, n=5;`

c) `m=3, n=3;`

d) `m=5, n=3.`

❻ **ПРИМЕНИТЕ!** Напишите программу, которая выводит на экран коды символов `'A'`, `'B'`, `'C'`, ..., `'Z'`.

❼ Вычислите для первых n элементов:

a) $1 + 3 + 5 + 7 + \ldots$ и $1 \cdot 3 \cdot 5 \cdot 7 \cdot \ldots$;

b) $2 + 4 + 6 + 8 + \ldots$ и $2 \cdot 4 \cdot 6 \cdot 8 \cdot \ldots$;

c) $3 + 6 + 9 + 12 + \ldots$ и $3 \cdot 6 \cdot 9 \cdot 12 \cdot \ldots$;

d) $4 + 8 + 12 + 16 + \ldots$ и $4 \cdot 8 \cdot 12 \cdot 16 \cdot \ldots$;

e) $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \frac{1}{6} + \ldots$.

Например: При n=3 имеем $1 + 3 + 5 = 9$; $1 \cdot 3 \cdot 5 = 15$.

❽ Напишите программу, которая считывает с клавиатуры n целых чисел и отображает на экране:
a) сколько из введенных чисел являются четными;
b) сумму положительных чисел, введенных с клавиатуры;
c) сумму всех чисел, введенных с клавиатуры;
d) среднее арифметическое значение введенных чисел.

❾ Напишите программу, которая считывает с клавиатуры n символов и отображает на экране, сколько раз в этой последовательности символов были набраны буквы A и B.

❿ Увлеченный садоводством Михай посадил несколько деревьев. В первый год плодоношения с одной из яблонь Михай собрал M яблок, а в следующем году урожай вырос на P процентов. Напишите программу, которая определит количество яблок в урожае N-го года. Числа M, P и N нужно считать с клавиатуры.

⓫ В парке, созданном при поддержке учеников-волонтеров из местной средней школы, для маленьких детей установили качели. В парке играют четверо детей. Напишите программу, которая определяет, могут ли двое из этих детей сесть на качели так, чтобы качели находились в равновесии. Вес каждого из детей считывается с клавиатуры.

3.14.Составной оператор

Синтаксис составного оператора:

ПАСКАЛЬ

<*Составной оператор*> ::= **`begin`** <*Оператор*> {`;` <*Оператор*>} **`end`**

C++

<*Составной оператор*> ::= `{` <*Оператор*> `;`{ <*Оператор*>`;`} `}`

Соответствующая синтаксическая диаграмма представлена на *рис. 3.10.*

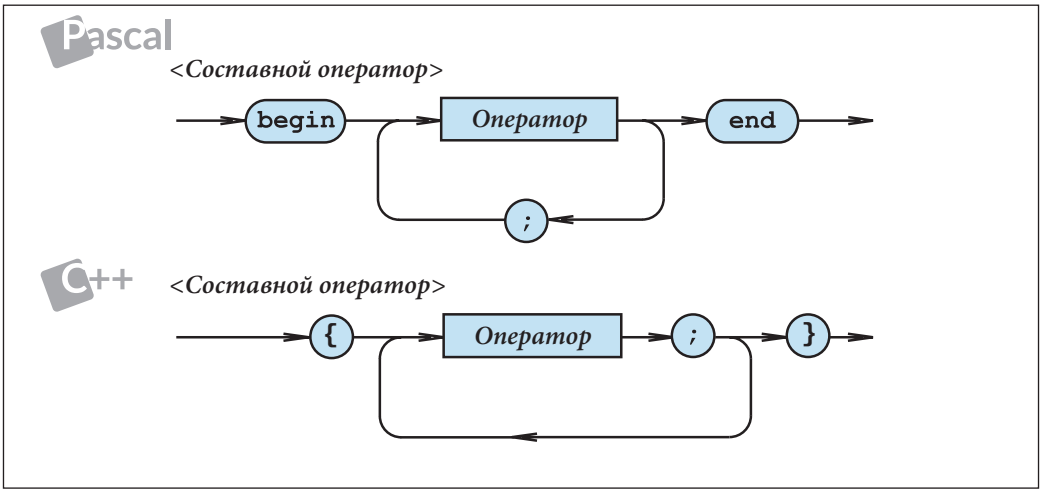


Рис. 3.10. Синтаксическая диаграмма <*Составной оператор*>

Примеры:

	ПАСКАЛЬ		C++

1)	`begin`
	`  a:=x+12;`
	`  p:=q and r;`
	`  writeln(p)`
	`end;`
		1)	`{`
			`a=x+12;`
			`p=q && r;`
			`cout<<p<<endl;`
			`}`

2)	`begin`
	`  write('x=');`
	`  readln(x)`
	`end;`
		2)	`{`
			`cout<<"x=";`
			`cin>>x;`
			`}`

В языке ПАСКАЛЬ ключевые слова **`begin`** и **`end`** соответственно, символы { и } выступают в роли "скобок". Последовательность операторов, заключенных в данные скобки, является, с точки зрения языка, одним оператором. Таким образом, составной оператор используется для того, чтобы несколько операторов поместить в те места программы, где разрешается наличие только одного оператора (см. операторы **`if, for, case, switch`** и др.).

Примеры:

ПАСКАЛЬ

1)
```
if a>0 then begin x:=a+b; y:=a*b end
       else begin x:=a-b; y:=a/b end;
```

2)
```
case c of
  '+' : begin y:=a+b; writeln('Сложение') end;
  '-' : begin y:=a-b; writeln('Вычитание') end;
  '*' : begin y:=a*b; writeln('Произведение') end;
  '/' : begin y:=a/b; writeln('Деление') end;
end ;
```

3)
```
for i:=1 to n do
  begin
    write('x=');
    readln(x);
    s:=s+x
  end;
```

C++

1)
```
if (a>0) {x=a+b; y=a*b ;}
    else {x=a-b; y=a/b;}
```

2)
```
switch (c)
{
case '+' : y=a+b; cout<<Сложение \n"; break;
case '-' : y=a-b; cout<<Вычитание \n"; break;
case '*' : y=a*b; cout<<Произведение \n"; break;
case '/' : y=a/b; cout<<Деление \n"; break;
}
```

3)
```
for (i=1; i<=n; i++)
{
 cout<<"x=";
 cin>>x;
 s=s+x;
}
```

Отметим, что выполняемая часть любой программы является составным оператором, так как она представляет собой последовательность операторов, заключенных в "скобки" **begin** и **end** в языке ПАСКАЛЬ или { и } в С++.

Для того чтобы программы были удобными для чтения, "скобки" операторов пишутся строго одна под другой, а операторы внутри "скобок" смещаются на несколько позиций вправо. Если составные операторы включаются в состав других операторов (**if**, **for**, **case**, **switch** и др.), то их "скобки" смещаются вправо.

В качестве примера рассмотрим программу на языке ПАСКАЛЬ и соответственно на С++, в которой вычисляется среднее арифметическое *n* чисел, считываемых с клавиатуры.

ПАСКАЛЬ

```
Program P65;
 { Среднее арифметическое n чисел }
var x, Suma, Media : real;
    i, n : integer;
begin
   write('n='); readln(n);
   Suma:=0;
   writeln('Введите ', n, ' чисел:');
   for i:=1 to n do
```

```
      begin
         write('x='); readln(x);
         Suma:=Suma+x;
      end;
   if n>0 then
      begin
         media:=Suma/n;
         writeln('среднее=', Media);
      end
   else
      writeln('Media= *****');
   readln;
end.
```

C++

```
// Программа P65
// Среднее арифметическое n чисел
#include <iostream>
using namespace std;
int main()
{
  double x, Suma, Media;
  int i, n;
  cout<<"n="; cin>>n;
  Suma=0;
  cout<<"Введите "<<n<<" чисел:"<<endl;
  for (i=1; i<=n; i++)
  {
  cout<<"x="; cin>>x;
  Suma=Suma+x;
  };
  if (n>0)
  {
  Media=Suma/n;
  cout<<"среднее= "<<Среднее<<endl;
  }
  else cout<<"Media= *****";
  return 0;
}
```

Вопросы и упражнения

❶ Для чего необходим составной оператор?

❷ Укажите на синтаксической диаграмме *рис. 3.10* пути, которые соответствуют составному оператору из программы P65,вариант ПАСКАЛЬ, или соответственно вариант C++.

❸ **П**РИМЕНИТЕ! Напишите программу, которая считывает с клавиатуры *n* чисел и затем выводит на экран:
a) сумму и среднее арифметическое считанных чисел;
b) сумму и среднее арифметическое положительных чисел;
c) сумму и среднее арифметическое отрицательных чисел;

❹ Напишите программу, которая считывает с клавиатуры n символов и затем выводит на экран:
a) количество считанных десятичных цифр;
b) количество четных цифр;
c) количество нечетных цифр;
d) количество считанных букв;
e) количество гласных;
f) количество согласных.
Вводимые символы разделяются нажатием клавиши <ENTER>. Предполагается, что будут вводиться десятичные цифры 0, 1, 2, ... 9 и прописные буквы латинского алфавита *A, B, C, ... Z*.

❺ Напишите программу, которая считывает с клавиатуры целое число *n* и выводит на экран «фигуры», состоящие из цифр:

a)
```
1
12
123
1234
...
1234...n
```

b)
```
123456...n
...
1234
123
12
1
```

c)
```
1
22
333
4444
...
nnn...n
```

d)
```
   1
  222
 33333
  ...
nnnnnnn
  ...
 33333
  222
   1
```

3.15. Оператор `while`

Как и любой другой язык программирования, языки ПАСКАЛЬ и С ++ имеют несколько операторов для управления повторным выполнением других операторов, другими словами, для организации циклов. С одним из таких операторов мы познакомились в предыдущих параграфах, а именно с оператором **`for`**.

Напоминаем, что оператор **`for`** повторно выполняет другой оператор, простой или составной, определенное количество раз, известное на момент написания программы. Однако бывают также ситуации, когда количество повторений неизвестно на момент написания программы, например, когда требуется повторять вычисления до тех пор, пока не будет выполнено определенное условие или не будет достигнута определенная цель. Такие ситуации возникают при написании программ, предназначенных для вычислений, направленных на получение максимальной прибыли, минимизацию возможных затрат, прогнозирование доходов или убытков компании, моделирование процессов распространения эпидемий и т. д.

Операторы повторения, которые позволяют организовать циклы с заранее неизвестным числом повторений, в зависимости от текущих значений других переменных в программе, подразделяются на:

- операторы повторения с начальным тестом (предусловием);
- операторы повторения с конечным тестом (постусловием).

В этом параграфе мы изучим оператор повторения с начальным тестом. Этот оператор обозначается ключевым словом **while**. Он содержит логическое выражение, которое контролирует повторное выполнение другого оператора. Синтаксис этого оператора:

ПАСКАЛЬ

<*Оператор* **while**> ::= **while** <*Логическое выражение*> **do** <*Оператор*>

C++

<*Оператор* **while**> ::= **while** (<*Логическое выражение*>) <*Оператор*>

Синтаксическая диаграмма оператора представлена на *рис. 3.11.*

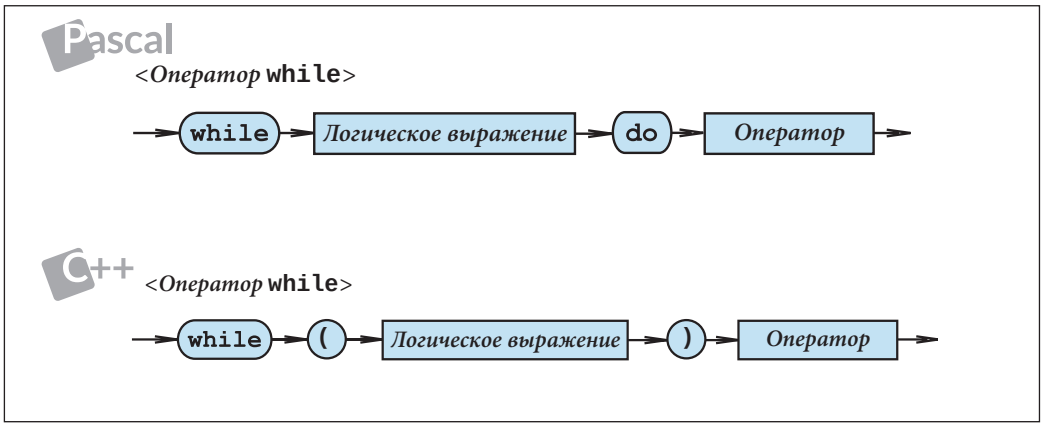


Рис. 3.11. Синтаксическая диаграмма <*Оператор* **while**>>

Примеры:

	ПАСКАЛЬ		C++
1)	`while x>0 do x:=x-1;`	1)	`while (x>0) x=x-1;`

2)	`while x<3.14 do`
	`  begin`
	`  x:=x+0.001;`
	`  writeln(sin(x));`
	`  end;`
		2)	`while (x<3.14)`
			`  {`
			`  x=x+0.001;`
			`  cout<<sin(x);`
			`  }`

3)	`while p do`
	`  begin`
	`  x:=x+0.001;`
	`  y:=10*x;`
	`  p:=y<1000;`
	`  end;`
		3)	`while (p)`
			`  {`
			`  x=x+0.001;`
			`  y=10*x;`
			`  p=y<1000;`
			`  }`

Выполнение простого либо составного оператора из состава оператора **while** повторяется до тех пор, пока значение логического выражения равно true. Если значение логического выражения становится `false`, соответствующий оператор больше не выполняется. Рекомендуется, чтобы логическое выражение было как можно более простым, так как оно вычисляется при каждой итерации.

Обычно оператор **while** используется при организации вычислений с переменными контроля вещественного типа.

В нижеследующих программах оператор **while** применяется для вывода на экран значений функции $y=2x$. Аргумент x принимает значения от x_1 до x_2 с шагом Δx.

ПАСКАЛЬ

```
Program P66;
 { Таблица функции y=2*x }
var x, y, x1, x2, deltaX : real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX=');
  readln(deltaX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      y:=2*x;
      writeln(x:10, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

C++

```
// Programul P66
// Таблица функции y=2*x
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(20)
<<'y';
cout<<endl;
x=x1;
while (x<=x2)
  {
   y=2*x;
   cout<<setw(10)<<x<<setw(20)
          <<y<<endl;
   x=x+deltaX;
  }
return 0;
}
```

Оператор **while** особенно полезен в случаях, в которых трудно вычислить число повторений некоторой последовательности операторов.

Для примера представляем программу P67, которая отображает на экране среднее арифметическое положительных чисел, считанных с клавиатуры.

ПАСКАЛЬ

```
Program P67;
{Среднее арифметическое
положительных чисел,
считанных с клавиатуры}
var x, suma : real;
    n : integer;
begin
  n:=0;
  suma:=0;
```

C++

```
// Программа P67
#include <iostream>
/* Среднее арифметическое
положительных чисел, считанных
с клавиатуры */
#include <iomanip>
using namespace std;
int main()
{
```

```
  writeln('Введите
положительные числа:');
  readln(x);
  while x>0 do
    begin
      n:=n+1;
      suma:=suma+x;
      readln(x);
    end;
  writeln('Вы ввели ',n, '
положительных чисел.');
  if n>0 then writeln
('среднее=', suma/n)
         else
writeln('среднее=*****');
  readln;
end.
```

```
double x, suma;
int n;
n=0;
suma=0;
cout<<"Введите положительные
числа: \n";
cin>>x;
while (x>0)
 {
   n=n+1;
   suma=suma+x;
   cin>>x;
  };
cout<<"Вы ввели "<<n<<
"положительных чисел.\n";
if (n>0) cout<<среднее="<<su-
ma/n<<endl;
else cout<<среднее=*****";
return 0;
}
```

Обратите внимание, что количество повторных выполнений оператора, входящего в состав оператора **while**, не может быть рассчитано заранее. Выполнение оператора **while** заканчивается, когда пользователь вводит число x≤0.

Вопросы и упражнения

❶ Как выполняется оператор while?

❷ Укажите на синтаксических диаграммах *рис. 3.11* пути, которые соответствуют операторам while в программах P66 и P67 для варианта ПАСКАЛЬ или C++, соответственно.

❸ **ПРИМЕНИТЕ!** Используя оператор while, напишите программу, которая выводит на экран значения функции $y = f(x)$ для аргумента, принимающего значения от x_1 до x_2 с шагом Δx:

a) $y = \frac{x}{3} + 2;$

b) $y = \frac{x}{2};$

c) $y = 3x - 4;$

d) $y = 4x - 13.$

❹ **ПРИМЕНИТЕ!** Пользователь вводит с клавиатуры целые положительные числа, разделяемые нажатием клавиши <ENTER>. Признаком конца последовательности является число 0. Напишите программу, которая выводит на экран:
a) сумму и среднее арифметическое четных чисел;
b) сумму и среднее арифметическое нечетных чисел.

❺ **ПРОАНАЛИЗИРУЙТЕ И ПРИМЕНИТЕ!** Напишите программу, которая выводит на экран значения функции $y = f(x)$. Аргумент x принимает значения от x_1 до x_2 с шагом Δx:

a) $y = \begin{cases} x, & x > 3; \\ 2x, & x \le 3; \end{cases}$

b) $y = \begin{cases} 6x, & x \ge 0; \\ 4x, & x < 0; \end{cases}$

c) $y = \begin{cases} x + 6, & x > 5; \\ x - 6, & x \le 5; \end{cases}$

d) $y = \begin{cases} 3 - x, & x > 4; \\ 3 + x, & x \le 4. \end{cases}$

Пример: $y = \begin{cases} x + 1, & x > 8; \\ x - 2, & x \le 8. \end{cases}$

ПАСКАЛЬ

```
Program P68;
 { Таблица функции }
var x, y, x1, x2, deltaX :
real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(del-
taX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      if x>8 then y:=x+1 else
y:=x-2;
      writeln(x:20, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

C++

```
// Программа P68
// Таблица функции
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
double x, y, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(20)
<<'y';
cout<<endl;
x=x1;
while (x<=x2)
 {
    if (x>8) y=x+1; else y=x-2;
    cout<<setw(10)<<x<<setw(20)
<<y
    <<endl;
    x=x+deltaX;
 }
return 0;
}
```

❻ **ОТКРОЙТЕ!** Оператор повторения

ПАСКАЛЬ

```
for i:=i1 to i2 do
              writeln(ord(i))
```

C++

```
for (i=i1; i<=i2; i++) cout<<i;
```

эквивалентен следующей последовательности операторов:

ПАСКАЛЬ	C++

`i:=i1;`
`while i<=i2 do`
`  begin`
`  writeln(ord(i));`
`  i:=succ(i);`
`  end.`
	`i=i1;`
	`while (i<=i2)`
	`  {`
	`  cout<<i;`
	`  i:=i+1;`
	`  }`

Напишите эквивалентную последовательность операторов для оператора повторения:

ПАСКАЛЬ	C++
`for i:=i1 downto i2 do` `  writeln(ord(i))`	`for (i=i1; i>=i2, i--) cout<<i;`

❼ **ПРОАНАЛИЗИРУЙТЕ И ОТКРОЙТЕ!** Даны описания:

ПАСКАЛЬ	C++
`var x1, x2, deltaX : real;` `  i, n : integer;`	`double x1, x2, deltaX;` `int i, n;`

Какие из следующих последовательностей операторов эквивалентны?

