Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Уральский энергетический институт

А. М. ЗЮЗЕВ
К. Е. НЕСТЕРОВ

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Пересчет расстояния: Мили -> Километры.");
            Console.Write("Задайте расстояние в милях: ");
            string m=Console.ReadLine();
            double k = Convert.ToDouble(m)*1.6;
            Console.WriteLine("{0} миль это {1} км.",m,k);
            Console.ReadLine();
        }
    }
}
```

Министерство науки и высшего образования
Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

А. М. Зюзев
К. Е. Нестеров

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие

Рекомендовано методическим советом
Уральского федерального университета
для студентов вуза,
обучающихся по направлению подготовки
13.03.02 — Электроэнергетика и электротехника

Екатеринбург
Издательство Уральского университета
2019

УДК 004.42(075.8)
ББК 32.973я73
З-98

Рецензенты:
кафедра информатики УГГУ (завкафедрой канд. техн. наук, доц. *Дружинин А. В.*);
директор ООО «Тяжпромэлектропривод» канд. техн. наук, доц. *Зеленцов В. И.*

Научный редактор — канд. техн. наук, доц. *Метельков В. П.*

Зюзев, А. М.

З-98 Объектно ориентированное программирование : учеб.-метод. пособие / А. М. Зюзев, К. Е. Нестеров. — Екатеринбург : Изд-во Урал. ун-та, 2019. — 116 с.

ISBN 978-5-7996-2634-1

В пособии рассмотрены основы программирования на примере языка Паскаль, описаны принципы разработки программ с графическим интерфейсом пользователя в среде объектно-визуального программирования Delphi, приведены базовые сведения по языку Microsoft Visual C#. Для запуска содержащихся в издании примеров могут быть использованы бесплатные версии сред разработки приложений, такие как PascalABC. NET и Microsoft Visual C# Express Edition.

Предназначено для бакалавров, изучающих электронику и электротехнику.

Библиогр.: 6 назв. Рис. 9.

УДК 004.42(075.8)
ББК 32.973я73

ISBN 978-5-7996-2634-1

Введение

Языки программирования условно можно разделить на три основных группы: языки низкого уровня (ассемблер), языки высокого уровня (Basic, C, Pascal) и языки объектно-визуального программирования (Delphi, LabVIEW, языки из пакета Microsoft Visual Studio). Языки низкого уровня используются в основном для программирования простейших контроллеров, т. к. позволяют получить эффективно исполняемый код минимальной длины. Языки высокого уровня применяются при разработке программ для более сложных контроллеров и персональных компьютеров (ПК). Языки объектно-визуального программирования используются при написании программ для ПК и позволяют значительно сократить время разработки пользовательского интерфейса программы.

1. Язык Object Pascal

1.1. Общие сведения

Алфавит языка Паскаль включает буквы, цифры, шестнадцатеричные цифры, специальные символы и зарезервированные слова.

Буквы — это буквы латинского алфавита от *a* до *z* и от *A* до *Z*. В Паскале нет различия между прописными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры — арабские цифры от 0 до 9. Каждая шестнадцатеричная цифра имеет значение от 0 до 15. Первые 10 значений обозначаются арабскими цифрами от 0 до 9, остальные шесть — латинскими буквами от *A* до *F* или от *a* до *f*.

Специальные знаки Паскаля — это символы + – * / = , ' . : ; < > [] () {} ^ @ $ #. К специальным знакам относятся также следующие пары символов: <> <= >= := (* *). В программе такие пары символов нельзя разделять пробелами, если они используются как знаки операций отношения или ограничители комментария. Символы (* и *) могут употребляться соответственно вместо {и}.

Особое место в алфавите языка занимают пробелы, к которым относятся любые символы ASCII в диапазоне кодов от 0 до 32. Эти символы рассматриваются как ограничители идентификаторов, констант, чисел, зарезервированных

слов. Несколько следующих друг за другом пробелов считаются одним пробелом (последнее не относится к строковым константам).

В Паскале имеются следующие зарезервированные слова:

and	*end*	*nil*	*shr*
asm	*file*	*not*	*string*
array	*for*	*object*	*then*
begin	*function*	*of*	*to*
case	*goto*	*or*	*type*
const	*if*	*packed*	*unit*
constructor	*implementation*	*procedure*	*until*
destructor	*in*	*program*	*uses*
div	*inline*	*record*	*var*
do	*interface*	*repeat*	*while*
downto	*label*	*set*	*with*
else	*mod*	*shl*	*xor*

Зарезервированные слова не могут использоваться в качестве идентификаторов (что такое идентификатор, см. ниже).

Стандартные директивы первоначально связаны с некоторыми стандартными объявлениями в программе. К ним относятся:

absolute	*far*	*near*
assembler	*forward*	*private*
external	*interrupt*	*virtual*

Можно переопределить любую стандартную директиву, т. е. объявить одноименный идентификатор. Стандартные директивы *PRIVATE* и *VIRTUAL* действуют только в пределах объявления объектов.

Идентификаторы в Паскале — это имена констант, переменных, меток, типов, объектов, процедур, функций, модулей, программ и полей в записях. Идентификаторы могут иметь произвольную длину, но значащими (уникальными в области определения) являются только первые 63 символа. Идентификатор всегда начинается с буквы или знака подчеркивания, за которым могут следовать буквы, цифры и знак подчеркива-

ния (код 95 в ASCII). Пробелы и специальные символы алфавита не могут входить в идентификатор. Примеры правильных идентификаторов:

```
a
ALPHA
MyProgramIsTheBestProgram
date_27_ sep_ 39
external
_beta
```

Примеры неправильных идентификаторов:

```
1Program (начинается с цифры)
block#1 (содержит специальный символ)
My Program (содержит пробел)
mod (зарезервированное слово)
```

В качестве констант в Паскале могут использоваться целые, вещественные и шестнадцатеричные числа, логические константы, символы, строки символов, конструкторы множеств и признак неопределенного указателя NIL.

Целые числа записываются со знаком или без него. Диапазон допустимых значений зависит от разрядности переменной или константы, в которой хранится число. Например, для 32-битной переменной диапазон значений составляет от –2 147 483 648 до +2 147 483 647. Следует учесть, что, если целочисленная константа выходит за допустимые границы, компилятор дает сообщение об ошибке. В этом случае необходимо использовать другой тип переменной.

Вещественные числа записываются со знаком или без него с использованием десятичной точки и (или) экспоненциальной части. Экспоненциальная часть начинается символом «e» или «E», за которым могут следовать знаки «+» или «–» и число, равное десятичному порядку. Например,

3.14E5 — 3.14 умножить на 10 в степени 5;

—17e–2 — минус 17 умножить на 10 в степени минус 2.

Если в записи вещественного числа присутствует десятичная точка, перед точкой и за ней должна быть хотя бы одна цифра. Если используется символ экспоненциальной части «e» («E»), за ним должна следовать хотя бы одна цифра десятичного порядка.

Шестнадцатеричное число состоит из шестнадцатеричных цифр, которым предшествует знак доллара «$» (код 36 в ASCII). Диапазон шестнадцатеричных чисел от $00000000 до $FFFFFFFF.

Логическая константа — это либо слово «*FALSE*» (ложь или 0), либо слово «*TRUE*» (истина или 1).

Символьная константа — это любой символ ПК, заключенный в апострофы: 'z' — символ z, 'Ф' — символ Ф. Если необходимо записать собственно символ апострофа, он удваивается: '''' — символ ' (апостроф). Допускается использовать запись символа путем указания его внутреннего кода, которому предшествует символ «#» (код 35), например: #97 — символ a, #90 — символ z, #39 — символ ', #13 — символ CR.

Строковая константа — любая последовательность символов (кроме символа «CR» — возврат каретки), заключенная в апострофы; если в строке нужно указать сам символ апострофа, он удваивается, например: 'Это — строка символов', 'That''s string'.

Строка символов может быть пустой, т. е. не иметь никаких символов в обрамляющих ее апострофах. Строку можно составлять из кодов нужных символов с предшествующими каждому коду символами, например, строка #83#121#109#98#11#108 эквивалентна строке 'Symbol'.

В строке можно чередовать части, записанные в обрамляющих апострофах, с частями, записанными кодами. Таким способом можно вставлять в строки любые управляющие символы, в т. ч. и символ «CR», например: #7'Ошибка! '#13'Нажмите любую клавишу ... '#7.

Конструктор множества — список элементов множества, обрамленный квадратными скобками, например:

```
[1,2,4..7,12];
[blue, red];
[];
[true].
```

В Паскале разрешается в объявлении констант использовать произвольные выражения, операндами которых могут быть ранее объявленные нетипизированные константы, имена типов и объектов, а также следующие функции от них:

abs	*lo*	*ptr*	*swap*
chr	*odd*	*round*	*trunc*
hi	*ord*	*sizeof*	
length	*pred*	*succ*	

Например:

```
const
MaxRea1 = Maxint div SizeOf (real);
NumChars = ord ('Z') — ord ('a') + 1;
Ln10 = 2,302585092994;
Ln10R = 1/Ln10.
```

Основными элементами, из которых конструируется исполняемая часть программы, являются константы, переменные и обращения к функциям. Каждый из данных элементов характеризуется своим значением и принадлежит к какому-либо типу данных. С помощью знаков операций и скобок из них можно составлять выражения, которые фактически представляют собой правила получения новых значений. Частным случаем выражения может быть просто одиночный элемент, т. е. константа, переменная или обращение к функции. Значение такого выражения имеет тот же тип, что и сам элемент. В бо-

лее общем случае выражение состоит из нескольких элементов (операндов) и знаков операций, а тип его значения определяется типом операндов и видом примененных к ним операций.

Примеры выражений:

```
Y
21
(a + b) * c
sin (t)*a > 2
a > 2
not Flag and (a = b)
NIL
[1, 3..7] * set1
```

В Паскале определены следующие операции:
унарные — *not*, *@;*
мультипликативные — *, /, *div*, *mod, and, shl, shr;*
аддитивные — +, —, *or, xor*;
отношения — =, <>, <, >, <=, >=, *in*.

Приоритет операций убывает в указанном порядке, т. е. наивысшим приоритетом обладают унарные операции, низшим — операции отношения. Порядок выполнения нескольких операций равного приоритета устанавливается компилятором из условия оптимизации кода программы и не обязательно слева направо. При исчислении логических выражений, операции равного приоритета всегда вычисляются слева направо, причем будут вычисляться все или только достаточные операции в зависимости от значения опции компилятора *COMPLETE BOOLEAN EVAL*: при установленном значении этой опции вычисляются все операции отношения; при неустановленном — только те, которые достаточны для получения результата.

При действиях с вещественным типом одним из операндов может быть значение любого целого типа. Результат операций имеет тип *DOUBLE*.

Унарная операция @ применяется к операнду любого типа и возвращает результат типа *POINTER*, в котором содержится адрес операнда. Пусть, например, задано описание

```
type
TwoChar = array [1..2] of char;
var Int : integer;
TwoCharPtr: ^TwoChar;
```

В таком случае оператор TwoCharPtr := @int; приведет к тому, что в *TWOCHARPTR* будет храниться адрес целочисленной переменной *int,* которая теперь будет интерпретироваться как совокупность двух символов. Поэтому возможен, например, такой оператор:

if TwoCharPtr [1]^ = 'C' then...

Если операция @ применяется к процедуре, функции или методу в объекте, ее результатом будет адрес точки входа в эту процедуру (функцию, метод). Этот адрес можно использовать только в подпрограмме, написанной на ассемблере, или в фрагментах *INLINE.*

В Паскале определены следующие логические операции:

not — логическое НЕ;

and — логическое И;

or — логическое ИЛИ;

xor — исключающее ИЛИ.

Логические операции применимы к операндам целого и логического типов. Если операнды — целые числа, то результат логической операции есть тоже целое число. К логическим операциям в Паскале обычно относятся и две сдвиговые операции над целыми числами:

i shl j — сдвиг содержимого *i* на *j* разрядов влево; освободившиеся младшие разряды заполняются нулями;

i shr j — сдвиг содержимого *i* на *j* разрядов вправо; освободившиеся старшие разряды заполняются нулями.

Логические операции над логическими данными дают результат логического типа.

Операция отношения IN применяется к двум операндам: первым (левым) операндом должно быть выражение любого порядкового типа; вторым — множество, состоящее из элементов того же типа, или идентификатор множественного типа. Операция даст TRUE, если левый операнд принадлежит множеству, например:

```
Var c : Char;
Type digit = set of '0'..'9';
...
if c in digit then...
```

1.2. Операторы ввода-вывода и управления вычислениями

Программа на языке Паскаль создается в текстовом виде и имеет следующую структуру: в начале программы располагается раздел описаний пользовательских констант, переменных, меток, функций и процедур, следом идет текст основной программы. В любом месте программы могут быть размещены комментарии, заключенные в фигурные скобки. Ниже приведен текст простейшей программы.

```
Program Hello; {имя программы}
Begin {начало программы}
  Writeln('Hello!'); {вывод текста на экран}
End. {конец программы}
```

Каждая программа начинается с ключевого слова «*Program*», расположенного в начале программы. После слова *«Program»* следует имя программы, которое может состоять из символов латинского алфавита, цифр и знака подчеркивания (использо-

вание остальных символов не допускается). Синтаксис языка требует добавления знака точки с запятой (;) после большинства команд, в т. ч. после указания имени программы. Слово «*Begin*» в приведенном примере обозначает начало программы, слово «*End*» — конец. В конце программы ставится точка (после слова «*End*»). Команда *Writeln* выводит на экран информацию, содержащуюся в аргументах этой команды, и переводит курсор на следующую строку. Аналогичная команда *Write* делает то же самое, но оставляет курсор сразу после выведенной информации. Аргументы командам передаются в круглых скобках через запятую (если требуется передать несколько аргументов). В приведенном примере команде *Writeln* передан аргумент типа «строковая константа» (строковая константа — строка текста, заключенная в апострофы).

Если в программе используются переменные, то они должны быть перечислены в разделе описания переменных с указанием их типа. Раздел описания переменных начинается с ключевого слова «*Var*», после которого приводится список переменных и их типов, оформленный следующим образом: *«имя_переменной»:«тип_переменной»*; где *«имя_переменной»* — идентификатор переменной (может состоять из символов латинского алфавита, цифр и знака подчеркивания), *«тип_переменной»* — слово, обозначающее один из стандартных или пользовательских типов данных. Пример программы, использующей переменные, приведен ниже.

```
Program Mil_km;  { Преобразование расстояния из миль в километры }
Var Mil:real;          { расстояние в милях }
   Kilom:real;         { расстояние в километрах }
Begin
 Writeln('Пересчет расстояния: Мили -> Километры.');
 Write('Задайте расстояние в милях: ');
 Readln(Mil);
 Kilom:=Mil*1.6;
 Writeln(Mil:5:2,' миль это', Kilom:5:2,' км.');
End.
```

В программе описаны две переменные типа *real* — вещественного типа данных. После вывода текстовых сообщений, при помощи команд *Writeln* и *Write* выполняется команда *Readln* (*Read*), которая используется здесь для ввода данных с клавиатуры: при исполнении этой команды программа ожидает ввода данных и нажатия кнопки ENTER, после чего помещает введенные данные в переменную, указанную в качестве параметра (*Mil*). Если тип введенных данных не совпадает с типом параметра переменной, то программа завершает работу с сообщением об ошибке. Оператор присваивания := (двоеточие, равно) копирует значение выражения, расположенного справа от него, в переменную, расположенную слева. В последней команде программы применено форматирование вывода: *Mil:5:2*, где число 5 — количество символов, отводимое для вывода всего числа, хранящегося в переменной *Mil*; 2 — количество знаков после запятой. Если значение переменной *Mil* окажется короче 5 символов (включая запятую), то выводимое значение будет дополнено слева пробелами. Если значение переменной *Mil* окажется длиннее 5 символов, то будет выведено все число.

Задания к примеру

1. Измените представление результата, модифицировав формат вывода.
2. На основе приведенного примера составьте программу, выставляющую запрос на ввод имени пользователя и приветствие с обращением по имени, применив переменную типа *String* (строковый тип данных).

Контрольные вопросы

1. Какова структура программы Паскаль?
2. Каков порядок организации ввода-вывода данных?
3. Как форматируется ввод-вывод?

Рассмотренные примеры программ имели линейную структуру, т. е. все команды программы выполнялись друг за другом

от ее начала до конца. Если в программе требуется выполнить какое-либо действие только при выполнении некоторого условия, то необходимо применить условный оператор. Порядок записи условного оператора в Паскале следующий:

IF «условие» THEN «оператор1» [ELSE «оператор2»];

где *«условие»* — выражение логического типа, например, $a > b$; *«оператор1»* — оператор, исполняемый при выполнении условия; *«оператор2»* — оператор, исполняемый при невыполнении условия. Часть *ELSE «оператор2»* является необязательной и может отсутствовать. Выражения логического типа могут принимать одно из двух значений: TRUE или FALSE. Если в качестве *«оператор1»* или *«оператор2»* необходимо выполнить несколько операторов, то они заключаются в операторные скобки *BEGIN ...END*. Пример использования условного оператора и операторных скобок показан ниже.

