Министерство образования и науки РФ
Государственное образовательное учреждение
высшего профессионального образования
Владимирский государственный университет
Кафедра физики и прикладной математики

ЯЗЫКИ ПРОГРАММИРОВАНИЯ И МЕТОДЫ ТРАНСЛЯЦИИ

Методические указания
к лабораторным занятиям

Составители:
А.В. ДУХАНОВ
О.Н. МЕДВЕДЕВА
М.В. ШИШКИНА

Владимир 2011

УДК 004.43
ББК 32.973.26-018.1
Я41

Рецензент
Кандидат технических наук,
генеральный директор ООО «ФС Сервис»
Д.С. Квасов

Печатается по решению редакционного совета
Владимирского государственного университета

Языки программирования и методы трансляции : метод. указания к лаб. занятиям / Владим. гос. ун-т ; сост. : А. В. Духанов, О. Н. Медведева, М. В. Шишкина. – Владимир : Изд-во Владим. гос. ун-та, 2011. – 68 с.

Я41

Приведены лабораторные работы по дисциплине «Языки программирования и методы трансляции».

Предназначены для студентов 2-го курса очной формы обучения специальности 010501 – прикладная математика и информатика и направления подготовки бакалавров 010500 – прикладная математика и информатика.

Рекомендованы для формирования профессиональных компетенций в соответствии с ФГОС 3-го поколения.

Ил. 2. Табл. 3. Библиогр.: 6 назв.

УДК 004.43
ББК 32.973.26-018.1

ВВЕДЕНИЕ

Быстрое развитие информационных технологий повышает требования к уровню знаний современного специалиста. Огромное количество программных средств разработки требует систематизированного подхода на этапах выбора необходимых системных ресурсов. Именно поэтому актуально и необходимо освоение принципов организации, состава и схемы работы систем программирования, способов разработки прикладных программных средств на высокоуровневых языках программирования, основных приемов программирования.

Цель преподавания дисциплины – ознакомление с основными методами, средствами и стандартами языков программирования и разработки программного обеспечения, а также системами программирования.

В процессе изучения дисциплины студенты изучат следующие темы:

1) операторный базис языков программирования;

2) реализация алгоритмических структур;

3) объектно-ориентированное программирование и визуальные компоненты;

4) методы трансляции.

Лабораторная работа № 1. МАШИНА ТЬЮРИНГА

Цель работы: *изучение работы автомата на примере машины Тьюринга.*

1.1. Краткая теоретическая часть

Автомат считывает символы с ленты, которая имеет бесконечную длину. В каждый момент времени считывающее устройство автомата (каретка) располагается над одним символом на ленте. Автомат работает бесконечно долго.

Алфавит A = $\{a_i, i = \overline{1, n}\}$, описывающий задачу, состоит из любых символов, является конечным, содержит символ разделения (пробел) «Λ».

Автомат всегда находится в одном состоянии из множества $Q = \{q_j, j = \overline{1, m}\}$. Множество состояний Q определяется условием задачи.

После считывания очередного символа автомат должен записать новый символ a на ленту, переместить считывающее устройство в определенном направлении $h = \{Н, Л, П\}$ (Н – «не перемещать», Л – «налево», П – «направо»), перейти в новое состояние q.

Работа автомата записывается таблицей, в ячейку которой помещается тройка a, h, q.

Табличное представление машины Тьюринга

Алфавит / Состояния	Λ	a_1	…	a_n
q_1				
…			a, h, q	
q_m				

Перед запуском автомата требуется указать его начальное состояние и положение каретки на ленте.

1.2. Задания к работе

1. На ленте записано двоичное число. Считывающее устройство расположено над правым символом. Прибавить 1.

2. На ленте записано десятичное число. Считывающее устройство находится над любым символом. Прибавить 1.

3. На ленте записано десятичное число. Считывающее устройство расположено над правым символом. Прибавить 10.

4. На ленте записано двоичное число. Считывающее устройство находится над любым символом. Выполнить инверсию (замена 0 → 1 и 1 → 0).

5. На ленте – слово из символов «*a*», «*b*». Считывающее устройство находится над любым символом. Справа от слова поставить символ «*X*».

6. На ленте – слово из символов «*a*», «*b*». Считывающее устройство находится над любым символом. В начале и конце слова поставить символ «*X*».

7. На ленте записано положительное десятичное число. Считывающее устройство расположено над правым символом. Вычесть 1.