	ПАСКАЛЬ		C++

1)	`x:=x1;`
	`while x<=x2 do`
	`begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`end;`
		1)	`x=x1;`
			`while (x<=x2)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`};`

2)	`n:=trunc((x2-x1)/deltaX)+1;`
	`x:=x1;`
	`for i:=1 to n do`
	`  begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`  end;`
		2)	`n=trunc((x2-x1)/deltaX)+1;`
			`x=x1;`
			`for (i=1; i<=n; i++)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`  };`

3)	`n:=round((x2-x1)/deltaX)+1;`
	`x:=x1;`
	`  for i:=1 to n do`
	`  begin`
	`  writeln(x);`
	`  x:=x+deltaX;`
	`  end;`
		3)	`n=round((x2-x1)/deltaX)+1;`
			`x=x1;`
			`  for (i=1; i<=n; i++)`
			`  {`
			`  cout<<x<<endl;`
			`  x=x+deltaX;`
			`  };`

Аргументируйте свой ответ.

❽ **ПРОГРАММИРУЙТЕ**! Используя оператор повторения while, напишите программу, которая вводит с клавиатуры натуральное ненулевое число N и отобразит на экране:

a) последнюю цифру числа N;
b) первую цифру числа N;
c) сумму цифр числа N
d) произведение цифр числа N;
e) сколько раз цифра C появляется в записи числа N. Цифра C считывается с клавиатуры;
f) наибольшую цифру из записи числа N;
g) наименьшую цифру из записи числа N;
h) перевертыш числа N (число, записанное в обратном порядке). Например, для N = 12345 будет выведено 54321;
i) число N, в котором первая и последняя цифры поменяны местами. Например, для N = 1234 будет выведено 4231;
j) сообщение "палиндром" или "не палиндром" в зависимости от того, обладает или нет число таким свойством. Число является палиндромом, если оно совпадает со своим перевертышем. Например, для N = 1234321 будет выведено "палиндром".

ЗНАЕТЕ ЛИ ВЫ, ЧТО
Самый длинный в мире палиндром написан на финском языке Теему Паавolайненом (Teemu Paavolainen) в 1992 году. В нем 49 935 знаков. (Источник ro.wikipedia.org)

❾ Напишите программу, которая:
a) сначала вводит с клавиатуры символ, который обозначим *c*;
b) затем по одному вводить с клавиатуры символы, количество которых априори неизвестно;
c) процесс ввода символов прерывается вводом символа "!";
d) выводит на экран количество появлений символа с во введенной с клавиатуры последовательности символов.

❿ Банк предоставляет годовую процентную ставку, равную p процентам. Напишите программу, которая вычисляет и отображает на экране:
a) сумму денег, которая будет через x лет у клиента, изначально внесшего S_i лей. Значения p, x и S_i считываются с клавиатуры.
b) через сколько лет на счету клиента, который изначально внес S_i лей, будет S_f лей? Значения p, S_i и S_f считываются с клавиатуры.

3.16.Оператор повторения с постусловием

В операторах повторения с постусловием проверка условия окончания цикла осуществляется после выполнения входящих в их состав операторов.

В языке ПАСКАЛЬ оператор повторения с постусловием обозначается ключевыми словами **repeat … until** (повторять… пока), а в C++ – ключевыми словами **do … while** (делать… пока).

Синтаксис этих операторов:

ПАСКАЛЬ

<*Оператор* **repeat**> ::=
repeat <*Оператор*> {**;**<*Оператор*>} **until** <*Логическое выражение*>

C++

< *Оператор* **do...while**> ::=
do {<*Оператор*>**;**{<*Оператор*>**;**}} **while** (<*Логическое выражение*>)

Синтаксические диаграммы представлены на *рисунке 3.12.*

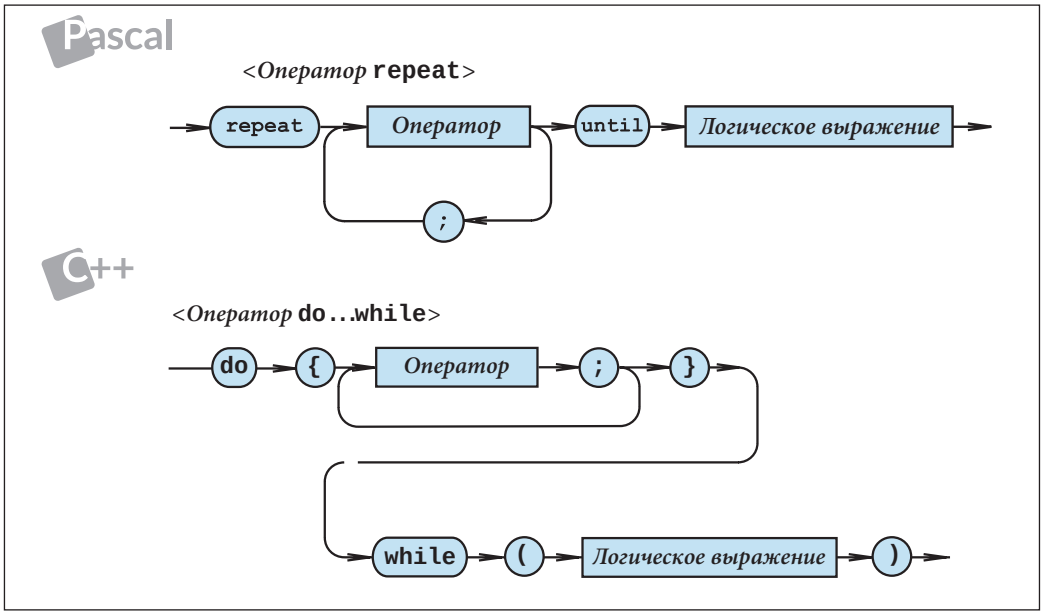


Fig. 3.12. Синтаксические диаграммы операторов повторения с постусловием

	ПАСКАЛЬ		C++
1)	`repeat x:=x-1 until x<0;`	1)	`do x=x-1; while (x>=0);`

2)	`repeat`
	`  y:=y+delta;`
	`  writeln(y)`
	`until y>20.5;`
		2)	`do`
			` { y=y+delta;`
			`  cout<<y<<endl; }`
			`while  (y<=20.5);`

3)	`repeat`
	`  readln(i);`
	`  writeln(odd(i))`
	`until i=0;`
		3)	`do`
			`{  cin>>i;`
			` (i%2==0)? cout<<"True \n" :`
			`  cout<<"False \n";`
			` }`
			`while (i!=0);`

Простые или составные операторы, входящие в состав оператора повторения с постусловием, выполняются повторно до тех пор, пока логическое выражение ложно, в случае языка ПАСКАЛЬ и истинно в случае языка C++. Когда это выражение становится истинным (ПАСКАЛЬ) и соответственно ложным (C++), управление передается оператору, следующему за оператором повторения.

Очевидно, что операторы из состава оператора повторения с постусловием будут выполнены по крайней мере один раз, потому что вычисление логического выражения происходит после их выполнения.

Обычно операторы повторения с постусловием используются вместо операторов повторения с начальным тестом, когда оценку выражения, управляющего повторением, необходимо осуществить после выполнения последовательности операторов из их состава.

Следующие программы отображают на экране четность целых чисел, считанных с клавиатуры.

ПАСКАЛЬ

```
Program P69;
 { Четность целых чисел,
считанных с клавиатуры }
var i : integer;
begin
 writeln('Введите целые чис-
ла:');
  repeat
   readln(i);
   if odd(i) then
   writeln(i:6, '- нечетное
число')
              else
   writeln(i:6, ' - четное
число');
  until i=0;
  readln;
end.
```

C++

```
// Программа P69
/* Четность целых чисел,
считанных с клавиатуры */
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
 int i;
 cout<<"Введите целые числа:";
 do
  {
   cin>>i;
  if (i%2!=0) cout<<setw(6)
<<i<<" нечетное число\n";
   else  cout<<setw(6)<<i<<
"четное число\n";
  }
 while (i!=0);
return 0;
}
```

Выполнение оператора повторения заканчивается, когда пользователь вводит `i`=0.

Оператор повторения с постусловием часто применяется для проверки правильности ввода с клавиатуры исходных данных.

Например, предположим, что необходимо написать программу, считывающую с клавиатуры вещественное число x и выводящую на экран квадратный корень $y = \sqrt{x}$. Очевидно, отрицательные значения переменной x недопустимы.

ПАСКАЛЬ

```
Program P70;
 { Вычисление квадратного
корня }
var x, y : real;
begin
  repeat
    write('Введите
неотрицательное число x=');
    readln(x);
  until x>=0;
  y:=sqrt(x);
  writeln('Квадратный корень
y=', y);
  readln;
end.
```

C++

```
// Программа P70
/* Вычисление квадратного
корня */
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
 double x, y;
 do
  {
    cout<<"Введите
   неотрицательное число x=";
    cin>>x;
   }
```

```
 while (x<0);
 y=sqrt(x);
 cout<<"Квадратный корень
y="<<y;
 return 0;
}
```

После запуска этих программ на выполнение пользователю предлагается ввести число, большее или равное нулю. Если пользователь случайно вводит отрицательное число, операторы чтения и записи, входящие в состав оператора повторения, будут выполнены снова. Повторяющийся процесс будет продолжаться до тех пор, пока пользователь не введет правильное значение.

Из приведенных примеров видно, что оператор повторения с постусловием полезен в ситуации, когда количество повторных выполнений последовательности операторов трудно оценить.

Вопросы и упражнения

❶ Как выполняется оператор повторения с постусловием?

❷ Укажите на синтаксических диаграммах, приведенных на рисунке 3.12, пути, соответствующие операторам повторения с постусловием из программ данного параграфа.

❸ **ПРОАНАЛИЗИРУЙТЕ!** Даны следующие операторы:

	ПАСКАЛЬ		C++

a)	`repeat`
	`  <Оператор 1>;`
	`  <Оператор 2>;`
	`  ...`
	`  <Оператор n>;`
	`until p`
		a)	`do`
			`{ <Оператор 1>;`
			`  <Оператор 2>;`
			`  ...`
			`  <Оператор n>;`
			`}`
			`while (!p)`

b)	`while not p do`
	`  begin`
	`  <Оператор 1>;`
	`  <Оператор 2>;`
	`  ...`
	`  <Оператор n>;`
	`  end.`
		b)	`while (!p)`
			`{`
			`  <Оператор 1>;`
			`  <Оператор 2>;`
			`  ...`
			`  <Оператор n>;`
			`}`

Являются ли эти операторы эквивалентными? Аргументируйте свой ответ.

❹ **ПРИМЕНИТЕ!** Разработайте программу, которая считывает с клавиатуры последовательность символов и отображает на экране:

a) количество прочитанных АРАБСКИХ цифр;

b) количество четных цифр;

c) количество нечетных цифр.

Введенные символы разделяются нажатием клавиши <ENTER>. Допускаются десятичные цифры `0, 1, 2, ..., 9` и символ *, обозначающий конец последовательности.

❺ Разработайте программу, которая считывает с клавиатуры вещественное число x и отображает значение выражения $\frac{1}{x}$. Очевидно, что значение $x=0$ недопустимо. Проведите проверку правильности ввода данных с клавиатуры с помощью оператора повторения с постусловием.

❻ **ПРОАНАЛИЗИРУЙТЕ!** Является ли эквивалентным оператор

ПАСКАЛЬ

```
for i:=i1 to i2 do
                writeln(ord(i))
```

C++

```
for (i=i1; i<=i2; i++) cout<<i;
```

с последовательностью операторов?

ПАСКАЛЬ

```
i:=i1;
repeat
  writeln(ord(i));
  i:=succ(i) ;
until i>i2;
```

C++

```
i=i1;
do
 {
   cout<<i;
   i=i+1;
 }
while (i<=i2);
```

Аргументируйте свой ответ.

❼ **ПРИМЕНИТЕ!** Напишите программу, которая выведет на экран значения функции $y = f(x)$. Аргумент функции x принимает значения от x_1 до x_2 с шагом Δx. Цикл необходимо организовать с помощью оператора повторения с постусловием.

a) $y = 2x;$

b) $y = \frac{x}{3} + 9;$

c) $y = x - 4;$

d) $y = \frac{x}{8} - 6.$

❽ **ПРИМЕНИТЕ!** Разработайте программу, которая считывает с клавиатуры последовательность символов и отображает на экране:

a) количество считанных букв;

b) количество заглавных букв;

c) количество строчных букв.

Введенные символы разделяются нажатием клавиши <ENTER>. Допускаются только буквы латинского алфавита и символ *, обозначающий конец последовательности.

❾ В банке, в целях безопасности, код сейфа меняется еженедельно. Код считается безопасым, если количество четных цифр, входящих в состав кода, превышает количество нечетных цифр. Напишите программу, которая проверяет, удовлетворяет ли введенный код условиям безопасности, чтобы банк мог его использовать. Цифры кода вводятся до тех пор, пока не будет введена цифра 0, которая указывает на конец кода и не является его частью.

3.17. Оператор goto

Как правило, операторы программы выполняются последовательно в том порядке, в котором они появляются в тексте программы. Оператор безусловного перехода **goto** позволяет изменить этот порядок и продолжить работу в другой части текста программы.

Pascal

В языке ПАСКАЛЬ синтаксис рассматриваемого оператора:

<*Оператор* **goto**> ::= **goto** <*Метка*>

Напомним, что метка в ПАСКАЛЕ – это целое число без знака, которое предшествует некоторому оператору программы (*рис. 3.1, стр. 93*). Метки перечисляются в разделе описаний программы после ключевого слова **label**. Синтаксис данного описания:

<*Раздел описания меток*> ::= **label** <*Метка*> {**,** <*Метка*>}**;**

Синтаксические диаграммы соответствующих грамматических единиц представлены на *рис. 3.13.*

Отметим, что перечисление меток в разделе описаний **label** обязательно.

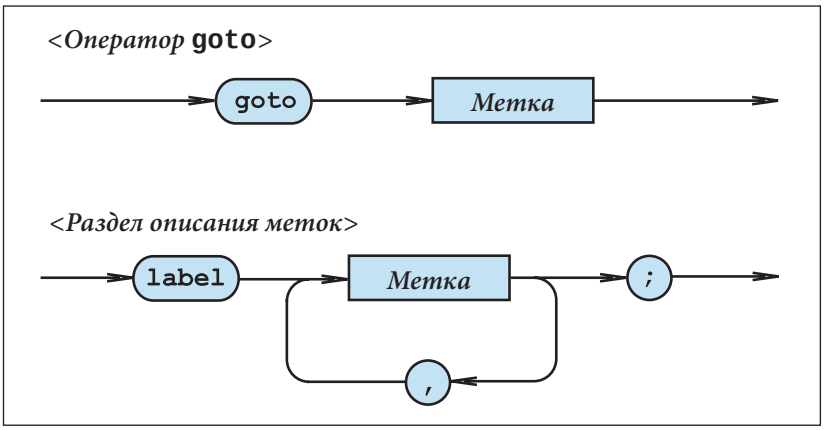


Рис. 3.13. Синтаксические диаграммы <*Оператор* goto> и <*Раздел описания меток*>, язык ПАСКАЛЬ

При выполнении оператора **goto** управление передается оператору, помеченному соответствующей меткой.

В качестве примера представим программу P71, которая вычисляет значение функции

$$y = \begin{cases} x, & x \geq 0; \\ 2x, & x < 0. \end{cases}$$

Значения аргумента *x* считываются с клавиатуры.

```
Program P71;
 { Выполнение оператора goto }
label 1, 2;
```

```
var x, y : real;
begin
  write('x='); readln(x);
  if x>=0 then goto 1;
  y:=2*x;
  writeln('x<0, y=', y);
  goto 2;
1: y:=x;
  writeln('x>=0, y=', y);
2: readln;
end.
```

Если пользователь набирает любое значение x, $x \geq 0$, то выполняется оператор **goto** 1 и управление передается оператору присваивания:

```
1: y:=x;
```

После этого выполняются операторы:

```
   writeln('x>=0 , y=', y);
2: readln;
```

Если пользователь набирает любое значение $x < 0$, то выполняются операторы:

```
y:= 2*x;
writeln('x<0 , y=', y);
goto 2;
```

Последний оператор передает управление оператору:

```
2: readln;
```

В любой программе для меток и операторов должны выполняться следующие правила:
1) каждая метка должна быть описана с помощью ключевого слова **label**;
2) каждая метка должна стоять перед одним и только одним оператором;
3) переходы внутрь структурированных операторов (**if, for, while**, ... и др.) запрещаются.

Пример:

1)
```
if i>5 then 1: writeln('i>5') else writeln('i<=5');
...
goto 1; {Ошибка}
```

2)
```
for i:=1 to 10 do
begin
10: writeln('i=', i);
end;
...
goto 10; {Ошибка}
```

При отсутствии оператора **goto** все операторы программы выполняются в том порядке, в котором они появляются в тексте программы. Следовательно, операторы **goto** нарушают соответствие между текстом программы и порядком выполнения операторов, что усложняет разработку, проверку и отладку программ. Таким образом, использование оператора **goto** нежелательно.

Например, программу P71 можно переписать в следующем виде:

```
Program P72;
 { Исключение оператора goto из программы P71 }
var x, y : real;
begin
  write('x=');
  readln(x);
  if x>=0 then
   begin
    y:=x;
    writeln('x>=0, y=', y);
   end
    else
   begin
    y:=2*x;
    writeln('x<0, y=', y);
   end;
 readln;
end.
```

Как правило, оператор **goto** используется в исключительных случаях, например для уменьшения размеров программы.

C++

В языке C++ оператор безусловного перехода имеет следующий синтаксис:

<*Оператор* **goto**> ::= **goto** <*Метка*> **;**

Напомним, что в языке C++ метка — это идентификатор, за которым следует символ двоеточие, предшествующий оператору программы (*рис. 3.13**).

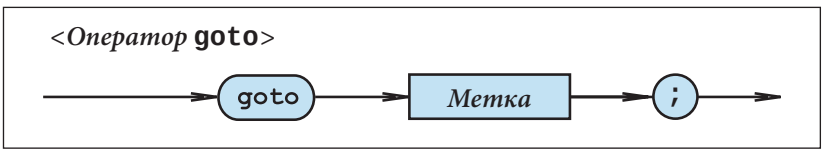


*Рис. 3.13**. Синтаксические диаграммы <Оператор **goto**>, Язык C++

Метка не может появляться более одного раза в теле одной и той же функции, область видимости метки ограничена телом функции, в которой она появляется. Эти метки используются только оператором **goto**, в любом другом контексте помеченный оператор выполняется без учета наличия метки.

