1. Элементы языка Object Pascal.

Язык программирования Object Pascal является последней версией семейства языков Pascal, реализующей принципы объектно-ориентированного программирования. Этот язык является основой системы визуального программирования Delphi. Наиболее существенным отличием от традиционного языка Pascal является наличие достаточно сложных структур данных (классы) и возможность средствами Pascal обращаться к функциям Windows API для создания полноценных Windows- приложений.

Object Pascal позволяет использовать множество самых разнообразных типов и структур данных. Все типы данных можно разбить на две группы: *простые* (*базовые*) и *структурированные* (*пользовательские*) типы, которые создаются на основе базовых и объединяют несколько переменных разных типов в одной структуре данных.

1.1. Базовые (простые) типы.

Любые переменные, используемые в программе, должны быть описаны в разделах описаний программы, соответствующих процедур, функций или библиотечных модулей. При описании указывается имя переменной и ее тип. Тип данных определяет:

1) множество допустимых значений, которые может принимать переменная;
2) набор допустимых операций над этой переменной;
3) формат внутреннего представления данных в оперативной памяти, в частности размер памяти, отводимый под хранение переменной.

Основные группы, относящиеся к базовым типам, перечислены в таблице 1.1. Объявление переменных простых типов производится непосредственно в разделе описания переменных с использованием стандартных идентификаторов.

К простым переменным часто относят и переменные строкового типа (строка символов). С точки зрения структуры данных строковые типы являются массивами символов, т.е. структурированными типами. Однако, учитывая широкое использование строк в программировании, их отнесли к базовым типам, введя для них стандартные идентификаторы.

Примеры объявления переменных базовых типов:

```
var
  I, J : integer;
  Ch : char;
  X, Y, Z: real;
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

St : string;

Таблица 1.1.

Типы	Идентификатор	Размер, байт	Допустимые значения
Целые	**byte**	1, без знака	0 .. 255
	word	2, без знака	0 .. 65535
	integer	4, со знаком	–2147483648 .. 2147483647
Символьные	**char, AnsiChar**	1	**Chr(0)** .. **Chr(255)**
	WideChar	2	– " –
Логические	**boolean, ByteBool**	1	**true** (1), **false** (0)
	WordBool	2	**true** (<>0), **false** (0)
Вещественные	**single**	4	$\pm 1.5 \cdot 10^{-45} .. \pm 3.4 \cdot 10^{38}$
	double, **real**	8	$\pm 5.0 \cdot 10^{-324} .. \pm 1.7 \cdot 10^{308}$
Строковые	**string, AnsiString**	От 4B до 2GB	Строка символов в кодировке ASCII (8 бит)
	WideString	От 4B до 2GB	Строка символов в кодировке UNICODE (16 бит)

1.2. Структурированные (пользовательские) типы.

Эти данные представляют собой структуры, состоящие из одного или нескольких базовых типов. При объявлении пользовательского типа ему необходимо присвоить имя. Разнообразные пользовательские типы данных широко используются в стандартных библиотеках Delphi. Ниже описываются некоторые из пользовательских типов, применение которых необходимо при выполнении заданий по курсу «Информатика».

Массивы.

Этот тип данных представляет собой однотипную совокупность элементов, упорядоченных по номерам. При решении задач, как правило, используются одномерные (с одним индексом), двумерные (с двумя индексами) и трехмерные (с тремя индексами) массивы. Массивы большей размерности на практике встречаются редко. В качестве индексов обычно используют целые числа. Ниже приведены примеры описания переменных

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

типа массив. Описание массивов производится с помощью зарезервированного слова **array.**

```
var
   Xx : array[1..100] of real;    {одномерный массив вещественных чисел}
   Charge  : array[1..5,1..5] of integer; {двумерный массив целых чисел}
   MyMessage : array[1..50] of string; {массив строк}
……………………………………………………………………
Xx[25]:=3.28e-5;
Charge[1,4,2]:=35;
MyMessage[43]:='Ошибка в вычислениях';
……………………………………………………………………
```

Записи.

Переменные этого типа представляют собой совокупность данных разных типов. Фиксированные записи состоят из нескольких *полей*. *Поле* – это переменная любого типа. Ниже приведены примеры описания этого типа. Описание записей производится с помощью зарезервированного слова **record.**

```
type
   TCoord = record
       X,Y : real;    {поля записи}
   end;                    {конец объявления записи}

   TParticle = record
       Charge: integer;
       Coord  : TCoord;
   end;
var
   Particle: TParticle ;
……………………………………….
Particle.Charge:=10;
Particle.Coord.X:=158.35;
with Particle do
   begin
       Coord.X:=Coord.X-56.5;
       Coord.Y:=88;
       Charge:=Charge*3;
   end;
………………………………………..
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Доступ к полям записи осуществляется с помощью так называемого *квалифицируемого* или *уточненного* идентификатора, который представляет собой имя переменной типа записи и имя поля, разделенные между собой точкой (выражения типа **Particle.Charge** или **Particle.Coord.X**). Кроме этого для доступа к полям записи можно использовать инструкцию присоединения **with**, как показано в примере.

Указательные типы.

Указатели представляют собой переменные, в которых записаны адреса тех ячеек памяти, в которых находятся данные. Обычно указатели применяют при работе с динамическими переменными, которые создаются в процессе выполнения программы и которые не имеют заранее определенного при компиляции адреса. Подробное описание этого типа выходит за рамки данного пособия.

Файлы.

Структура данных (*логический* файл), которая создается для последующей записи этих данных во внешнюю память (*физический* файл). Структура типа файл представляет собой набор однородных по типу данных и в этом смысле напоминает массив. В отличие от массива у файла нет заранее определенного количества элементов. Конец файла определяется специальным символом **Eof** (ASCII код – 26). Доступ к элементам файла для их считывания или записи осуществляется с помощью стандартных процедур языка программирования, предназначенных для работы с файлами.

Пример работы с файлами приведен ниже. Здесь рассмотрены два файловых типа – *текстовые* и *типизированные* файлы. Более сложные случаи работы с *нетипизированными* файлами и *файловыми потоками* в данном пособии не рассматриваются.

```
var
  F1: TextFile;   {текстовый файл}
  F2: file of real; {типизированный файл (файл вещественных чисел)}
  St: string;
  A,B,C : real;
……………………………………
AssignFile (F1,'TextFile.txt'); {связывание логического файла с физическим, имеющим  имя " TextFile.txt "}
ReWrite(F1);   {создать и открыть новый файл}
……………………………………………………………
St:= 'Создаем новый файл';
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
Write(F1,St); {запись в файл строковой переменной}
CloseFile(F1);{закрытие файла}
…………………………………………………….
AssignFile(F2,'MyFile.dat');
Reset(F2);   {открытие существующего файла}
…………………………………………………….
Read(F2,A,B,C);  {считывание из файла данных в переменные типа real}
CloseFile(F2);
…………………………………………………
```

1.3. Классы.