```
Program Sqroot; { Решение квадратного уравнения }
Var a,b,c:real; { коэффициенты уравнения }
   x1,x2:real; { корни уравнения }
   d:real; { дискриминант }
begin
  writeln('*Решение квадратного уравнения*');
  write('Введите значения коэффициентов');
  write(' в одной строке и нажмите <Enter>');
  readln(a,b,c); { ввод коэффициентов }
  d:=b*b-4*a*c; { вычисление дискриминанта }
  if d >= 0 then begin
    x1:=(-b+sqrt(d))/(2*a);
    x2:=(-b-sqrt(d))/(2*a);
    writeln('Корни уравнения');
    writeln('x1=',x1,' x2=',x2);
  end
  else writeln('Корней нет');
end.
```

В программе использована функция извлечения квадратного корня *SQRT (x)*, возвращающая значение типа *real*.

Задание к примеру

Составьте программу расчета корней биквадратного уравнения.

Контрольные вопросы

1. Каковы основные правила записи выражений?
2. В каких случаях необходимо применение операторных скобок?
3. Модифицируйте программу, применив оператор *IF ... THEN*.

Условные операторы могут быть вложены друг в друга, это продемонстрировано на следующем примере.

```
Program Optves; { Контроль веса }
var w:real;   { вес }
   h:real;   { рост }
   opt:real; { оптимальный вес }
   d:real;   { отклонение от оптимального веса }
begin
 writeln('Введите в одной строке через пробел');
 writeln ('вес (в кг) и рост (в см), затем нажмите <Enter>');
 write('->');
 readln(w,h);
 opt:=h-100;
 if w=opt then writeln('Ваш вес оптимален!')
 else if w<opt then begin
   d:=opt-w;
   writeln('Вам надо поправиться на ',d:5:2,' кг.');
 end else begin
   d:=w-opt;
   writeln('Вам надо похудеть на',d:5:2,' кг');
 end;
end.
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

Вопрос

Каков порядок выполнения действий при использовании составного оператора IF?

Выражения логического типа могут содержать операции И, ИЛИ, НЕ, при этом наивысший приоритет имеет инвертирование (НЕ), операции И и ИЛИ имеют равные приоритеты. Для явного указания порядка выполнения логических и арифметических операций можно использовать скобки. Применение логических операторов рассмотрено ниже.

```
Program Razvor;{ Вычисление стоимости телефонного разговора с учетом
предоставляемой по субботам и воскресеньям скидки}
var Time:integer; { длительность разговора }
   Day:integer;  { день недели }
   Summa:real;   { стоимость разговора }
begin
  write('Длительность разговора');
  write('(целое кол-во минут) ->');
  readln(Time);
  write('День недели');
  write(' (1-понедельник,..,7-воскресенье)-> ');
  readln(Day);
  Summa:= 150*Time;
  if (Day = 6) OR (Day = 7) then Summa:=Summa*0.8;
  writeln('Стоимость разговора:',Summa:8:3);
end.
```

В примере использован тип данных *integer* — целые числа со знаком, диапазон доступных значений зависит от версии Паскаля.

Задание к примеру

Модифицируйте программу, введя в расчет скидку на ночной тариф.

Контрольные вопросы

1. Какого типа переменные могут употребляться в действиях с логическими операторами?
2. Какие значения может принимать выражение (Day = 6)?
3. Какие значения может принимать выражение (Day = 6) OR (Day = 7)?

В следующем примере рассмотрен оператор множественного выбора *CASE*, формат записи которого следующий:

case «ключ_выбора» of
«константа_выбора_1»:«оператор_1»;
«константа_выбора_2»:«оператор_2»;
...
«константа_выбора_n»:«оператор_n»;
[else «оператор_else»;]
end;

Здесь *«ключ_выбора»* — выражение перечислимого типа (например, *integer*); *«константа_выбора»* — константа того же типа, что и *«ключ_выбора»*; *«оператор_x»* выполнится при условии *«ключ_выбора» = «константа_выбора_x»*. Если ключ выбора не равен ни одной из констант выбора, то выполнится *«оператор_else»*. Часть *else «оператор_else»;* является необязательной.

```
Program Propis; {Дописывание после числа слова "рубль" в правильной
форме}
var n: integer; { число }
   r: integer; { остаток от деления n на 10 }
begin
    { здесь инструкции программы, определяющие значение n }
 writeln('Введите целое число (от 1 до 999) и нажмите <Enter>');
 write('-> ');
 readln(n);
 { здесь вывод поясняющего текста }
 if n>100 then n:=n mod 100;
 if (n>=11) and (n<=14) then writeln(' рублей')
 else begin
```

```
   r:=n mod 10;
   case r of
    0, 5 .. 9: writeln(' рублей');
    1:   writeln (' рубль');
    2..4: writeln (' рубля');
   end;
 end;
end.
```

В программе использован оператор получения остатка от деления нацело *MOD*. Формат его записи следующий: *a MOD b*, где *a* — делимое; *b* — делитель, а возвращаемый результат — остаток от деления.

Задание к примеру

Изобразите алгоритм определения окончания слова «рубль».

Контрольные вопросы

1. Можно ли решить эту же задачу на основе оператора *IF*...?
2. В чем преимущества оператора *CASE*...?
3. Какого типа переменная может быть в качестве ключа выбора?

Для организации переходов в программе может использоваться оператор *GOTO*. Однако настоятельно не рекомендуется применять данный оператор, т. к. это нарушает принципы объектно ориентированного программирования. Формат записи оператора следующий:

GOTO «метка»;

где *«метка»* — идентификатор, описанный в разделе описания меток и расположенный в тексте программы. Раздел описания меток начинается с ключевого слова «*LABEL*», после которого через запятую перечисляются все метки программы. Метки могут состоять из символов латинского алфавита, цифр и знака подчеркивания. Пример использования оператора перехода приведен ниже.

```
Program Go_to; {вход по паролю}
label finish; { раздел описания меток }
var password:integer; { запрашиваемый пароль }
begin
  write('Пароль-> ');
  readln(password);
  if password<>3737 then begin { 3737 - правильное значение пароля }
     writeln('Неверный пароль!');
    writeln('Доступ запрещен!');
    goto finish;
  end;
{ здесь инструкции программы }
finish:
end.
```

Задание к примеру

Организуйте запуск любого из предыдущих примеров в составе данной программы при правильно набранном пароле.

Контрольные вопросы

1. В каких случаях целесообразно использовать оператор *GOTO*?
2. Каковы общие правила применения оператора *GOTO*?

Использование логические констант TRUE и FALSE рассмотрено на следующем примере.

```
Program Nextd;{ Вычисление даты следующего дня }
var day,month,year:integer;
    last:boolean;{TRUE, если текущий день – последний день месяца }
    r:integer;{ если год високосный, то остаток от деления year на 4 равен нулю }
begin
   writeln;
   writeln('Задайте цифрами сегодняшнюю дату');
   write('(  число месяц год  )-> ');
   readln(day,month,year);
   last:=FALSE;
   case month of
```

```
    1,3,5,7,8,10,12:if day=31 then last:=TRUE;
    4,6,9,11: if day=30 then last:=TRUE;
    2: if (day=28) then begin
                        r:=year mod 4;
                        if r<>0 then
                        last:=TRUE;
                       end;
  end;
  if last
      then begin
      writeln('Последний день месяца!');
      day:=1;
      if month=12 then begin
          month:=1;
          year:=year + 1;
          writeln('С наступающим Новым годом!');
      end
      else month:=month + 1;
  end
    else day:=day + 1;
  writeln('Завтра ',day,'.',month,'.',year );
end.
```

Задание к примеру

Модифицируйте программу таким образом, чтобы она накануне дня рождения выставляла поздравление.

Контрольные вопросы

1. Укажите границы действия оператора *CASE*... в примере.
2. Какой смысл имеет действие "last:=FALSE;" в примере?

1.3. Операторы циклов

Если в программе необходимо выполнить одни и те же действия несколько раз, то удобно воспользоваться операторами повторений (циклами). В Паскале доступны три вида цикло-

вых операторов: счетный, с предпроверкой условия и с постпроверкой условия.

Синтаксис счетного оператора цикла следующий:

FOR «параметр_цикла»:=«нач_знач» TO «кон_знач» DO «оператор»;

где *«параметр_цикла»* — переменная перечислимого типа; *«нач_знач»* — начальное значение (присваивается параметру цикла при первом проходе цикла); *«кон_знач»* — конечное значение (при достижении которого цикл закончится); *«оператор»* — оператор, выполняемый в цикле. Параметр цикла автоматически увеличивает свое значение на один шаг после каждого прохода цикла. Если ключевое слово «*TO*» заменить на «*DOWNTO*», то параметр цикла будет уменьшаться от начального значения до конечного. Если начальное значение равно конечному, то цикл выполнится один раз.

```
Program Summr;{ Вычисление суммы ряда: 1+ 1/2 + 1/3 + ... }
var i:integer;    { номер элемента ряда }
   elem:real;     { значение элемента ряда }
   summ:real;   { сумма элементов ряда }
begin
 summ:=0;
 for i:=1 to 5 do begin
   elem:=1/i;
   summ:=summ+elem;
   writeln(i:2,'  ',elem:4:2);
 end;
 writeln('Сумма элементов ряда: ',summ:4:2);
end.
```

Задания к примеру

1. Изобразите структурную схему алгоритма работы программы.
2. Дополните программу вводом произвольного числа элементов ряда.

Контрольные вопросы

1. Какого типа может быть переменная цикла?
2. Как оформить выполнение нескольких действий в операторе *FOR*?
3. Каким образом на основе оператора *FOR* организовать табулирование функции с заданным шагом?

Счетный оператор цикла удобен при работе с данными типа массив — объекта, состоящего из элементов одного типа, у каждого из которых есть свой индекс — порядковый номер. Описание переменной типа «массив» выглядит следующим образом:

«имя_массива»: array [«нач_индекс»..«кон_индекс»] of «тип_элементов»;

Ниже приведен пример использования переменной типа «строковый массив». Число 11, указанное после слова «*string*», означает, что длина строки не может быть больше 11 символов.

```
Program Days; { Вывод на экран массива строк }
var day:array[1..7] of string[11];i:integer;
begin
  day[1]:='Понедельник';day[2]:='Вторник';
  day[3]:='Среда';day[4]:='Четверг';
  day[5]:='Пятница';day[6]:='Суббота';
  day[7]:='Воскресенье';
  for i:=1 to 7 do writeln(i,'  ',day[i]);
end.
```

Задание к примеру

Измените порядок вывода дней недели на обратный, модифицировав оператор цикла *FOR*.

Контрольные вопросы

1. Как описывается переменная типа «массив»?
2. Какой смысл имеет запись числа в описании строковой переменной?

Оператор цикла с предпроверкой условия записывается следующим образом:

WHILE «условие» DO «оператор»;

где *«условие»* — выражение логического типа, в случае истинности которого оператор цикла будет выполнен, а цикл продолжен. «Оператор» может не выполниться ни разу, т. к. «условие» проверяется до его выполнения.

```
Program Chislo_Pi; { Вычисление числа ПИ }
var pi:real;      { вычисляемое значение ПИ}
   t:real;        { точность вычисления}
   n:integer;     { номер члена ряда}
   elem:real;     { значение члена ряда}
begin
 pi:=0;n:=1;
 elem:=1; { начальное значение}
 write('Задайте точность вычисления ПИ-> ');
 readln(t);
 writeln('Вычисление ПИ с точностью',t:9:6);
 while elem >= t do begin
   elem:=1/(2*n-1);
   if (n mod 2) = 0 then pi:=pi-elem
   else pi:=pi+elem;
   n:=n+1;
 end;
 pi:=pi*4;
 writeln('Значение ПИ с точностью',t:9:6,' равно',pi:9:5);
 writeln('Просуммировано ',n,' членов ряда.');
end.
```

Задания к примеру

1. Изобразите структурную схему алгоритма работы программы.
2. На основе оператора *WHILE* напишите программу табулирования функции *SIN* с заданным шагом.

Контрольные вопросы

1. Укажите действия, составляющие тело цикла программы.
2. В каких случаях применяются операторные скобки в теле цикла программы?
3. Могут ли игнорироваться действия, составляющие тело цикла программы?

Оператор цикла с постпроверкой условия имеет следующий синтаксис:

REPEAT
Оператор_1;
...
Оператор_n;
UNTIL «условие»;

Составляющие тело цикла операторы 1 ... *n* исполняются процессором до тех пор, пока не выполнится *«условие»*, причем проверка «*условия*» осуществляется после выполнения тела цикла, т. е. данный цикл всегда выполняется хотя бы один раз.

```
Program Sumrep;{ Вычисление суммы положительных чисел }
var numb:integer;    { число, введенное с клавиатуры }
   summ:integer;    { сумма введенных чисел }
begin
 writeln('Вычисление суммы положительных целых чисел.');
 summ:=0;
 repeat
   write('->');readln(numb);
   if numb>0 then summ:=summ+numb;
 until numb <= 0;
 writeln('Сумма введенных чисел: ',summ);
end.
```

Задание к примеру

Модифицируйте программу таким образом, чтобы она выполняла подсчет двух сумм — положительных и отрицательных чисел, с выходом при введении 0.

Контрольные вопросы

1. Укажите действия, составляющие тело цикла программы.
2. Требуются ли операторные скобки для оформления тела цикла программы?
3. Могут ли игнорироваться действия, составляющие тело цикла программы?

Еще один пример использования оператора цикла с постпроверкой условия показан ниже.

```
Program Prost;{ Проверка, является ли целое число простым }
var n,d,r: integer;
begin
   write('Введите целое число-> ');
   readln(n);
   d:=2;       { сначала будем делить на два }
   repeat
      r:=n mod d;
      if r<>0 { n не разделилось нацело на d }
         then d:=d + 1;
   until r=0; { пока не нашли число, на которое делится n }
   if d=n
      then writeln( n,' - простое число')
      else writeln(n,' - непростое число');
end.
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

Контрольные вопросы

1. Поясните действие, определяемое выражением "r:=n mod d;".
2. Есть ли разница в толковании условий продолжения цикла в операторах *WHILE* и *REPEAT*?

1.4. Работа с символьными и строковыми переменными

Кроме численных (*integer*, *real*), логических (*boolean*) и строковых (*string*) типов данных Паскаль поддерживает символьный тип *char*. Переменная типа *char* может хранить только один символ, каждому символу соответствует определенный числовой код. Функция *ORD* возвращает этот код в ответ на переданный ей в качестве параметра символ. Обратная функция *CHR* возвращает символ, соответствующий переданному в качестве параметра числу.

Следующая программа выводит таблицу символов, используя функцию *CHR*.

```
Program Ascii;{ Вывод таблицы ASCII кодировки символов }
var
   ch:char;    { символ }
   dec:integer; { десятичный код символа }
   i,j:integer;

begin
   dec:=0;
   for i:=0 to 15 do { шестнадцать строк }
      begin
         dec:=i;    { чтобы получить таблицу кодировки для символов }
                    { с кодами 128-255, эту инструкцию надо заменить }
                    { на dec:=i+128;}
         for j:=1 to 8 do { восемь колонок }
            begin
               if(dec<7) or (dec>=14)
                  then
                     write(dec:4,'-',' ',
                        chr(dec):1,chr(179))
                  else { символы CR,LF,TAB не отображаются }
                     write(dec:4,'-  ',chr(179));
               dec:=dec+16;
            end;
```

```
        writeln;    { переход к новой строке экрана }
    end;
end.
```

Задание к примеру

Выведите последовательно первую и вторую половины таблицы кодировки, сравните коды символов строчных и прописных букв.

Контрольные вопросы

1. Поясните принцип организации табличного вывода данных.
2. Поясните действие операции *CHR*.
3. Каковы отличия кодов символов строчных и прописных букв?
4. Для чего используются спецсимволы и графические символы?
5. Применимы ли к символьным переменным операции сравнения?

Работа с массивом символов рассмотрена на следующем примере.