Для примера представим программу, которая вычисляет значение функции

$$y = \begin{cases} x, & x \geq 0; \\ 2x, & x < 0. \end{cases}$$

Значения аргумента x считываются с клавиатуры.

```
// Программа P71
#include <iostream>
using namespace std;
int main()
{
  double x, y;
  cout<<"x="; cin>>x;
  if (x>=0) goto A1;
  y=2*x;
  cout<<"x<0, y="<<y;
  goto A2;
A1: y=x;
    cout<<"x>=0, y="<<y;
A2:
return 0;
}
```

Если пользователь вводит значение $x \geq 0$, выполняется оператор **goto** A1 и контроль передается оператору присваивания

```
A1: y=x;
```

После этого выполняются операторы

```
    cout<<"x>=0, y="<<y;
A2: return 0;
```

Если пользователь вводит значение $x < 0$, выполнятся операторы

```
y= 2*x;
cout<<"x<0 , y="<<y;
goto A2;
```

Последний оператор передает управление оператору

```
A2: return 0;
```

Очевидно, что любая метка должна быть префиксом только одного оператора.

При отсутствии **goto** операторы программы выполняются в том порядке, в котором они написаны. Следовательно, операторы **goto** нарушают соответствие между текстом программы и порядком выполнения операторов. Это затрудняет разработку, проверку и отладку программ. Следовательно, использование оператора **goto** не рекомендуется.

Например, указанную выше программу можно переписать следующим образом:

```
// Программа P72
#include <iostream>
using namespace std;
int main()
{
double x, y;
cout<<"x=";
cin>>x;
if (x>=0)
   {
     y=x;
     cout<<"x>=0, y="<<y;
   }
      else
   {
     y=2*x;
     cout<<"x<0, y="<<y;
   }
return 0;
}
```

Как правило, оператор **goto** используется в исключительных случаях, например, для уменьшения размеров программы.

В языке C++ оператор безусловного перехода имеет следующий синтаксис:

Вопросы и упражнения

❶ Для чего необходим оператор **goto**?

❷ Укажите на синтаксических диаграммах *рис. 3.13* (соответственно, *рис. 3.13**) пути, которые соответствуют меткам и операторам **goto** из программ этого параграфа.

❸ **ПРИМЕНИТЕ!** Перепишите следующие программы без использования оператора **goto**:

ПАСКАЛЬ

```
Program P73;
 { Вывод приветствия на экран }
label 1, 2, 3;
var i : 6..23;
begin
  write('Который час?'); re-
adln
(i);
  if i>12 then goto 1;
  writeln('Доброе утро!');
  goto 3;
```

C++

```
// Программа P73
// Вывод приветствия на экран
#include <iostream>
using namespace std;
int main()
{
 unsigned short int i;
  cout<<"Который час? \n";
cin>>i;
 if (i>12) goto Timp1;
 cout<<"Доброе утро!";
 goto Timp3;
```

```
1: if i>17 then goto 2;
  writeln('Добрый день!');
  goto 3;
2: writeln('Добрый вечер!');
3: readln;
end.
```

```
Timp1: if (i>17) goto Timp2;
 cout<<"Добрый день!";
 goto Timp3;
Timp2: cout<<"Добрый вечер!";
Timp3: return 0;
}
```

❹ **ПРОАНАЛИЗИРУЙТЕ!** Прокомментируйте следующую программу:

ПАСКАЛЬ	C++

`Program P74;`
`{ Ошибка }`
`label 1;`
`var i : 1..5;`
`begin`
`  i:=1;`
`1: writeln(i);`
`  i:=i+1;`
`  goto 1;`
`end.`
	`// Программа P74`
	`// Ошибка`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`unsigned short int i;`
	`i=1;`
	`E1: cout<<i;`
	`  i+=1;`
	`  goto E1;`
	`return 0;`
	`}`

❺ **ПРОАНАЛИЗИРУЙТЕ!** Что выведет на экран следующая программа?

ПАСКАЛЬ	C++

`Program P75;`
`{ Ошибка }`
`label 1;`
`var x : real;`
`begin`
`  x:=0;`
`1: writeln(x);`
`  x:=x+1e-30;`
`  goto 1;`
`end.`
	`// Программа P75`
	`// Ошибка`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`
	`double i;`
	`x=0;`
	`E1: cout<<x;`
	`  x=x+1e-30;`
	`  goto E1;`
	`return 0;`
	`}`

Напомним, что в ПАСКАЛЕ работу программы можно прервать нажатием клавиш <CTRL+C> или <CTRL+BREAK>, а в C++, просто закрыв окно **Run**.

❻ **ПРОАНАЛИЗИРУЙТЕ!** Прокомментируйте следующую программу:

ПАСКАЛЬ	C++

`Program P76;`
`{ Ошибка }`
`label 1;`
`var i : integer;`
`begin`
`  i:=1;`
	`// Программа P76`
	`// Ошибка`
	`#include <iostream>`
	`using namespace std;`
	`int main()`
	`{`

```
  while i<=20 do
    begin
      writeln(i);
      1: i:=i+1;
    end;
  goto 1;
end.
```

```
int i;
i=1;
  while (i<=20)
   {
      cout<<i<<endl;
 A1:   i+=1;
   }
  goto A1;
  return 0;
}
```

❼ **ИЗУЧИТЕ!** Работу над заданием можно организовать индивидуально или в группах, разделив на отдельные подзадания.
В языке C ++ есть и другие операторы, которые позволяют делать переходы, например: **`break`**, **`continue`** и **`return`**. Проанализируйте, как работает каждый из них. Измените программу из задания 3 таким образом, чтобы не использовать оператор **`goto`**.

3.18. Структура программы ПАСКАЛЬ / C++

Pascal

Программа на языке ПАСКАЛЬ имеет следующую структуру:

<*Программа*> ::= <*Заголовок программы*>
<*Тело*>**.**

Заголовок программы содержит имя программы и, если необходимо, список формальных параметров:

<*Заголовок программы*> ::= **`Program`** <*Идентификатор*> [(<*Идентификатор*> {**,** <*Идентификатор*>})]**;**

Примеры:

1) **`Program`** `A1;`

2) **`Program`** `B6(Intrare);`

3) **`Program`** `C15(Intrare, Iesire);`

Обычно формальные параметры используются для связи программы со средой ввода/вывода.

Тело программы состоит из раздела объявлений и выполняемой части:

<*Тело*> ::= <*Объявления*>
<*Составной оператор*>

Синтаксис **раздела объявлений:**:

<*Объявления*> ::= [<*Метки*>]
[<*Константы*>]
[<*Типы*>]
[<*Переменные*>]
[<*Подпрограммы*>]

Выполняемая часть представляет собой составной оператор **`begin...end`**.

Синтаксические диаграммы соответствующих грамматических единиц представлены на *рис. 3.14.*

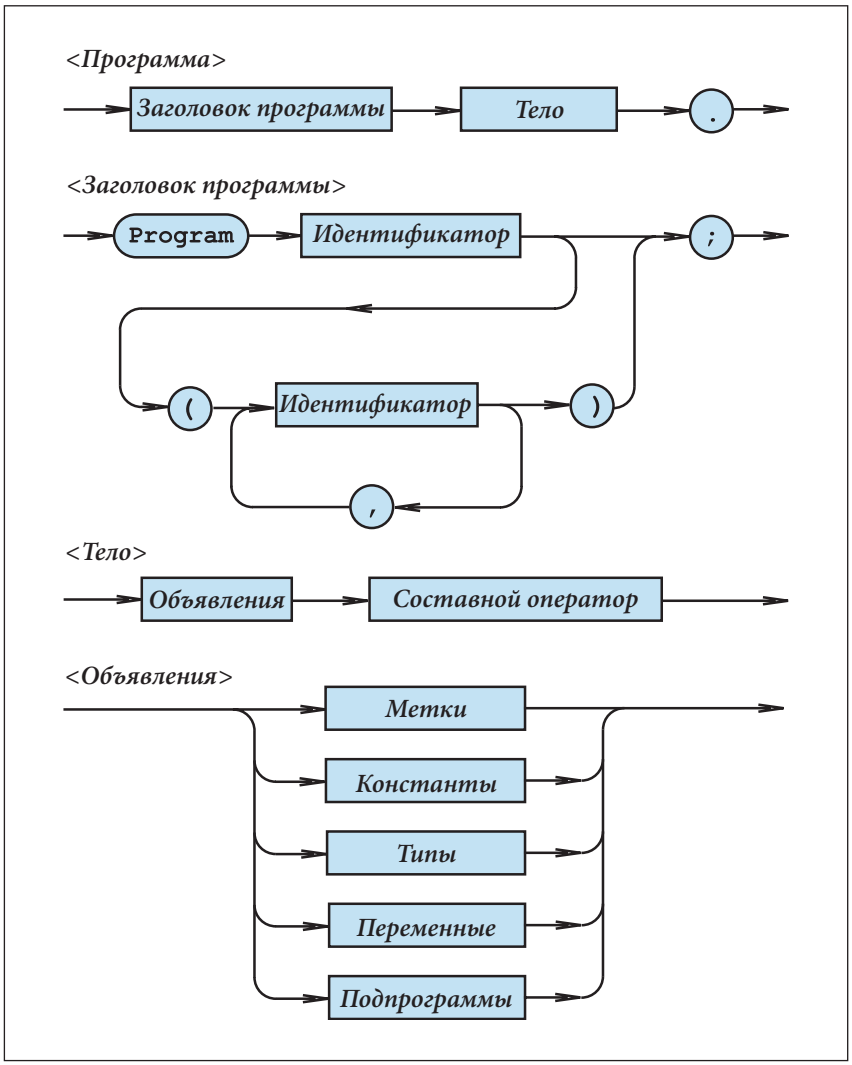


Рис. 3.14. Структура программы на языке ПАСКАЛЬ

Отметим, что конец программы обозначается символом “ **.** ” (точка).

В заключение в качестве примера представляем программу `P77`, которая вычисляет длину дуги окружности, на которую опирается центральный угол в α градусов, и площадь соответствующего сектора.

```
Program P77;
 { Длина дуги окружности и площадь соответствующего сектора }
label 1, 2;
const Pi=3.141592654;
```

```
type grade=0..360;
var alfa : grade;
    raza, lungimea, aria : real;
begin
  write('радиус='); readln(raza);
  if raza<0 then goto 1;
  write('alfa='); readln(alfa);
  lungimea:=Pi*raza*alfa/180;
  writeln('длина=', lungimea);
  aria:=Pi*sqr(raza)*alfa/360;
  writeln('площадь=', aria);
  goto 2;
1: writeln('Ошибка: радиус<0');
2: readln;
end.
```

Программа на С ++ состоит из следующих компонентов:

– директивы;

– глобальные объявления, которые описывают типы пользователей, объявляют типы данных, переменные, глобальные константы, используемые в программе;

– пользовательские функции;

– основная функция main.

Отметим, что программа, написанная на языке С++, представляет собой набор функций, каждая из которых выполняет четко определенное действие. Функция **main** – это основная функция. Именно с нее начинается выполнение любой программы на С++ и ее наличие обязательно. Все остальные функции являются необязательными, и их имена задает программист.

Форма основной функции **main**:

```
int main ()  // Заголовок функции
{            // Начало тела функции
  Локальные объявления;;
  Операторы;;
  return 0;
}           // Конец тела функции, завершение программы.
```

Оператор **return** используется для завершения выполнения функции и возврата значения указанного в ней выражения.

Напомним, что на языке С++ программы пишутся строчными буквами.

Для примера ниже представляем программу, которая вычисляет длину дуги окружности, на которую опирается центральный угол в α градусов и площадь соответствующего сектора.

```
// Программа P77
#include <iostream>
#include <cmath>
using namespace std;
/* Программа P76 на языке C++ */
/* Длина дуги окружности и площадь соответствующего
   сектора */
int main()
{
 const double Pi=3.141592654;
 unsigned int alfa;
 double raza, lungimea, aria;
 cout<<»радиус=»; cin>>raza;
 if (raza<0) goto R1;
 cout<<»alfa=»; cin>>alfa;
 lungimea=Pi*raza*alfa/180;
 cout<<»длина=»<<lungimea<<endl;
 aria=Pi*pow(raza,2)*alfa/360;
 cout<<»площадь=»<<aria<<endl;
 goto R2;
R1: cout<<»Ошибка: радиус<0»;
R2: return 0;
}
```

Вопросы и упражнения

❶ (ПАСКАЛЬ) Для чего необходим заголовок программы? Как обозначается конец программы?

(C++) Каково назначение заголовка функции? Как обозначается конец функции?

❷ Укажите на синтаксических диаграммах *рис. 3.14* пути, которые соответствуют грамматическим единицам из программ данного параграфа.

❸ **ПРИМЕНИТЕ!** Перепишите программу P77 без использования оператора goto. Укажите составные части созданной программы.

❹ (C++) Укажите директивы, основную функцию, начало и конец тела основной функции, локальные объявления и константы программы P77.

Тест для самопроверки № 3

1. Запишите в соответствии с правилами языка программирования ПАСКАЛЬ/C++ следующие выражения:

a)	$(a + b) - 2ab$;	*d)*	$2\alpha\beta - 5\pi r$;
b)	$6a^2 + 15ab - 13b^2$;	*e)*	$\pi r^2 + \alpha\beta^2$;
c)	$(a + b)\,(a - b)$;	*f)*	$xy \vee xz$.

2. Запишите в обычном виде выражения, представленные в соответствии с правилами языка программирования ПАСКАЛЬ/С++ :

	ПАСКАЛЬ		С++
1)	`sqr(a)+2/sqr(b)`	*1)*	`pow(a,2)+2/pow(b,2)`
2)	`2*a/(b+c)`	*2)*	`2*a/(b+c)`
3)	`15*sqrt(a/(a-b))`	*3)*	`15*sqrt(a/(a-b))`
4)	`not(x and y) or z`	*4)*	`!(x && y) || z`
5)	`sqr((a+b)/2)`	*5)*	`pow((a+b)/2,2)`
6)	`(x<>0) and (q<p)`	*6)*	`(x!=0) && (q<p)`

3. Какие из нижеследующих выражений на языке, который вы изучаете, ошибочны?

	ПАСКАЛЬ		С++
1)	`2*a+2*b`	*1)*	`2*a+2*b`
2)	`4*sinx+4*cosy`	*2)*	`4*sinx+4*cosy`
3)	`3*sqr(x)+3/sin(y)`	*3)*	`3*pow(x,2)+3/sin(y)`
4)	`a+2*-b`	*4)*	`a+2*-b`
5)	`not (q and p)`	*5)*	`! (q && p)`
6)	`2*(+x)+((-y))`	*6)*	`2*(+x)+((-y))`

4. Пусть x=1, y=2 и z=3. Вычислите значения нижеследующих выражений:

	ПАСКАЛЬ		С++
1)	`x+2*y+3*z`	*1)*	`x+2*y+3*z`
2)	`(1+x+y-2)*z`	*2)*	`(1+x+y-2)*z`
3)	`x*y+y*(-z)`	*3)*	`x*y+y*(-z)`
4)	`not(x+y+z>0)`	*4)*	`!(x+y+z>0)`
5)	`x*y<y+z`	*5)*	`x*y<y+z`
6)	`(x>y)or(2*x<y+z)`	*6)*	`(x>y)||(2*x<y+z)`

5. Пусть заданы объявления переменных:

ПАСКАЛЬ	С++

`var x : real;`
`i : integer;`
`p : boolean;`
`s : char;`
`Zi : (Luni, Marti, Miercuri, Joi, Vineri, Sambata, Duminica);`
	`double x;`
	`int i;`
	`bool p;`
	`char s;`
	`enum {Luni, Marti, Miercuri, Joi, Vineri, Sambata, Duminica} Zi;`

Определите тип каждого из следующих выражений:

	ПАСКАЛЬ		C++
a)	`i mod 9`	*a)*	`i % 9`
b)	`i/9`	*b)*	`i/9`
c)	`i+x`	*c)*	`i+x`
d)	`ord(pred(Zi))`	*d)*	`Zi+1`
e)	`ord(Zi)+trunc(x)`	*e)*	`Zi+trunc(x)`
f)	`sqr(ord(s))`	*f)*	`pow(s,2)`
g)	`p or (x>i)`	*g)*	`p ||(x>i)`
h)	`chr(i+ord(p))`	*h)*	`char(i+p)`

6. Напишите программу, которая выводит на экран значение выражения $15i(x+y)$. Значения целой переменной i и вещественных переменных x, y считываются с клавиатуры.

7. Даны следующие объявления:

ПАСКАЛЬ	C++

`type FunctiaOcupata = (Muncitor, SefDeEchipa, Maistru, SefDeSantier, Director);`
`StareaCivila = (Casatorit, Necasatorit);`
`var i : integer;`
`x : real;`
`f : FunctiaOcupata;`
`s : StareaCivila;`
	`enum FunctiaOcupata {Muncitor, SefDeEchipa, Maistru, SefDeSantier, Director};`
	`enum StareaCivila {Casatorit, Necasatorit};`
	`int i;`
	`double x;`
	`FunctiaOcupata f;`
	`StareaCivila s;`

Какие из нижеприведенных операторов являются правильными?

	ПАСКАЛЬ		C++
1)	`i:=ord(f)+15`	*1)*	`i=f+15`
2)	`f:=Casatorit`	*2)*	`f=Casatorit`
3)	`x:=ord(f)+1`	*3)*	`x=f+1`
4)	`i:=2*x-15`	*4)*	`i=2*x-15`
5)	`s:=pred(s)`	*5)*	`s=s-1`
6)	`f:=succ(SefDeEchipa)`	*6)*	`f=SefDeEchipa+1`

8. Напишите программу, которая вычисляет значение функции:

$$y = \begin{cases} 9x + 3x^2, x > 15; \\ 3x - 5\sqrt{x + 28}, x \le 15. \end{cases}$$

Значение вещественной переменной *x* считывается с клавиатуры.

9. Используемые в Республике Молдова монеты имеют значения 1, 5, 10, 25 или 50 бань. Напишите программу на языке ПАСКАЛЬ/C++, которая считывает с клавиатуры числовое значение монеты и выводит на экран это же значение, выраженное словами. Например, если пользователь вводит „25”, то на экран выводится „двадцать пять бань”. Если же пользователь вводит отличное от 1, 5, 10, 25 или 50 значение, то на экран выводится сообщение „недопустимое значение”.