Переменные этого типа представляют собой сложную структуру, состоящую из *полей, методов* и *свойств*. Поля, методы и свойства класса называются его компонентами или членами. Классы описывают структуру динамической переменной, которая появляется только в процессе выполнения программы. Такая переменная является экземпляром класса или **объектом**. Объекты могут появляться и уничтожаться как динамически распределенные блоки памяти, структура которых задается типом их класса. Поэтому переменная типа класс фактически является *указателем* на объект. До тех пор, пока объект не создан, эта переменная имеет значение **nill**, означающее, что данный указатель ни на что не указывает.

Поле – это переменная произвольного типа, играющая ту же роль, что и поле записи. Поля класса представляют собой элементы данных, которые копируются в каждом экземпляре класса.

Метод – процедура или функция, связанная с классом. Большинство методов оперирует с компонентами объекта. Специальные методы, носящие название **Constructor** и **Destructor** используются для создания экземпляров класса и их уничтожения.

Свойство – это поля, но защищенные от непосредственного доступа. Прочитать значение свойства или изменить его можно только с помощью так называемых спецификаторов доступа, т. е. методов, связанных с данным свойством.

Важным признаком класса является *наследование*. Каждый новый класс является "потомком" какого-либо класса. При этом все поля, методы и свойства "родителя" переходят к "потомку", а кроме них добавляются новые. Таким образом, каждый новый класс включает в себя все свойства, поля и методы всех своих "предков". Все классы образуют иерархическую структуру, во главе которой находится абстрактный класс **TObject**, который является для них общим "предком". Благодаря наследованию нет необходимости при описании класса перечислять все компоненты "родителя". В описании класса указываются только новые компоненты и имя "родителя". Описание классов располагается *в начале* основной программы

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

или в интерфейсном разделе библиотечного модуля. Описание классов внутри процедур и функций не допускается. При описании класса его методы указываются только в виде *заголовков* процедур и функций. Сами же описания этих процедур и функций располагаются в другом месте программы или библиотечного модуля (в разделе *implementation*). Пример описания класса приведен ниже:

```
type
TForm1 = class(TForm) {имя класса TForm1 и "родителя" TForm }
    {список полей и их типов}
    Edit1: TEdit;
    Button1: TButton;
    ………………………………
    {список методов, только заголовки процедур и функций}
    procedure  Button1Click(Sender: TObject);
    procedure  Edit1Change(Sender: TObject);
    ……………………………….
Public
    X,Y: real;
    ………………………………..
end;        {конец описания класса}
var
    Form1: TForm1; {объявление переменной типа класс}
……………………………………………………..
```

Если имя родителя не указывается, это означает, что данный класс является прямым потомком **TObject**. При описании компонентов класса используются специальные *директивы*, обеспечивающие доступность данного компонента из различных *модулей* программы. Для обеспечения доступа к данному компоненту из любой части программы, его следует помещать в раздел, отмеченный директивой **public.**

Доступ к компонентам переменной типа класс возможен только тогда, когда создан экземпляр класса (объект), на который указывает данная переменная. Доступ при этом осуществляется, как и для записей с помощью квалифицируемого идентификатора или с помощью инструкции присоединения.

```
………………………
Form1. Button1Click (Sender);
with Form1 do
   begin
       X:=Width;
       Y:=Height;
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
   end;
………………………..
```

Если обращения к компонентам класса происходит из метода этого же класса, то квалифицируемые идентификаторы не используются, достаточно просто указать имя компонента.

1.4. Операторы и выражения.

Операторы – это один из видов предопределенных функций, встроенных в язык программирования. Они используются для построения и вычисления *выражений*. В выражения кроме операторов входят *операнды*, в качестве которых могут выступать переменные, константы и другие выражения. Оператор называется *унарным*, если он воздействует на один операнд, и *бинарным*, если операндов два. По типу операндов и типу возвращаемого результата операторы делятся на несколько групп.

Арифметические операторы.

Используются для построения арифметических выражений. В таблице 1.2 приведены характеристики арифметических операторов. Если один из операндов для первых четырех операторов имеет вещественный тип, то и результат будет вещественным.

Таблица 1.2.

Оператор	Действие	Тип операндов	Тип результата	Пример
+	Сложение	**integer, real**	**integer, real**	**X + Y**
-	Вычитание	**integer, real**	**integer, real**	**Result - 1**
*	Умножение	**integer, real**	**integer, real**	**P * IntRate**
/	Деление	**integer, real**	**real**	**X / 2**
div	Целочисленное деление	**integer**	**integer**	**Total div Usize**
mod	Остаток от деления	**integer**	**integer**	**Y mod 6**
+ (унарный)	Знак числа	**integer, real**	**integer, real**	**+7**
- (унарный)	Знак числа	**integer, real**	**integer, real**	**-X**

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Логические операторы.

Используются для построения и вычисления логических выражений. В таблице 1.3 приведены характеристики логических операторов.

Таблица 1.3.

Оператор	Действие	Тип операндов	Тип результата	Пример
not	Отрицание	**boolean**	**boolean**	**not C**
and	Логическое "И" (конъюнкция)	**boolean**	**boolean**	**Done and Total**
or	Логическое "ИЛИ" (дизъюнкция)	**boolean**	**boolean**	**A or B**
xor	Исключающее "ИЛИ"	**boolean**	**boolean**	**A xor B**

В таблице 1.4 показаны результаты действия логических операторов (правила булевой алгебры).

Таблица 1.4.

Операнды		Операции			
A	**B**	**not A**	**A and B**	**A or B**	**A xor B**
false	**false**	**true**	**false**	**false**	**false**
false	**true**	**true**	**false**	**true**	**true**
true	**false**	**false**	**false**	**true**	**true**
true	**true**	**false**	**true**	**true**	**false**

Битовые (поразрядные) операторы.

Эти операторы применяются к целым числам (**integer**) и воздействуют на отдельные разряда в двоичном представлении числа (биты). Характеристики битовых операторов приведены в таблице 1.5.

Результат действия битовых операторов приведен в таблице 1.6. Для наглядности взяты операнды типа **byte**.

Поразрядный сдвиг влево на N соответствует *умножению* числа на 2^N, соответственно поразрядный сдвиг вправо на N соответствует *делению* числа на 2^N, при этом, как видно из таблицы "лишние" биты отбрасываются (результат округляется), а "освободившиеся" разряды заполняются нулями.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Таблица 1.5.

Оператор	Действие	Тип операндов	Тип результата	Пример
Not	Поразрядное отрицание	**integer**	**integer**	**not X**
And	Поразрядное "И"	**integer**	**integer**	**X and Y**
Or	Поразрядное "ИЛИ"	**integer**	**integer**	**X or Y**
Xor	Поразрядное исключающее "ИЛИ"	**integer**	**integer**	**X xor Y**
Shl	Поразрядный сдвиг влево	**integer**	**integer**	**X shl 2**
Shr	Поразрядный сдвиг вправо	**integer**	**integer**	**Y shr I**

Таблица 1.6.

	Десятичное представление	Двоичное представление
Операнд **A**	11	00001011
Операнд **B**	2	00000010
not A	244	11110100
A and B	2	00000010
A or B	11	00001011
A xor B	9	00001001
A shl 3	88	01011000
A shr 2	2	00000010

Строковый оператор.