```
Program Getstr;{ Ввод в символьный массив строки символов }
const SIZE=30;
var buf:array[1..SIZE] of char;//массив символов
   n:integer; { реальная длина введенной строки }
begin
   write('->');
   n:=0;
   repeat
                  if NOT EOLN then
                        begin
                              n:=n+1;
                              read(buf[n]);
                        end;
        until EOLN or (n=SIZE);
```

```
    writeln('Длина введенной строки ',n,' символов.');
end.
```

В примере использована функция *EOLN* (*End Of Line*), возвращающая значение TRUE, если при вводе данных достигнут конец строки.

Задания к примеру

1. Проверьте работу программы при различных размерах буфера.
2. Дополните программу поиском и выводом номера заданного пользователем символа в строке.

Контрольные вопросы

1. Поясните организацию счетчика символов.
2. Какой смысл имеет введение константы SIZE?

В следующем примере строчные буквы русского языка заменяются на соответствующие им прописные.

```
Program Toup;
const
   BSIZE=80;{ предельная длина строки }
var
   buf:array[1..BSIZE] of char;
   n:integer;{ длина введенной строки }
   i:integer;
begin
   n:=0;
   write('-> ');
   repeat
      if not EOLN
         then begin
            n:=n+1;
            read(buf[n]);
         end;
   until EOLN or (n=BSIZE); { здесь прочитана строка и записана в массив buf }
```

```
    for i:=1 to n do { преобразование прописных букв строки в строчные }
        begin
            case buf[i] of
            'a'..'я':buf[i]:=chr(ord(buf[i])-32);
            end;
        end;
    writeln;
    for i:=1 to n do
        write(buf[i]); { вывод преобразованной строки }
end.
```

Задания к примеру

1. Модифицируйте программу так, чтобы она меняла прописные кириллические буквы на строчные.
2. Дополните программу так, чтобы она меняла прописные латинские буквы на строчные.

Контрольные вопросы

1. Объясните смысл операций *CHR* и *ORD*.
2. Объясните назначение и происхождение числа 32 в выражении chr(ord(buf[i])-32).

Пример использования функции *CHR* для вывода специальных символов показан ниже.

```
Program Ramka;
var g,v:char;
   i:integer;
begin
    g:=chr(4);    {символ горизонтальной линейки}
    v:=chr(3);    {символ вертикальной линейки}
    for i:=1 to 34 do write(g);
    writeln;
    writeln(v,'Пример использования функции CHR',v);
    writeln(v,'     для рисования рамки.     ',v);
    for i:=1 to 34 do write(g);
    writeln;
end.
```

Задание к примеру

Измените размер и оформление рамки, нарисуйте двойную рамку.

Контрольные вопросы

1. Где найти символы псевдографики?
2. В чем отличие растровой графики от псевдографики?

Для работы со строковыми переменными (*string*), предусмотрены функции *POS* («подстрока», «строка») и *LENGTH* («строка»). Первая ищет «подстроку» в «строке» и возвращает номер символа в «строке», с которого начинается «подстрока». Вторая возвращает длину «строки». Команда *DELETE* («строка», *N*, *M*) используется для удаления *M* символов из «строки», начиная с символа номера *N*. Пример работы со строковой переменной показан ниже.

```
Program Delsp; { Удаление начальных пробелов строки }
var st:string[30];
begin
   write('Введите строку:');
   readln(st);
   while (pos(' ',st) = 1) and (length(st)>0) do
      delete(st,1,1);
   write('Строка без начальных пробелов:',st);
end.
```

Задание к примеру

Дополните программу выводом позиции заданного символа и выводом символа из заданной позиции в строке.

К строковым переменным можно применить операции сравнения, как в следующем примере.

```
Program Strpal;{ Пароль (сравнение символьных строк) }
var name:string[30];
```

```
   parol:string[6];
begin
    write('Пароль (6 символов)?');
    readln(parol);
    if parol='секрет' then begin
        write('Ваша фамилия? ');
        readln(name);
        writeln(name,', ВАМ доступ разрешен.');
    end
    else begin
        writeln('Неверный пароль! Доступ запрещен!');
    end;
end.
```

Задание к примеру

Дополните программу подтверждением пароля.

Контрольные вопросы

1. Какого типа операции применимы к строковым переменным?
2. Что означает число 30 в записи: «name:string[30]»?

В следующем примере для хранения списка строк использован массив, содержащий строковые элементы.

```
Program Inlist;{ввод/вывод списка}
const N=35;
var student:array[1..N] of string;
   name:string[30];
   i:integer;
begin
    i:=1;
    repeat
        write('Введите фамилию или <Enter> для завершения -> ');
        readln(name);
        if name[1]<>chr(13) // первый символ строки не равен символу
                                перевода каретки
            then begin
```

```
            student[i]:=name;
            i:=i+1;
          end;
   until (name[1]=chr(13)) or (i=N);

   { вывод введенного списка }
   i:=1;
   while (length(student[i])<>0) and (i<=N) do begin writeln(student[i]);i:=i+1
end;
end.
```

Задание к примеру

Дополните программу поиском и выводом номера строки из списка по введенному пользователем шаблону.

Контрольные вопросы

1. Чем ограничивается размер списка и длина вводимого имени?
2. Какой результат и почему выдает программа, когда длина вводимого имени или их количество превышают заданные?

Для преобразования строкового типа в числовой предназначена команда *VAL*, имеющая следующий синтаксис: *VAL* («строка», «переменная», «код ошибки»). Команда считывает из «строки» число и записывает его в «переменную» численного типа. Если преобразование не удалось выполнить, то «код ошибки» оказывается не равен нулю.

```
Program Month_8;{ Использование VAL с проверкой правильности }
                        { преобразования }
var month:integer;
   buf:string[10];   { буфер для ввода строки}
   code:integer;     { код ошибки преобразования строки в число }
begin
    repeat
       write('Месяц->');
```

```
      readln(buf);    { вводим номер месяца как строку }
      val(buf,month,code);{ преобразуем строку в число }
      if code<>0   { введенная строка не цифра }
         then writeln('Ошибка! Месяц следует задать цифрой.');
   until code=0;
end.
```

Задание к примеру

Дополните программу арифметическими операциями над введенными данными.

Контрольные вопросы

1. Объясните действия функции VAL.
2. В каких случаях переменная CODE=0?
3. Когда следует применять функцию VAL?

1.5. Работа с массивами

Переменные типа «массив» удобно использовать, если требуется хранить или обрабатывать однотипные данные. В следующем примере массив *day* использован для хранения 7 элементов типа *string*, а массив *temper* будет содержать в себе 7 элементов типа *single* — числа с плавающей точкой одинарной точности.

```
Program Midtem;{ Вычисление средней (за неделю) температуры воздуха }
var day:array[1..7] of string [11]; { названия дней недели }
   temper:array[1..7] of single; { массив температур }
   sum: single; { сумма температур за неделю }
   sredn: single; { средняя температура за неделю }
   i:integer;
begin
    day[1]:='Понедельник';
    day[2]:='Вторник';
    day[3]:='Среда';
```

```
    day[4]:='Четверг';
    day[5]:='Пятница';
    day[6]:='Суббота';
    day[7]:='Воскресенье';
    writeln('Задайте температуру воздуха за неделю.');
    for i:=1 to 7 do
        begin
            write(day [i],'->');
            readln(temper[i]);
        end;
    { вычисление средней температуры за неделю }
    sum:=0;
    for i:=1 to 7 do
        sum:=sum + temper [i];
    sredn:=sum/7;
    writeln ('Средняя температура за неделю:',sredn:6:2);
end.
```

Задание к примеру

Модифицируйте программу на расчет среднегодовой температуры.

Контрольные вопросы

1. Что такое многомерный массив? Как он записывается?
2. Можно ли объединять в одном массиве данные разных типов?
3. Примените при выполнении задания многомерный массив, например, для записи текущих и средних температур.

Поиск элемента массива по шаблону может осуществляться путем перебора всех элементов массива, как в следующем примере.

```
Program Poisk_2;
var massiv:array[1..10] of integer; { массив целых}
   obrazec:integer; { образец для поиска }
   naiden:boolean; { признак совпадения с образцом }
   i:integer;
```

```
begin
   { ввод 10 целых чисел }
   writeln('Поиск в массиве.');
   write('Введите 10 целых в одной строке через пробел ');
   writeln ('и нажмите <Enter>');
   write('->');
   for i:=1 to 10 do read(massiv[i]);
   { числа введены в массив }
   write('Введите образец для поиска (целое число)-> ');
   readln(obrazec);
   { поиск простым перебором }
   naiden:=FALSE; { совпадений нет }
   i:=1; { проверяем с первого элемента массива }
   repeat
      if massiv[i] = obrazec
         then naiden:=TRUE    { совпадение с образцом }
         else i:=i+1;    { переход к следующему элементу }
   until (i>10) or (naiden); { завершим, если совпадение с образцом или
                               { проверен последний элемент массива }
   if naiden
      then writeln('Совпадение с элементом номер', i:3,'. ')
      else writeln('Совпадений с образцом нет.');
end.
```

Задание к примеру

Модифицируйте программу на поиск в массиве из строковых переменных, например, имен людей.

Контрольные вопросы

1. Объясните действия: for i:=1 to 10 do read(massiv[i]);
2. Предложите варианты сокращения времени поиска.

Если элементы массива упорядочены, например, по возрастанию, то для поиска необходимого элемента может использоваться алгоритм бинарного поиска:

```
Program Poisk_3;{ Бинарный поиск в упорядоченном массиве }
var a:array[1..9] of integer; { массив целых }
```

```
   obrazec:integer; { образец для поиска }
   sred,verh,niz:integer;  { номера среднего, верхнего и нижнего элементов }
                           { массива}
   naiden:boolean;{ признак совпадения с образцом }
   n:integer; { счетчик сравнений с образцом }
   i:integer;
begin
   { ввод 9 целых чисел }
   writeln('Бинарный поиск в массиве.');
   writeln('Введите 9 целых чисел, расположив их по возрастанию');
   for i:=1 to 9 do read(a[i]);
   { здесь числа в массив введены }
   writeln('Введите образец для поиска (целое число) ');
   readln(obrazec);
   { бинарный поиск }
   verh:=1; niz:=9; naiden:=FALSE; n:=0;
   writeln(' verh  niz   sred');
   repeat
      sred:=(niz-verh) div 2+verh;
      writeln(verh:5,niz:5,sred:5);
      n:=n+1;
      if a[sred]=obrazec then naiden:=TRUE
      else begin
         if obrazec<a[sred]
            then niz:=sred-1
            else verh:=sred+1;
      end;
   until (verh>niz) or naiden;
   if naiden
      then write('Совпадение с элементом номер ',
              sred,'. Выполнено ', n, ' сравнений.')
      else writeln('Образец в массиве не найден.');
end.
```

Задание к примеру

Модифицируйте программу на произвольное число членов массива.

Контрольные вопросы

1. Проанализируйте программу, объясните суть метода бинарного поиска.
2. Будет ли осуществляться поиск, если массив не упорядочен?
3. Предложите алгоритм упорядочивания массива.

В ряде случаев для хранения данных удобно применить массив, имеющий более одной размерности, например, двумерный. Двумерный массив можно сравнить с таблицей: каждый элемент таблицы имеет 2 координаты — номер строки и номер столбца. Точно так же и элемент двумерного массива имеет 2 индекса. Пример использования двумерного массива показан ниже.

```
Program Zavod;{Ввод, обработка, вывод в двумерный массив}
const ZAV=3; IZD=4;
var prod:array[1..ZAV+1, 1..IZD+1] of integer;
   i:integer; { номер завода }
   j:integer; { номер изделия }
begin
    writeln('Ввод исходных данных.');
    for i:=1 to ZAV do
    begin
        writeln('Завод',i:2);
        for j:=1 to IZD do
            begin
                write('Изделие',j:2,'->');
                readln(prod[i, j]);
            end;
    end;
    { вычислим общее количество изделий, выпускаемых каждым заводом }
    for i:=1 to ZAV do
        begin
            prod [i, IZD+1] := 0; { общее число изделий, произведенное i-м }
                                  { заводом}
            for j:=1 to IZD do
                prod [i, IZD+1] := prod[i, IZD+1] + prod[i, j];
```

```
        end;
    { вычислим количество изделий одного наименования, выпущенных }
    { всеми заводами }
    for j:=1 to IZD do
        begin
            prod [ZAV+1, j] := 0;
            for i:=1 to ZAV do
                prod [ZAV+1, j] := prod[ZAV+1, j] + prod[i, j];
        end;
    { вывод итоговой таблицы }
    writeln ('          1   2   3   4   Всего');
    for i:=1 to ZAV+1 do
        begin
            if i<=ZAV then write('Завод ', i:2) else write('Всего   ');
            for j:=1 to IZD+1 do write(prod[i, j]:5);
            writeln;
        end;
end.
```

Задания к примеру

1. Добавьте расчет числа всех изделий, выпущенных всеми заводами.
2. Модифицируйте программу на произвольное число членов массива.

Контрольные вопросы

1. Каков принцип формирования данных многомерного массива?
2. Поясните действие операции prod[i,IZD+1] := prod[i,IZD+1] + prod[i, j], сравните их с действиями операции prod[ZAV+1,j]:= prod[ZAV+1,j] + prod[i,j].

Сортировка элементов массива может осуществляться различными способами. Простейший вариант алгоритма сортировки массива рассмотрен в следующем примере.

```
Program Sortar_5; {Сортировка массива простым выбором}
const SIZE=5;
var a:array[1..SIZE] of integer;
   i:integer; { номер элемента, от которого ведется поиск }
            { минимального эл-та }
   min:integer; { номер минимального элемента в части }
              { массива от i до верхней границы массива }
   j:integer; { номер эл-та, сравниваемого с минимальным }
   buf:integer; { буфер, используемый при обмене эл-тов массива }
   k:integer;
begin
    writeln('Сортировка массива.');
    write('Введите',SIZE:3,' целых чисел в одной строке ');
    writeln('через пробел и нажмите <Enter>');
    for k:=1 to SIZE do read(a[k]);
    writeln('Сортировка');
    for i:=1 to SIZE-1 do
    begin
       { поиск минимального элемента в части массива от a[i] до a[SIZE]}
       min:=i;
       for j:=i+1 to SIZE do if a[j]<a[min] then min:=j;
       { поменяем местами a[min] и a[i] }
       buf:=a[i];
       a[i]:=a[min];
       a[min]:=buf;
       {выведем массив }
       for k:=1 to SIZE do write(a[k],' ');
       writeln;
    end;
    writeln('Массив отсортирован.');
end.
```

Задание к примеру

Модифицируйте программу для работы с массивом символьного типа.

Контрольные вопросы

1. Проанализируйте программу, объясните суть метода сортировки простым выбором.

2. Каково назначение и использование переменной *buf*?
3. Как можно модифицировать программу для работы с массивом строкового типа?

Программа сортировки массива пузырьковым методом представлена ниже.