10. Напишите программу, которая, используя оператор for, вычисляет для первых n элементов сумму

$$s = \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{n}$$

и произведение

$$p = \frac{1}{1} \cdot \frac{1}{2} \cdot \frac{1}{3} \cdot ... \cdot \frac{1}{n}.$$

11. Напишите программу, которая, используя оператор **`while`**, вычисляет и выводит на экран значения функции

$$y = \begin{cases} 2\sqrt{x + 6}, x \ge 4; \\ 3 - abs(x), x < 4 \end{cases}$$

для значений аргумента x от x_1 до x_2 с шагом Δx.

12. Специальные сообщения мобильной телефонной связи определяются с помощью следующих металингвистических формул:

<*Цифра*> ::= `0|1|2|3|4|5|6|7|8|9`
<*Специальное сообщение*> ::= `*`{<*Цифра*>}`#`

Напишите на языке ПАСКАЛЬ/ C++ программу, которая, используя оператор повторения с постусловием, вычисляет и выводит на экран количество цифр, содержащихся в специальном сообщении. Например, если пользователь вводит:

```
*104#<ENTER>
```

на экран должно выводиться число 3.

Глава 4

МОДУЛИ ПО ВЫБОРУ

Модули из этой главы не являются обязательными. Что это значит? Вы можете выбрать один из модулей и изучить его в удобном для вас темпе, самостоятельно или с одноклассниками, на основе практики и разработки проектов, с помощью компьютерного обучения. Выбрав модуль, вы, совместно с учителем информатики, решите, какие прикладные программы будут использоваться, убедитесь, что эти программы лицензированы или свободно распространяются, что они установлены на компьютерах в компьютерных классах и/или на личных устройствах, в зависимости от обстоятельств.

4.1. Веб-дизайн

В широком смысле *Веб-дизайн* означает разработку веб-сайтов, начиная с их структуры и заканчивая содержанием и формой, в которой они будут предлагаться пользователю. Первым веб-дизайнером был Тим Бернерс-Ли (Tim Berners-Lee), изобретатель одного из самых распространенных интернет-сервисов WWW (*World Wide Web* – Всемирная паутина). Эта «паутина» состоит из множества связанных друг с другом сайтов, документов и информации, к которым можно получить доступ через глобальную сеть Интернет.

В области веб-дизайна используются следующие термины:

Веб-страница – файл, содержащий простой неформатированный текст, отображение компонентов которого указывается с помощью языка разметки, называемого HTML (*Hyper Text Markup Language*). Помимо информации о фактическом стиле отображения текста (символы, абзацы, списки, таблицы и т. д.) язык HTML позволяет создавать элементы управления (кнопки, текстовые поля, раскрывающиеся списки, счетчики и т. д.) и ссылки (*link*-и). Ссылки могут указывать как на объекты внутри веб-страницы, так и на различные внешние объекты. Внешние объекты, такие как изображения, аудиоклипы, видеоклипы и даже другие веб-страницы, могут храниться как на локальном компьютере, так и на любом другом компьютере, подключенном к Интернету.

Веб-документ – набор взаимосвязанных веб-страниц вместе с их внешними объектами, хранящимися на локальном компьютере.

Веб-сайт – Веб-документ, опубликованный в Интернете.

Процесс изучения веб-дизайна будет осуществляться в соответствии с этапами разработки веб-документов:

1. Выбор приложений для разработки веб-документов. Эти приложения могут быть общего или специального назначения.

В случае приложений общего назначения наиболее часто используются офисные приложения, изучаемые в гимназических классах: текстовые редакторы, редакторы электронных таблиц, приложения для электронных презентаций. Эти приложения позволяют сохранять документы в веб-форматах: HTML (текстовый формат, предназначенный для представления веб-страниц) и MHT (формат для архивированных веб-документов). Подчеркнем, что приложения общего назначения полезны только на начальных этапах изучения веб-дизайна, так как не дают больших возможностей использовать все инструменты для персонализации веб-страниц, вставки мультимедийных элементов, обеспечения интерактивности, выбора информации, предоставляемой пользователю в соответствии с его профилем.

В случае специализированных приложений, изучение которых требует больше времени и усилий, ученики смогут детально спроектировать структуру веб-документа и работать на уровне каждого его компонента. Им предоставится возможность развивать свой творческий потенциал, сосредоточив деятельность на художественных аспектах создаваемых ими цифровых продуктов.

Отметим, что крупные компании, предоставляющие Интернет-услуги, предлагают пользователям онлайн-приложения для разработки веб-документов, которые сопровождаются системами поддержки и электронными руководствами, предназначенными для самостоятельного изучения процессов разработки веб-документов**.**

2. Определение целевой группы. В подавляющем большинстве случаев веб-документы размещаются в Интернете и к ним могут получить доступ многие пользователи. Пользователи могут иметь самые разные профили: возрастная категория (дети, подростки, молодежь, взрослые), уровень образования (начальное, среднее, лицейское, профессионально-техническое, высшее), профессиональный статус (школьники, студенты, сотрудники, безработные, пенсионеры), профессиональная сфера (сельское хозяйство, промышленность, торговля, образование, здравоохранение, правоохранительные органы, юстиция, искусство, государственное управление и др.), предпочитаемые языки и т. д.

Очевидно, что влияние веб-документа тем сильнее, чем больше его содержание и стиль представления соответствуют профилю целевой группы, для которой он разработан. Таким образом, веб-дизайнер должен точно определить характеристики целевой группы, для которой предназначен разрабатываемый документ.

Например, документ, предназначенный для младших школьников, будет содержать меньше текста и больше изображений, шрифт текста будет достаточно крупным, чтобы не создавать визуального дискомфорта, структура документа будет максимально простой, без ненужных многоуровневых ссылок. Предлагаемое детям содержание будет соответствовать их возрастным особенностям, оно будет адаптировано к их способу восприятия и познания мира через элементы наблюдения и игры.

Документ для старшеклассников будет иметь более строгую структуру, без отвлекающей графики, с логической системой навигации, без образующих замкнутые круги ссылок.

3. Определение цели веб-документа. Веб-документы могут преследовать различные цели: информация, продвижение, художественное выражение, социализация, предоставление определенных услуг, развлечения, отдых и др.

Исходя из намеченной цели, веб-дизайнер будет выбирать содержание и стили его представления, избегая предоставления нерелевантной по отношению к намеченной

цели информации и загромождения документа ненужными стилями, которые иллюстрируют лишь индивидуальную способность автора обращаться с определенными цифровыми инструментами.

Например, в случае веб-документа о продвижении волонтерства основное внимание будет уделяться содержанию, которое подчеркивает роль волонтерства в жизни сообщества, мотивацию участвовать в такой деятельности, влияние волонтерства на благополучие сообщества в целом или отдельных категорий граждан в частности. Содержимое веб-документа (текстовое, графическое, звуковое и видео) будет мобилизующим, корректным по отношению к персональной информации отдельных бенефициаров, будет содержать примеры передовой практики, предложит пользователям возможные области расширения волонтерской деятельности.

В случае веб-документа, продвигающего гендерное равенство, упор будет делаться на успешные примеры, на передовой опыт. Следует избегать загромождения содержания длинными цитатами из юридических документов, предназначенных только для специалистов в области права. Следует избегать также морализаторского тона. Достижения и недостатки, которые все еще сохраняются в этой области, будут рассматриваться на равной основе с точки зрения обоих полов. Содержание будет больше ориентировано на преодоление стереотипов, на осознание благотворного влияния гендерного равенства на развитие человечества.

4. Разработка структуры веб-документа. Традиционно в повседневной деятельности, которая день ото дня все больше оцифровывается, значение термина *документ* считается очень близким к значению термина *файл*.

В случае веб-документов такая ассоциация может быть неверной, поскольку веб-документ обычно содержит несколько файлов. Основной файл, в формате HTML, читается и отображается программой навигации (*browser* – браузером), который, следуя содержащимся в нем ссылкам, отображает/запускает другие файлы. Обычно это файлы в текстовом, графическом, аудио- и видеоформатах. Очевидно, что ссылки в основном файле могут вести к другим файлам HTML, создавая таким образом многоуровневую структуру.

Следовательно, структуры веб-документа могут быть следующие.

Структура веб-страницы, отображаемая на экране браузером. Эта структура состоит из текстов, изображений и элементов управления: навигации, ввода информации, воспроизведения аудио- и видеопоследовательностей и т. д.

При разработке структуры веб-страницы, отображаемой на экране, дизайнер учтет специфику целевой группы, цель документа, а также особенности оборудования, с помощью которого будут открываться соответствующие страницы: стационарное, мобильное, большой или маленький экран, с помощью клавиатуры, мыши или сенсорных экранов, разрешение экрана, пропускную способность интернет-канала и т. д.

Структура веб-документа, состоящая из главной страницы, подчиненных страниц и внешних объектов. Главная страница находится на верхнем уровне иерархии (уровень 0), а подчиненные страницы - на уровнях 1, 2, 3 и так далее. Практически уровень подчиненной страницы определяется количеством кликов, необходимых для перехода на нее, начиная с главной страницы.

При разработке веб-документов рекомендуется следовать «золотому правилу Интернета»: к любой подчиненной странице можно получить доступ не более чем за три щелчка мышью.

Обычно, чтобы упростить процесс проектирования, структура веб-документа представляется с помощью рисунка, состоящего из небольших прямоугольников, представляющих веб-страницы, и линий, иллюстрирующих переход от одной страницы к другой. Рекомендуется, чтобы рассматриваемый рисунок не содержал «замкнутых кругов», т.е. чтобы его можно было ассоциировать с деревом, ветви которого растут вниз. В «корне» дерева находится главная страница (уровень 0), а подчиненные страницы символизируются его «листьями».

Структура файловой системы веб-документов. Как и любая другая файловая структура, она состоит из каталогов, содержащих локальные внешние объекты, такие как текст, графика, аудио, видео и, конечно же, файлы HTML. Обычно в случае приложений общего назначения эта структура создается автоматически при сохранении веб-документа и состоит из единственного каталога, имя которого происходит от имени HTML-страницы. В случае специализированных приложений соответствующая структура представляет собой древовидную структуру, причем соответствующие каталоги создаются автоматически или веб-дизайнером. В большинстве случаев веб-дизайнеры предпочитают, чтобы один из этих каталогов содержал графические файлы, другой – аудиофайлы, третий – видеофайлы и так далее.

5. Создание веб-страниц. Этот шаг включает в себя создание каждой из веб-страниц, составляющих веб-документ, и состоит из вставки и форматирования составляющих их объектов. В некоторой степени этот процесс похож на процесс создания традиционных документов:

– установление иерархической структуры веб-страницы;
– вставка текстов, списков, таблиц и их форматирование;
– вставка и форматирование диаграмм, изображений, формул, аудиофрагментов, видеофрагментов, элементов навигации и управления;
– вставка ссылок на внешние объекты;
– макет страницы.

Подчеркнем, что объекты, которые будут вставлены на веб-страницы, могут быть взяты из других источников или созданы авторами веб-документа с помощью специализированных приложений, например, с помощью программ для создания и обработки растровых и векторных изображений, цифровой обработки аудио и видео. Очевидно, что в таких случаях работа по созданию веб-документов может быть организована в командах, причем каждый член команды специализируется на определенном типе объектов.

6. Тестирование веб-документа. Для начала веб-документ будет протестирован локально с помощью программы навигации (браузера). В процессе тестирования проверяется доступность всех объектов в веб-документах, правильность ссылок, работа инструментов навигации, правильность отображения изображений, точность воспроизведения аудио и видео.

В случае возникновения ошибок необходимо вернуться к этапу создания веб-страниц и объектов, из которых они состоят.

Подчеркнем, что для тестирования в специализированных приложениях для разработки веб-документов есть специальные средства, которые упрощают проверку документов на стадии доработки.

7. Проверка соблюдения авторских прав. Как правило, в веб-документ можно вставлять объекты, созданные другими авторами. В таких случаях особое внимание необходимо уделить соблюдению авторских прав. Перед вставкой объекта, загруженного из Интернет-источника, печатной или мультимедийной публикации, например, книги с аудио- или видеокомпакт-диска и т. д., веб-дизайнеры проверяют, защищен ли объект авторским правом, и если да, то какой лицензией он защищен (проприетарная или бесплатная). В зависимости от типа лицензии веб-дизайнеры уточнят требования для использования этих объектов в своих веб-документах, и если они решат их использовать, то должны обязательно указать источники.

8. Проверка соблюдения требований относительно защиты персональных данных и безопасности в Интернете. Поскольку подавляющее большинство веб-документов предназначено для публикации в Интернете, они не должны включать персональные данные, будь то личные, одноклассников или учителей. Очевидно, что веб-документ не должен включать средства запроса и сбора персональных данных от пользователей, которые будут иметь доступ к этому документу.

Веб-документ не должен содержать объекты, в которых могут быть вредоносные программы (вирусы) или ссылки на другие веб-сайты, содержащие такие программы.

Как правило, следует избегать ссылок на сайты, на которых явно не указаны выходные данные: владелец сайта, авторы материалов на сайте, дата последнего обновления этих материалов, адреса и номера телефонов компаний, управляющих сайтом.

9. Публикация веб-документа в Интернете. Существует несколько способов публикации веб-документов, основные из которых:

– на личном сайте;

– на сайте учебного заведения;

– на сайте общественной организации.

В зависимости от статуса сервера, на котором был размещен веб-документ, формулируются требования к содержанию и дизайну его веб-страниц, моральная и материальная ответственность авторов, обладателей и владельцев сайта.

В случае личных вебсайтов вся ответственность ложится на их обладателей и владельцев. Очевидно, что в случае учащихся, которые еще не достигли совершеннолетия, создание личных веб-сайтов требует согласия их родителей/законных представителей.

В случае веб-сайтов образовательных учреждений и общественных объединений ответственность несут соответствующие организации и авторы веб-документов. Очевидно, что в случае образовательных учреждений соответствующие документы должны отвечать миссии и целям образовательной системы, а в случае общественных организаций – их миссии, целям, задачам и сферам деятельности.

10. Поддержка и развитие веб-документа. В зависимости от цели веб-документы могут размещаться в Интернете в течение относительно короткого или более длительного периода времени.

Для коротких сообщений веб-документы не нужно обновлять, их следует удалить из виртуального пространства и заархивировать по истечении ожидаемого времени. Такой подход позволяет избежать явлений «информационных помех», «информационной инфляции» и «информационного насыщения».

Для длительных сообщений веб-документы необходимо обновлять. Эти обновления могут производиться как по запросу пользователей, так и по собственной инициативе их авторов. Элементом хорошего тона считается включение в веб-документы некоторых автоматизированных средств, которые позволили бы пользователям сигнализировать о возможной ошибке в содержании или структуре соответствующих документов.

Если веб-документ был разработан и опубликован в онлайн-сервисе, обновления производятся в реальном времени, заменяя и / или изменяя желаемые объекты. Если веб-документ был создан на локальном компьютере, а затем размещен в Интернете, очевидно, что обновление также выполняется на локальном компьютере с последующей публикацией обновленной версии в том же месте в виртуальном пространстве.

Как уже упоминалось выше, рекомендуется четко указывать дату последнего обновления на каждой из веб-страниц, что поможет пользователю оценить новизну полученной информации.

Рекомендуемая учебная деятельность и продукты

Упражнения:

- идентификация объектов в составе веб-страницы;
- идентификация объектов в составе веб-документа;
- представление структуры веб-страницы;
- представление структуры веб-документов в виде рисунка;
- разработка структуры веб-страницы;
- разработка структуры веб-документа;
- разработка файловой структуры веб-документа;
- разграничение этапов разработки веб-документа и разъяснение содержания каждого этапа;
- создание веб-документов с использованием офисных приложений;
- создание веб-документов с использованием специализированных приложений;
- создание веб-документов с помощью онлайн-приложений;
- проверка соблюдения авторских прав (сайты, предложенные преподавателем, и веб-документы, разработанные в классе);
- проверка соблюдения правил интернет-безопасности и защиты персональных данных (сайты, предложенные преподавателем, и веб-документы, разработанные в классе);
- публикация веб-документов в локальной сети и в Интернете.

Исследования:

- Сообщения, отправляемые сайтами, часто посещаемыми учениками.
- Цели сайтов, часто посещаемых учениками.
- Структура, графический вид, функциональность и удобство использования сайтов, предложенных преподавателем и часто посещаемых учениками.
- Взаимосвязь между целями и дизайном сайтов, часто посещаемых учениками.

– Взаимосвязь между спецификой целевых групп и дизайном сайтов, часто посещаемых учениками.
– Актуальность, оригинальность, значимость, объективность, справедливость и правильность информации на сайтах, часто посещаемых учениками.

Разработка веб-сайтов:

– Мой дом / Моя школа / Мой город / Родное село.
– Мой класс / Мой школьный кружок / Моя спортивная секция.
– Ученический совет / Молодежный совет.
– Волонтерство в моей школе, в моем селе / городе.
– Книжный магазин / Библиотека / Дом культуры / Музыкальный салон.
– Музей села / Городской музей.
– История моего села / города.
– Замечательные люди из моего села / города.
– Искусство в моей жизни (в сферах: музыка, пластика, декоративное искусство).
– Спорт в моей жизни.
– Здорово быть здоровым.
– Тренажерные залы / фитнес-залы.
– Салоны красоты / Модные салоны.
– Швейные / Ремонтные мастерские (автомобили, компьютеры, бытовая техника, аудио, видео).
– Крестьянское хозяйство.
– Магазины (детские товары, школьные товары, хозяйственные товары, одежда).

Проекты, охватывающие обширные темы, например касающиеся города, села, школы, будут разрабатываться в командах учащихся, причем каждый член команды специализируется на определенном аспекте, например историческом, демографическом, экономическом, культурном, этнографическом и др.

Также в командах будут разрабатываться проекты, которые характеризуются богатым мультимедийным контентом, специализация членов команды осуществляется по жанрам соответствующих материалов, например растровая графика, векторная графика, анимация, фото, аудио, видео и т. д.

4.2. Компьютерная графика

Компьютерная графика, часто называемая цифровой графикой, – это область компьютерных наук, которая занимается теоретическими основами и прикладными аспектами, связанными с обработкой изображений с помощью компьютеров. Теоретические и практические знания в этой очень важной области информатики, которая имеет тесную связь с изобразительным искусством, используются для синтеза, модификации, хранения и управления изображениями, а также для обработки визуальной информации, полученной из окружающей действительности. Знание основных элементов

компьютерной графики необходимо инженерам, ученым, художникам-оформителям, дизайнерам, фотографам, аниматорам и др. Также для многих компьютерная графика стала приятным занятием вне профессиональной деятельности (хобби).