Над строками определено только одно действие – *конкатенация*, или объединение строк. Оператором конкатенации служит символ +. В качестве операндов могут участвовать переменные и константы *символьного* и *строкового* типов (**char, string**), результатом будет строка. Пример конкатенации строк:

'Добро' + ' ' + **'пожаловать'**+ **'!'**

Результатом такой операции будет: **'Добро пожаловать!'**

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Операторы отношения.

Эти операторы используются для сравнения двух операндов. В результате такой операции формируется логическое выражение. Описание операторов отношения приведено в таблице 1.7. В таблице указаны только описанные в данном разделе пособия типы данных.

Таблица 1.7.

Оператор	Действие	Тип операндов	Тип результата	Пример
=	Равенство	Простые, указательные, **string, class**	**boolean**	**I = Max**
<>	Неравенство	Простые, указательные, **string, class**	**boolean**	**X <> Y**
<	Меньше	Простые, **string**	**boolean**	**X < Y**
>	Больше	Простые, **string**	**boolean**	**Len > 0**
<=	Меньше или равно	Простые, **string**	**boolean**	**Cnt <= I**
>=	Больше или равно	Простые, **string**	**boolean**	**J >= 1**

Необходимо отметить, что в качестве операндов в выражении отношения могут стоять только совместимые типы, например, нельзя сравнить целое число и структуру типа **class**, в то время, как целые и вещественные числа, строки и символы можно сравнивать друг с другом.

1.5. Инструкции.

Основная часть программы (тело программы) состоит из последовательности инструкций, которые задают алгоритм программы. Инструкция определяет набор действий, который должен выполнить процессор и составляют основу любого языка программирования. Инструкции в Object Pascal делятся на две группы. Это *простые* и *составные* инструкции. Инструкции в программе, как правило, отделяются друг от друга символом (**;**).

Простые инструкции.

Эти инструкции не включают в себя другие инструкции. К ним относятся, в частности, инструкция *присвоения* и *вызовы процедур*.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

1) ***Инструкция присвоения***. Общий вид:

Переменная **:=** выражение**;**

С помощью этой инструкции текущее значение переменной изменяется на значение выражения. Символ "**:=**" называют *оператором присвоения*. При использовании этой инструкции необходимо следить за совместимостью типов слева и справа от оператора присвоения. Примеры использования инструкции присвоения:

X := Y + Z;
Done := (I >= 1) and (I < 100);
Form1.Timer1.Enabled:=true;

Если переменная **X** имеет тип **integer**, то следующее выражение будет ошибкой:

X:= Y/Z; {ошибка -несоответствие типов}

2) ***Вызовы процедур***. Для вызова процедуры необходимо указать имя процедуры и список параметров. Примеры инструкций вызова:

Refresh;
Form1.Button1Click(Sender);
Canvas.TextOut(X,Y,'График функции y=sin(x)');

Составные инструкции.

Этот тип инструкций состоит из нескольких простых инструкций. Здесь будут рассмотрены так называемый *составной оператор*, инструкция *присоединения*, *условные* инструкции и инструкции *цикла*.

1) *Составной оператор.* При необходимости можно несколько инструкций объединить с помощью т.н. операторных скобок (**begin … end**) в одну. Подобный прием используется в тех случаях, где по правилам языка можно использовать только одиночные инструкции. Примеры будут приведены ниже.

2) *Инструкция присоединения.* Инструкция присоединения используется для *доступа* к полям записей, а также полям, свойствам и методам объектов. Общий вид:

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
with  переменная  do
   begin
       инструкция;
       ……………….
       инструкция ;
   end;
```

Здесь имеется в виду переменная типа запись или класс. Ниже показан пример доступа к свойствам и методам объекта **PaintBox1.Canvas:**

```
……………………………………….
with PaintBox1.Canvas do
     begin
          Brush.Color :=clWhite;
          Rectangle(0, 0, Width, Height );
     end;
………………………………………..
```

В этом примере область графического объекта **Paintbox1**, доступная для рисования, закрашивается белым цветом.

Следующие два типа инструкций объединяются под общим названием *условных инструкций*, так они определяют действия в зависимости от выполнения некоторого условия.

3) ***Инструкция* if**. Общий вид инструкции:

```
If  логическое выражение  then
   begin
      Инструкция;
      ………………….
      Инструкция;
   end
else
   begin
      Инструкция;
      …………………..
      Инструкция;
   end;
```

Если значение логического выражения – **true** ("истина"), то выполняется составной оператор, стоящий после зарезервированного слова **then**, если же значение логического выражения – **false** ("ложь"), то выполняется

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

составной оператор, стоящий после зарезервированного слова **else**. При использовании инструкции **if** раздел, начинающийся со слова **else,** может быть опущен. Кроме этого необходимо помнить, что перед **else** *точка с запятой не ставится.* Пример использования инструкции:

```
…………………………………………..
If (cavas.pixels[X,Y]=clRed) or (cavas.pixels[X,Y]=clYellow) then
   begin
       N:=N+1;
       Cavas.Pixels[X,Y]:=clYellow;
   end
else
   Cavas.Pixels[X,Y]:=clBlack;
……………………………………………
```

4) ***Инструкция выбора*** (**case**). Инструкция выбора используется, когда необходимо использовать несколько альтернативных путей выполнения программы в зависимости от значения некоторого выражения. Общий вид инструкции:

```
case  выражение выбора  of
    Значение: begin
                 Инструкция;
                 ……………….
                 Инструкция;
              end;
       ………………………………
    Значение: begin
                 Инструкция;
                 ……………….
                 Инструкция;
              end;
    else
begin
            Инструкция;
            ………………..
            Инструкция;
        end;
end;            {of case}
```

Выражение выбора может иметь тип: **integer**, **real**, **char**. Составной оператор, стоящий после зарезервированного слова **case** выполняется, ко-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

гда выражение выбора не принимает ни одного из значений, перечисленных после слова **of**. Пример использования инструкции выбора:

```
…………………………………….
case Canvas.Pixels[X,Y] of
    clRed:    begin
                  N:=N+1;
                  Canvas.Pixels[X,Y]:=clYellow;
              end;
    clYellow:        N:=N+1;
    else
       Canvas.Pixels[X,Y]:=clBlack;
end; {конец инструкции выбора}
……………………………………..
```

В этом примере выполняются те же действия, что и в предыдущем.

Инструкции *цикла* используются в тех случаях, когда требуется неоднократное выполнение одного и того же набора инструкций до тех пор, пока не будет выполнено некоторое условие. В Object Pascal определены три типа циклов: **for –** циклы, **while –** циклы и **repeat –** циклы.

5) ***Циклы со счетчиком***. В этом виде цикла заранее определяется количество шагов с помощью *счетчика цикла* - переменной, обычно *целого типа*, для которой задаются начальное и конечное значения. После выполнения очередного шага величина счетчика изменяется на единицу. Когда счетчик достигает своего конечного значения, выполняется последний шаг цикла.

Общий вид этой инструкции:

```
for  счетчик := начальное значение to конечное значение do
    begin
        Инструкция;
        …………………..
        Инструкция;
    end;
```

При такой организации цикла значение счетчика *увеличивается.*