8. На ленте – последовательность из символов «*X*». Считывающее устройство находится над любым символом. Вставить перед правым символом «0».

9. На ленте – слово из символов «*a*», «*b*». Считывающее устройство расположено над левым символом. Отделить первые буквы «*b*» слова (перед первой буквой «*a*» вставить пробел).

10. На ленте – последовательность из символов «*X*». Считывающее устройство расположено над правым символом. Получить копию последовательности.

11. На ленте – последовательность из символов «*X*». Считывающее устройство расположено над левым символом. После правого символа вывести «Ч», если число символов четное, или «Н», если нечетные.

12. На ленте записано двоичное число. Считывающее устройство находится над левым символом. Выполнить последовательное сложение бит без переноса (1 + 1 = 0) и записать результат справа от числа.

13. На ленте – слово из символов «*a*», «*b*». Считывающее устройство расположено над правым символом. Получить зеркальную копию слова.

14. На ленте – слово из символов «*a*», «*b*». Считывающее устройство находится над левым символом. Получить зеркальное отражение исходного слова.

15. На ленте записаны два десятичных числа, разделенных знаком пробела (последнее число – однозначное). Считывающее устройство расположено над вторым числом. Необходимо вычислить разность. Результат может быть отрицательным.

1.3. Контрольные вопросы

1. Что представляет собой машина Тьюринга?
2. Перечислите основные состояния машины Тьюринга.
3. Можно ли в одну ячейку записать три символа?
4. Назовите направления перемещения каретки.

Лабораторная работа № 2. ВВЕДЕНИЕ В TURBO DELPHI

Цель работы: *ознакомление с основами создания простейшего приложения на языке Turbo Delphi, изучение основных типов языка и структуры программы.*

2.1. Краткая теоретическая часть

Структура любой программы на языке включает в себя:

```
Program имя_программы;
Uses
 Подключение модулей
Const
 Раздел описания констант
Label
 Раздел описания меток
Type
 Раздел описания типов
Var
 Раздел описания переменных
Begin
 Тело программы
End.
```

Элементы программы – это минимальные неделимые ее части, еще несущие в себе определенную значимость для компилятора. К элементам относятся:

- зарезервированные слова;
- идентификаторы;
- типы;
- константы;
- переменные;
- метки;
- подпрограммы;
- комментарии.

Зарезервированные слова – это английские слова, указывающие компилятору на необходимость выполнения определенных действий. Зарезервированные слова не могут использоваться в программе ни для каких иных целей, кроме тех, для которых они предназначены.

Зарезервированные слова:

And	Object	Div
Except	Then	Inherited
Label	Class	Program
Resourcestring	Function	Until
Array	Of	Do
Exports	Threadvar	Initialization
Library	Const	Property
Set	Goto	Uses
As	Or	Downto
File	To	Inline
Mod	Constructor	Raise
Shl	If	Var
Asm	Out	Else
Finalization	Try	Interface
Nil	Destructor	Record
Shr	Implementation	While
Begin	Packed	End
Finally	Type	Is
Not	Dispinterface	Repeat
String	In	With
Case	Procedure	Xor
For	Unit	

Имена операторов, переменных, констант, типов величин, имя самой программы назначаются программистом и называются идентификаторами. Существуют правила, которым должны отвечать все идентификаторы:

• идентификатор должен быть уникальным, то есть разные объекты не могут быть названы одним и тем же именем;

• идентификатор имеет ограничение по длине (зависит от конкретной реализации языка на компьютере);

• идентификатор может состоять только из символов латинского алфавита, цифр и знака подчеркивания ("_");

• идентификатор не может начинаться с цифры.

Типы – это специальные конструкции языка, которые рассматриваются компилятором как образцы для создания других элементов программы, таких как переменные, константы и функции.

К основным типам данных языка Delphi относятся:

• целые числа (Integer);

• дробные числа (Real);

• символы (Char);

• строки (String);

• логический тип (Boolean).

Помимо основных типов есть и другие. Их диапазоны значений и размер занимаемой памяти указаны в таблице вместе со значениями стандартных типов.