В зависимости от того, как изображения представлены в памяти компьютера, различаем *точечно-ориентированную графику и объектно-ориентированную графику.*

Точечно-ориентированная графика

Напомним, что в точечно-ориентированной графике изображения представлены в памяти компьютера путем разделения их на микрозоны, называемые *точками* или *пикселями*. Разложение изображения на пиксели производится с помощью *растра* (от латинского слова *raster*, буквально «грабли»).

Растр представляет собой плоскую, обычно прямоугольную поверхность, на которой нанесены два набора параллельных линий, перпендикулярных друг другу. Плотность линий и плотность пикселей соответственно характеризуют разрешающую способность оборудования для воспроизведения или создания изображений.

В процессе оцифровки изображения обход пикселей происходит в том порядке, в котором они читаются: слева направо, сверху вниз. Каждому пикселю соответствует двоичное число, которое в компьютерной форме представляет информацию о его яркости и цвете.

Другими словами, в цифровой форме, в точечно-ориентированной графике, изображение представлено последовательностью двоичных чисел, и вся его обработка выполняется операциями с этими числами.

Последовательности двоичных чисел, содержащие информацию о яркости и цвете пикселей, называются *растровыми цифровыми изображениями* или, проще, *растровыми изображениями.*

Чаще всего растровые изображения используются для обработки визуальной информации, полученной из окружающей среды. Обычно большая часть оборудования для оцифровки изображений, такого как фотоаппараты, видеокамеры, сканеры и т. д., обеспечивает вывод растровых изображений, точнее файлов в форматах, специально разработанных для представления, хранения и обработки растровых изображений. Растровые изображения также используются для отображения визуальной информации на экранах цифровых устройств и для их печати.

Основное преимущество растровых изображений – их достоверность, то есть аккуратность и точность представления или воспроизведения реальности. Из недостатков отметим большой объем памяти, необходимый для хранения растровых изображений, и ухудшение качества при их увеличении.

Объектно-ориентированная графика

Чтобы обеспечить лучшее качество цифровых изображений при изменении их размера и уменьшение объема памяти, необходимой для их хранения, в объектно-ориентированной графике изображения состоят из простых графических объектов: линий, квадратов, прямоугольников, кругов, эллипсов и т. д.

В компьютере каждый графический объект в изображении кодируется набором двоичных чисел обычно называемых *вектором*. Такой набор двоичных чисел содержит всю

информацию, необходимую для рисования объекта: координаты центра и радиуса каждого круга, координаты вершин каждого прямоугольника и т. д. Очевидно, что вектор, который характеризует графический объект, также включает информацию о цвете и толщине линий, которые его образуют, цвете заливки, цветовых градиентах и т.д.

Обработка векторных изображений производится путем пересчета координат и размеров каждого графического объекта, содержащегося в изображении, по формулам из аналитической геометрии. Важно знать, что в аналитической геометрии, в отличие от традиционной геометрии, фигуры определяются не изображениями, а с помощью формул, и их преобразование чисто алгебраическое. Для этого плоскость, в случае двухмерных изображений, и пространство, в случае трехмерных изображений, снабжены системами координат, обычно декартовыми.

Векторные изображения можно увеличивать и уменьшать без ухудшения их качества, пересчитывая их размеры в соответствии с математическими формулами, связанными с каждым графическим объектом. Более того, с помощью расчетов графические объекты можно анимировать, перемещать в пространстве, перекрашивать, трансформировать и т.д., что особенно важно при создании цифровых симуляторов и компьютерных игр. Очевидно, что в случае приложений цифровой графики компоненты аналитической геометрии, на которых основана обработка векторных изображений, то есть формулы и вычислительные алгоритмы, «невидимы» для пользователя, он оперирует только терминами, характерными для графического дизайна.

Недостатком векторных изображений является то, что кодирование сложной визуальной информации простыми графическими объектами, описываемыми с помощью математических формул, приводит к снижению достоверности представления или воспроизведения реального мира. Очевидно, что снижение точности вызвано не самими математическими формулами, а сложностью и большим количеством объектов, которые могут потребоваться для обеспечения желаемого уровня точности. Однако с увеличением производительности современных компьютеров сложность и количество графических объектов перестают быть непреодолимым препятствием, что можно легко наблюдать в случае компьютерных игр, изображения в которых становятся более сложными и близкими к таким, какими мы видим их в реальном мире.

Преобразование изображений

Растровые изображения можно преобразовывать в векторные изображения и наоборот, векторные изображения – в растровые. Более того, современные приложения для обработки изображений позволяют создавать и обрабатывать смешанные цифровые изображения, т. е. изображения, содержащие как растровые, так и векторные компоненты.

Обычно преобразование растровых изображений в векторные изображения производится для того, чтобы уменьшить занимаемый ими объем, объединить реальный мир с элементами виртуального мира (дополненная реальность), распознавать формы. Преобразование векторных изображений в растровые производится с целью их отображения на экранах цифровой техники и для печати.

Обработка растровых изображений

Для формирования и развития навыков обработки растровых изображений рекомендуется прохождение следующих тем.

1. Повторение понятий из гимназических классов: пиксель, растр, растровое изображение, разрешение, размеры, цветовая модель.

2. Рабочее пространство графического редактора: панели, меню, инструменты, линейки, мобильные линейки, направляющие, настройка рабочего пространства.

3. Инструменты рисования: перо, кисть, фон, графические фигуры / примитивы.

4. Управление свойствами инструмента рисования.

5. Инструменты для выделения и редактирования: селектор, маски, ножницы, пипетка, ластик, нож.

6. Инструменты для обработки текста: написание, редактирование, форматирование.

7. Базовая обработка растровых изображений:

– рисование / перерисовка изображений стандартными инструментами;
– импорт изображений из внешних источников (камеры, сканеры);
– вставка и форматирование текстов;
– добавление контента;
– регулировка уровней прозрачности и затенения;
– применение художественных эффектов (мозаика, мокрое стекло, кристаллизация, текстура, размытие).

8. Работа с графическими объектами: создание, редактирование, клонирование, сортировка, группировка / разгруппировка / перегруппировка.

9. Работа со слоями: добавление / редактирование; выделение, группировка и привязка; перемещение, наложение и блокировка.

10. Расширенная обработка текста: масштабирование, поворот, применение геометрических эффектов, применение художественных эффектов.

11. Применение спецэффектов: трехмерная геометрия, размытие, фильтрация, контурная декомпозиция, текстура, четкость.

12. Хранение и распространение изображений: архивирование, подготовка к печати, организация в локальных альбомах и / или в Интернете.

Обработка векторных изображений

Навыки, необходимые для обработки векторных изображений, можно сформировать и развить, изучив следующие темы.

1. Основы объектно-ориентированной графики: точка / узел, прямые линии, кривые Безье (*Bezier*), разрешение, размеры, цветовые модели.

2. Рабочее пространство редактора векторной графики и его настройка.

3. Инструменты рисования: трассировки линий, преобразователь линий, четырехугольники, многоугольники, графические примитивы, построитель сетки.

4. Управление свойствами инструмента рисования.

5. Инструменты для выбора и редактирования: селектор объекта / узла, ножницы, пипетка, нож.

6. Обработка текста.

7. Создание и редактирование объектов.

8. Растеризация векторных изображений.

Предлагаемые для изучения темы можно найти в системах поддержки редакторов растровых и векторных изображений, в открытых образовательных ресурсах, разме-

щенных в сети Интернет. В процессе создания, редактирования и распространения изображений особое внимание необходимо уделить соблюдению правил интернет-безопасности, цифровой этики, авторского права.

Рекомендуемая учебная деятельность и продукты

Упражнения:

- выявление открытых образовательных ресурсов для изучения цифровой графики;
- настройка рабочего пространства графических редакторов;
- идентификация элементов и свойств изображений, предложенных учителем, и изображений, загруженных из Интернета;
- идентификация графических объектов в составе изображений и их свойств;
- рисование / перерисовка растровых и векторных изображений;
- применение эффектов к изображениям и объектам, их составляющим;
- импорт и экспорт изображений;
- преобразование форматов цифровых изображений;
- векторизация растровых изображений;
- растеризация векторных изображений;
- создание альбомов и архивов, локальных и в Интернете.

Исследования:

- Художественное и социально-экономическое влияние цифровых изображений.
- Эволюция цифровых фотоаппаратов.
- Эволюция графических редакторов в бесплатном распространении.
- Эволюция проприетарных графических редакторов.
- Веб-платформы для цифровых изображений.
- Веб-сервисы для цифровых альбомов.
- Графические средства выделения актуальности и важности идей и отражения настроений.
- Художественные трансформации форм в процессе создания и редактирования изображений.
- Художественная и эстетическая ценность цифровых изображений, часто встречающихся в повседневной жизни.
- Реализм и абстракционизм цифровых изображений.
- «Воровство» и «вдохновение» в цифровых изображениях.
- Защита авторских прав на цифровые изображения.
- Компьютерная графика и цифровая этика.

Проекты:

- Тематические выставки цифровой графики.
- Цифровые альбомы: моя школа, мой город, мои друзья, праздничные мероприятия, развлекательные мероприятия, спортивные состязания.
- Художественные тексты: замечательные цитаты, цитаты с выделением основной идеи, эпиграфы.
- Профили выдающихся личностей.
- Генеалогическое древо одного из знаменитостей-земляков.

– Флаеры, тематические постеры, плакаты (права детей, гражданская активность, волонтерство, здоровый образ жизни, экология, свободная тема).
– Цифровые коллекции указателей, знаков (дорожных, по охране труда, предупреждающих, информационных).
– Рисунки для разных школьных предметов.

4.3.Цифровая фотография

Термин *фотография* имеет тройное значение. Под ним мы понимаем: (а) технику создания изображений под действием света; (б) изображение, полученное с помощью этой техники; и (в) отрасль графического искусства, в которой используется эта техника.

Изначально фотографическая техника была основана на использовании светочувствительных материалов, но с развитием информационных и коммуникационных технологий изображения стали получать с помощью матриц, состоящих из светочувствительных ячеек, называемых *пикселями*.

Современные цифровые камеры используют матрицы, содержащие десятки и сотни миллионов пикселей, и хранят эти изображения в графических файлах. Простая цифровая обработка фотографий выполняется компьютерами, встроенными в камеры, но более сложная обработка выполняется с помощью графических редакторов, работающих на персональных компьютерах.

Отметим тот факт, что цифровые фотографии можно получать не только с помощью самих фотоаппаратов, но и с помощью камер, встроенных в мобильные телефоны и персональные планшеты. Безусловно, качество цифровых фотографий зависит в первую очередь от используемого оборудования, поскольку фотографии, сделанные с помощью мобильных телефонов, планшетов и камер для любителей, по качеству ниже, чем цифровые фотографии, сделанные с помощью камер для профессионалов. Более того, в случае изобразительного искусства некоторые профессиональные фотографы предпочитают использовать классические пленочные фотоаппараты, а проявленные изображения впоследствии сканируются и обрабатываются с помощью графических редакторов.

Очевидно, что цифровые фотографии можно отображать на экранах цифрового оборудования, проецировать с помощью мультимедийных устройств, печати и распечатывания на различных статических носителях, таких как бумага, ткань, фольга, керамические предметы и т. д.

Развитие навыков создания цифровых фотографий основано на изучении следующих тем.

1. Основы цифровой фотографии: размер, разрешение, разрешающая способность, цветовые схемы, графические форматы.

2. Классификация и технические характеристики цифровых фотоаппаратов.

3. Устройство и работа цифровых фотоаппаратов.

4. Оборудование для цифровых фотоаппаратов: объективы, фильтры, вспышки, штативы, фотометрические приборы, аксессуары.

5. Факторы качества: композиция, фокус, глубина, выдержка.

6. Техника цифровой фотографии: портреты и люди, натюрморты, репортажи, архитектура, пейзажи, спорт, животные, абстрактные сюжеты.

7. Программные продукты для технической и художественной обработки цифровых фотографий.

8. Методы цифровой обработки фотографий. Преобразования: форматные, геометрические, колористические, художественные.

9. Хранение и распространение цифровых фотографий: архивирование, подготовка к печати, организация в локальных альбомах и / или в Интернете.

Упражнения:

- выявление открытых образовательных ресурсов для изучения цифровой фотографии;
- определение свойств цифровых фотографий, предложенных преподавателем, и фотографий, загруженных из Интернета;
- идентификация компонентов и элементов управления цифровых фотоаппаратов;
- съемка и последующее управление файлами с помощью элементов управления цифрового фотоаппарата;
- подсчет количества информации в цифровых фотографиях;
- расчет возможных разрешений для фотографий;
- изменение размера цифровых фотографий;
- вырезка и редактирование фрагментов цифровых фотографий;
- изменение цветовых схем, контрастности, кривых интенсивности основных цветов;
- определение соотношения размеров экрана и физических размеров фотографии после печати;
- нанесение эффектов витража, кристаллизации, ветра, дождя, плаката;
- редактирование цифровых фотографий;
- художественная обработка цифровых фотографий: пейзаж, индивидуальный портрет, групповой портрет, натюрморт, репортажи, путешествия, архитектурные объекты, спортивные соревнования, животные, абстрактная фотография;
- идентификация знаков, декларирующих авторские права;
- разъяснение правил соблюдения авторских прав;
- использование лицензий на распространение.

Исследования:

- История фотографии.
- История цифровой фотографии.
- Художественное и социально-экономическое влияние цифровой фотографии.
- Эволюция цифровых фотоаппаратов.
- Специфика цифровых зеркальных фотоаппаратов DSLR (*digital single-lens reflex* – цифровой однообъективный зеркальный).
- Факторы, влияющие на качество цифровых фотографий.
- «Воровство» и «вдохновение» в цифровой фотографии.
- Защита авторских прав на цифровые фотографии.
- Цифровая фотография и этика Интернета.
- Десять самых удачных фотографических портретов.

– Животные в природе.
– Самые экзотические цветы.
– Впечатляющие фотоотчеты.
– Карнавалы в картинках.

Проекты:

– Фотоотчеты с праздничных мероприятий, школьных конкурсов, творческих мероприятий, развлекательных мероприятий, общественных мероприятий, волонтерской деятельности.
– Тематические выставки цифровых фотографий.
– Цифровые альбомы: школьная жизнь, жизнь класса, праздники, моя деревня / город, портреты, натюрморты, отчеты, путешествия, архитектурные объекты, пейзажи, спортивные соревнования, животные, абстрактная фотография.
– Коллекции цифровых фото для школьного музея, музея родного села / города.
– Коллекции дидактических цифровых фотографий по различным школьным предметам.
– Коллекции цифровых фотографий для индивидуальных интернет-профилей.

Рекомендуем ученикам загрузить из Интернета наиболее подходящие для каждого из них открытые образовательные ресурсы в области цифровой фотографии, руководства по использованию цифровой фотоаппаратуры и программные продукты цифровой графики. В процессе создания, редактирования и распространения фотографий особое внимание необходимо уделить соблюдению правил интернет-безопасности, цифровой этики, авторского права.

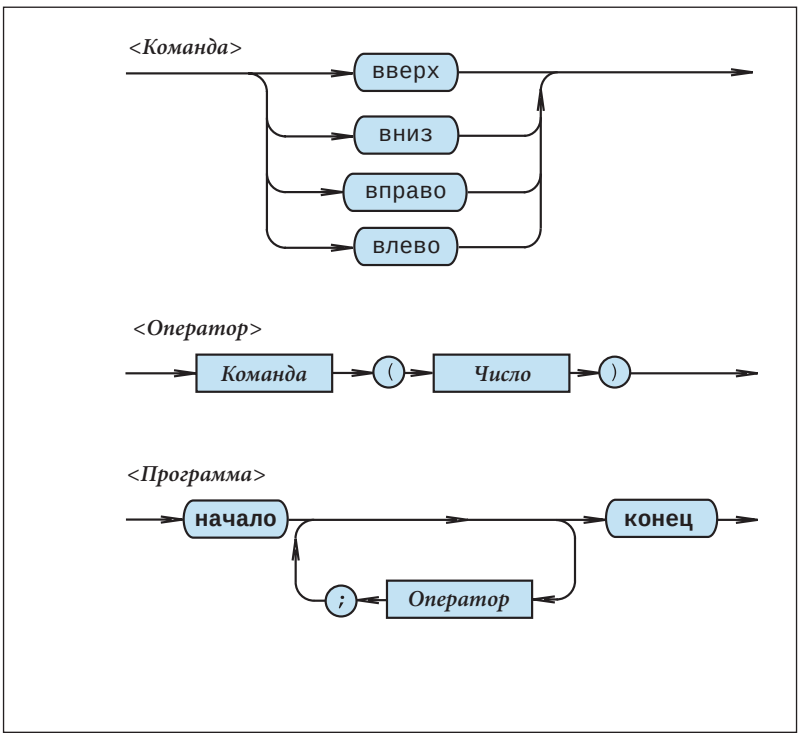
Ответы на задания из тестов для самопроверки

Тест № 1

Pascal

1. *a, c, d* – верно; *b, e, f, g* – ошибочно.

2.



3. *a, c, e, f, h, i, m, o* – верно; *b, d, g, j, k, l, n* – ошибочно.

4. *<Восьмеричная цифра>* ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7`

<Восьмеричное число> ::= [+|–]*<Восьмеричная цифра>*{*<Восьмеричная цифра>*}

5. *<Цифра>* ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<Буква> ::= `a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z`

<Идентификатор> ::= *<Буква>* { *<Буква>* | *<Цифра>* }

6.

1) `x1`

2) `x2`

3) `Delta`

4) `UnghiulAlfa`

5)	`UnghiulBeta`	*8)*	`Vocala`
6)	`DistantaParcursa`	*9)*	`Pixel`
7)	`ListaElevilor`	*10)*	`Culoare`

7.

a)	`3.14`	*f)*	`-984.52`	*k)*	`-3628.297e12`
b)	`265`	*g)*	`-523`	*l)*	`-38.00001`
c)	`23.4635`	*h)*	`+28`	*m)*	`35728.345452e-8`
d)	`+0.000001`	*i)*	`+28000000`	*n)*	`24815`
e)	`6.1532e-5`	*j)*	`614.45e-12`	*o)*	`-296.0020001`

8.

a)	6124,485;	*f)*	$-0{,}03428\cdot10^{-8}$;	*k)*	2005;
b)	+18,315;	*g)*	232847,5213;	*l)*	$+23{,}08\cdot10^{-5}$;
c)	$-218{,}034\cdot10^{-3}$;	*h)*	$-0000012\cdot10^{+2}$;	*m)*	–17502;
d)	193526;	*i)*	18,45;	*n)*	+1;
e)	$1000{,}01\cdot10^{23}$;	*j)*	$623{,}495\cdot10^{-6}$;	*o)*	$-46341{,}2\cdot10^{-6}$.