```
for  счетчик := начальное значение downto конечное значение do
    begin
        Инструкция;
        …………………..
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
      Инструкция;
   end;
```

В этом случае на каждом шаге значение счетчика *уменьшается*. Необходимо отметить, что в качестве счетчика могут использоваться переменные *символьного* и *интервального* типа.

Пример использования **for** - цикла для вывода на экран графика функции y=sin(x):

```
……………………………………………..
for I:=0 to PaintBox1.Width do
  begin
     Y:=Round(100*Sin(I/20));
     PaintBox1.Canvas.Pixels[I,Y]:=clAqua;
  end;
……………………………………………...
```

6) ***Циклы с предусловием***. Здесь набор инструкций повторятся до тех пор, пока выполняется некоторое условие. В качестве условия используется *логическое выражение*. Проверка условия происходит *перед тем*, как выполняется следующий шаг. Если значение логического выражения равно **true** ("истина"), продолжается выполнение цикла, если – **false** ("ложь"), цикл заканчивается. Общий вид инструкции:

```
while  логическое выражение  do
   begin
      Инструкция;
      ………………………….
      Инструкция;
   end;
```

Пример использования цикла с предусловием

```
………………………………………
with Image1 do
while ((X>0) and (X<Width)) and ((Y>0) and (Y<Height)) do
  begin
     EField(x,y);
     X:=X+dl*Ex/E;
     Y:=Y+dl*Ey/E;
     Canvas.Pixels[Round(X),Round(Y)]:=clLime;
     Refresh;
  end;
……………………………………………..
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

7) ***Циклы с постусловием***. В циклах с постусловием после выполнения каждого шага проверяется *логическое выражение*, определяющее условие *окончания* цикла. Если значение логического выражения – **false**, цикл продолжается, если **true**, то цикл завершается. Следует обратить внимание на то, что использование логического выражения здесь *противоположно* тому, как оно используется в **while –** циклах. Общий вид инструкции:

```
…………………………..
repeat
   Инструкция;
         ………………
   Инструкция;
until логическое выражение;
……………………………
```

В циклах данного типа нет необходимости повторяющийся набор инструкций заключать в *операторные скобки*. На конец структуры цикла указывает зарезервированное слово **until**.

Пример использования **repeat –** цикла:

```
……………………………………………
repeat
  if Random > ver then
      Break;
  L:=-L0*(1+Ln(Random));
  Phi:=2*Pi*Random;
  X:=X+L*Cos(Phi);
  Y:=Y+L*Sin(Phi);
  Canvas.LineTo(Round(X),Round(Y));
until X>D;
……………………………………………
```

Существуют две стандартные процедуры для работы с циклами. Процедура **Break** используется, когда надо досрочно выйти из цикла, процедура **Continue** используется, когда есть необходимость прервать выполнение текущего шага и перейти к следующему. Обычно эти процедуры используются в составе условных инструкций **if** внутри цикла.

1.6. Структура программы.

Программа представляет собой набор команд (инструкций), *последовательное* выполнение которых процессором приводит к достижению за-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

планированного результата. Последовательность инструкций определяется *алгоритмом* задачи.

Программа в Object Pascal состоит из нескольких разделов:

{заголовок}

program имя программы**;**

{раздел описаний}

uses модуль**, ….. ,** модуль**;** {список используемых модулей}
{$…..} {директивы компилятору}
type ……. {описание типов}
var ……… {описание переменных}
const ……. {описание констант}
procedure …. {описание процедур}
function …… {описание функций}

{раздел инструкций}

begin
Инструкция **;**
…………………
Инструкция **;**
end. {конец программы}

В разделе описаний программы описываются *глобальные* переменные, константы и типы, доступные из любого блока программы. С помощью зарезервированного слова **uses** подключаются *библиотечные модули*. При этом программе становятся доступны переменные, константы, типы, процедуры и функции, описанные в этих модулях, что фактически многократно увеличивает раздел описаний самой программы. *Директивы компилятору* в основном используются для подключения к программе так называемых ресурсных файлов, где в откомпилированном виде содержится дополнительная информация о *визуальных компонентах*, не включенная в программу.

Конец программы обозначается зарезервированным словом **end** с *точкой* на конце.

Раздел описаний.

Описания типов производится в соответствии с правилами описания типов, приведенными выше:

type
Имя типа = описание**;**

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

……………………….
Имя типа= описание**;**

Описание переменных:

var
имя переменной**, …. ,** имя переменной **:** тип**;**
………………………..
имя переменной**, …. ,** имя переменной **:** тип**;**

Описание констант:

const
Имя константы = значение**;**
……………………………..
Имя константы **:** тип = значение**;** {типизированная константа}
……………………………..

Значение типизированной константы можно изменять внутри программы. В этом случае она выступает как переменная с начальным присвоением. Последовательность описания типов, переменных и констант в разделе описаний произвольная.

1.7. Процедуры и функции.

Процедуры и функции являются *подпрограммами*, представляют собой замкнутые блоки инструкций, которые могут быть вызваны для исполнения из разных частей программы. *Функция* – это подпрограмма, которая, которая после завершения возвращает *результат* – данные определенного типа. Поэтому функции используются в выражениях или в инструкциях присвоения. *Процедура* – это подпрограмма, которая после завершения не возвращает результата, и она вызывается в программе как самостоятельная инструкция. Функцию также можно вызывать как самостоятельную инструкцию, но при этом ее результат отбрасывается. Объявление процедур и функций производится в разделе описаний соответствующих блоков программы (основная программа, библиотечный модуль, другие процедуры и функции).

Объявление процедур и функций.

Процедуры и функции состоят из *заголовка*, включающего в себя *имя* и *список передаваемых параметров* и *тела*, куда входит раздел *локальных описаний* и раздел *инструкций*. При объявлении указывается заго-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

ловок подпрограммы и описывается ее структура. Раздел локальных описаний и раздел инструкций полностью аналогичен разделу описаний и разделу инструкций программы.

{заголовок процедуры}
procedure имя процедуры (параметры**:** тип**; ……;** параметры**:** тип**);**

или

{заголовок функции}
function имя функции (параметры**:** тип**; ……;** параметры**:** тип**):** тип результата**;**

{раздел локальных описаний}
type ……. {описание типов}
var ……… {описание переменных}
const ……. {описание констант}
procedure …. {описание внутренних процедур}
function …… {описание внутренних функций}

{раздел инструкций}
begin
Инструкция **;**
…………………
имя функции **:=** результат**;** {присвоение результата функции (только при описании функции)}
…………………

Инструкция **;**
end;

Раздел локальных описаний присутствует, если в подпрограмме определены локальные типы данных, локальные переменные и константы. Локальные описания действуют только в пределах той подпрограммы, где они определены.

В подпрограммах можно использовать стандартную процедуру **Exit** для досрочного завершения процедуры или функции. Обычно она используется в составе условных инструкций **if** внутри подпрограммы.