```
Program Sortar_6; {Сортировка массива перестановкой элементов}
const SIZE=5;
var a:array[1..SIZE] of integer;
   i:integer; { счетчик циклов }
   k:integer; { текущий индекс элемента массива }
   buf:integer;
begin
    writeln('Сортировка массива пузырьковым методом.');
    write('Введите',SIZE:3,' целых в одной строке через пробел ');
    writeln ('и нажмите <Enter>');
    for k:=1 to SIZE do read(a[k]);
    writeln('Сортировка.');
    for i:=1 to SIZE-1
    do begin
       for k:=1 to SIZE-1
       do begin
          if a[k]>a[k+1]
          then begin
             { обменяем k-й и (k+1)-й элементы }
             buf:=a[k];
             a[k]:=a[k+1];
             a[k+1]:=buf;
          end;
       end;
       for k:=1 to SIZE do write(a[k],' ');
       writeln;
    end;
    writeln('Массив отсортирован.');
end.
```

Задание к примеру

Модифицируйте программу для работы с массивом строкового типа.

Контрольные вопросы

1. Проанализируйте программу, объясните суть метода сортировки перестановкой.
2. Как можно модифицировать программу для работы с двумерным массивом?

1.6. Подпрограммы: процедуры и функции

Использование подпрограмм позволяет разбить большую программу на несколько относительно самостоятельных блоков, что упрощает отладку программы и делает код более структурированным. Существует два вида подпрограмм: процедуры и функции.

Процедура — это имеющий идентификатор и оформленный особым образом фрагмент программы. Функция отличается от процедуры тем, что после ее выполнения она возвращает некоторое значение в программу, поэтому при описании функции необходимо указывать тип этого значения (тип функции). Процедуры и функции располагаются в разделе описаний.

Для передачи данных в подпрограмму и из нее могут использоваться параметры вызова, которые бывают двух видов: параметр-значение и параметр-переменная. Параметр-значение сохраняет свое значение после вызова подпрограммы, даже если он был изменен внутри подпрограммы. Параметр-переменная, при изменении его внутри подпрограммы, напротив, сохраняет измененное значение и поэтому может использоваться для передачи данных из подпрограммы в программу.

Ниже рассмотрен пример использования функции с параметром-значением.

```
Program Chet_1;{проверка целого числа на четность}
var m:integer;
```

```
 function Chet(n:integer):boolean; { Функция проверки целого числа на }
                                   { четность }
 begin
  if (n mod 2) = 0 then Chet:=TRUE else Chet:=FALSE;
 end;
begin
 writeln('Введите целое число и нажмите <Enter>');
 writeln('Для завершения введите 100');
 repeat
  write('->');read ( m );
   if chet(m) then writeln( m,' - четное число') else writeln( m,' - нечетное число');
 until m = 100;
end.
```

Задание к примеру

Напишите и примените функцию проверки числа на нечетность.

Контрольные вопросы

1. Объясните область действия переменной m:integer.
2. Каково действие операции if chet(m) и каков возможный результат?

Подпрограммы могут иметь собственные (локальные) переменные, которые доступны внутри подпрограммы и недоступны из основной программы. В следующем примере использована функция с локальной переменной *pr* типа *real*.

```
Program Cubrt_2;{Функция "Кубический корень"}
 Function cubrt(x:real):real;
 var pr:real;    { приближенное значение кубического корня}
 begin
  pr:=sqr(x); { первое приближение, в качестве второго приближения }
              {выбираем (x:pr):pr }
   while abs(pr-x/(pr*pr))>0.001 do begin { новое приближение - среднее }
            { арифметическое удвоенного приближения на прошлом шаге }
            { и текущего }
    pr:=(2*pr+x/(pr*pr))/3;
```

```
    end;
  cubrt:=pr;
 end;
begin{ Проверим, как работает cubrt }
 writeln(cubrt(3));
end.
```

Задание к примеру

Модифицируйте программу на вычисление кубического корня введенного по запросу числа.

Контрольные вопросы

1. Объясните содержание записи: Function cubrt(x:real):real.
2. Каковы границы раздела описаний в приведенном примере?

Ниже рассмотрен еще один пример оформления функции.

```
{ Функция "Факториал" }
function Factor(n:integer):integer;
var f,i:integer;
begin
 f:=1;
 for i:=2 to n do f:=f*i;
 factor:=f;
end;
```

Задание к примеру

Напишите программу с применением приведенной в примере функции.

Контрольные вопросы

1. Существует ли связь между типами параметра функции и возвращаемого функцией значения?
2. Какова область действия переменных f,i:integer?

Подпрограмма может вызывать другие подпрограммы и даже саму себя. Подпрограмма, вызывающая саму себя, называется рекурсивной. Пример рекурсивной функции приведен ниже.

```
Program Usefac;{Вычисление факториала с использованием рекурсивной }
                    { функции}
var n:integer;    { число, факториал которого надо вычислить}
   f:integer; {результат}
  function factorial(k:integer):integer; { Рекурсивная функция "Факториал" }
   begin
    if k = 1 then factorial:=1 else factorial:= k*factorial(k-1);
   end;
begin { основная программа }
 writeln('Вычисление факториала с использованием рекурсивной функции.');
 write('Введите число, факториал которого надо вычислить ->');
 readln(n);
 f:=factorial(n);
 writeln('Факториал числа ',n,' равен ',f);
end.
```

Задание к примеру

Изобразите алгоритм программы и функции.

Контрольные вопросы

1. Как реализовать функцию расчета факториала без рекурсии?
2. Можно ли в приведенном примере отказаться от операции f:=factorial(n)?

Далее представлена программа, использующая процедуру для вывода на экран 30 символов:

```
Program Prline;{Вывод строки из 30 черточек}
var x:integer;y:real;
  Procedure Line30;
  var i:integer;
  begin
```

```
   for i:=1 to 30 do write('-');
   writeln;
 end;
begin { основная программа }
 writeln('Таблица логарифмов чисел от 1 до 10');
 Line30;    { нарисовать линию }
 for x:=1 to 10 do begin
   y:=ln(x); { функция вычисления логарифма x }
   writeln(x:3,y:8:3);
 end;
 Line30;    { нарисовать линию }
end.
```

Задание к примеру

Модифицируйте программу на рисование строки из заданного пользователем числа символов, также заданных пользователем.

Контрольные вопросы

1. Имеются ли у процедуры, приведенной в примере, параметры?
2. Можно ли оформить приведенную в примере процедуру в виде функции?

Следующий пример иллюстрирует использование подпрограммы с параметрами-переменными (признаком параметра-переменной является ключевое слово «*VAR*», расположенное перед описанием параметра подпрограммы).

```
Program Sqring;{Вычисление длины и площади окружности}
var r,l,s:real; { радиус, длина и площадь окружности }
                { глобальные переменные }
 procedure SqLeOkr(r1:real;var s1:real; var l1:real);
 { r1 – радиус, s1 – площадь круга, l1 – длина окружности }
 const pi=3.14159;{локальная константа}
 begin
  s1:=pi*r1*r1;
```

```
    l1:=2*pi*r1;
  end;
begin{ основная программа }
  writeln('Вычисление длины окружности и площади круга:');
  write('Задайте радиус и нажмите <Enter> ');
  readln(r);
  l:=0;s:=0;
  SqLeOkr(r,s,l);
  writeln('Радиус окружности: ',r:6:3);
  writeln('Длина: ',l:6:3,',площадь: ',s:7:3);
end.
```

Задание к примеру

Модифицируйте программу, включив в нее процедуры графического оформления результатов.

Контрольные вопросы

1. Соотнесите выражения: Procedure SqLeOkr(r1:real; var s1:real; var l1:real); и SqLeOkr(r,s,l).
2. Каков смысл понятия «параметр-переменная»?

1.7. Работа с файлами

Существует два основных вида файлов: текстовые и двоичные. В текстовых файлах вся информация хранится в виде текста, в двоичных — в виде двоичного кода. Преимущество текстовых файлов перед двоичными заключается в возможности их редактирования в любом текстовом редакторе — для просмотра или правки двоичного файла необходим специальный редактор. В текстовом файле может одновременно храниться информация различных типов (строки, числа), в двоичном — только одного жестко заданного типа (*integer*, *char*, *double*). Однако текстовые файлы можно читать только последовательно

(последовательный доступ), в двоичном же файле можно обратиться к любому элементу (произвольный доступ).

При работе с файлами необходимо выполнить следующие действия:

1) связать файловую переменную с конкретным файлом;
2) открыть файл в режиме чтения или записи;
3) прочитать или записать данные;
4) закрыть файл.

Команда, используемая для связи файловой переменной с файлом, записывается так: *ASSIGN («файловая переменная», «имя файла»)*, где *«файловая переменная»* — переменная типа *TEXT* или *FILE*, а *«имя файла»* — строковая константа, содержащая имя файла.

Для открывания файла в режиме чтения предназначена команда *RESET*, в режиме записи — *REWRITE*, для закрывания файла используется команда *CLOSE*; всем командам в качестве параметра передается файловая переменная. Для чтения и записи данных используются рассмотренные ранее команды *READ/READLN* и *WRITE/WRITELN*, которым в качестве первого параметра также передается файловая переменная.

Пример программы, выводящей данные в текстовый файл, показан ниже:

```
Program Sqrfile; {Решение квадратного уравнения}
var a,b,c,x1,x2:real; {коэффициенты и корни уравнения }
    d:real; { дискриминант }
    f:text; { файловая переменная для вывода результатов работы }
begin
 writeln('*Решение квадратного уравнения*');
 writeln('Введите значения коэффициентов');
 write(' в одной строке через пробел и нажмите <Enter> ');
 readln(a,b,c);
 d:=b*b-4*a*c;
 if d >= 0 then begin
  x1:=(-b+sqrt(d))/(2*a);
```

```
    x2:=(-b-sqrt(d))/(2*a);
  end;
{ вывод результатов в файл }
  assign(f,'rezult.txt'); { назначаем файловой переменной имя}
  rewrite(f); { открываем файл в режиме замещения существующего}
  writeln(f,'Решение квадратного уравнения');
  writeln(f,'Коэффициенты уравнения: ',a:6:3,b:6:3,c:6:3);
  if d <0 then writeln(f,'У уравнения нет корней.')
    else writeln(f,'Корни уравнения: x1=',x1:6:3,' x2=',x2:6:3);
  close(f); { закрываем файл }
  writeln('Результаты работы программы в файле rezult.txt');
end.
```

Задания к примеру

1. Убедитесь в том, что в вашем рабочем каталоге появился файл результатов, просмотрите его.
2. Запустите программу с новыми данными, убедитесь, что содержимое файла изменилось.
3. Выведите в файл таблицу значений синуса с шагом 30°.

При использовании команды *REWRITE* данные в файле перезаписываются, т. е. старая информация заменяется новой. Если необходимо дописать информацию в конец файла, не удаляя существующие данные, то вместо команды *REWRITE* используется команда *APPEND*, применение которой рассмотрено ниже.

```
Program Aptemper;{Добавление в файл данных о дневной температуре воздуха}
var DayTemp:record { дневная температура }
       day:integer;    { число }
       month:integer;    { месяц }
       temper:integer;    { температура }
     end;
     f:text;    { файл дневной температуры}
begin
  assign(f,'temperat.txt');
  append(f);    { открываем файл для добавления записи }
  write('Введите в одной строке, разделяя пробелами');
  writeln(' число, номер месяца и температуру');
```

```
 write('->');
 with DayTemp do begin
          readln(day,month,temper); {ввод данных с клавиатуры}
          writeln(f,day,' ',month,' ',temper); {вывод данных в файл}
 end;
 close(f);
 writeln('Данные добавлены'); {вывод на экран}
end.
```

Задание к примеру

Убедитесь, что при повторном запуске программа добавляет данные в файл.

В показанном примере задействована переменная *DayTemp* типа *RECORD* (запись), которая содержит поля *day*, *month* и *temper* типа *integer*. Переменные типа *запись* удобно использовать для объединения связанных по смыслу переменных в одну группу. Обращение к полям записи может осуществляться с использованием конструкции *WITH «запись» DO «оператор»*, как в рассмотренном примере, или напрямую в виде *«запись».«поле»*, т. е. код

```
with DayTemp do begin
        readln(day,month,temper); {ввод данных с клавиатуры}
        writeln(f,day,' ',month,' ',temper); {вывод данных в файл}
end;
```

может быть заменен на

```
readln(DayTemp.day, DayTemp.month, DayTemp.temper); {ввод данных }
                                                    {с клавиатуры}
writeln(f, DayTemp.day,' ', DayTemp.month,' ', DayTemp.temper); {вывод }
                                                    { данных в файл}
```

Следующая программа поясняет процедуру чтения данных из текстового файла.

```
Program Sredtemp; {расчет средней температуры по данным файла}
var DayTemp:record { дневная температура }
       day:integer;    { число }
       month:integer;    { месяц }
       temper:integer;    { температура }
    end;
    f:text;    {файл со значениями температуры}
    mes:integer;   { номер месяца }
    n:integer;    { кол-во дней месяца, о которых есть данные в файле }
    summ:integer; { сумма температур месяца }
    sredn:real; { средняя температура }
begin
 n:=0; summ:=0;
 writeln('Вычисление среднемесячной температуры.');
 write('Месяц->');
 readln(mes);
 assign(f,'temperat.txt'); reset(f);
 while NOT EOF(f) do begin
  with DayTemp do readln(f,day,month,temper);
  if DayTemp.month = mes then begin
   n:=n+1;
   summ:=summ+DayTemp.temper;
  end;
 end;
 close(f);
 if n<>0 then begin
  sredn:=summ/n;
  writeln('Средняя температура:',sredn:6:3);
 end else writeln('Данных о температуре за',mes:3,' месяц нет.');
end.
```

Задания к примеру

1. Убедитесь в том, что запрашиваемый файл присутствует в вашем рабочем каталоге и формат его данных соответствует задаче.
2. Убедитесь, что программа считывает данные из файла.

В примере использована функция *EOF* (*End Of File*) — логическая функция, возвращающая значение *TRUE*, если при чтении достигнут конец файла.

1.8. Графика

Работа с графикой рассматривается в среде PascalABC. NET, для других версий языка принципы и команды работы с графикой отличаются. Графические команды содержатся в библиотеке GraphABC, которую необходимо подключить при помощи объявления ее имени в разделе описаний в блоке *USES*. Следующий пример иллюстрирует использование базовых графических команд.

```
Program sinus;
uses GraphABC; {подключаем библиотеку для работы с графикой}
var n:integer;
begin
 line(0,150,400,150);{рисуем ось X}
 line(10,10,10,300);{рисуем ось Y}
 moveto(10,150);{переместить перо в начало системы координат}
 for n:=0 to 360 do
  lineto(10+n,150-round(100*sin(n/180*pi)));{рисование графика синуса}
end.
```

Задания к примеру

1. Измените цвет и толщину линии графика, обратившись к встроенной справке.
2. Добавьте на экран график косинуса.

Ниже рассмотрен более сложный пример, включающий анимацию.