9. Ключевые слова: **`Program`**, **`var`**, **`begin`**, **`if`**, **`then`**, **`end`**, **`and`**.
Специальные символы: `;` , `,` , `:` , `(` , `)` , `<>` , `:=` , `/` , `'` , `=` , `..`
Идентификаторы: `TA1`, a, b, x, `real`, `readln`, `writeln`.
Числа: `0`.
Строки символов: *`'Уравнение имеет единственный корень'`*,
`'Уравнение имеет бесконечное множество корней'`,
`'Множество корней уравнения пусто'`.

10. *Строка 1* – заголовок; *Строка 2* – раздел объявлений; *Строки 3–15* – раздел операторов.

11. Ошибки будут распознаны компилятором интегрированной среды развития программ на языке ПАСКАЛЬ.

C++

1. *a*, *c*, *d* – верно; *b*, *e*, *f*, *g* – ошибочно.

2. Совпадает с рисунком из ответа на задание № 2, язык ПАСКАЛЬ (см. страницу 186)

3. *a*, *c*, *e*, *f* , *h*, *i*, *m*, *o* – верно; *b*, *d*, *g*, *j*, *k*, *l*, *n* – ошибочно.

4. <*Восьмеричная цифра*> ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7`
<*Восьмеричное число*> ::= [+|–]<*Восьмеричная цифра*>{<*Восьмеричная цифра*>}

5. *<Цифра>* ::= `0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9`

<Буква> ::= `a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z`

<Идентификатор> ::= *<Буква>* { *<Буква>* | *<Цифра>* }

6.

1)	`x1`	*6)*	`DistantaParcursa`
2)	`x2`	*7)*	`ListaElevilor`
3)	`Delta`	*8)*	`Vocala`
4)	`UnghiulAlfa`	*9)*	`Pixel`
5)	`UnghiulBeta`	*10)*	`Culoare`

7.

a)	`3.14`	*f)*	`-984.52`	*k)*	`-3628.297e12`
b)	`265`	*g)*	`-523`	*l)*	`-38.00001`
c)	`23.4635`	*h)*	`+28`	*m)*	`35728.345452e-8`
d)	`+0.000001`	*i)*	`+28000000`	*n)*	`24815`
e)	`6.1532e-5`	*j)*	`614.45e-12`	*o)*	`-296.0020001`

8.

a)	6124,485;	*f)*	$-0{,}03428 \cdot 10^{-8}$;	*k)*	2005;
b)	+18,315;	*g)*	232847,5213;	*l)*	$+23{,}08 \cdot 10^{-5}$;
c)	$-218{,}034 \cdot 10^{-3}$;	*h)*	$-0000012 \cdot 10^{+2}$;	*m)*	–17502;
d)	193526;	*i)*	18,45;	*n)*	+1;
e)	$1000{,}01 \cdot 10^{23}$;	*j)*	$623{,}495 \cdot 10^{-6}$;	*o)*	$-46341{,}2 \cdot 10^{-6}$.

9. Директивы: `#include`
Ключевые слова: **`ussing`**, **`namespace`**, **`int`**, **`float`**, **`if`**, **`return`**
Специальные символы: `;`, `,`, `==`, `!=`, `&&`, `(`, `)`, `{`, `}`, `<`, `>`
Идентификаторы: a, b, x, `iostream`
Числа: `0`.
Строки символов: *"Уравнение имеет единственный корень"*, *"Уравнение имеет бесконечное множество корней"*, *"Множество корней уравнения пусто"*.

10. *Строка 1* – директива; *Строка 3* – основная функция; *Строка 5* – декларативная часть основной функции.

11. Ошибки будут распознаны компилятором интегрированной среды для разработки программ на языке C++.

Тест № 2

Pascal

1. Под типом данных понимается множество значений и множество операций, которые могут выполняться с соответствующими значениями. Примеры типов данных: `integer, real, char`. Значения `1, 2` и `3` – типа `integer`; Значения `1.0, 2.0` и `0.5e+07` – типа real, а значения `'A', 'B'` и `'+'` – типа `char`.

2. В программах на языке ПАСКАЛЬ величины являются объектами, предназначенными для представления данных. Существуют величины двух видов: переменные и константы. Значение любой переменной во время выполнения программы может быть изменено, в то время как значения констант не могут быть модифицированы..

3. `i, j` ↓ переменные типа `integer; a, b, c`↓ переменные типа `real`; `s` ↓ переменная типа `char; p` ↓ переменная типа `boolean; 5, 9` ↓ константы типа `integer; 1.0, 1.0e-01, -2.001` ↓ константы типа `real; 'A'` ↓ константа типа `char; true` ↓ константа типа `boolean`.

4. Множество значений типа данных `integer` состоит из целых чисел, которые могут быть представлены в хост-компьютере языка программирования ПАСКАЛЬ. Наибольшее допустимое значение можно узнать с помощью константы `MaxInt`, доступной в любой программе ПАСКАЛЬ. Обычно наименьшее допустимое целое число данного типа равно `-MaxInt` или `–MaxInt+1`). Примеры операций, допустимых над целыми значениями: `+, -, *, mod, div` и др.

5. Ошибки переполнения возникают в случае, если `x*y>MaxInt`. Например, для `MaxInt=32767` (версия Turbo PASCAL 7.0) ошибки переполнения возникают, если пользователь вводит для `x` и `y` значения больше, чем `200`.

6. Множество значений типа данных `real` состоит из вещественных чисел, которые могут быть представлены в хост-компьютере языка программирования ПАСКАЛЬ. Примеры операций, допустимых над вещественными значениями: `+, -, *, /` (деление) и др. Результаты данных операций являются приближенными вследствие ошибок округления.

7. Ошибки переполнения появляются в случае, когда результат операции `x*y` выходит за область значений типа данных `real`. В версии Turbo PASCAL 7.0 областью значений типа `real` являются –1,7·1038, ..., +1,7·1038. Следовательно, ошибки переполнения возникают в случае, если пользователь вводит для x и y значения больше, например, чем 10^{20}.

8. Тип данных `boolean` включает значения истинности `false` и `true`. Предопределенными операциями типа данных `boolean` являются логические операции **`not`**, **`and`** и **`or`**.

9. См. *рис. 2.1.*

10. Оператор `readln(p, q)` ошибочен вследствие того, что значения переменных логического типа не могут быть считаны с клавиатуры.

11. Множество значений типа `char` включает все печатные символы, упорядоченные согласно таблице кодов *ASCII*. Примеры операций, допустимых над значениями типа `char: ord, pred, succ`, операции отношения.

12.

```
Program RTA1;
 { Порядковые номера десятичных цифр }
begin
  writeln(ord('0'));
  writeln(ord('1'));
  writeln(ord('2'));
  writeln(ord('3'));
  writeln(ord('4'));
  writeln(ord('5'));
  writeln(ord('6'));
  writeln(ord('7'));
  writeln(ord('8'));
  writeln(ord('9'));
end.
```

13. Множество значений перечисляемого типа данных задается списком идентификаторов. Первый в списке идентификатор задает минимальное значение, его порядковый номер равен нулю. Второй идентификатор определяет значение с порядковым номером один, третий – с номером два и т.д. Примеры операций, допустимые над значениями перечисляемого типа: `ord, pred, succ`, операции отношения.

14.

```
Program RTA2;
 { Порядковые номера значений перечисляемого типа }
type FunctiaOcupata = (Muncitor, SefDeEchipa, Maistru,
                       SefDeSantier, Director);
     StareaCivila = (Casatorit, Necasatorit);
begin
  writeln(ord(Muncitor));
  writeln(ord(SefDeEchipa));
  writeln(ord(Maistru));
  writeln(ord(SefDeSantier));
  writeln(ord(Director));
  writeln(ord(Casatorit));
  writeln(ord(Necasatorit));
end.
```

15. Интервальный тип определяет подмножество значений некоторого предопределенного типа: `integer, boolean, char` или *перечисляемого*, называемого базовым типом. Над значениями интервального типа допустимы все операции базового типа.

16. p – интервальный тип, базовый тип `char`. Может принимать значения *'A'*, *'B'*, *'C'*, ..., *'Z'*;
q – интервальный тип, базовый тип `integer`. Может принимать значения 1, 2, 3, ..., 9;
r – интервальный тип, базовый тип `char`. Может принимать значения *'0'*, *'1'*, *'2'*, ..., *'9'*;
s – тип `char`. В качестве значения может принимать любой печатный символ *ASCII*;
t – тип `integer`. В качестве значения может принимать любое целое число, допустимое в конкретной машинной реализации языка;
u – перечисляемый тип. Может принимать значения `Alfa, Beta, Gama, Delta`.

17. Порядковые типы данных: integer, `boolean`, `char`, перечисляемый, интервальный. Общие свойства:
a) любое значение порядкового типа имеет порядковый номер, который можно узнать с помощью предопределенной функции ord;
b) к значениям любого порядкового типа можно применить операции отношения;
c) для порядковых типов существуют предопределенные функции `pred` (предшествующий) и `succ` (последующий).

18. Построчно будут выведены следующие значения:

```
Y    E    -6    10    1    3
```

19. Построчно будут выведены следующие значения:

```
1    0    2    true    false    true
```

20. Два типа являются идентичными, если они описаны тем же именем.

Два типа являются совместимыми в случае истинности одного из следующих утверждений:
a) рассматриваемые типы идентичны;
b) один тип является интервальным типом второго;
c) оба типа являются интервальными, с одинаковыми базовыми типами.
Тип данных является анонимным, если он описан неявно при объявлении переменных.

21. Идентичные типы:: *a)* T1, `integer` şi T2; *b)* T3 şi T4;
Совместимые типы: *a)* T1, `integer`, T2, T3, T4, T5 şi `1..100`; *b)* T6, T7 şi T8; *c)* T9 şi T10.
Анонимные типы: *a)* `1..100`; *b)* `(Alfa, Beta, Gama, Delta)`.

22. `Alfa – integer; Beta – real; Indicator – boolean; Mesaj –` строка символов; `Semn – char; Inscris –` строка символов.

23.

```
const a = 3.14;
      b = 9.8;
var i, j : integer;
    x, y : real;
```

1. Под типом данных подразумевается множество значений и множество операций, которые могут быть выполнены с этими значениями. Примеры типов данных: **`int`**, **`double`**, **`char`**. Значения `1`, `2` и `3` относятся к целочисленному типу **`int`**; значения `1.0`, `2.0` и `0.5e + 07` относятся к вещественному типу **`double`**, а значения `'A'`, `'B'` и `'+'` – к типу **`char`**.

2. В программе на C++ величины – это объекты, предназначенные для представления данных. Есть два вида величин: *переменные* и *константы* (постоянные). Во время выполнения программы значение любой переменной можно изменить, а значения констант изменить нельзя.

3. `i`, `j` – переменные типа **`int`**; `a`, `b`, `c` – переменные типа double; `s` – переменная типа **`char`**; `p` – переменная типа **`bool`**; 5, 9 – константы типа int; `1.0`, `1.0e-01`, `-2.001` – константы типа double; 'A' – константа типа **`char`**; **`true`** – логическая константа.

4. Множество значений типа данных **`int`** состоит из целых чисел, которые могут быть представлены на хост-компьютере языка программирования. Максимальное значение указывает константа `INT_MAX`, известная любой программе на C++. Обычно минимальное значение, допустимое для этого типа данных, составляет (`INT_MAX+1`). С целочисленными значениями можно выполнять следующие операции: +, -, *, % и др.

5. Ошибки переполнения будут возникать, когда `x*y>INT_MAX`. Например, для компиляторов, выделяющих 2 байта (16 бит), `INT_MAX = 32767`, ошибки переполнения будут возникать, если пользователь введет значения `x` и `y` больше `200`. Для компиляторов, выделяющих для типа данных **`int`** 4 байта (32 бита), `INT_MAX = 2147483647`, ошибки переполнения будут возникать, если пользователь введет значения x и y больше 46350.

6. Множество значений рассматриваемого типа данных состоит из вещественных чисел, которые могут быть представлены на хост-компьютере языка. Операции, которые могут быть выполнены с вещественными значениями: +, -, *, / (деление) и т. д. Результаты этих операций обычно являются приблизительными из-за ошибок округления.

7. Ошибки переполнения будут возникать, когда результат операции x*y не попадет в диапазон значений типа данных **`double`**. Для компиляторов, которые выделяют 8 байтов для типа данных **`double`**, его диапазон значений составляет $-1,7 \cdot 10^{308} \ldots +1,7 \cdot 10^{308}$. Поэтому ошибки переопределения будут возникать, когда пользователь введет для x и y значения большие, чем, например, 10^{160}.

8. Тип данных **`bool`** включает значения истинности `false` и `true`. Предопределенные операции с типом данных **`bool`**: !, && и `||`.

9. См. *рис. 2.1*.

10. При выполнении оператора `cin>>p>>q` логическим переменным с клавиатуры могут быть присвоены только числовые значения: 0 (вместо `false`) или 1 (вместо `true`).

11. Множество значений типа данных **`char`** включает все печатные символы, отсортированные в соответствии с таблицей кодов ASCII. Со значениями типа данных **`char`** можно выполнять операции отношения.

12.

```
//Программа RTA1
// порядковые номера десятичных чисел
#include <iostream>
using namespace std;
int main()
{
cout<<int('0')<<endl;
cout<<int('1')<<endl;
cout<<int('2')<<endl;
cout<<int('3')<<endl;
cout<<int('4')<<endl;
cout<<int('5')<<endl;
cout<<int('6')<<endl;
cout<<int('7')<<endl;
cout<<int('8')<<endl;
cout<<int('9')<<endl;
return 0;
}
```

13. Множество значений любого *перечисляемого* типа данных определяется списком идентификаторов. Первый идентификатор в списке обозначает наименьшее значение с нулевым порядковым номером. Второй идентификатор будет иметь порядковый номер один, третий – порядковый номер два и так далее. В C++ *перечисляемые* типы являются целочисленными типами, и идентификаторы перечисляемого типа могут использоваться так же, как целочисленные переменные. В языке C++ нет предопределенных функций для прямого определения последующего или предшествующего значения. В арифметических выражениях любые данные *перечисляемого* типа обрабатываются как целые числа, причем преобразование в **int** является неявным, но преобразование целого числа в перечисляемый тип должно быть запрошено явно. Таким образом, для перехода к последующему или предшествующему элементу будут применяться операции изменения типа с использованием так называемых операторов преобразования. Например, пусть x будет целым числом, с помощью (**char**)x (или **char**(x)) мы изменим тип x с **int** на **char**.

14.

```
//Программа RTA2
//Порядковые номера значений перечисляемого
#include <iostream>
using namespace std;
int main()
{
 enum FunctiaOcupata {Muncitor, SefDeEchipa, Maistru,
  SefDeSantier, Director};
 enum StareaCivila {Casatorit, Necasatorit};
  cout<<Muncitor<<endl;
  cout<<SefDeEchipa<<endl;
  cout<<Maistru<<endl;
  cout<<SefDeSantier<<endl;
```

```
  cout<<Director<<endl;
  cout<<Casatorit<<endl;
  cout<<Necasatorit<<endl;
 return 0;
}
```

15. p – T1 это тип **char**. В качестве значения может принимать любой печатный символ *ASCII*;

q – T2 – это тип **int**. В качестве значения может принимать любое целое число, которое может быть представлено на хост-компьютере языка программирования;

r – T3 – это тип **char**. В качестве значения может принимать любой печатный символ *ASCII*;

s – типа **char**. В качестве значения может принимать любой печатный символ *ASCII*;

t – типа **int**. В качестве значения может принимать любое целое число, которое может быть представлено на хост-компьютере языка программирования;

u – перечисляемого типа. Может принимать значения Alfa, Beta, Gama, Delta.

16. Порядковые типы данных: **int, bool, char, enum.** Общие свойства:

a) любое значение порядкового типа имеет порядковый номер, который можно узнать с помощью **оператора преобразования** (**int**)x или **int**(x), (x – принадлежит порядковому типу);

b) над значениями любого порядкового типа данных допустимы операции отношения.

17. Построчно будут выведены следующие значения:

```
Y   E   -6   10   1   3
```

18. Построчно будут выведены следующие значения:

```
1   0   2   1   0   1
```

19. Два типа являются идентичными, если они были определены одинаковым именем или если у них имена разные, однако типы эквивалентны через транзитивность.

Компиляторы проверяют совместимость типов в следующих случаях:

– при присваивании
– при передаче параметров
– при вычислении значений выражений.

Тип данных является анонимным, если он был определен по умолчанию, в объявлении переменных.

20. Идентичные типы: *a)* T1, T2, T3, T4, T5 şi int; *b)* T6,T7 и T8; *c)* T9, T10 şi **char**; Анонимные типы: (Alfa, Beta, Gama, Delta).

21. Alfa – **int**; Beta – **float**; Indicator – **bool**; Mesaj – строка символов; Semn – **char**; Inscris – строка символов.

22.

```
const float a = 3.14, b = 9.8;
int i, j;
float x, y;
```

Тест № 3

Pascal

1.

a) `(a+b)-2*a*b;`

b) `6*sqr(a)+15*a*b-13*sqr(b);`

c) `(a+b)*(a-b);`

d) `2*Alfa*Beta-5*Pi*r;`

e) `Pi*sqr(r)+Alfa*sqr(Beta);`

f) `x and y or x and z.`

2.

a) $a^2+\frac{2}{b^2};$

b) $\frac{2a}{b+c};$

c) $15\sqrt{\frac{a}{a-b}};$

d) $\overline{xy}\vee z;$

e) $\left(\frac{a+b}{2}\right)^2;$

f) $(x\neq 0)\ \&\ (q<p).$

3. *a*, *c*, *e*, *f* ↓ правильно; *b*, *d* ↓ ошибочно.

4. *a*) 14; *b*) 6; *c*) –4; *d*) `false`; *e*) `true`; *f*) `true`.

5.

a) `integer`

b) `real`

c) `real`

d) `integer`

e) `integer`

f) `integer`

g) `boolean`

h) `char`

6.

```
Program RTA3;
var i : integer;
    x, y : real;
begin
  writeln('Введите i='); readln(i);
  writeln('Введите x='); readln(x);
  writeln('Введите y='); readln(y);
  writeln(15*i*(x+y));
  readln;
end.
```

7. $a, c, e, f \downarrow$ правильно; $b, d \downarrow$ ошибочно.