Примеры описания процедур и функций:

…………………………………..
procedure EField (X,Y: real);
var
R: real;

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
begin
    R:= Sqrt(sqr(X-X0)+Sqr(Y-Y0));
    Ex:=Q*(X-X0)/(R*R*R);
    Ey:=Q*(Y-Y0)/(R*R*R);
    E:=Sqrt(Sqr(Ex)+Sqr(Ey));
end;

function Potential (X,Y: real): real;
var
    R: real;
begin
    R:= Sqrt(Sqr(X-X0)+Sqr(Y-Y0));
    Potential:=Q/R;     {присвоение результата функции}
end;

procedure Degree (A,B: real; var C: real);
begin
    C:=Exp(B*Ln(A));
end;
……………………………………………

{вызов подпрограмм из основного блока}
EField (X,Y);
Phi:=Potential(150,200.5);
Degree(2.58e-3,56.4,C);
…………………………………………….
```

Передаваемые параметры.

Список передаваемых параметров представляет собой набор данных, которыми обмениваются между собой отдельные самостоятельные блоки программы. При описании процедуры или функции в их заголовке в круглых скобках указываются *формальные параметры*. Формальные параметры рассматриваются как переменные, дополняющие список локальных переменных. При вызове подпрограммы вместо формальных параметров в список подставляются фактические параметры, т.е. данные, определенные в том блоке программы, из которого производится вызов. Тип и последовательность подстановки фактических параметров должны соответствовать типу и последовательности расположения формальных параметров в списке при объявлении процедуры или функции.

Формальные параметры чаще всего объявляются как *параметры – значения* и *параметры – переменные*. Параметры – значения используются для передачи данных из основного блока в подпрограмму. При их измене-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

нии внутри подпрограммы, измененное значение не возвращается в основной блок. При необходимости возврата измененного значения параметр объявляется как параметр – переменная. При этом в списке параметров эти параметры обозначаются зарезервированным словом **var**. Кроме формальных параметров, локальных переменных, констант и типов в подпрограмме доступны и переменные, константы и типы, объявленные в основном блоке (*глобальные* переменные, константы и типы).

1.8. Библиотечные модули.

Библиотечный модуль представляют собой самостоятельный (отдельный от основной программы) блок, в котором описаны типы, переменные, константы, процедуры и функции, которые можно использовать в основной программе, если подключить к ней этот модуль. Сам модуль хранится в отдельном файле с расширением "**.pas**", имя которого *обязательно* совпадает с именем модуля. Поскольку модуль не является программой, его можно откомпилировать, но *нельзя* запустить на исполнение. Так же как и программа библиотечный модуль имеет свою строго определенную структуру.

Модуль состоит из двух основных частей:

Интерфейсной, содержащей объявления типов, переменных, констант, заголовков процедур и функций, доступных из любого блока программы. Эта часть модуля располагается после зарезервированного слова **interface**.

Выполняемой, где в основном приведены полные описания процедур и функций, заголовки которых имеются в интерфейсной части, а также есть свои разделы описаний, необходимые для работы этих подпрограмм. Все описания, приведенные в этой части модуля, недоступны из других блоков программы. Эта часть модуля располагается после зарезервированного слова **implementation**.

Кроме этих двух основных частей модуль может содержать раздел *инициализации*, содержащий набор инструкций, которые выполняются при совместной компиляции основной программы и модуля. Этот раздел используется для инициализации переменных модуля (задания начальных значений).

Структура модуля:

{заголовок модуля}

unit имя модуля**;**

{интерфейсная часть модуля}

interface

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

uses **…………..**{список используемых модулей}
type **…………..**{описание типов}
var **…………..**{описание переменных}
const **………….**{описание констант}

{заголовки процедур и функции}
procedure имя процедуры (список параметров)**;**
……………………………..
procedure имя процедуры (список параметров)**;**
function имя функции (список параметров)**:** тип результата**;**
……………………………..
function имя функции (список параметров)**:** тип результата**;**

{выполняемая часть модуля}
implementation

uses **…………..**{список используемых модулей}
type **…………..**{описание типов}
var **…………..**{описание переменных}
const **………….**{описание констант}

{описание процедур и функций}
procedure имя процедуры (список параметров)**;**
**…………………….. **{описание процедуры}
…………………………………………………….
procedure имя процедуры (список параметров)**;**
**…………………….. **{описание процедуры}

function имя функции (список параметров)**:** тип результата**;**
………………………{описание функции}
…………………………………………………….
function имя функции (список параметров)**:** тип результата**;**
………………………{описание функции}

{раздел инициализации}
begin
   инструкция**;**
………………
   инструкция**;**
end. {конец модуля}

При создании модуля необходимо следить, чтобы заголовки одних и тех же подпрограмм в интерфейсной и выполняемой частях модуля совпа-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

дали друг с другом. Конец модуля, так же как и конец программы обозначается словом **end** с точкой. Если отсутствует раздел инициализации, соответствующий **begin** можно опустить.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

2. Программирование в Delphi.

2.1. Windows – приложение.

Приложения, работающие под управлением операционной системы Windows-95, обладают рядом общих черт.

Во-первых, каждое приложение работает в окне, имеющем стандартный набор свойств, таких как наличие кнопок системного меню, возможность изменять размеры и перемещаться по экрану, закрываться окном другого приложения и восстанавливать свой вид при активизации.

Во-вторых, приложения работают в режиме диалога с пользователем, когда пользователь, управляя работой приложения, может в любой момент вмешаться в действия программы, закрыть ее или переключиться на работу в другое приложение.

Это достигается с помощью механизма обработки *сообщений*, реализованного в операционной системе Windows – 95. При поступлении сигналов от внешних устройств, таких как клавиатура, мышь и т.д. или по сети, а также от системных устройств, например таймера, ядро операционной системы вырабатывает сообщение (windows message) о *событии* и посылает его приложению. Таких сообщений насчитывается более двухсот (например: **wm_Create** – создание окна, **wm_Char –** нажатие клавиши на клавиатуре, **wm_MouseMove** – перемещение мыши, и т.д.). Само сообщение представляет собой структуру типа *запись*, в полях которой записана вся информация о событии, вызвавшем сообщение, например, какая клавиша была нажата (сообщение **wm_Char**). Каждое приложение имеет в своей структуре *цикл обработки сообщений*. Он запускается сразу после запуска приложения. По мере поступления сообщений, они распознаются, и вызывается процедура – *обработчик события*. После завершения очередной процедуры обработки события приложение опять переходит в режим ожидания следующих сообщений. Это продолжается вплоть до поступления сообщения **wm_Quit**, которое генерируется при нажатии кнопки закрытия окна, после чего приложение закрывается.

2.2. Проект Delphi.

Приложение, создающееся в среде программирования Delphi, состоит из нескольких файлов, объединенных в *проект*. Схема сборки проекта показана на рис. 2.1.

Во-первых, это *основная программа*, которая представляет собой текстовый файл "**project1.dpr**" с текстом основной программы на языке Object Pascal.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

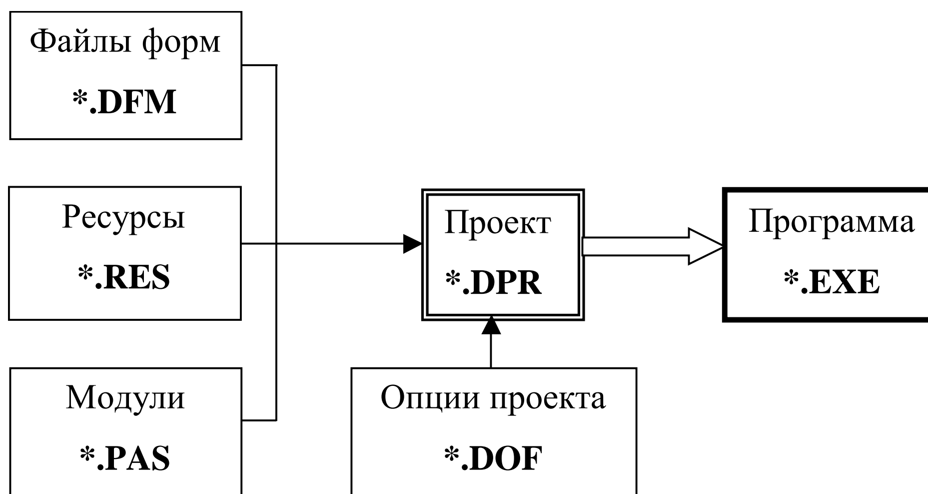

Рис. 2.1. Схема сборки проекта Delphi.

Во-вторых, *библиотечные модули*, содержащие пользовательские процедуры обработки событий, которые с помощью зарезервированного слова **uses** присоединены к основной программе. Они представляют собой текстовые файлы с расширением "**.pas**", имя которых совпадает с именем модуля. В простейшем случае – это один модуль "**form1.pas**".

Обязательной частью проекта является *файл формы*, в котором в бинарном виде записано изображение формы – окна, главного и обязательного элемента Windows – приложения. Файл формы имеет расширение "**.dfm**", создается автоматически и в нем описаны свойства формы, кнопок и других компонентов, заданные при создании формы визуальными средствами. Они включают в себя набор свойств, указываемых в окне "Object Inspector" среды Delphi. Файл формы можно увидеть, открыв его в окне редактора.

Кроме этого частью проекта являются некоторые *служебные файлы*, в которых записана информация, необходимая для работы с проектом.

Основная программа.

Основная программа создается автоматически и имеет стандартный вид:

```
program Project1;
uses
    Forms,
    Unit1 in 'Unit1.pas' {Form1};
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
{$R *.RES}

begin
    Application.Initialize;
    Application.CreateForm(TForm1, Form1);
    Application.Run;
end.
```

Основная программа содержит всего три инструкции, являющиеся вызовами методов объекта **Application**, являющегося глобальной переменной типа **TApplication**, класса, описанного в стандартном модуле **Forms**. Класс **TApplication** содержит все необходимые методы и свойства для того, чтобы программа работала как Windows – приложение. Первая инструкция (**Initialize**) программы инициализирует само приложение, вторая инструкция (**CreateForm(TForm1, Form1)**) создает главное окно приложения. При наличии нескольких окон эта инструкция повторяется для всех окон приложения. И, наконец, третья инструкция (**Run**) выводит форму на экран и запускает цикл обработки сообщений.

Пользовательские библиотечные модули.

Любая программа содержит как минимум один пользовательский модуль, в котором описан класс **TForm1**, являющийся потомком класса **TForm**. Класс **TForm1** является описанием главного окна приложения, он наследует все основные свойства окна Windows – приложения и включает в себя поля – *визуальные компоненты*, добавляемые в окно, и методы – процедуры обработки событий. Кроме этого в описание класса **TForm1** можно добавлять другие поля и методы, необходимые для работы приложения. Ниже приведен вид модуля, описывающего окно с кнопкой (поле **Button1**):

```
unit Unit1;

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
    TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
```


PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
private
  { Private declarations }
public
  { Public declarations }
end;

var
   Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
   Close;
end;

end.
```

Метод **Button1Click(Sender: TObject)** определен как процедура обработки события – "нажатия кнопки". Согласно описанию процедуры при нажатии кнопки окно закрывается (метод **Close**).

2.3. Визуальные компоненты.

Визуальные компоненты являются основными "кирпичиками", из которых строится приложение. С точки зрения Object Pascal они представляют собой объекты, классы которых описаны в стандартных библиотеках Delphi. Некоторые свойства этих объектов можно задавать с помощью специального окна "Object Inspector" среды Delphi (вкладка "Properties").

Для визуальных компонентов определены свойства особого типа – *события*. Событие – это переменная процедурного типа (указатель на метод объекта) и связано с обработкой определенного сообщения Windows. Используя вкладку "Events", данному свойству присваивается метод объекта, делая таким образом этот метод процедурой обработки сообщения. В процедуру обработки сообщения в качестве параметров передается информация из самого сообщения, например текущие координаты мыши или ASCII код нажатой клавиши.

В таблице 2.1. приведен список некоторых, наиболее часто используемых визуальных компонентов. *Форма* – это основной компонент программы. С его помощью строится главное окно приложения. Форма явля-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

ется контейнером для остальных визуальных компонентов. Компонент, встроенный в форму, становится полем класса **TForm1**, в свою очередь имеющим тип **class**.

Таблица 2.1.

Компонент	Страница	Описание	Использование
MainMenu	Standard	Главное меню	Запуск процедур
Button, BitBtn, SpeedButton	Standard, Additional	Кнопки	Запуск процедур
Edit	Standard	Окно редактирования	Ввод и вывод данных
Memo	Standard	Текстовый редактор	Ввод и вывод большого текста
Label	Standard	Статический текст	Подписи
Panel	Standard	Панель инструментов и статусная строка	Контейнер для других компонентов
Image	Additional	Окно рисунка	Размещение рисунков из файла. Рисование
Shape	Additional	Фигуры	Изображение прямоугольников и эллипсов
PaintBox	System	Контейнер для рисунков	Рисование
Timer	System	Таймер	Многократный запуск процедуры

Общие свойства компонентов.

Ниже описаны свойства, имеющиеся у большинства компонентов. Поскольку эти свойства имеют одинаковые названия, при обращении к ним необходимо использовать квалифицируемый идентификатор, т.е. указывать кроме имени свойства еще и имя объекта, свойство которого рассматривается.

Значение свойств компонентов можно не только задавать в окне "Object Inspector", но и менять в ходе выполнения программы. Для этого используется инструкция присвоения:

Timer1.Enabled:=True;
Label2.Caption:='Подождите, идут вычисления';

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Свойства, задающие размеры компонента и его расположение на экране:

Left – координата левой границы компонента в пикселях.
Top – координата верхней границы компонента в пикселях.
Width – ширина компонента в пикселях.
Height – высота компонента в пикселях.

Эти свойства имеют тип **integer**. Координаты отсчитываются от границ рабочей области формы, в которой установлен визуальный компонент.

Следующие свойства – переменные строкового типа (**string**), служат для идентификации компонента:

Name – имя компонента. Delphi самостоятельно именует свои компоненты и без особой необходимости изменять их имена не следует. Имя компонента используется в выражениях и инструкциях программы.
Caption – заголовок или подпись. По умолчанию в этом свойстве записано имя компонента.