```
Program Graf;{Пример построения графика и анимации}
Uses GraphABC;
Var
 T,Fi,H,H1,V,V1,A,Ho,Te,W:double;
 I,XX,XX1,SS,SS1,WW,WW1,UU,UU1:Integer;
```

```
Procedure GRAPHIC; {построение графика}
Var
  TT,POZ1,SPD1,ACC1:double;
Begin
  TT:=(20+T*100); XX1:=ROUND(TT);
  POZ1:=(280-H*50); SS1:=ROUND(POZ1); pen.Color:=clwhite;
                    Line(XX1,SS1,XX,SS);
  SPD1:=(180+V); WW1:=ROUND(SPD1); pen.Color:=clred;
                    Line(XX1,WW1,XX,WW);
  ACC1:=(180-A); UU1:=ROUND(ACC1); pen.Color:=cllime;
                    Line(XX1,UU1,XX,UU);
  XX:=XX1;SS:=SS1;WW:=WW1;UU:=UU1;
End;

procedure RedrawMech; {рисование механизма}
Begin
   pen.Color:=clblack; {закрашиваем}
   rectangle(250,350,450,500);
   pen.Color:=cllime; {рисуем}
   Line(250,400,350,Round(400+H*30));
   Line(350,Round(400+H*30),450,400);
   pen.Color:=clyellow;    {рисуем}
   Circle(350,Round(390+H*30),10);
End;

Begin
  Ho:=1;Te:=1;W:=10;
  XX:=20;
  XX1:=20;WW1:=180;UU1:=180;SS1:=280;
  brush.Color:=clblack;
  rectangle(0,0,2000,1300);
     {Рисование осей координат}
      pen.Color:=clgray;
      Line(20,0,20,400);line(0,180,2000,180);line(0,280,2000,280);
{Графики и анимация}
For I:=1 to 1000 do
 Begin
  T:=0.01*I;
  Fi:=W*T;
  H:=Ho*(1-(exp(-T/Te)*cos(Fi)));
  V:=(H-H1)*100;A:=(V-V1)*100;
```

```
 GRAPHIC;
 RedrawMech;
 sleep(20); {пауза между кадрами}
 H1:=H;V1:=V;
End;
End.
```

2. Основы визуального объектного программирования

Примером языка, относимого к языкам визуального объектного программирования, является Delphi — интегрированная среда разработки ПО для Microsoft Windows, созданная первоначально фирмой Borland и на данный момент принадлежащая и разрабатываемая Embarcadero Technologies.

Delphi — это:

высокопроизводительный компилятор в машинный код;

объектно ориентированная модель компонент;

визуальное (следовательно, и скоростное) построение приложений из программных прототипов;

масштабируемые средства для построения баз данных.

При создании программы разработчик выбирает из палитры компонентов готовые элементы (кнопки, надписи, поля ввода данных и т. д.) и помещает их на форму — окно, с которым будет взаимодействовать пользователь. Результаты разработки графической составляющей проекта видны еще до компиляции. Кроме графической (пользовательской) части необходимо написать текстовый код на языке, близком по синтаксису к Паскалю, для реализации алгоритма программы.

Среда Delphi включает в себя полный набор визуальных инструментов для скоростной разработки приложений (RAD — *rapid application development*), поддерживающей разработку пользовательского интерфейса и подключение к корпоративным базам данных.

VCL — библиотека визуальных компонент, включает в себя стандартные объекты построения пользовательского интерфейса, объекты управления данными, графические объекты, объекты мультимедиа, диалоги и объекты управления файлами, управление DDE и OLE.

Delphi поддерживает DLL, написанные на C и C++.

После запуска Delphi, в верхнем окне горизонтально располагаются иконки палитры компонент, из которой можно выбрать компоненты для построения приложения. Компоненты включают в себя как визуальные, так и невизуальные элементы. Такие компоненты, как кнопки, поля редактирования — это визуальные компоненты; таймеры, компоненты для организации взаимодействия по протоколу DDE — невизуальные. Поскольку в Delphi программа строится визуальным образом, все компоненты имеют свое графическое представление в поле форм для того, чтобы можно было ими соответствующим образом оперировать. Но в запущенной на выполнение программе видимыми остаются только визуальные компоненты. Компоненты сгруппированы на вкладках палитры по своим функциям. К примеру, компоненты, представляющие стандартные диалоговые окна Windows, размещены на вкладке палитры с названием DIALOGS.

В левой части экрана вертикально расположено окно инспектора объектов OBJECT INSPECTOR, в котором во время проектирования программы можно устанавливать значения свойств PROPERTIES и событий EVENTS объекта, выделенного в данный момент времени. Для различных объектов (окно, кнопка и др.) набор свойств и событий отличается. Примером свойств могут служить цвет, шрифт, размеры и др. Вкладка EVENTS служит для назначения процедур-обработчиков событий, т. е. сопоставления имени процедуры с событием, при возникновении которого эта процедура будет вызвана. Примерами событий могут быть: нажатие на кнопку формы, запуск или закрытие приложения, перемещение курсора мыши и др.

Центральная часть экрана отводится для окна разрабатываемого приложения и текстового кода.

Delphi обладает встроенным графическим отладчиком, позволяющим находить и устранять ошибки в коде. Можно установить точки останова, проверить и изменить переменные, при помощи пошагового выполнения проконтролировать работу программы.

Ключевой особенностью Delphi является возможность не только использовать визуальные компоненты для строительства приложений, но и создавать новые компоненты. Такая возможность позволяет встраивать новые инструменты в существующую среду. Кроме того, можно улучшить или заменить существующие в Delphi компоненты.

Проект приложения, разрабатываемого в Delphi, включает:

текстовый файл главной программы — файл с расширением dpr;

файлы с описанием форм программы (могут отсутствовать, например, при разработке консольного (выполняемого из командной строки) приложения) — файлы с расширением dfm;

файлы-модули, содержащие текст кода, реализующий алгоритм задачи, — файлы с расширением pas.

Формы — это объекты, в которые программист помещает другие объекты для создания пользовательского интерфейса приложения. Модули состоят из кода, который реализует функционирование приложения, обработчики событий для форм и их компонент. Для каждой формы среда Delphi создает свою пару файлов dfm-pas. Оба файла автоматически синхронизируются Delphi; если в проект добавить новую форму, связанный с ней файл.pas будет создан автоматически, а ссылка на него будет добавлена в проект. Такая синхронизация и делает Delphi two-way-инструментом, обеспечивая полное соответствие между кодом и визуальным представлением.

При добавлении нового объекта или кода, Delphi устанавливает так называемую «кодовую синхронизацию» между ви-

зуальными элементами и соответствующими им кодовыми представлениями. Например, для добавления описания поведения формы, при старте можно создать обработчик события ONCREATE, дважды щелкнув мышкой непосредственно на форму или дважды щелкнув мышкой по строке ONCREATE в таблице EVENTS в ИНСПЕКТОРЕ ОБЪЕКТОВ. Delphi создаст процедуру или заголовок метода, куда можно добавить код:

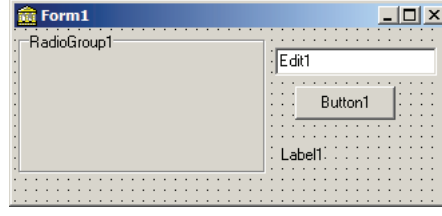
```
procedure TForm1.FormCreate(Sender: TObject);
begin
end;
```

Создавая этот код, Delphi автоматически формирует декларацию объекта TForm1, которая содержит процедуру FormCreate, представляющую собой обработчик события OnCreate.

Далее рассмотрен пример создания простейшей программы, выполняющей пересчет дюймов в сантиметры и сантиметров в дюймы. Для построения такой программы необходимо:

1) запустить среду Delphi;

2) поместить на окно формы следующие компоненты — RADIOGROUP, EDIT, BUTTON и LABEL (все компоненты расположены во вкладке STANDARD), разместив их так, как показано ниже:



3) перейти к редактированию свойства ITEMS элемента RADIOGROUP1. Для этого следует выделить указанный компонент на форме и дважды щелкнуть по значению поля ITEMS в окне OBJECT INSPECTOR;

4) в появившемся окне набрать две строки: «Дюймы в сантиметры» и «Сантиметры в дюймы» (без кавычек);

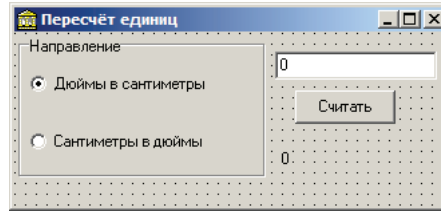
5) значение поля ITEMINDEX этого же компонента установить в 0;

6) дважды щелкнуть по компоненту BUTTON1 на форме для перехода к редактированию кода обработчика события BUTTON1CLICK;

7) в текстовом редакторе кода набрать следующее —

```
 if RadioGroup1.ItemIndex=0 then label1.Caption:=floattostr(strtofloat(edit1.
text)*2.54)
 else label1.Caption:=floattostr(strtofloat(edit1.text)/2.54);
```

8) изменить надписи на элементах формы в соответствии с их функциями. Для этого потребуется отредактировать значения полей CAPTION для компонентов FORM1, RADIOGROUP1, BUTTON1 и LABEL1 и поля TEXT для компонента EDIT1. Конечный вариант программы должен выглядеть следующим образом:



9) запустить программу, нажав кнопку клавиатуры F9, и проверить правильность работы приложения.

3. Язык C#

3.1. Общие сведения

C# (произносится «си шарп») — объектно ориентированный язык программирования. Разработан в 1998–2001 гг. группой инженеров компании Microsoft как язык разработки приложений для платформы Microsoft.NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в т. ч. операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщенные типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Среда программирования Microsoft Visual C# Express Edition позволяет создавать как приложения с пользовательским интерфейсом, так и консольные приложения. Далее приведены тексты консольных приложений, аналогичных рассмотренным ранее при изучении языка Паскаль.

3.2. Операторы ввода-вывода и управления вычислениями

В консольных приложениях для организации ввода и вывода данных можно использовать команды *Console.Write*, *Console.WriteLine* и *Console.ReadLine*. Переменные могут быть описаны в любом месте программы с помощью следующего синтаксиса: *«тип переменной» «имя переменной» = «начальное значение»*. Начальное значение переменной можно присвоить не при объявлении переменной, а после. В качестве начального значения может выступать результат, возвращаемый функцией, как в примере, показанном ниже.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Пересчет расстояния: Мили -> Километры.");
                                          //вывод на экран
      Console.Write("Задайте расстояние в милях: ");
      string m = Console.ReadLine(); //описание переменной и ввод данных
                                          пользователя
      double k = Convert.ToDouble(m)*1.6;  //преобразование типа и расчет
                                          нового значения
      Console.WriteLine("{0} миль это {1} км.",m,k); //вывод с параметрами
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

На основе приведенного примера составьте программу, выставляющую запрос на ввод имени пользователя и приветствие с обращением по имени, применив переменную типа String.

Условный оператор языка C# записывается так: *if («условие») «оператор 1» else «оператор 2»*. При выполнении условия будет выполнен оператор 1, иначе — оператор 2. Если в качестве оператора нужно выполнить несколько команд, то они заключаются в операторные скобки — {}. В следующем примере переменные *x1* и *x2* описаны внутри операторных скобок условного оператора и доступны только внутри этих скобок.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("*Решение квадратного уравнения*");
      double a, b, c, d;//описание переменных без присвоения начальных
                        значений
      Console.Write("Введите значение коэффициента a: ");
      a = Convert.ToDouble(Console.ReadLine());
      Console.Write("Введите значение коэффициента b: ");
      b = Convert.ToDouble(Console.ReadLine());
      Console.Write("Введите значение коэффициента c: ");
      c = Convert.ToDouble(Console.ReadLine());
      d = b * b - 4 * a * c;
      if (d >= 0) //условный оператор
      {// открывающая операторная скобка
        double x1, x2;
        x1 = (-b + Math.Sqrt(d)) / (2 * a);
        x2 = (-b - Math.Sqrt(d)) / (2 * a);
        Console.WriteLine("Корни уравнения");
        Console.WriteLine("x1= {0}   x2= {1}", Math.Round(x1, 2),
            Math.Round(x2, 2));
      }// закрывающая операторная скобка
      else
        Console.WriteLine("Корней нет.");
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Составьте программу расчета корней биквадратного уравнения.

Функция *Math.Round (x1, 2)* возвращает округленное до 2 знаков после запятой значение переменной *x1*.

В языке C# определены следующие операторы сравнения: равно (==), не равно (!=), больше (>), меньше (<), больше или равно (>=), меньше или равно (<=).

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Введите ваш вес в кг: ");
      float вес = Convert.ToSingle(Console.ReadLine());
      Console.Write("Введите ваш рост в см: ");
      float рост = Convert.ToSingle(Console.ReadLine());
      float оптимум = рост - 100;
      if (вес == оптимум)
        Console.WriteLine("Поздравляем, Ваш вес оптимален!");
      else if (вес < оптимум)
        {
          float разница = оптимум - вес;
          Console.WriteLine("Вам надо поправиться на {0} кг.",разница);
        }
      else
        {
          float разница = вес - оптимум;
          Console.WriteLine("Вам надо похудеть на {0} кг.",разница);
        }
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

Логические операции записываются таким образом: И — «&», ИЛИ — «|», НЕ — «!», Исключающее ИЛИ — «^».

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Введите продолжительность разговора: ");
      double dlitelnost = Convert.ToDouble(Console.ReadLine());
      Console.Write("Введите номер для недели: ");
      int day = Convert.ToInt32(Console.ReadLine());
      double SummaKOplate = dlitelnost*1;
      if (day == 6 | day==7)// | - это логическое "ИЛИ"
        SummaKOplate = SummaKOplate * 0.5;
      Console.WriteLine("Стоимость разговора: {0} р.",SummaKOplate);
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Модифицируйте программу, введя в расчет скидку на ночной тариф.

Оператор множественного выбора имеет следующий синтаксис:

switch («ключ выбора»)
{
case «константа выбора 1»: «оператор 1»; break;
case «константа выбора 2»: «оператор 2»; break;
...

default: «оператор default»; break;
}

«Оператор N» выполняется, если «ключ выбора» совпадает с «константой выбора N»; оператор *default* выполняется, если «ключ выбора» не совпал ни с одной из констант выбора.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Введите целое число (от 1 до 999): ");
         int n = Convert.ToInt32(Console.ReadLine());
         if (n > 100)
            n = n % 100; // остаток от деления на 100
         if ((n >= 11) & (n <= 14))
            Console.WriteLine("рублей");
         else
         {
            int r = n % 10;
            switch (r)
            {
               case 0:
               case 5:
               case 6:
               case 7:
               case 8:
               case 9:
                  Console.WriteLine("рублей");
                  break;
               case 1:
                  Console.WriteLine("рубль");
                  break;
               case 2:
               case 3:
               case 4:
                  Console.WriteLine("рубля");
                  break;
```

```
            }
        }
        Console.ReadLine();
    }
  }
}
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

Для организации переходов внутри программы, можно использовать оператор *GOTO*. Точка перехода обозначается меткой, признаком которой является символ «:», расположенный после имени метки.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      начало: //метка для перехода
      Console.Write("введите пароль: ");
      string parol=Console.ReadLine();
      if (parol == "123")
        Console.WriteLine(" Правильно!");
      else
      {
        Console.WriteLine(" Неверный пароль!");
        goto начало; //переход
      }
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Выполните любой из предыдущих примеров при правильно набранном пароле.

В следующем примере использованы логические переменные и константы. Программа определяет, является ли введенная дата последним днем месяца.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("Какой нынче год? ");
         int year = Convert.ToInt32(Console.ReadLine());
         Console.Write("А месяц? ");
         int month = Convert.ToInt32(Console.ReadLine());
         Console.Write("А число? ");
         int day = Convert.ToInt32(Console.ReadLine());
         bool last = false;
         switch (month)
         {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
               if (day == 31)
                  last = true;
               break;
            case 4:
            case 6:
            case 9:
            case 11:
               if (day == 30)
                  last = true;
               break;
            case 2:
               if (day == 28)
               {
```

```
                int r = year % 4;
                if (r != 0)
                    last = true;
            }
            break;
        }
        if (last)
        {
            Console.WriteLine("Последний день месяца!");
            day = 1;
            if (month == 12)
            {
                month = 1;
                year++; //это компактная запись выражения year = year + 1
                Console.WriteLine("С наступающим Новым годом!");
            }
            else
                month++;
        }
        else
            day++;
        Console.WriteLine("Завтра {0}.{1}.{2}",day,month,year);
        Console.ReadLine();
    }
  }
}
```

Эта же задача может быть решена по-другому:

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Какой нынче год? ");
      int year = Convert.ToInt32(Console.ReadLine());
      Console.Write("А месяц? ");
      int month = Convert.ToInt32(Console.ReadLine());
```

```
        Console.Write("А число? ");
        int day = Convert.ToInt32(Console.ReadLine());
        DateTime d1 = new DateTime(year, month, day);//структура для
                                                        хранения даты
        d1 = d1.AddDays(1);//прибавили к дате 1 день
        if (d1.Day==1)
        {
          Console.WriteLine("Последний день месяца!");
          if (month == 1)
          {
            Console.WriteLine("С наступающим Новым годом!");
          }
        }
        Console.WriteLine("Завтра {0}", d1.ToLongDateString());
        Console.ReadLine();
    }
  }
}
```

Задание к примеру

Применяя сведения, полученные ранее, модифицируйте программу таким образом, чтобы она ко всему прочему накануне выставляла поздравление с днем рождения.

3.3. Операторы циклов

В языке C# определены 4 оператора для организации циклов: *for*, *foreach*, *while*, *do-while*. Пример использования счетного оператора *for* показан ниже. Если внутри цикла необходимо выполнить несколько инструкций, то они заключаются в операторные скобки {}.