Диапазоны данных

Название типа	Диапазон значений	Размер в памяти, байт
Integer	–32768 ... 32767	2
Real	2,9E – 39 ... 1,7E + 38	6
Char	0…255	1
String	0 … 255 для одного символа	Зависит от длины строки
Boolean	False, True	1
Byte	0 ... 255	1
Word	0 ... 65535	2
Shortint	–128 … 127	1
Longint	–2147483648 … 2147483647	4
Double	5,0E – 324 ... 1,7E + 308	8
Extended	3,4E – 4932 ... 1,1E + 4932	10

Пример программы, вычисляющей площадь квадрата:

```
Program Square;
Uses SysUtils;
Var a, S : Integer;
Begin
  Writeln('Сторона квадрата - ', a);
  Readln(a);
  S:=sqr(a);
  Writeln(S:0:2);
End;
```

2.2. Задания к работе

Составьте блок-схему и программу для вычисления:

а) длины окружности и площади круга радиуса R;

б) длины медианы на сторону a в треугольнике со сторонами a, b, c;

в) длины медианы на сторону b в треугольнике со сторонами a, b, c;
г) длины медианы на сторону c в треугольнике со сторонами a, b, c;
д) длины биссектрисы на сторону a в треугольнике со сторонами a, b, c;
е) длины биссектрисы на сторону b в треугольнике со сторонами a, b, c;
ж) длины биссектрисы на сторону c в треугольнике со сторонами a, b, c;
з) площади треугольника со сторонами a, b, c;
и) площади квадрата с диагоналями d;
к) площади ромба с диагоналями d_1 и d_2;
л) площади трапеции с высотой h и основаниями a и b;
м) площади вписанного четырехугольника со сторонами a, b, c, d (через полупериметры);
н) площади кольца с радиусами R и r;
о) диагонали параллелепипеда с ребрами a, b, c;
п) объема и поверхности параллелепипеда с ребрами a, b, c.

2.3. Контрольные вопросы и задания

1. Основные компоненты программы.
2. Перечислите правила именования идентификаторов.
3. Назовите базовые типы данных и их основные характеристики.

Лабораторная работа № 3. ОПЕРАТОРЫ TURBO DELPHI. МАССИВЫ

Цель работы: *изучить особенности работы основных операторов языка Turbo Delphi и принципы работы с массивами в Turbo Delphi.*

3.1. Краткая теоретическая часть

Операторы языка

Оператор безусловного перехода

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

Общий вид:

```
Goto <метка>;
```

Метка объявляется в разделе описания меток и состоит из имени и следующего за ним двоеточия. Имя метки может содержать цифровые

и буквенные символы, максимальная длина имени ограничена 127 знаками. Раздел описания меток начинается зарезервированным словом *Label*, за которым следует имя метки.

Пример.

```
Program primer;
Label 999, metka;
Begin
  …
  Goto 999;
  …
  999: write ('Имя');
  …
  Goto metka;
  …
  Metka: write('Фамилия');
  …
End.
```

Использование безусловных передач управления в программе считается теоретически избыточным и подвергается критике, так как способствует созданию малопонятных и трудномодифицируемых программ, которые вызывают сложности при отладке. Рекомендуется минимальное использование оператора безусловного перехода с соблюдением следующих правил:

• следует стремиться применять операторы перехода для передачи управления только вниз (вперед) по тексту программы;

• расстояние между меткой и оператором перехода на нее не должно превышать одной страницы текста (или высоты экрана дисплея).

Пустой оператор

Пустой оператор не содержит никаких символов и не выполняет никаких действий. Его используют для организации перехода к концу блока в случаях, если необходимо пропустить несколько операторов, но не выходить из блока. Для этого перед зарезервированным словом *End* ставятся метка и двоеточие, например:

```
Label m;
  …
  Begin
```

```
  …
  Goto m;
  …
  m:
End;
```

Структурные операторы

Структурные операторы представляют собой конструкции, построенные из других операторов по строгим правилам. Их можно разделить на три группы: составные, условные и повтора. Применение структурных операторов в программе очень часто просто незаменимо потому, что они позволяют программисту сделать его программу зависимой от каких-либо условий, например, введенных пользователем значений. К тому же, применяя операторы повтора, вы получаете возможность обрабатывать большие объемы данных за сравнительно малый отрезок времени.

Составной оператор

Этот оператор представляет собой совокупность произвольного числа операторов, отделенных друг от друга точкой с запятой, и ограниченную операторными скобками *Begin* и *End*. Он воспринимается как единое целое и может находиться в любом месте программы, где возможно наличие оператора.

Условные операторы

Условные операторы предназначены для выбора к исполнению одного из возможных действий в зависимости от некоторого условия (при этом одно из действий может отсутствовать). Для программирования ветвящихся алгоритмов в Turbo Delphi существуют специальные операторы. Один из них – условный оператор *If*. Это одно из самых популярных средств, изменяющих порядок выполнения операторов программы.