Следующие свойства определяют графическое оформление компонента:

Color – цвет фона. Можно выбрать из предлагаемого списка.
Font – шрифт, с помощью которого будут отображаться стандартный текст в окне компонента. Можно выбрать вид шрифта, размер и цвет символов.

Свойства, которые определяют функциональность компонента. Это логические переменные (тип **boolean**):

Visible – свойство, с помощью которого компонент ставится видимым (**true**) или невидимым (**false**).
Enabled – свойство, позволяющее активизировать компонент (**true**) или сделать его неактивным (**false**). Активность компонента означает его способность реагировать на события от мыши, клавиатуры или таймера.

События, которые являются общими для всех компонентов:

onCreate – событие, связанное с созданием компонента.
onClick – событие, связанное с нажатием на клавишу мышки.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

onMouseMove – событие, связанное с перемещением мыши. В качестве параметров процедуры обработки передаются текущие координаты мыши и удерживаемые нажатыми клавиши.

onMouseDown – событие, связанное с нажатием клавиши мыши. Передаются координаты мыши, и имя нажатой клавиши.

onMouseUp – событие, связанное с отпусканием клавиши мыши. Передаются координаты мыши, и имя отпущенной клавиши.

onKeyPress – событие, связанное с нажатием клавиши на клавиатуре. В качестве параметра передается символ (тип **char**), соответствующий нажатой клавиши.

Некоторые методы, общие для всех компонент:

Refresh – обновление компонента. Этот метод необходимо вызывать, если в процессе выполнения процедуры изменились свойства компонента.

Close – закрыть компонент. Если этот метод применяется для главной формы, то закрывается приложение.

Вызов процедур.

При запуске программы на исполнение на экране возникает изображение главного окна приложения, после чего программа переходит в режим ожидания Windows – сообщений, которые она может обработать. Для выполнения необходимых действий надо запустить пользовательские процедуры. Основной способ вызова пользовательских процедур – это использование *меню* и *кнопок*. Вызываемая процедура подключается, как процедура обработки сообщения **onClick**, т.е. она запускается при нажатии клавиши мыши на пункте меню или кнопке. После завершения процедуры управление опять передается главному окну. Для запуска другой процедуры надо нажать клавишей мыши на другом пункте меню или кнопке.

Главное меню **MainMenu** состоит из пунктов, каждый из которых может раскрываться в виде подменю. Создать меню можно с помощью *Дизайнера меню*. Для каждого пункта меню необходимо задать свойство **Caption** (заголовок) и процедуру обработки **onClick**.

Стандартная кнопка **Button** содержит только подпись.

Кнопка **BitBtn** представляет собой кнопку с предопределенными возможностями обработки событий. При выборе этого компонента дополнительно можно задать свойства **Kind** – выбор типа кнопки и **Gliph** – выбор рисунка, помещаемого на кнопку. Свойство **Kind** определяет и действие кнопки, в частности, если выбрана кнопка типа **bkClose**, то ее нажатие закрывает форму.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Кнопка **SpeedButton** отличается тем, что она обычно используется без подписи, только с картинкой.

Для кнопки необходимо задать свойство **Caption** (подпись) и процедуру обработки **onClick**.

Ввод и вывод данных.

Для ввода и вывода данных используются *окна редактирования* (**Edit**). Ввод осуществляется с помощью клавиатуры. Данные вводятся в виде строки текста и сохраняются в свойстве **Text**. При необходимости вводить и выводить числа используются стандартные *функции преобразования типов:*

I:= StrToInt(Edit1.Text); {преобразование строки символов в целое число**}**
X:= StrToFloat(Edit2.Text); {преобразование строки символов в вещественное число**}**
Edit1.Text:=IntToStr(K); {преобразование целого числа в строку символов**}**
Edit2.Text:= FloatToStr(X); {преобразование вещественного числа в строку символов**}**

Преобразование строки в число используется для ввода данных через окно редактирования, обратное преобразование используется для вывода результатов расчетов в окно редактирования.

Событие **onChange** используется для преобразования введенного текста в число. Это событие наступает при изменении текста в окне редактирования. Пример процедуры обработки этого события (через окно редактирования **Edit1** вводится целое число **M**):

```
………………………………………..
procedure TForm1.Edit1Change(Sender: TObject);
begin
   if Edit1.Text >'' then     {проверка того, что строка непустая}
       M:=StrToInt(Edit1.Text);
end;
………………………………………..
```

Необходимость проверки того, что строка **Edit1.Text** непустая, связана с тем, что функции **StrToInt** и **StrToFloat** не работают с пустыми строками.

Компонент **Memo** представляет собой текстовый редактор, который можно встроить в приложение. Он обладает всеми качествами стандартно-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

го текстового редактора. Используется для вывода информации, состоящей из нескольких строк. В свойстве **Text** храниться полный текст, содержащийся в редакторе.

Свойство **Lines** используется для доступа к отдельным строкам текста. Это переменная класса **TStrings**. Методы этого класса используются для добавления, удаления и вставки строк. Например, следующие инструкции добавляют текст:

```
Memo1.Lines.LoadFromFile('c:\autoexec.bat'); {ввод текста из файла}
Memo1.Lines.Add('Еще одна строка');
```

Управление программой.

С помощью компонента **Timer** можно регулировать скорость выполнения отдельных процедур. Этот компонент подключает к программе системный таймер. Таймер вырабатывает событие **onTimer**, наступающее через определенные промежутки времени, которые задаются свойством **Interval**. Это переменная целого типа, задает интервал в миллисекундах между двумя последовательными событиями **onTimer.** В качестве обработчика события может выступать любая пользовательская процедура, выполнение которой автоматически повторяется через заданные промежутки времени.

Обычно процедура, определенная в качестве обработчика события **onTimer**, используется вместо цикла, но в отличие от него выполняется с регулируемой задержкой. Во время задержки управление программой передается главному окну, что позволяет, не дожидаясь его окончания, остановить выполнение этого цикла, запустить другую процедуру, управлять размерами и расположением окна или закрыть приложение.

Обычно включение таймера производится с помощью нажатия соответствующей кнопки. Пример процедуры обработки нажатия такой кнопки:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Timer1.Enabled:= true;
end;
```

Часто возникает необходимость прервать выполнение процедуры, не используя таймер, например по нажатию какой-либо клавиши. В этом случае в процедуре надо предусмотреть явный запуск функции обработки сообщений. Следующий пример показывает, как это можно сделать.

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