```
using System;
namespace ConsoleApplication1
{
```

```
class Program
{
    static void Main(string[] args)
    {
        double summa = 0;
        double elem;
        for (int i = 1; i <= 5; i++)//для целочисленной переменной i от 1 до
                                        5 с увеличением i
        {//выполнять
            elem = 1.0 / i;
            summa += elem;//это оптимизированная запись выражения
                                summa = summa + elem
            Console.WriteLine("{0} {1}",i,Math.Round(elem,3));
        }
        Console.WriteLine("Сумма элементов ряда: {0}", Math.Round(summa,3));
        Console.ReadLine();
    }
}
}
```

Задания к примеру

1. Изобразите структурную схему алгоритма работы программы.
2. Дополните программу вводом произвольного числа элементов ряда.

Для последовательного перебора элементов массива удобно применить оператор *foreach*, формат записи которого следующий:

foreach («тип переменной» «переменная» in «массив») «оператор».

«Оператор» выполнится для каждого элемента «массива», причем значение данного элемента будет доступно через «переменную», которая автоматически принимает значение текущего элемента массива на каждой итерации цикла. Переменная типа «массив» объявляется путем добавления квадратных скобок после указания типа переменной: *string [] days* — стро-

ковый массив *days*. После объявления массива можно перечислить значения его элементов, тогда указывать число элементов массива нет необходимости; или можно выделить память под заданное количество элементов массива: *string [] days = new string [5]* — массив из 5 элементов. Начальный элемент массива имеет индекс, равный нулю.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      string[] days = { "Понедельник", "Вторник", "Среда", "Четверг",
          "Пятница", "Суббота", "Воскресенье" };// массив строк
      foreach (string day in days)//для каждой строки day в массиве days
                                  выполнить
        Console.WriteLine(day);//вывод day на экран
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Измените порядок вывода дней недели на обратный, применив оператор цикла *FOR*.

Цикл с предпроверкой условия оформляется так: *while («условие») «оператор»*. «Оператор» выполняется, пока выполняется «условие». Код программы, вычисляющей число пи с использованием цикла while, приведен ниже.

```
using System;
namespace ConsoleApplication1
{
  class Program
```

```
  {
    static void Main(string[] args)
    {
      double pi = 0, elem=1;
      int n = 1;
      Console.Write("Задайте точность вычисления числа пи (например,
              0,001)->");
      double t = Convert.ToDouble(Console.ReadLine());
      Console.WriteLine("Вычисление числа пи с точностью {0}",t);
      while (elem >= t / 4)//пока выполняется условие elem >= t / 4
      {//выполнять следующие инструкции
        elem=1.0/(2*n-1);
        if ((n % 2) == 0)
          pi -= elem;//аналог pi = pi - elem
        else pi+=elem;
        n++;
      }
      pi*=4;//умножить pi на 4
      Console.WriteLine("Значение ПИ с точностью {0} равно {1}", t, pi);
      Console.WriteLine("Просуммировано {0} членов ряда.", n);
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

Оператор цикла с постпроверкой условия имеет следующий синтаксис: *do «оператор» while («условие»)*. «Оператор» будет выполняться до тех пор, пока выполняется «условие». Условие продолжения цикла проверяется после выполнения оператора, поэтому цикл всегда выполняется хотя бы один раз.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
```

```
    static void Main(string[] args)
    {
      int summa = 0,num;
      Console.WriteLine("Вычисление суммы целых чисел.");
      do//цикл с постпроверкой условия
      {
        Console.Write("->");
        num = Convert.ToInt32(Console.ReadLine());
        summa+=num;
      }
      while (num!=0);//условие продолжения цикла
      Console.WriteLine("Сумма введенных чисел: {0}",summa);
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Модифицируйте программу таким образом, чтобы она отдельно выполняла подсчет суммы положительных и отрицательных чисел.

Еще один пример использования цикла *do-while* показан ниже.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Введите целое число: ");
      int d = 2, r, n = Convert.ToInt32(Console.ReadLine());
      do
      {
        r = n % d;//остаток от деления n на d поместить в r
        if (r != 0)//n не разделилось нацело на d
          d++;
```

```
        }
        while (r != 0);
        if (d==n)
          Console.WriteLine(n.ToString() + " - простое число.");
        else
          Console.WriteLine(n + " - непростое число.");
        Console.ReadLine();
      }
    }
}
```

Задание к примеру

Изобразите структурную схему алгоритма работы программы.

3.4. Работа с символьными и строковыми переменными

Строку в C# можно считать массивом символов. В следующем примере введенные пользователем слова выводятся на экран построчно.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Введите несколько слов, разделяя их
пробелами:");
      string s = Console.ReadLine();
      for (int i = 0; i < s.Length; i++)
        if (s[i] != ' ')//s[i] - это символ строки - тип char
          Console.Write(s[i]);
        else
          Console.WriteLine();
```

```
        Console.ReadLine();
      }
   }
}
```

Задание к примеру

Измените программу таким образом, чтобы в качестве разделителя слов использовался не только пробел, но и запятая.

Разделить строку на отдельные слова можно при помощи специальной функции языка C# — *Split*:

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Введите несколько слов, разделяя их пробелами:");
         string s = Console.ReadLine();
         string[] str=s.Split(' ');//массив строк, содержащих отдельные слова
         foreach (string elem in str)
            Console.WriteLine(elem);
         Console.ReadLine();
      }
   }
}
```

Задание к примеру

Применив функцию *Replace*, напишите программу, которая заменяет во введенной строке все точки на запятые.

Для работы со строковыми переменными язык C# имеет множество инструментов. В следующем примере использована процедура *ToUpper*, преобразующая строчные буквы строки в прописные.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Введите строку:");
      string m=Console.ReadLine();
      m = m.ToUpper();
      Console.WriteLine(m);
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Модифицируйте программу на преобразование прописных букв строки в строчные.

Функция *TrimStart* удаляет начальные символы из строки, если они совпадают с символом, переданным функции в качестве параметра:

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.Write("Введите строку:");
      string m=Console.ReadLine();
      m=m.TrimStart(' ');
      Console.WriteLine("Строка без начальных пробелов:"+m);
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Дополните программу выводом позиции заданного символа и выводом символа из заданной позиции в строке.

3.5. Массивы

Основные принципы работы с массивами поясняются в следующем примере.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] day = new string[7];//строковый массив из 7 элементов
         day[0]="Понедельник";//начальный элемент массива имеет индекс,
                                    равный нулю
         day[1]="Вторник";
         day[2]="Среда";
         day[3]="Четверг";
         day[4]="Пятница";
         day[5]="Суббота";
         day[6]="Воскресенье";
         double[] temper = new double[7];//массив типа double
         Console.WriteLine("Задайте температуру воздуха за неделю.");
         int i;
         for (i = 0; i <= 6; i++)
         {
            Console.Write(day[i] + "->");
            temper[i] = Convert.ToDouble(Console.ReadLine());
         }
         double sum = 0;
         foreach (double t in temper)
            sum += t;
         double sredn = sum / 7;
```

```
        Console.WriteLine("Средняя температура за неделю: {0}",
                                Math.Round(sredn,1));
        Console.ReadLine();
      }
   }
}
```

Задание к примеру

Модифицируйте программу на расчет среднемесячной температуры.

В следующем примере рассмотрен поиск в числовом массиве.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Поиск в массиве из 10 целых чисел.");
         int[] massiv = new int[10];
         int i;
         for (i = 0; i < 10; i++)
         {
            Console.Write("->");
            massiv[i] = Convert.ToInt32(Console.ReadLine());//ввод элементов
                                                                  массива
         }
         Console.Write("Введите образец для поиска: ");
         int obrazec = Convert.ToInt32(Console.ReadLine());
         bool naiden = false;
         i=1;
         foreach (int elem in massiv)//поиск в массиве
            if (elem == obrazec)
            {
               naiden = true;
               break;
```

```
            }
            else
               i++;
         if (naiden)
            Console.WriteLine("Совпадение с элементом номер {0}",i);
         else
            Console.WriteLine("Совпадений с образцом нет.");
         Console.ReadLine();
      }
   }
}
```

Задание к примеру

Модифицируйте программу на поиск в массиве из строковых переменных, например, имен людей.

Задача поиска в массиве может быть решена другим способом:

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Поиск в массиве из 10 целых чисел.");
         int[] massiv = new int[10];
         int i;
         for (i = 0; i < 10; i++)
         {
            Console.Write("->");
            massiv[i] = Convert.ToInt32(Console.ReadLine());
         }
         Console.Write("Введите образец для поиска: ");
         int obrazec = Convert.ToInt32(Console.ReadLine());
         i = Array.IndexOf(massiv, obrazec);//поиск в массиве
         if (i>=0)
            Console.WriteLine("Совпадение с элементом номер {0}",i+1);
         else
            Console.WriteLine("Совпадений с образцом нет.");
```

```
        Console.ReadLine();
      }
   }
}
```

C# располагает большим числом встроенных функций для работы с массивами, например, сортировка элементов массива может быть выполнена путем вызова процедуры *Sort*.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Сортировка массива из 10 целых чисел.");
         int[] massiv = new int[10];
         for (int i = 0; i < 10; i++)
         {
            Console.Write("->");
            massiv[i] = Convert.ToInt32(Console.ReadLine());
         }
         Array.Sort(massiv);//сортировка элементов массива
         foreach (int i in massiv)
            Console.WriteLine(i);
         Console.ReadLine();
      }
   }
}
```

Задание к примеру

Модифицируйте программу на сортировку массива из строковых переменных, например, имен людей.

3.6. Функции

Следующая программа проверяет введенное число на четность при помощи функции *Chet*, которая возвращает значение логического типа.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static bool Chet(int n)//Функция проверки целого числа на четность
      {
         if (n % 2 == 0)//если n делится на 2 без остатка
            return true;//вернуть true
         else return false;//иначе вернуть false
      }
      static void Main(string[] args)//Главная функция программы
      {
         Console.WriteLine("Введите целое число.");
         Console.WriteLine("Для завершения введите 100.");
         int m;
         do
         {
            Console.Write("->");
            m = Convert.ToInt32(Console.ReadLine());
            if (Chet(m))//вызов функции
               Console.WriteLine("{0} - четное число", m);
            else
               Console.WriteLine("{0} - нечетное число", m);
         }
         while (m != 100);
      }
   }
}
```

Задание к примеру

Напишите и примените функцию, определяющую, является ли переданное ей число простым.

Описанная в следующем примере функция *cubrt* вычисляет кубический корень из переданного ей параметра типа *double* и возвращает значение этого же типа.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static double cubrt(double x)//приближенное значение кубического
корня
    {
      double pr = x * x;//первое приближение
      //в качестве второго приближения выбираем (x/pr)/pr
      while (Math.Abs(pr - x / (pr * pr)) > 0.001)
        pr=(2.0*pr+x/(pr*pr))/3.0;
      //новое приближение - среднее арифметическое
      //удвоенного приближения на прошлом шаге и текущего
      return pr;
    }
    static void Main(string[] args)
    {
      Console.WriteLine(cubrt(8));
      Console.ReadLine();
    }
  }
}
```

Задание к примеру

Модифицируйте программу на вычисление кубического корня введенного пользователем числа.

Функция, вычисляющая факториал числа, показана в примере ниже.

```
using System;
namespace ConsoleApplication1
{
  class Program
```

```
  {
    static double Factor(int n)//функция, вычисляющая факториал числа
    {
      int f=1;
      for (int i = 2; i <= n; i++)
        f *= i;//умножить f на i
      return f;
    }
    static void Main(string[] args)//главная функция программы
    {
    }
  }
}
```

Задание к примеру

Примените приведенную в примере функцию в программе.

Факториал может быть вычислен с использованием рекурсии:

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static int factorial(int k)//Рекурсивная функция "Факториал"
    {
      if (k==1)
        return 1;
      else return k*factorial(k-1);//функция вызывает саму себя
    }
    static void Main(string[] args)
    {
      Console.WriteLine("Вычисление факториала с использованием
рекурсивной функ-ции.");
      Console.Write("Введите число, факториал которого надо вычислить
->");
      int n,f;
      n=Convert.ToInt32(Console.ReadLine());
      f = factorial(n);
      Console.WriteLine("Факториал числа {0} равен {1}",n,f);
```

```
        Console.ReadLine();
      }
   }
}
```

Задание к примеру

Напишите рекурсивную функцию для расчета суммы ряда $1+1/2+1/3+...+1/n$, где n — параметр функции.

Если от функции требуется не вернуть некоторое значение, а изменить значения переданных ей переменных, то при описании переменных-параметров функции нужно указать ключевое слово *out*. Это же слово указывается при передаче параметров в функцию при ее вызове. Следующий пример поясняет передачу параметров-переменных в функцию.

```
using System;
namespace ConsoleApplication1
{
   class Program
   {
      static void SqLeOkr(double r, out double s, out double l)//s и l – выходные
переменные
      {
         s = Math.PI * r * r;
         l = 2 * Math.PI * r;
      }
      static void Main(string[] args)
      {
         Console.WriteLine("Вычисление длины окружности и площади круга.");
         Console.Write("Задайте радиус: ");
         double l, s, r = Convert.ToDouble(Console.ReadLine());
         SqLeOkr(r, out s, out l);//передали переменные s и l для записи в них
новых значений
         Console.WriteLine("Длина: {0}   площадь: {1}", Math.Round(l, 2), Math.
Round(s, 2));
         Console.ReadLine();
      }
   }
}
```

Задание к примеру

Напишите и примените функцию для расчета площади и периметра прямоугольника.

Вместо ключевого слова *out* при описании переменных-параметров функции может использоваться слово *ref*, однако в этом случае перед вызовом функции необходимо присвоить передаваемым переменным начальные значения, как это сделано в следующем примере.

```
using System;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("*Решение квадратного уравнения*");
      double a,b,c;
      Console.Write("Введите значение коэффициента a: ");
      a = Convert.ToDouble(Console.ReadLine());
      Console.Write("Введите значение коэффициента b: ");
      b = Convert.ToDouble(Console.ReadLine());
      Console.Write("Введите значение коэффициента c: ");
      c = Convert.ToDouble(Console.ReadLine());
      double x1=0,x2=0;
      if (korni(a, b, c, ref x1, ref x2))
        Console.WriteLine("x1={0}   x2={1}", Math.Round(x1, 3), Math.
Round(x2, 3));
      else Console.WriteLine("Корней нет");
      Console.ReadLine();
    }

    static bool korni(double a, double b, double c, ref double x1, ref double x2)
    {
      double d = b * b - 4 * a * c;
      if (d >= 0)
      {
        x1 = (-b + Math.Sqrt(d)) / (2 * a);
```

```
            x2 = (-b - Math.Sqrt(d)) / (2 * a);
            return true;
        }
        else return false;
    }
  }
}
```

Задание к примеру

Модифицируйте пример на расчет корней биквадратного уравнения.

3.7. Файлы

Для работы с текстовыми файлами в C# могут использоваться классы *StreamReader* и *StreamWriter*. Пример программы, записывающей в файл результаты работы, показан ниже.

```
using System;
using System.IO;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Пересчет расстояния: Мили -> Километры.");
      Console.Write("Задайте расстояние в милях: ");
      string m=Console.ReadLine();
      double k = Convert.ToDouble(m)*1.6;
      StreamWriter f = new StreamWriter("D:\\rezult.txt");//объект для
вывода в файл
      f.WriteLine("{0} миль это {1} км.",m,k);//вывод в файл
      f.Close();//закрыть файл
      Console.WriteLine("Результаты работы программы в файле
D:\rezult.txt");
```

```
        Console.ReadLine();
      }
   }
}
```

Задания к примеру

1. Убедитесь в том, что после работы программы появился файл результатов; ознакомьтесь с ним.
2. Запустите программу с новыми данными, убедитесь в том, что содержимое файла изменилось.
3. Выведите в файл таблицу значений синуса от 0 до 2π с шагом $\pi/6$.

Если при создании экземпляра класса *StreamWriter* в качестве второго параметра указать логическое значение TRUE, то файл будет открыт для добавления данных, а не перезаписан.

```
using System;
using System.IO;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Пересчет расстояния: Мили -> Километры.");
         Console.Write("Задайте расстояние в милях: ");
         string m=Console.ReadLine();
         double k = Convert.ToDouble(m)*1.6;
         StreamWriter f = new StreamWriter("D:\\rezult.txt",true);//добавление
данных в файл
         f.WriteLine("{0} миль это {1} км.",m,k);
         f.Close();
         Console.WriteLine("Результаты работы программы в файле D:\rezult.
txt");
         Console.ReadLine();
      }
   }
}
```

Задания к примеру

1. Убедитесь в том, что после работы программы появился файл результатов; ознакомьтесь с ним.
2. Запустите программу с новыми данными, убедитесь в том, что содержимое файла изменилось.