8.

```
Program RTA4;
var x, y : real;
begin
  write('x='); readln(x);
  if x>15 then y:=9*x+3*sqr(x)
          else y:=3*x-5*sqrt(x+28);
  writeln('y=', y);
  readln;
end.
```

9.

```
Program RTA5;
var i : integer;
begin
  write('Введите числовое значение монеты: ');
  readln(i);
  case i of
    1 : writeln('один бан');
    5 : writeln('пять банов');
    10 : writeln('десять банов');
    25 : writeln('двадцать пять банов');
    50 : writeln('пятьдесят банов');
    else writeln('недопустимое значение');
  end;
  readln;
end.
```

10.

```
Program RTA6;
var n, i : integer;
    s, p : real;
begin
  write('n='); readln(n);
  s:=0; p:=1;
  for i:= 1 to n do
    begin
      s:=s+(1/i); p:=p*(1/i);
    end;
  writeln('s=', s);
  writeln('p=', p);
  readln;
end.
```

11.

```
Program RTA7;
var y, x, x1, x2, deltaX : real;
begin
  write('x1='); readln(x1);
  write('x2='); readln(x2);
  write('deltaX='); readln(deltaX);
  writeln('x':10, 'y':20);
  writeln;
  x:=x1;
  while x<=x2 do
    begin
      if x>=4 then y:=2*sqrt(x+6) else y:=3-abs(x);
      writeln(x:20, y:20);
      x:=x+deltaX;
    end;
  readln;
end.
```

12.

```
Program RTA8;
var c : char; { символ, считываемый с клавиатуры }
    n : integer; { количество цифр в сообщении }
begin
  n:=0;
  writeln('Введите сообщение:');
  repeat
    read(c);
    if (c<>'*') and (c<>'#') then n:=n+1;
  until c='#';
  writeln('количество цифр n=', n);
readln;
end.
```

1.

a) `(a+b)-2*a*b;`

b) `6*pow(a,2)+15*a*b-13*pow(b,2);`

c) `(a+b)*(a-b);`

d) `2*Alfa*Beta-5*Pi*r;`

e) `Pi*pow(r,2)+Alfa*pow(Beta,2);`

f) `x && y || x && z.`

2.

a) $a^2+\frac{2}{b^2}$;

b) $\frac{2a}{b+c}$;

c) $15\sqrt{\frac{a}{a-b}}$;

d) $\overline{xy}\vee z$;

e) $\left(\frac{a+b}{2}\right)^2$;

f) $(x\neq 0)\ \&\ (q<p)$.

3. *a*, *c*, *d*, *e*, *f* ↓ правильно; *b* ↓ ошибочно.

4. *a*) 14; *b*) 6; *c*) –4; *d*) 0; *e*) 1; *f)* 1.

5.

a) **int**;
b) **int**;
c) **double**;
d) **int**;
e) **int**;
f) **int**;
g) **bool**;
h) **char**.

6.

```
// Programul RTA3
#include <iostream>
using namespace std;
int main()
{
int i;
double x, y;
cout<<"Dati i="; cin>>i;
cout<<"Dati x="; cin>>x;
cout<<"Dati y="; cin>>y;
cout<<15*i*(x+y);
return 0;
}
```

7. *a*, *c*, ↓ правильно; *b*, *e*, *f* ↓ ошибочно.

8.

```
// Программа RTA4
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
double x, y;
cout<<"x="; cin>>x;
```

```
if (x>15) {y=9*x+3*pow(x,2);} else {y=3*x-5*sqrt(x+28);}
cout<<"y="<<y<<endl;
return 0;
}
```

9.

```
// Программа RTA5
#include <iostream>
using namespace std;
int main()
{
int i;
cout<<"Введите значение монеты: ";
cin>>i;
switch (i)
{
    case 1 : cout<<"один бан"; break;
    case 5 : cout<<"пять банов"; break;
    case 10 : cout<<"десять банов"; break;
    case 25 : cout<<"двадцать пять банов"; break;
    case 50 : cout<<"пятьдесят банов"; break;
    default : cout<<"недопустимое значение";
}
  return 0;
}
```

10.

```
// Программа RTA6
#include <iostream>
using namespace std;
int main()
{
int n, i;
double s, p;
cout<<"n="; cin>>n;
s=0; p=1;
for (i=1; i<=n; i++)
{s=s+(double)1/i; p=p*(double)1/i;}
cout<< "s="<<s<<endl;
cout<<"p="<<p;
return 0;
}
```

11.

```
// Программа RTA7
#include <iostream>
#include <cmath>
#include <iomanip>
using namespace std;
int main()
{
double y, x, x1, x2, deltaX;
cout<<"x1="; cin>>x1;
cout<<"x2="; cin>>x2;
cout<<"deltaX="; cin>>deltaX;
cout<<setw(10)<<'x'<<setw(10)<<'y'<<endl;
x=x1;
while (x<=x2)
{
    if (x>=4) {y=2*sqrt(x+6);} else {y=3-abs(x);}
    cout<<setw(10)<<x<<setw(10)<<y<<endl;
    x=x+deltaX;
}
return 0;
}
```

12.

```
// Программа RTA8
#include <iostream>
using namespace std;
int main()
{
char c;     // читается символ с клавиатуры
int n;  // количество цифр в сообщении
n=0;
cout<<"Введите сообщение:"<<endl;
do
{cin>>c;
if ((c!='*')&&(c!='#')) { n=n+1;}
}
while (c!='#');
cout<<"Количество цифр n="<<n<<endl;
return 0;
}
```

Приложение 1. Словарь языка ПАСКАЛЬ

1. <*Буква*> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

2. <*Цифра*> ::= 0|1|2|3|4|5|6|7|8|9

3. <*Специальный символ*> ::= +|-
|*|/|=|<|>|]|[|,|(|)|:|;|^|.|@|{|}|$|#|<=|>=|
<> | := |..| <*Ключевое слово*> | <*Эквивалентный символ*>

4. <*Эквивалентный символ*> ::= (*|*)|(.|.)

5. <*Ключевое слово*> ::= **and** | **array** | **begin** | **case**| **const** | **div** | **do** | **downto** | **else** | **end** | **file** | **for** | **function** | **goto** | **if** | **in** | **label** | **mod** | **nil** | **not** | **of** | **or** | **packed** | **procedure** | **program** | **record** | **repeat** | **set** | **then** | **to** | **type** | **until** | **var** | **while** |**with**

6. <*Идентификатор*> ::= <*Буква*> { <*Буква*> | <*Цифра*> }

7. <*Директива*> ::= <*Буква*> {<*Буква*> | <*Цифра*>}

8. <*Целое без знака*> ::= <*Цифра*> {<*Цифра*>}

9. <*Знак*> ::= + | –

10. <*Целое число*> ::= [<*Знак*>] <*Целое без знака*>

11. <*Масштабный множитель*> ::= <*Целое число*>

12. <*Вещественное число*> ::= <*Целое число*> e <*Масштабный множитель*> |
<*Целое число*> . <*Целое без знака*>
[e <*Масштабный множитель*>]

13. <*Число*> ::= <*Целое число*> | <*Вещественное число*>

14. <*Строка символов*> ::= '<*Элемент строки*> { <*Элемент строки*> } '

15. <*Элемент строки*> ::= '' | <*Любой печатный символ*>

16. <*Метка*> ::= <*Целое без знака*>

17. <*Комментарий*> ::= (* <*Любая последовательность символов, заключенная внутри фигурных скобок*> *)

Примечание. Терминальные символы { и } из формулы (17) переданы эквивалентными символами (* и *) соответственно.

Приложение 2. Синтаксис языка ПАСКАЛЬ

1. <*Программа*> ::= <*Заголовок программы*>
<*Тело*>**.**

2. <*Заголовок программы*> ::= **Program** <*Идентификатор*> [**(**<*Идентификатор*>{**,** <*Идентификатор*>}**)**]**;**

3. <*Тело*> ::= <*Объявления*>
<*Составной оператор*>

4. <*Объявления*> ::= [<*Метки*>]
[<*Константы*>]
[<*Типы*>]
[<*Переменные*>]
[<*Подпрограммы*>]

5. <*Метки*> ::= **label** <*Метка*> {**,** <*Метка*>}**;**

6. <*Константы*> ::= **const** <*Определение константы*>**;** { <*Определение константы*>**;**}

7. <*Определение константы*> ::= <*Идентификатор*> **=** <*Константа*>

8. <*Константа*> ::= [**+** | **-**] <*Число без знака*> | [**+** | **-**] <*Имя константы*> | <*Строка символов*>

9. <*Типы* > ::= **type** <*Описание типа*>**;** { <*Описание типа*>**;**}

10. <*Описание типа*> ::= <*Идентификатор*> **=** <*Тип*>

11. <*Переменные*> ::= **var** <*Объявление переменных*>**;** {<*Объявление переменных*>**;**}

12. <*Объявление переменных*> ::= <*Идентификатор*> {**,** <*Идентификатор*>} **:** <*Тип*>

13. <*Подпрограммы*> ::= { <*Функция*>**;** | <*Процедура*>**;** }

14. <*Тип*> ::= <*Идентификатор*> |
<*Перечисляемый тип*> |
<*Интервальный тип*> |
<*Тип-массив*> |
<*Тип-запись*> |
<*Тип-множество*> |
<*Файловый тип*> |
<*Ссылочный тип*>

15. <*Перечисляемый тип*> ::= **(**<*Идентификатор*> {**,** <*Идентификатор*>}**)**

16. <*Интервальный тип*> ::= <*Константа*>**..**<*Константа*>

17. <*Тип-массив*> ::= [**packed**] **array** **(** **.** <*Тип*> {**,** <*Тип*>} **.)** **of** <*Тип*>

18. <*Тип-множество*> ::= [**packed**] **set of** <*Тип*>

19. <*Файловый тип*> ::= [**packed**] **file of** <*Тип*>

20. <*Ссылочный тип*> ::= **^**<*Тип*>

21. <*Тип-запись*> ::= [**packed**] **record** <*Список полей*> [**;**] **end**

22. <*Список полей*> ::= <*Фиксированная часть*> [**;** <*Область вариантов*>] | <*Область вариантов*>

23. <*Фиксированная часть*> ::= <*Раздел записи*> { **;** <*Раздел записи*> }

24. <*Раздел записи*> ::= <*Имя поля*> {**,** <*Имя поля*> } **:** <*Тип*>

25. <*Область вариантов*> ::= **case** [<*Идентификатор*> **:**] <*Тип*> **of** <*Вариант*> {**;** <*Вариант*> }

26. <*Вариант*> ::= <*Константа*> {**,** <*Константа*> } **:** ([<*Список полей*>] [**;**])

27. <*Функция*> ::= <*Заголовок функции*>**;** <*Тело*> | <*Заголовок функции*>**;** <*Директива*> | **function** <*Идентификатор*> **;** <*Тело*>

28. <*Заголовок функции*> ::= **function** <*Идентификатор*> [<*Список формальных параметров*>] **:** <*Идентификатор*>

29. <*Процедура*>::=<*Заголовок процедуры*>**;**<*Тело*> | <*Заголовок процедуры*>**;** <*Директива*> | **procedure** <*Идентификатор*> **;** <*Тело*>

30. <*Заголовок процедуры*> := **procedure** <*Идентификатор*> [<*Список формальных параметров*>]

31. <*Список формальных параметров*> ::= **(**<*Формальный параметр*> {**;** <*Формальный параметр*>}**)**

32. <*Формальный параметр*> ::= [**var**] <*Идентификатор*> {**,** <*Идентификатор*>} **:** <*Идентификатор*> |<*Заголовок функции*> | <*Заголовок процедуры*>

33. <*Оператор*> ::= [<*Метка*> **:**] <*Оператор без метки*>

34. <*Оператор без метки*> ::= <*Присваивание*> | <*Вызов процедуры*> | <*Составной Оператор*> | <*Оператор* **if**> | <*Оператор* **case**> | <*Оператор* **while**> | <*Оператор* **repeat**> | <*Оператор* **for**> | <*Оператор* **with**> | <*Оператор* **goto**> | <*Пустой оператор*>

35. <*Присваивание*> ::= <*Переменная*> **:=** <*Выражение*> | <*Имя функции*> **:=** <*Выражение*>

36. <*Вызов процедуры*> ::= <*Имя процедуры*> [<*Список актуальных параметров*> | <*Список параметров вывода*>]

37. <*Список актуальных параметров*> ::= **(**<*Актуальный параметр* > {**,** <*Актуальный параметр*>}**)**

38. <*Актуальный параметр* > ::= <*Выражение*> | <*Переменная*> | <*Имя функции*> | <*Имя процедуры* >

39. <*Имя функции*> ::= <*Идентификатор*>

40. <*Имя процедуры*> ::= <*Идентификатор*>

41. <*Список параметров вывода*> ::= **(**<*Параметр вывода*> { **,** <*Параметр вывода*> }**)**

42. <*Параметр вывода*> ::= <*Выражение*> [**:**<*Выражение*> [**:** <*Выражение*>]]

43. <*Составной Оператор*> ::= **begin** <*Оператор*> { **;** <*Оператор*> } **end**

44. <*Оператор* **if** > ::= **if** <*Логическое выражение*> **then** <*Оператор*> [**else** <*Оператор*>]

45. <*Оператор* **case**> ::= **case** <*Выражение*> **of** [<*Вариант*>{**;** <*Вариант*>}] [**;**] **end**

46. <*Вариант*> ::= <*Константа*> {**,** <*Константа*>} **:** <*Оператор*>

47. <*Оператор* **while**> ::= **while** <*Логическое выражение*> **do** <*Оператор*>

48. <*Оператор* **repeat**> ::= **repeat** <*Оператор*> {**;**<*Оператор*>} **until** <*Логическое выражение*>

49. <*Логическое выражение*> ::= <*Выражение*>

50. <*Оператор* **for**> ::= **for** <*Переменная*> **:=** <*Выражение*> <*Шаг*> <*Выражение*> **do** <*Оператор*>

51. <*Шаг*> ::= **to** | **downto**

52. <*Оператор* **with**> ::= **with** <*Переменная*> {**,**<*Переменная*>} **do** <*Оператор*>

53. <*Оператор* **goto**> ::= **goto** <*Метка*>

54. <*Пустой оператор*> ::=

55. <*Переменная*> ::= <*Идентификатор*> | <*Переменная*> **(.** <*Выражение*> { **,** <*Выражение*>} **.)** | <*Переменная*> **.** <*Имя поля*> | <*Переменная*>

56. <*Имя поля*> ::= <*Идентификатор*>

57. <*Выражение*> ::= <*Простое выражение*>{<*Оператор отношения*> <*Простое выражение*>}

58. <*Оператор отношения*> ::= **<** | **<=** | **=** | **>=** | **>** | **<>** | **in**

59. <*Простое выражение*> ::= [**+** | **–**] <*Терм*>{<*Аддитивный оператор*> <*Терм*>}

60. <*Аддитивный оператор*> ::= **+** | **–** | **or**

61. <*Терм*> ::= <*Фактор*> {<*Мультипликативный оператор*> <*Фактор*>}

62. <*Мультипликативный оператор*> ::= ***** | **/** | **div** | **mod** | **and**

63. <*Фактор*> ::= <*Переменная*> | <*Константа без знака*> | <*Вызов функции*> | **not** <*Фактор*> | **(**<*Выражение*>**)** | <*Конструктор множества*>

64. <*Вызов функции*> ::= <*Имя функции*> [<*Список актуальных параметров*>]

65. <*Константа без знака*> ::= <*Число без знака*> | <*Строка символов*> | <*Идентификатор*> | **nil**

66. <*Конструктор множества*> ::= **(.** [<*Спецификация элемента*> { **,** <*Спецификация элемента*> }] **.)**

67. <*Спецификация элемента*> ::= <*Выражение*> [**..** <*Выражение*>]

Примечание. Терминальные символы [и] из формул (17), (55) и (66) переданы эквивалентными символами (. и .) соответственно.

Приложение 3. Компиляция и отладка программ на языке ПАСКАЛЬ

После написания программы на языке ПАСКАЛЬ она редактируется, компилируется и отлаживается.

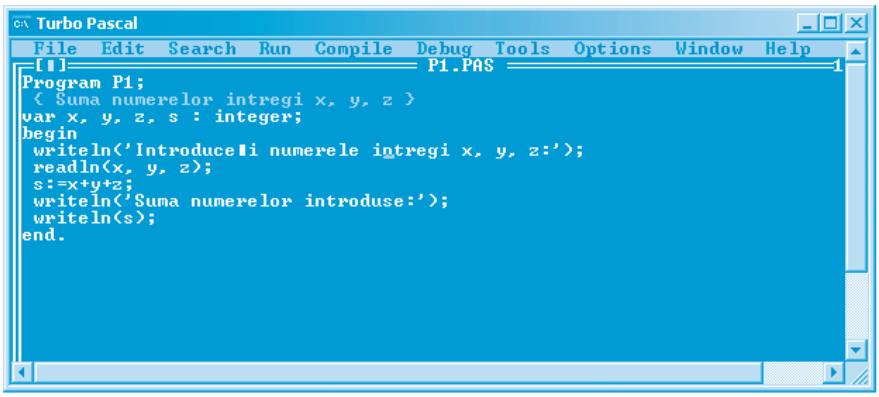
Редактирование заключается во вводе программы в компьютер. Введенные программы можно хранить в текстовых файлах с расширением *.pas*.

Компиляция – это процесс автоматического перевода программы, написанной на языке ПАСКАЛЬ, в программу, написанную на компьютерном языке. После компиляции программу на машинном языке можно запустить или сохранить в исполняемом файле с расширением *.exe*.

Отладка – это процесс обнаружения и исправления синтаксических и семантических ошибок в программе на языке ПАСКАЛЬ.

Обычно все эти операции выполняются с помощью специальных программ, называемых *интегрированными средами разработки* (IDE *Integrated Development Environment*).

В случае интегрированных сред разработки взаимодействие пользователя с компьютером осуществляется с использованием графических интерфейсов, которые отображают на экране окна приложений, диалоговые окна, окна навигации, окна проводника и окна документов.



Как правило, окна интегрированной среды разработки программ содержат стандартные графические элементы, изученные в предыдущих классах: строка меню, меню, команды, кнопки, курсоры, текстовые поля и т. д.

Представим далее часто используемые команды из меню программных средств Turbo PASCAL и Free PASCAL, установленных в большинстве компьютерных классов в школах Республики Молдова.

Меню `File` (Файл)

`New` (Новый) – создает новое окно, в котором пользователь может вводить и редактировать текст, обычно это программа на языке ПАСКАЛЬ.

Open... (**Открыть...**) – читает указанный пользователем файл и отображает его содержимое в новом окне. Затем это содержимое может быть обработано по желанию.

Save (**Сохранить**) – сохраняет содержимое текущего окна в ранее открытый с помощью команды **Open** файл. Однако, если текущее окно было создано с помощью команды **New**, будет создан новый файл, и пользователю будет предложено дать ему имя.

Save as... (**Сохранить как...**) – сохраняет содержимое текущего окна в новый файл. Пользователю предлагается дать имя для вновь созданного файла.

Print (**Печать**) – печать содержимого текущего окна на принтере.

Exit (**Выход**) – выход из интегрированной среды разработки программ. Перед выходом из программы приложение предложит пользователю сохранить содержимое открытых окон.

Меню Edit (Редактирование)

Cut (**Вырезать**) – вырезает выделенный фрагмент текста. Вырезанный фрагмент помещается в буфер памяти.