```
Procedure ……………….;
……………………..
var
Msg: TMsg; {объявление переменной, в которой будет записана структура
        Windows – сообщения}
……………………
begin
……………………
PeekMessage(Msg,0,wm_KeyDown,wm_KeyDown,pm_Remove);    {функ-
        ция, которая заполняет структуру Msg, в случае, если поступило со-
        общение}
If Msg.Message = wm_KeyDown then {анализ типа сообщения (нажатие
        клавиши)}
Exit; {завершение процедуры, если клавиша нажата}
………………………
end;
```

2.4. Графические операции в Delphi.

Рисовать на экране можно, используя готовые графические компоненты Delphi, а также благодаря свойству **Canvas**.

Графические компоненты.

Визуальный компонент **Shape** предназначен для изображения различных геометрических фигур – окружностей, эллипсов, прямоугольников и квадратов. Вид этих фигур определяется свойствами:

Shape – определяет тип фигуры.
Pen – определяет стиль и цвет рамки.
Brush – определяет стиль и цвет закраски.

Компонент **Image** используется для отображения в окне различных графических изображений, хранимых во внешних файлах, а также для рисования и сохранения созданных рисунков. Для загрузки и хранения рисунка используется свойство **Picture** (класс **TPicture**). При выборе этого свойства в окне "Object Inspector" открывается графический редактор, с помощью которого можно выбрать графический файл для загрузки.

Рисовать в процессе выполнения программы в окне компонента **Image** можно с помощью объекта **Canvas**, описанного ниже. Для загрузки

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

рисунка в процессе выполнения программы и сохранения нарисованного в файле используют методы **TPicture**:

Image1.Picture.LoadFromFile('map.bmp'); {загружается рисунок из файла "**map.bmp**"**}**
Image1.Picture.SaveToFile('mypict.bmp'); {содержимое **Image1** сохраняется в файле "**mypict.bmp**"**}**

При использовании компонента **Image** в качестве поля для рисования каждый раз *необходимо* применять метод **Refresh** для обновления и перерисовки окна **Image**.

Инструменты рисования.

Возможность выводить на экран текст и рисовать на нем обеспечивается свойством **Canvas** (класс **TCanvas**), которое определено для многих визуальных компонентов Delphi. В этом классе определены инструменты и методы рисования.

Инструментами рисования являются три основных свойства **Canvas**: перо – **Pen** (класс **TPen), кисть** – **Brush** (класс **TBrush**) и шрифт – **Font** (класс **TFont**). Ниже приведены основные свойства и методы этих классов, а также примеры их использования.

1) ***Класс TPen***. Используется для рисования линий и фигур

Свойства:

Style – задает стиль линии. Возможные значения:
- **psSolid** – сплошная;
- **psDash** – прерывистая;
- **psDot** – пунктирная;
- **psDashDot** – штрих пунктирная;
- **psClear** – линия цвета фона.

Width – задает толщину линии. Это целое число.
Color – задает цвет линии. Это переменная *интервального* типа, содержащего набор целых чисел.

В Delphi предусмотрено несколько вариантов задания цвета.

Во-первых, это предопределенные константы: **clBlack, clWhite, clGreen, clRed, clBlue,** и т.д. Например:

Canvas.Pen.Color:= clRed;

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

Во-вторых, цвет можно задавать непосредственно в виде целого числа:

Canvas.Pen.Color:=TColor($ff0000);

При такой записи учитывается, что цвет задается в виде трехбайтового числа, первый байт которого задает интенсивность *красного* цвета, второй – *зеленого*, третий – *синего*.

В-третьих, для задания цвета можно использовать стандартную функцию **RGB(R,G,B)**. Параметры этой функции типа **byte** – десятеричные числа, показывающие интенсивности трех основных цветов. При использовании этой функции можно цвет задавать с помощью переменных, вычисляемых в программе, например:

Canvas.Pen.Color:=RGB(145,15+I,255-J*B);

2) ***Класс TBrush***. Используется для заполнения (закраски) различных геометрических фигур.

Свойства:

Style – задает стиль заполнения. Возможные значения:
- **bsSolid –** сплошная закраска;
- **bsClear –** закраска цветом фона;
- **bsHorizontal** – штриховка горизонтальными линиями;
- **bsVertical** – штриховка вертикальными линиями;
- **bsDiagonal** – штриховка наклонными линиями;
- **bsCross**, **bsDiagCross** – штриховка "крест-накрест".

Color – задает цвет заполнения.
Bitmap - с помощью этого свойства можно указать рисунок, который будет использоваться в качестве стиля кисти.

3) ***Класс TFont***. Этот класс используется для задания свойств шрифта при выводе на экран текста и символов.

Свойства:

Color – задает цвет символов.
Height – высота символов в пикселях.
Size – размер шрифта в типографских единицах (пунктах).
Style – стиль шрифта. Возможные значения:

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

fsBold – "жирный" шрифт;
fsItalic – "курсив";
fsUnderLine – подчеркнутые буквы;

Name – задает название гарнитуры шрифта, например:
Canvas.Font.Name:='Arial';

Методы рисования.

Для рисования указанными выше инструментами используют методы класса **TCanvas.** Наиболее часто используемые из них:

MoveTo(X,Y) – изменяет координаты текущей точки. Используется для указания начала линии. **X** и **Y** – целые числа, обозначающие координаты точки в пикселях *от верхней левой границы* того визуального компонента, чье свойство **Canvas** используется для рисования.

LineTo(X,Y) – проводит линию в точку с координатами **X** и **Y**. Стиль и цвет линии определяется заданными свойствами объекта **Pen**.

Ellipse(X1,Y1,X2,Y2) – рисует эллипс, вписанный в прямоугольник с заданными координатами верхнего левого (**X1,Y1**) и нижнего правого (**X2,Y2**) углов. Параметры граничной линии и закраски определяются заданными свойствами объектов **Pen** и **Brush**.

Rectangle(X1,Y1,X2,Y2) – рисует прямоугольник, верхний левый и нижний правый угол которого задается координатами **X1, Y1** и **X2, Y2**. Параметры граничной линии и закраски определяются заданными свойствами объектов **Pen** и **Brush**.

FloodFill(X,Y,Color,FillStyle) – закрашивает *замкнутую* область, внутри которой находится точка с координатами **X,Y**. Параметры закраски определяются заданными свойствами объекта **Brush**. Параметр **FillStyle** определяет, каким образом будет происходить закраска. Он может принимать следующие значения:
fsBorder – будет закрашиваться область, ограниченная замкнутой линией, нарисованной цветом **Color**;
fsSurface – область заполняется до тех пор, пока в ней присутствует цвет, заданный параметром **Color**.

TextOut(X,Y,Text) – выводит на экран строку текста, которая задается параметром **Text** типа **string**. Координаты **X,Y** определяют точку на эк-

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com

ране, начиная с которой будет выводиться текст. Параметры шрифта задаются свойствами объекта **Font**. Пример:

Canvas.TextOut(Round(Width/2),Round(Height/2),'Привет!');

Для вывода на экран изображения по точкам используют свойство **Pixels** класса **TCavas**. Это двумерный массив, в котором записаны цвета всех пикселей, находящихся в текущей области, доступной для рисования. Доступ к пикселям производится с помощью инструкции присвоения:

Canvas.Pixels[X,Y]:=clYellow **{**точка с координатами **X,Y** закрашивается в желтый цвет**}**

Canvas.Brush.Color:=Canvas.Pixels[I,J]; **{**цвет кисти задается цветом пикселя с координатами **I,J}**.

If Canvas.Pixels[X,Y]=clRed then
Canvas.Pixels[X,Y]:=clBlack; **{**красная точка заменяется на черную**}**

PDF created with FinePrint pdfFactory Pro trial version http://www.fineprint.com