Далее приведен пример чтения данных из файла.

```
using System;
using System.IO;
namespace ConsoleApplication1
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Чтение чисел из файла и вычисление их суммы");
      StreamReader f = new StreamReader("D:\\data.txt");//объект для
чтения данных из файла
      string s;
      int sum=0;
      while ((s = f.ReadLine()) != null) //пока прочитанная строка не пустая
      {
        sum += Convert.ToInt32(s);//добавлять прочитанное число к sum
        Console.WriteLine(s);
      }
      Console.WriteLine("Сумма чисел: {0}", sum);
      f.Close();
      Console.ReadLine();
    }
  }
}
```

Задания к примеру

1. Скопируйте файл data.txt в корневой каталог диска D.
2. Запустите программу и убедитесь в ее работоспособности.
3. Измените программу таким образом, чтобы она вычисляла 2 суммы: сумму положительных и сумму отрицательных чисел.

Если при работе с файлами требуется контролировать ошибки ввода-вывода, то может использоваться конструкция *try ... catch*, показанная ниже.

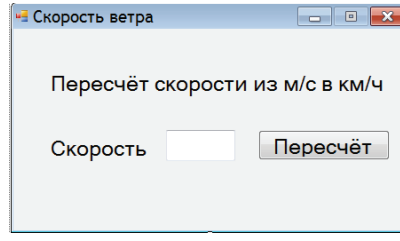
```
using System;
using System.IO;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Чтение чисел из файла и вычисление их суммы");
         try //попытаться выполнить блок инструкций
         {
            using (StreamReader f = new StreamReader("D:\\newdata.txt"))
            {
               string s;
               int sum = 0;
               while ((s = f.ReadLine()) != null)
               {
                  sum += Convert.ToInt32(s);
                  Console.WriteLine(s);
               }
               f.Close();
               Console.WriteLine("Сумма чисел: {0}", sum);
            }
         }
         catch (Exception e)// в случае ошибки выполнить следующий блок
инструкций
         {
            Console.Write("Ошибка: ");
            Console.WriteLine(e.Message);
         }
         Console.ReadLine();
      }
   }
}
```

Задания к примеру

1. Запустите программу.
2. Скопируйте файл newdata.txt в корневой каталог диска D.
3. Запустите программу и убедитесь в ее работоспособности.
4. Измените программу таким образом, чтобы она вычисляла сумму чисел, записанных через пробел в одну строку (см. п. 3.4).

3.8. Приложения с графическим интерфейсом

При разработке приложения с графическим интерфейсом программист должен скомпоновать окно (форму) программы, разместив в нем необходимые элементы (кнопки, поля ввода данных, изображения, надписи, ...), и написать код для обработки требуемых событий (нажатие кнопок, ввод текста и др.). Пример окна программы, выполняющей пересчет скорости из метров в секунду в километры в час, показан ниже:



В окне размещены следующие элементы: надписи (*Label*), поле для ввода текста (*TextBox*) и кнопка (*Button*).

Для создания процедуры обработки события «нажатие кнопки Пересчет» нужно, находясь в режиме разработки, дважды

щелкнуть по этой кнопке. В появившемся окне текстового редактора необходимо написать код обработчика события:

```
private void button1_Click(object sender, EventArgs e)
{
        double ms = Convert.ToDouble(textBox1.Text);//метры в секунду
        double kmh = ms * 3.6;//пересчет в км/ч
        label3.Text = Convert.ToString(ms) + "м/с - это " + Convert.
ToString(kmh) + "км/ч";
}
```

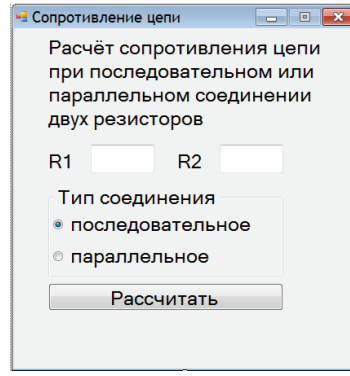
Для одного и того же элемента может быть определено несколько разных событий. Чтобы увидеть список возможных событий, нужно:

1) открыть конструктор формы;
2) открыть окно свойств, щелкнув по кнопке СВОЙСТВА в окне ОБОЗРЕВАТЕЛЬ РЕШЕНИЙ;
3) выбрать необходимый элемент на форме;
4) нажать кнопку СОБЫТИЯ в окне СВОЙСТВА.

Задания к примеру

1. Убедитесь в правильности работы программы.
2. Удалите кнопку ПЕРЕСЧЕТ, сохранив функциональность программы, т. е. реализуйте пересчет сразу при вводе данных. Для этого необходимо создать обработчик события *TextChanged* для элемента ввода *textBox1* и поместить в него код из обработчика события «нажатие кнопки Пересчет».

В следующем примере окно программы содержит такие элементы: надписи (*Label*), поля для ввода текста (*TextBox*), группу (*GroupBox*), элементы выбора (*RadioButton*) и кнопку (*Button*).



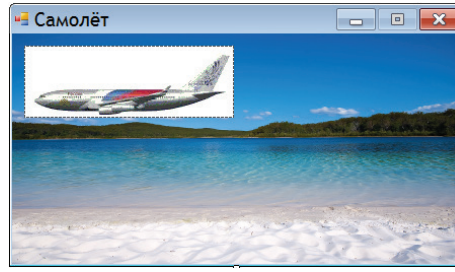
Обработчик события *button1_Click* имеет следующий код:

```
private void button1_Click(object sender, EventArgs e)
{
        double r,r1 = Convert.ToDouble(textBox1.Text),r2 = Convert.
ToDouble(textBox2.Text);
        if (radioButton1.Checked)
           r = r1 + r2; //выбрано последовательное соединение
        else
           r = (r1 * r2) / (r1 + r2);//иначе параллельное
        label5.Text = "Сопротивление цепи " + Convert.ToString(Math.
Round(r,3)) + " Ом";
}
```

Задание к примеру

Удалите кнопку РАССЧИТАТЬ, сохранив функциональность программы, т. е. реализуйте расчет при вводе данных или изменении типа соединения.

В следующем примере в окне программы размещены изображение самолета (*PictureBox*), таймер (*Timer*), а в качестве фона окна выбрана фотография пляжа (*BackgroundImage*):



При создании окна выполняется настройка параметров элемента *PictureBox*:

```
public Form1()
{
        InitializeComponent();
        Bitmap myBitmap = new Bitmap(pictureBox1.Image);//создать новое
изображение
        myBitmap.MakeTransparent();//сделать его прозрачным
        pictureBox1.Image = myBitmap;//установить новое изображение
в pictureBox1
        pictureBox1.Left = -300;//переместить изображение
        pictureBox1.Top = 100;
}
```

В свойствах таймера указаны параметры *Enabled = True* и *Interval = 1* — таймер включен, а интервал между таймерными событиями равен 1 мс. Обработчик таймерного события содержит такой код:

```
private void timer1_Tick(object sender, EventArgs e)
{
        if (pictureBox1.Left > -300)
          pictureBox1.Left--;//уменьшать координату X самолета, пока она
больше -300
        else //если самолет вышел за границы экрана
        {
          pictureBox1.Left = Width;//координата X - это правый край окна
```

```
            Random rnd = new Random();//генератор случайных чисел
            pictureBox1.Top = rnd.Next(Height / 3);//координата Y
        }
}
```

Задания к примеру

1. Измените скорость движения самолета.
2. Измените траекторию движения самолета, изобразив взлет или посадку.

Окно следующего примера, так же как и в предыдущем случае, содержит элементы *Timer* и *PictureBox*, причем для последнего параметр выравнивание (*Dock*) выбран равным «Заполнить» (*Fill*). Таймер настроен на интервал 10 мс и находится в активном состоянии (*Enabled = true*).

Полный код приложения приведен ниже.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    Graphics g;//объект, позволяющий рисовать
    int x_old=0, x=0, y_old=220, y=0;//глобальные переменные

    public Form1()//создание окна
    {
      InitializeComponent();
      Bitmap bmp = new Bitmap(2000, 1100);//создание холста
      pictureBox1.Width = bmp.Width;
      pictureBox1.Height = bmp.Height;
```

```
        pictureBox1.Image = bmp;
        g = Graphics.FromImage(bmp);
    }

    private void timer1_Tick(object sender, EventArgs e)//таймерное событие
    {
        x++; y = 220 - (int)(200 * Math.Sin(x / 100.0));//вычислить координаты
        g.DrawLine(new Pen(Color.Red, 2), x_old, y_old, x, y);//нарисовать
прямую линию
        x_old = x; y_old = y;//запомнить координаты для рисования
следующей линии
        if (x > 800) timer1.Enabled = false;//отключить таймер, если график
нарисован
        pictureBox1.Invalidate();//обновить изображение
    }

    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        g.Dispose();//удалить графический объект при закрытии окна
    }
  }
}
```

Задание к примеру

Добавьте к графику синуса график косинуса.

В следующем примере в окне программы размещены элементы *Chart* и *Timer*. Элемент *Chart* содержит 2 объекта *Series*, для которых тип графика выбран равным *FastLine*.

Полный код приложения показан ниже.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
```

```
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    double x = 0, integral = 0;//глобальные переменные
    const double dt = 0.01;//глобальная константа

    public Form1()//создание окна
    {
      InitializeComponent();
      chart1.Series[0].Points.Clear();
      chart1.Series[1].Points.Clear();
      chart1.Series[0].Name = "Функция";
      chart1.Series[1].Name = "Интеграл функции";
      chart1.ChartAreas.First().AxisX.LabelStyle.Format = "{0.00}";
    }

    private void timer1_Tick(object sender, EventArgs e)//таймерное событие
    {
      double s = Math.Cos(x);
      integral += s * dt;//интегрирование переменной s
      x += dt;
      chart1.Series[0].Points.AddXY(x, s);//добавить точку на график
      chart1.Series[1].Points.AddXY(x, integral);
      timer1.Enabled = (x < 6.28);//отключить таймер, если график
нарисован
    }
  }
}
```

Задания к примеру

1. Замените выражение s = Math.Cos(x) на s = Math.Sin(x), проанализируйте результат.
2. Добавьте график производной функции по времени. Производную можно определить по формуле $\frac{dS}{dt} \approx \frac{\Delta S}{\Delta t}$, где ΔS — изменение переменной S, а Δt — изменение переменной t.

3.9. Библиотеки динамической компоновки

Библиотека динамической компоновки, или динамически подключаемая библиотека, — это файл, содержащий в себе исполняемый код (функции) и ресурсы, которые могут вызываться и использоваться другими приложениями или библиотеками. Файл динамически подключаемой библиотеки имеет расширение DLL (*Dynamic Link Library*) и не может быть запущен на исполнение напрямую, т. к. он содержит лишь набор функций и не содержит основной программы. Одна и та же библиотека может одновременно использоваться разными программами. Драйверы устройств часто имеют формат DLL.

Изначально предполагалось, что применение DLL позволит эффективно организовать память и дисковое пространство, используя только один экземпляр библиотечного модуля для различных приложений. Это было особенно важно для старых ПК с ограниченным объемом оперативной памяти. Также предполагалось улучшить эффективность разработок и использования системных средств за счет модульности. Замена DLL-программ с одной версии на другую должна была позволить независимо наращивать систему, не затрагивая приложений. Кроме того, одни и те же динамические библиотеки могли использоваться разнотипными приложениями, например, Microsoft Office, Microsoft Visual Studio.

Однако всех преимуществ от внедрения динамически подключаемых библиотек получить не удалось из-за явления, названного DLL hell («DLL-ад»). DLL hell возникает, когда несколько приложений требуют одновременно различные, не полностью совместимые версии одних и тех же библиотек, что приводит к сбоям в этих приложениях, резко снижая общую надежность операционных систем. Поздние версии Microsoft Windows разрешают параллельное использование разных версий DLL (технология Side-by-side assembly), но это свело на нет преимущества изначального принципа модульности.

Для создания динамически подключаемой библиотеки в среде C#, необходимо выполнить следующие действия:

1) выбрать пункт меню ФАЙЛ → СОЗДАТЬ ПРОЕКТ → БИБЛИОТЕКА КЛАССОВ;
2) в появившемся шаблоне описать функцию, например, для проверки числа на четность

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ClassLibrary1
{
  public class Class1
  {
    public static bool chet(int i)
    {
      if (i % 2 == 0) return true;
      else return false;
    }
  }
}
```

3) сохранить проект и создать DLL-файл (пункт меню ПОСТРОЕНИЕ → ПОСТРОИТЬ РЕШЕНИЕ).

После этого в подпапке\Bin\Release проекта библиотеки появится файл с расширением DLL, содержащий описанную функцию. Для использования библиотеки в программе нужно:

1) создать в среде C# приложение — ФАЙЛ → СОЗДАТЬ ПРОЕКТ → КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ;
2) щелкнуть правой кнопкой мыши на пункте ССЫЛКИ в окне ОБОЗРЕВАТЕЛЬ РЕШЕНИЙ и выбрать ДОБАВИТЬ ССЫЛКУ... В появившемся окне во вкладке ОБЗОР открыть папку с созданной библиотекой (...\ClassLibrary1\ClassLibrary1\Bin\Release) и выбрать файл библиотеки ClassLibrary1.dll;

3) в процедуру Main шаблона программы дописать код, использующий функцию библиотеки:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine(ClassLibrary1.Class1.chet(2));//вызов функции chet
         Console.ReadKey();
      }
   }
}
```

Задание к примеру

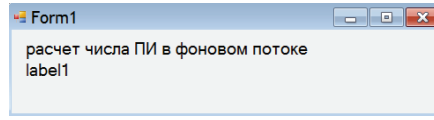
Модифицируйте программу таким образом, чтобы она запрашивала у пользователя число и выводила на экран надпись: «*X* — четное число» или «*X* — нечетное число», где *X* — введенное пользователем число.

3.10. Многопоточные приложения

Поток выполнения — это последовательность инструкций, описанных в программе и выполняемых друг за другом. Программа может создавать один поток, тогда она называется однопоточной; если программа создает несколько потоков, то она — многопоточная. Несколько потоков могут выполняться параллельно (при достаточном количестве ядер процессора) или вперемежку (если вычислительных ресурсов ПК недоста-

точно) — некоторое время выполняется один поток, потом он приостанавливается и выполняется другой поток и т. д. В последнем случае на скорость выполнения каждого потока влияет его приоритет: чем он выше, тем больше процессорного времени предоставляется потоку. Использование многопоточных приложений способно существенно повысить производительность вычислений, выполняемых на ПК с многоядерными процессорами. Однако разработка таких приложений оказывается сложнее однопоточных, а для некоторых задач не имеет смысла.

В следующем примере рассматривается многопоточное приложение, создающее основной и фоновый потоки выполнения. Фоновый поток рассчитывает число π, используя итерационный алгоритм. Основной поток отображает данные этого расчета. Окно приложения содержит две надписи (*Label*), таймер (*Timer*) и компонент для организации фонового потока (*backgroundWorker*):



Ниже представлен код программы.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
      static decimal valuePi;//переменная для хранения числа ПИ
```

```
    static ulong iter;//количество итераций
    public Form1()
    {
      InitializeComponent();
      backgroundWorker1.RunWorkerAsync();
    }
    private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
    { //фоновый поток, расчет числа ПИ
      decimal elem = 1, pi = 0;
      while (iter < 18446744073700000000)
      {
        elem = 1m / (2 * iter - 1);//1m - это единица типа decimal
        if ((iter % 2) == 0)
          pi -= elem;
        else pi += elem;
        valuePi = pi * 4;
        iter++;
      }
    }
    private void timer1_Tick(object sender, EventArgs e)//обработчик таймера
    {
      label1.Text = "Значение ПИ: " + valuePi.ToString() + "\nКол-во
итераций: " + Con-vert.ToString(iter/1000000)+" млн.";
    }
  }
}
```

Задание к примеру

Модифицируйте программу на расчет суммы бесконечного ряда 1–1/2 + 1/3–1/4 + ...

Фоновый поток назван так из-за того, что он имеет низший приоритет — он выполняется, если процессор свободен от задач с более высоким приоритетом. В следующем примере рассматривается способ создания дополнительного потока при помощи класса *Thread*.