Copy (**Копировать**) – копирует выделенный фрагмент текста. Скопированный фрагмент помещается в буфер памяти.

Paste (**Вставить**) – фрагмент текста в буфере памяти вставляется в место нахождения курсора.

Clear (**Очистить**) – удаляет выделенный фрагмент текста, не помещая его в буфер памяти.

Меню Search (Поиск)

Find (**Найти**) – ищет, начиная с текущей позиции курсора, фрагмент текста, обозначенный в текстовом поле диалогового окна.

Replace (**Заменить**) – заменяет выбранный фрагмент текста на фрагмент, указанный пользователем.

Меню Run (Запуск)

Run (**Выполнить**) – компилирует и запускает программу на языке ПАСКАЛЬ из текущего окна. Если программа содержит синтаксические ошибки, отобразится сообщение с указанием типа и места возникновения ошибки.

Меню Compile (Компиляция)

Compile (**Компилировать**) – компилирует программу на языке ПАСКАЛЬ из текущего окна, но не запускает ее на выполнение.

Build (**Создать**) – компилирует программу на языке ПАСКАЛЬ из текущего окна и сохраняет результат компиляции в исполняемом файле.

Меню Windows содержит команды, обеспечивающие расположение окон и переключение между ними, а также команды меню **Help**, обеспечивающие доступ к справочному руководству.

Приложение 4. Словарь языка C++

1. <*Буква*> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

2. <*Цифра*> ::= 0|1|2|3|4|5|6|7|8|9

3. <*Специальный символ*> ::= + | – | * | / | = | < | > |] | [| , | (|) | : | ; | ” | . | \ | { | } | $ | # | <= | >= | == | != | % | &

4. <*Ключевое слово*>::= **auto** | **asm** | **bool** | **break** |**case** | **catch**| **char** | **const** | **continue** | **class** | **default** | **delete** | **do** | **double** | **else** | **enum** | **extern** | **float** | **for** | **friend** | **goto** | **if** | **inline** | **int** | **long** | **namespace** | **new** | **operator** | **private** | **public** | **protected** | **plate** | **register** | **return** | **short** | **signed** | **sizeof** | **static** | **struct** | **switch** | **typedef** | **union** | **unsigned** | **using** | **void** | **volatile** | **virtual** | **while**

5. <*Идентификатор*> ::= <*Буква*> { <*Буква*> | <*Цифра*> }

6. <*Директива*> ::= # <*Ключевое слово*> <*Исходный файл*>| # <*Ключевое слово*> <*Идентификатор*> [<*Число*>|<*Строка символов*>]

7. <*Целое без знака*> ::= <*Цифра*> {<*Цифра*>}

8. <*Знак*> ::= + | –

9. <*Целое число*> ::= [<*Знак*>] <*Целое без знака*>

10. <*Масштабный множитель*> ::= <*Целое число*>

11. <*Вещественное число*> ::= <*Целое число*> e <*Масштабный множитель*> | <*Целое число*>.<*Целое без знака*> [e <*Масштабный множитель*>]

12. <*Число*> ::= <*Целое число*> | <*Вещественное число*>

13. <*Строка символов*> ::= “<*Элемент строки*> { <*Элемент строки*> } ”

14. <*Элемент строки*> ::= “ ”| <*Любой печатный символ*>

15. <*Метка*> ::= <*Идентификатор*> <:>

16. <*Комментарий*> ::= // <*Любая последовательность символов и конец строки* > | /* < *Любая последовательность символов* >*/

Приложение 5. Компиляция и отладка программ C++

Программа – это исполняемый файл в двоичном коде. Текстовый файл, содержащий реализацию программы на языке программирования, представляет собой ***файл исходного кода*** или просто ***исходный код***, например, исходный код C++, исходный код Java и т. д. Процесс записи файла исходного кода называется ***написанием исходного кода*** или ***написанием кода.***

После написания программы на C++ она редактируется, компилируется, компонуется и выполняется.

Редактирование заключается во вводе программы в компьютер. Введенные программы можно хранить в текстовых файлах с расширением .cpp.

Компиляция – это процесс автоматического перевода программы, написанной на C++, в программу, написанную на компьютерном языке (машинном коде). Процесс компиляции выполняется с помощью компилятора. В варианте языка C++ на первом этапе компиляции вызывается *препроцессор*. Сначала он распознает и анализирует *директивы* процессора. Затем проверяется исходный код, чтобы убедиться, что он соответствует синтаксису и семантике языка. Если есть ошибки, о них пользователю выдаются сообщения. Пользователь должен исправить ошибки (изменив исходную программу). Только после этого исходный код переводится в код ассемблера и, наконец, в двоичный машинный код, соответствующий компьютеру. Этот двоичный код называется *объектным кодом* и обычно хранится в другом файле, называемом *объектным файлом*. Объектный файл обычно имеет то же имя, что и исходный файл, и расширение *.obj*.

Компоновка. После того как исходная программа была переведена в объектную программу, она будет подвергнута операции компоновки. Цель этой операции – получить окончательную форму программы для ее выполнения. Компоновщик «связывает» объектные модули, выполняет ссылки на внешние функции и процедуры в библиотеках и создает исполняемый код, хранящийся в другом файле, который называется *исполняемым файлом* с тем же именем и расширением *.exe*.

Выполнение. Запуск на выполнение заключается в загрузке исполняемой программы в память и ее выполнении.

Обычно все эти операции выполняются с помощью специальных программ, называемых и*нтегрированными средами разработки* (IDE – *Integrated Development Environment*).

При использовании интегрированных сред разработки взаимодействие пользователя с компьютером осуществляется с помощью графических интерфейсов, которые отображают на экране окна приложений, диалоговые окна, окна навигации, окна проводника и окна документов.

Если интегрированная среда не используется, программист вызовет (в командной строке) текстовый редактор, компилятор, компоновщик. Запуск на выполнение также будет производиться из командной строки.

Мы будем использовать *интегрированные среды разработки*. Как правило, окна интегрированных сред разработки программ содержат стандартную графику, изученную в предыдущих классах: строка меню, меню, команды, кнопки, курсоры, текстовые поля и т. д.

Представляем далее часто используемые команды из меню среды программирования **Code :: Blocks**, которую можно скачать из Интернета по адресу http://www.codeblocks.org/. Этот продукт представляет собой бесплатную интегрированную среду разработки (IDE), которая содержит все, что нужно программисту для создания приложений. Перечислим некоторые из необходимых инструментов:

1) *Текстовые редакторы*. Необходимы для написания программы или исходного кода – файлы с расширением .c (для языка C) или .cpp (для языка C++);

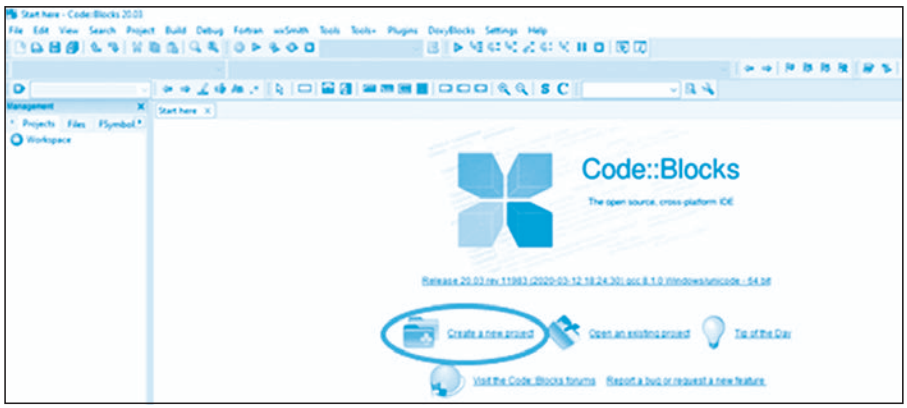
2) *Компиляторы*. Необходимы для преобразования исходного кода в команды, понятные процессору, и создания исполняемой программы или скомпилированных файлов. Существует несколько компиляторов для языков C / C++, наиболее часто используемый – gcc (*GNU C Compiler*).

3) *Библиотеки* – файлы заголовков, содержащие описание определенных функций, наиболее распространенными из которых являются чтение и отображение данных (файл *iostream*). Любые дополнительные функции могут быть добавлены путем установки плагинов (например, компиляторов).

Написание программы на языке C++ предполагает выполнение следующих шагов.

Шаг 1. Запуск среды программирования. Осуществляется выбором соответствующей команды из меню **Start** операционной системы либо двойным щелчком на пиктограмму **Code::Blocks** на рабочем столе.

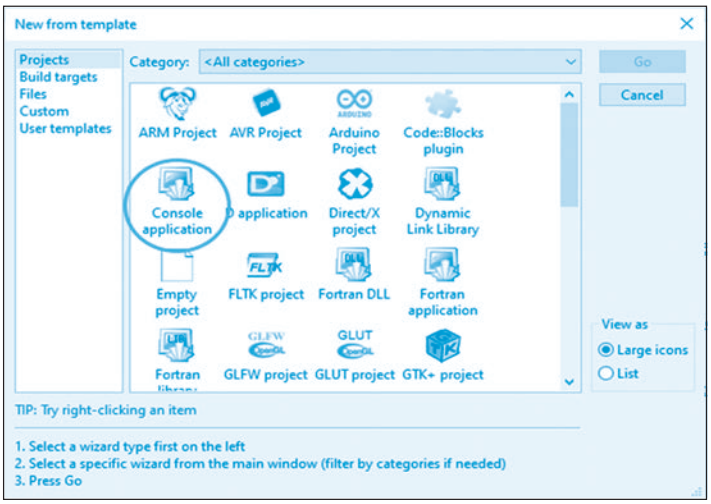
Шаг 2. Создание нового проекта. В окне приложения, которое появляется сразу после запуска среды программирования, выберите ***Create a new project.***

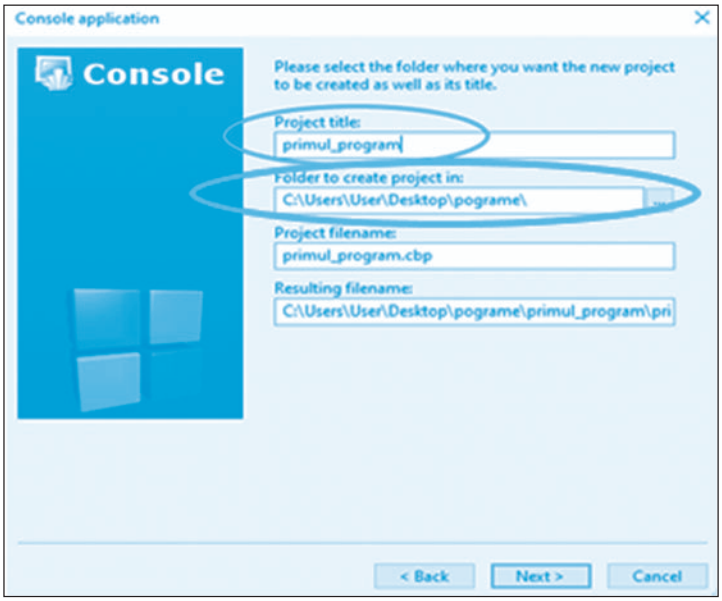


Шаг 3. Выбор типа приложения. В целом среда разработки программ **Code :: Blocks** дает пользователям возможность создавать разные типы приложений, такие как *Консольное приложение, Библиотека динамической компоновки* (Dynamic Link Library), *Arduino* (программы для роботов) и др. Очевидно, что в нашем случае будет выбран вариант *Консольного приложения* (**Console application):**

Шаг 4. Выбор языка, на котором будет написана программа. Конечно, в нашем случае необходимо выбрать язык C ++.

Шаг 5. Определение названия проекта и места, где он будет сохранен.





Например, на приведенном рисунке проект называется primul_program и сохраняется в папке `C:\Users\User\Desktop\programe\`

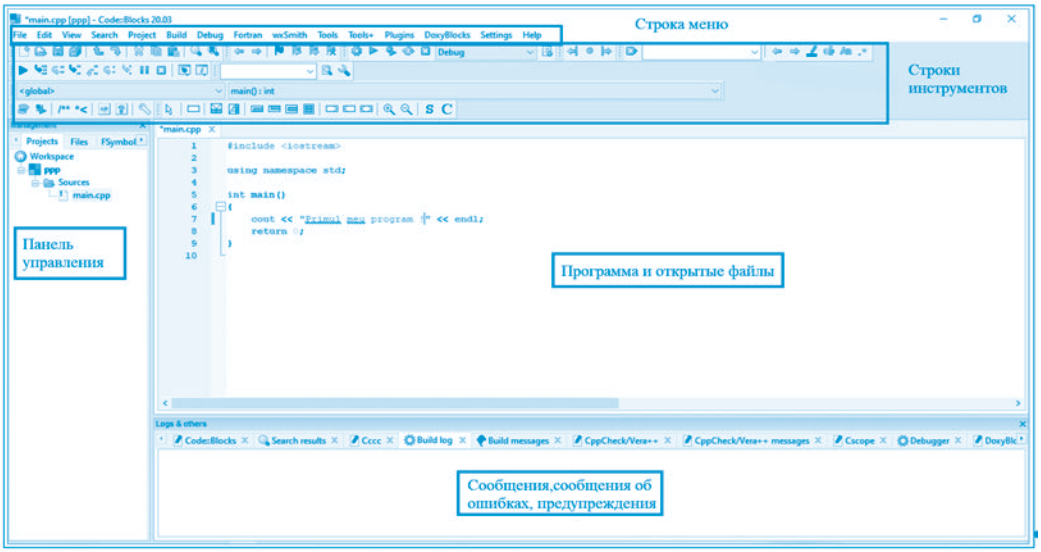
Шаг 6. Редактирование программы. Редактирование производится в главном окне среды разработки программ.

Это окно имеет стандартную структуру и включает:

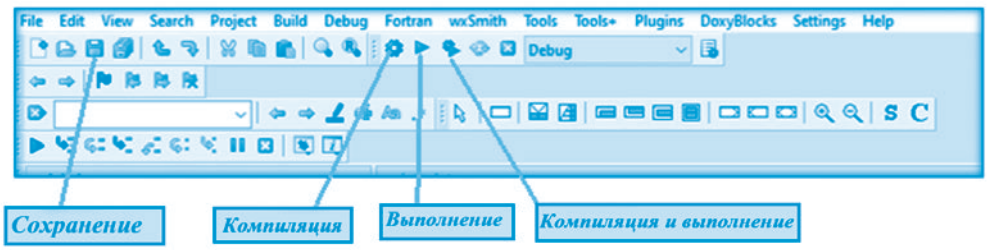
- Строку меню, содержащую в том числе наиболее часто используемые в процессе изучения языка C++ меню: **File, Edit, Build** (Построение), **Debug** (Отладка).
- Строки/панели, содержащие часто используемые команды, обозначенные пиктограммами;
- Панель управления, в которой отображается древовидная структура проекта;

– Окно редактирования, в котором пишутся разрабатываемые программы;
– Окно, содержащее вкладки, показывающие сообщения об ошибках, значения переменных, команды, выполняемые в пошаговом режиме. Вкладка для отображения выбирается с помощью закладок из верхней части этого окна.

Подчеркнем, что пользователь может настроить главное окно с помощью команд в меню **View** (Просмотр).



Шаг 7. Компиляция и выполнение программы. Основные инструменты, используемые при компиляции, выполнении и сохранении программы, находятся в меню **Build**, а наиболее часто используемые обозначаются значками из соответствующей панели:

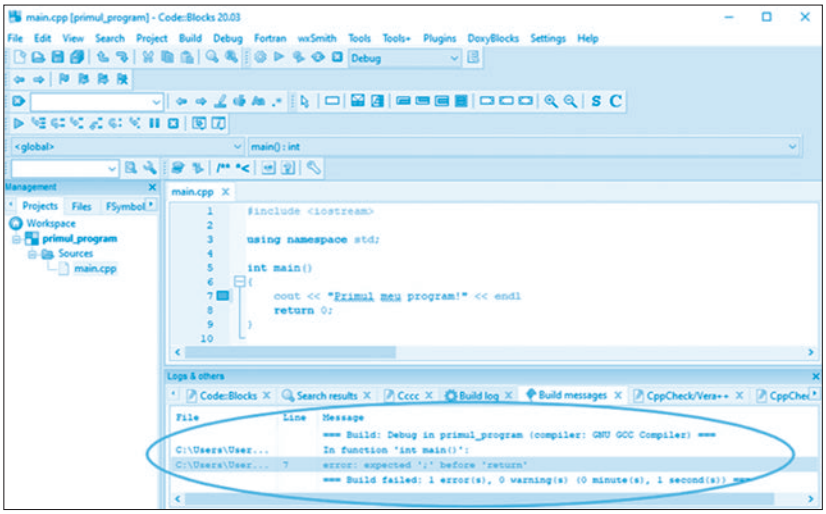


Результат выполнения программы будет выведен на экран:

```
C:\Users\User\Desktop\pograme\primul_program\bin\Debug\primul_program.exe
Primul meu program!

Process returned 0 (0x0)   execution time : 0.089 s
Press any key to continue.
```

Если же программа содержит синтаксические ошибки, соответствующие сообщения выводятся в специальной области, отмеченной на нижеследующем рисунке.



После исправления ошибок программу необходимо опять запустить на компиляцию и выполнение.

Шаг 8. Отладка программы (устранение неполадок в программе). За исключением некоторых очень простых программ, разрабатываемые программы содержат не только синтаксические, но и логические ошибки. Среды разработки программ не могут автоматически обнаруживать такие ошибки, но они предоставляют пользователю инструменты, облегчающие этот процесс.

Чтобы обнаружить возможные логические ошибки, программист должен подготовить набор тестов. Каждый из этих тестов содержит исходные данные и стандартные ответы. Запуская программу для каждого из разработанных тестов, пользователь сравнивает ответы, предоставленные программой, со стандартными ответами. В случае несоответствий программист может выполнить операторы программы в пошаговом режиме и отобразить текущие значения переменных в программе.

Операции по устранению неполадок выполняются с помощью следующих команд:

a) **Debug → Debugging windows → Watches** позволяет обзор значений переменных в реальном режиме времени.

b) **Debug → Step** into позволяет построчно/пошагово увидеть, как работает ваша программа.

Вообще говоря, обнаружение логических ошибок требует от программиста особого внимания. Даже в случае программ, разработанных для учебных целей, их написание занимает около 30% рабочего времени ученика, а отладка этих программ – остальные 70%. Поэтому в процессе изучения информатики ученикам рекомендуется распределять время на обучение таким образом, чтобы у них была возможность не только писать, но и устранять неполадки в разработанных ими программах.

Более подробную информацию о среде разработки программы можно найти в руководстве пользователя, доступном по адресу http://www.codeblocks.org/user-manual