```
using System;
using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    static decimal valuePi;
    static ulong iter;
    Thread pi;//объект для создания потока
    public Form1()
    {
      InitializeComponent();
      pi = new Thread(calc_pi);//создание дополнительного потока
      pi.Start();//в доп. потоке будет выполняться процедура calc_pi
    }
    private void calc_pi()//расчет числа ПИ
    {
      decimal elem = 1, pi = 0;
      while (iter < 18446744073700000000)
      {
        elem = 1m / (2 * iter - 1);
        if ((iter % 2) == 0)
          pi -= elem;
        else pi += elem;
        valuePi = pi * 4;
        iter++;
      }
    }
    private void timer1_Tick(object sender, EventArgs e)//отображение данных
доп. потока
    {
      label1.Text = "Значение ПИ: " + valuePi.ToString() + "\nКол-во
итераций: " + Con-vert.ToString(iter/1000000)+" млн.";
    }
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
```

```
            pi.Abort();//завершение дополнительного потока
        }
    }
}
```

Задания к примеру

1. Добавьте в программу второй поток, рассчитывающий сумму бесконечного ряда 1–1/2 + 1/3–1/4 + ...
2. Результат расчета отобразите аналогично результату работы первого потока.

3.11. Обмен данными между приложениями

Существует несколько технологий обмена данными между приложениями, например, динамический обмен данными (DDE), каналы (Pipes), сокеты (Sockets), общая память и др. Далее рассматриваются варианты организации обмена данными между приложениями, построенные по принципу «клиент — сервер», который заключается в том, что одно из приложений (сервер) ожидает запроса от другого приложения (клиента) и в ответ на полученный запрос пересылает клиенту данные.

В следующем примере окно приложения-сервера содержит компоненты pictureBox и backgroundWorker, используемые для отображения графики и организации локального обмена данными с приложением-клиентом посредством именованного канала (Named Pipe). Ниже показан код программы.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Pipes;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    Graphics g;
    Pen p = new Pen(Color.Red, 2);
    NamedPipeServerStream server = new NamedPipeServerStream("Test
Pipe", PipeDirec-tion.InOut, 1, PipeTransmissionMode.Byte, PipeOptions.
Asynchronous, 8, 8);//создание сервера
    public Form1()
    {
      InitializeComponent();
      Bitmap bmp = new Bitmap(2000, 1100);
      pictureBox1.Width = bmp.Width;
      pictureBox1.Height = bmp.Height;
      pictureBox1.Image = bmp;
      g = Graphics.FromImage(pictureBox1.Image);
      backgroundWorker1.RunWorkerAsync();//запуск фонового потока для
обмена данными
    }
      private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
      p.Dispose();
      g.Dispose();
    }
    private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
    {
      server.WaitForConnection();//ожидание подключения
      byte[] buf=new byte[8];//буфер для чтения
      while (server.IsConnected)
      {
        server.Read(buf, 0, 8);//считать данные в буфер
        int x1 = buf[0] * 256 + buf[1];//вычислить координаты точек
        int y1 = buf[2] * 256 + buf[3];
        int x2 = buf[4] * 256 + buf[5];
        int y2 = buf[6] * 256 + buf[7];
        g.DrawLine(p, x1, y1, x2, y2);//нарисовать линию
```

```
            pictureBox1.Invalidate();//обновить картинку
        }
    }
  }
}
```

Задачей сервера в этом примере является прием координат двух точек (начало и конец отрезка) и рисование прямой линии между этими точками. Задача приложения-клиента — рисовать линии, следя за координатами и кнопками мыши, и отправлять данные о координатах серверу. Окно приложения-клиента кроме компонентов pictureBox и backgroundWorker содержит кнопку ОЧИСТИТЬ, при нажатии которой выполняется очистка рисунка. Текст программы-клиента представлен ниже.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Pipes;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    Graphics g;
    Pen p = new Pen(Color.Red, 2);
    bool mouseDown = false;//переменная для контроля состояния кнопки
мыши
    int x1, y1;
    NamedPipeClientStream client = new NamedPipeClientStream(".",
"TestPipe", PipeDirec-tion.InOut, PipeOptions.Asynchronous);//создание
клиента
    public Form1()
    {
      InitializeComponent();
```

```
        Bitmap bmp = new Bitmap(2000, 1100);
        pictureBox1.Width = bmp.Width;
        pictureBox1.Height = bmp.Height;
        pictureBox1.Image = bmp;
        g = Graphics.FromImage(pictureBox1.Image);
        backgroundWorker1.RunWorkerAsync();//запуск фонового потока для
обмена данными
    }
    private void Form1_FormClosed(object sender, FormClosedEventArgs e)
    {
        p.Dispose();
        g.Dispose();
    }
    private void pictureBox1_MouseDown(object sender, MouseEventArgs e)
    {
        mouseDown = true;//кнопка мыши нажата
    }
    private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
    {
        mouseDown = false; //кнопка мыши отпущена
    }
    private void pictureBox1_MouseMove(object sender, MouseEventArgs e)
    {
        if (mouseDown)//если кнопка мыши нажата,
        {
            g.DrawLine(p, x1, y1, e.X, e.Y);//то рисуем линию
            pictureBox1.Invalidate();
            if (client.IsConnected)
            {
                byte[] buf = new byte[8];//подготовка данных для отправки на
сервер
                buf[0] = (byte)(x1 / 255);
                buf[1] = (byte)(x1 % 255);
                buf[2] = (byte)(y1 / 255);
                buf[3] = (byte)(y1 % 255);
                buf[4] = (byte)(e.X / 255);
                buf[5] = (byte)(e.X % 255);
                buf[6] = (byte)(e.Y / 255);
                buf[7] = (byte)(e.Y % 255);
                client.Write(buf, 0, 8);//отправить данные
            }
```

```
        }
        x1 = e.X;//запомнить координаты для рисования следующего отрезка
        y1 = e.Y;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        g.Clear(pictureBox1.BackColor);//очистка рисунка
        pictureBox1.Invalidate();
    }
    private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
    {
        client.Connect();//подключение к серверу
    }

  }
}
```

Задания к примеру

1. Измените программы таким образом, чтобы при нажатии кнопки ОЧИСТИТЬ в обеих программах выполнялась очистка экрана. Реализовать это можно, добавив в клиенте отправку серверу сообщения об очистке, например, вместо координат точек можно отправить числа, превышающие возможные значения координат. В приложении-сервере необходимо добавить проверку принятых координат и команду очистки окна, которая будет вызываться, если координаты превышают некоторое значение.

2. Измените программы таким образом, чтобы рисовать мышью можно было и в программе-сервере, при этом рисунок должен повторяться в программе-клиенте. Для этого в программу-сервер необходимо добавить контроль состояния кнопки мыши (события MouseDown и MouseUp), построение линий при перемещении мыши (MouseMove) и отправку данных с координатами для рисования клиенту. В приложении-клиенте в процедуре фонового потока, после подключения к серверу, нужно добавить чтение данных от сервера и построение по этим данным линий.

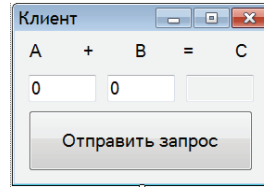
В следующем примере рассмотрен сетевой обмен данными с использованием сокета (*Socket*). Сервер выполнен в виде консольного приложения, принимающего два числа, отображающего их на экране и отправляющего в качестве ответа клиенту сумму этих чисел.

Код сервера приведен ниже.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Net;
using System.Net.Sockets;
using System.IO;
namespace ConsoleApplication1
{
   class Program
   {
      static void Main(string[] args)
      {
         //создаем объект типа Socket
         Socket server = new Socket(AddressFamily.InterNetwork, SocketType.
Stream, Proto-colType.Tcp);
         //задаем область видимости сервера (локальный ПК или сеть) и порт
         server.Bind(new IPEndPoint(IPAddress.Any, 5000));
         //переключаем сервер в режим ожидания подключений
         server.Listen(10);//очередь до 10 подключений
         while (true)
         {
            Socket link = server.Accept();//приняли запрос на подключение
            //объект типа Socket передает только байты
            //для передачи переменных других типов нужно подключить
к нему поток вво-да/вывода
            NetworkStream data=new NetworkStream(link);
            //подключаем к потоку объект, способный читать данные
стандартных типов
            BinaryReader reader = new BinaryReader(data);
            double a = reader.ReadDouble();//прочитали переменную a
            double b = reader.ReadDouble();
            Console.WriteLine("{0}: {1}, {2}", link.RemoteEndPoint, a, b);
```

```
            //подключаем к потоку объект, способный выводить данные
стандартных типов
            BinaryWriter writer = new BinaryWriter(data);
            writer.Write(a + b);//вывели сумму
            data.Dispose();//освободили ресурсы
            link.Close();//закрыли соединение
        }
    }
  }
}
```

Окно приложения-клиента содержит поля ввода, текстовые надписи и кнопку:



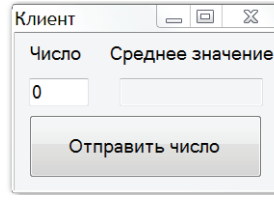
Ниже показан код приложения-клиента.

```
using System;
using System.Windows.Forms;
using System.Net;
using System.Net.Sockets;
using System.IO;
namespace WindowsFormsApplication1
{
  public partial class Form1 : Form
  {
    public Form1()
    {
      InitializeComponent();
    }
    private void button1_Click(object sender, EventArgs e)
    {
      try//попытаться выполнить следующий блок инструкций
      {//это защита от вылета программы при ошибках преобразования
```

```
типов/связи
            double a = Convert.ToDouble(textBox1.Text);
            double b = Convert.ToDouble(textBox2.Text);
            //создаем объект типа Socket
            Socket client = new Socket(AddressFamily.InterNetwork, SocketType.
Stream, Proto-colType.Tcp);
            //пытаемся подключиться к указанному компьютеру
            client.Connect("E200B_ST1", 5000);
            //объект типа Socket передает только байты
            //для передачи переменных других типов нужно подключить к
нему поток вво-да/вывода
            NetworkStream data = new NetworkStream(client);//создали поток
            //подключаем к потоку объект, способный выводить данные
стандартных типов
            BinaryWriter writer = new BinaryWriter(data);
            writer.Write(a);//вывели значение переменной a
            writer.Write(b);
            //подключаем к потоку объект, способный читать данные
стандартных типов
            BinaryReader reader = new BinaryReader(data);
           double x = reader.ReadDouble();//прочитали из потока переменную x
            textBox3.Text = x.ToString();//вывели результат
            data.Dispose();//освободили ресурсы
            client.Close();//закрыли соединение
        }
        catch//перехватываем ошибку
        {
            textBox3.Text = "ошибка";
        }
    }
  }
}
```

Задания к примеру

1. Измените внешний вид приложения-клиента, чтобы оно выглядело так:



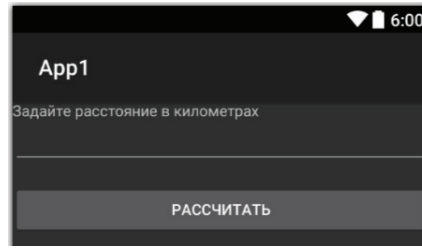
2. Измените код клиента таким образом, чтобы он отправлял серверу одно число, принимал и отображал ответ.
3. Измените сервер таким образом, чтобы он вычислял среднее значение всех полученных от клиентов чисел и отправлял его в качестве ответа.

3.12. Разработка Android-приложений

Среда Visual Studio позволяет разрабатывать приложения для операционных систем Android и iOS. Далее рассмотрено создание Android-приложения, пересчитывающего расстояние из километров в мили. Чтобы создать это приложение, необходимо запустить среду разработки Visual Studio и в главном окне выбрать пункт СОЗДАТЬ ПРОЕКТ. В предлагаемых шаблонах нужно выбрать VISUAL C# → ANDROID → BLANK APP. После этого среда создаст простейшее приложение. Его можно запустить, используя эмулятор устройства с операционной системой Android. В меню главного окна выберите пункт СЕРВИС → ANDROID → ANDROID EMULATOR MANAGER. В появившемся окне выберите или создайте эмулятор с нужными параметрами (разрешение и размер экрана, версия ОС) и нажмите кнопку START. Подтвердите запуск, нажав кнопку LAUNCH. Дождитесь загрузки операционной системы в эмуляторе и запустите созданную программу из среды Visual Studio, нажав комбинацию клавиш CTRL+F5. Разблокируйте окно эмулятора, нажав кнопку мыши на изображении закрытого замка и проведя мышью в сторону открытого замка. После компиляции и загрузки, приложение запускается автоматически.

Для модификации приложения необходимо проделать следующее. В окне ОБОЗРЕВАТЕЛЬ РЕШЕНИЙ откройте файл RESOURCES → LAYOUT → MAIN.AXML. Удалите существующую кнопку и добавь-

те из ПАНЕЛИ ЭЛЕМЕНТОВ новые компоненты в следующем порядке: TEXTVIEW, NUMBER, TEXTVIEW, BUTTON. Дважды щелкните по первому элементу TEXTVIEW в окне приложения и вместо надписи «Text» введите «Задайте расстояние в километрах». Во втором элементе TEXTVIEW удалите надпись «Text». Надпись «BUTTON» на кнопке замените на «РАССЧИТАТЬ». Окно должно выглядеть следующим образом:



Если нажать на кнопку SOURCE, то можно увидеть код разметки окна, который должен выглядеть так:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
  <TextView
    android:text="Задайте расстояние в километрах"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/textView1" />
  <EditText
    android:inputType="number"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/editText1" />
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
```

```
    android:id="@+id/textView2" />
  <Button
    android:text="Рассчитать"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/button1" />
</LinearLayout>
```

В окне ОБОЗРЕВАТЕЛЬ РЕШЕНИЙ щелкните по файлу MAINACTIVITY.CS и замените существующий код на следующий:

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
namespace App1
{
  [Activity(Label = "App1", MainLauncher = true, Icon = "@drawable/icon")]
  public class MainActivity : Activity
  {
    protected override void OnCreate(Bundle bundle)
    {
      base.OnCreate(bundle);
      SetContentView(Resource.Layout.Main);
      Button button = FindViewById<Button>(Resource.Id.button1);
      button.Click += Button_Click;
    }
    private void Button_Click(object sender, EventArgs e)
    {
      EditText km = FindViewById<EditText>(Resource.Id.editText1);//поле
ввода
      double mil = Convert.ToDouble(km.Text) / 1.6;//пересчитать км в мили
      TextView otvet = FindViewById<TextView>(Resource.Id.textView2);//
поле ответа
      otvet.Text = String.Format("{0} километров - это {1} миль", km.Text, mil);
    }
  }
}
```

Запустите программу в эмуляторе и проверьте ее работоспособность.

Задание к примеру

Удалите кнопку РАССЧИТАТЬ и реализуйте автоматический пересчет при вводе данных.

Библиографический список

Иванова Г. С. Объектно-ориентированное программирование : учебник для студентов вузов / Г. С. Иванова, Т. Н. Ничушкина. Москва : МГТУ им. Н. Э. Баумана, 2014. 455 с.

Культин Н. Б. Программирование в Turbo Pascal 7.0 и Delphi / Н. Б. Культин. 3-е изд., перераб. и доп. Санкт-Петербург : БХВ-Петербург, 2007. 400 с.

Николаев Е. И. Объектно-ориентированное программирование : учебное пособие / Е. И. Николаев. Ставрополь : СКФУ, 2015. 225 с.

Павловская Т. А. C/C++. Процедурное и объектно-ориентированное программирование : учебник для студентов вузов / Т. А. Павловская. Москва ; Санкт-Петербург ; Нижний Новгород : Питер, 2015. 496 с.

Павловская Т. А. C#. Программирование на языке высокого уровня : учебник для вузов / Т. А. Павловская. Санкт-Петербург : Питер, 2012. 432 с.

Рацеев С. М. Программирование на языке Си : учебное пособие / С. М. Рацеев. Ульяновск : УлГУ, 2015. 318 с.

Оглавление

Учебное издание

Зюзев Анатолий Михайлович
Нестеров Константин Евгеньевич

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Редактор И. В. Меркурьева
Верстка О. П. Игнатьевой

Подписано в печать 29.06.2019. Формат 60×84/16.
Бумага офсетная. Цифровая печать. Усл. печ. л. 6,7.
Уч.-изд. л. 4,1. Тираж 40 экз. Заказ 167

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: +7 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620083, Екатеринбург, ул. Тургенева, 4
Тел.: +7 (343) 358-93-06, 350-58-20, 350-90-13
Факс: +7 (343) 358-93-06
http://print.urfu.ru


ПУЛЬТ УНИВЕРСИТЕТСКОГО КНИГОИЗДАНИЯ
УНИВЕРСИТЕТСКАЯ КНИГА
2018


9 785799 626341