Он может принимать одну из форм:

```
If <условие> Then <оператор1>
Else <оператор2>;
```

или

```
If <условие> Then <оператор>;
```

Оператор выполняется следующим образом: сначала вычисляется выражение, записанное в условии, в результате чего получается значение логического (булевского) типа. Если это значение – «истина», то выполняется *оператор1*, указанный после слова *Then*. Если же в результате имеем «ложь», то выполняется *оператор2*. В случае, если вместо *оператора1* или *оператора2* следует серия операторов, то ее необходимо заключить в операторные скобки *Begin…End*.

Следует обратить внимание, что перед словом *else* точка с запятой не ставится.

Пример. Необходимо составить программу, которая запрашивает возраст ребенка и затем выдает решение о приеме ребенка в школу (возраст для приема в школу – 7 лет).

```
Program sh;
Var v: Integer;
Begin
Write('Введите возраст ребенка');
Readln(v);
If v>=7 Then Writeln('Принимаем в школу')
Else Writeln ('Не принимаем в школу');
End.
```

Задание. Необходимо модифицировать данную программу, чтобы верхняя граница приема в школу составляла 16 лет.

Решение:

```
Program sh;
Var v: Integer;
Begin
  Write('Введите возраст ребенка');
  Readln(v);
  If (v>=7) and (v<=16)
  Then Writeln('Принимаем в школу')
  Else Writeln ('Не принимаем в школу');
End.
```

Если оператор *If* обеспечивает выбор из двух альтернатив, то существует оператор, который позволяет сделать выбор из произвольного числа вариантов, – это *оператор выбора Case*. Он организует переход на один из нескольких вариантов действий в зависимости от значения выражения, называемого селектором.

Общий вид:

```
Case k Of
  <const1>: <оператор1>;
  <const2>: <оператор2>;
  …
  <constN>: <операторN>
Else <операторN+1>
End;
```

Здесь *k* – выражение-селектор, которое может иметь только простой порядковый тип (целый, символьный, логический). *<const1>*, … *<constN>* – константы того же типа, что и селектор.

Оператор *Case* работает следующим образом: сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна значению селектора, то выполняется оператор, стоящий за словом *Else*. Если же это слово отсутствует, то активизируется оператор, находящийся за границей *Case*, т.е. после слова *End*.

При использовании оператора *Case* должны выполняться следующие правила:

1. Выражение-селектор может иметь только простой порядковый тип (целый, символьный, логический).

2. Все константы, которые предшествуют операторам альтернатив, должны иметь тот же тип, что и селектор.

3. Все константы в альтернативах должны быть уникальны в пределах оператора выбора.

Формы записи оператора:

• *селектор интервального типа:*

```
Case I of
  1..10 : Writeln('число в диапазоне 1-10');
  11.. 20 : Writeln('число в диапазоне 11-20');
  Else  Writeln('число  вне  пределов  нужных
  диапазонов')
End;
```

• *селектор целого типа:*

```
Case I Of
  1 : y:= I+10;
```

```
  2 : y:= I+20;
  3: y:= I +30;
End;
```

Пример. Необходимо составить программу, которая по введенному номеру дня недели выводит на экран его название.

```
Program days;
Var day : Byte;
Begin
  Write('Введите номер дня недели');
  Readln(day);
  Case day Of
    1: Writeln('Понедельник');
    2: Writeln('Вторник');
    3: Writeln('Среда');
    4: Writeln('Четверг');
    5: Writeln('Пятница');
    6: Writeln('Суббота');
    7: Writeln('Воскресенье')
    Else Writeln('Такого дня нет');
  End;
End.
```

Пример. Необходимо составить программу, которая по введенному номеру месяца выводит на экран название времени года.

```
Program m;
Var k:Byte;
Begin
  Write('Введите номер месяца');
  Readln(k);
  Case k Of
    1, 2, 12: Writeln('Зима');
    3, 4, 5: Writeln('Весна');
    6, 7, 8: Writeln('Лето');
    9, 10, 11: Writeln('Осень')
    Else Writeln('Такого месяца нет');
  End;
End.
```

Операторы повтора (цикла)

Если в программе возникает необходимость неоднократного выполнения некоторых операторов, то для этого используются операторы повтора (цикла).

В языке Turbo Delphi различают три вида операторов цикла:

• с предусловием (*While*);

• с постусловием (*Repeat*);

• с параметром (*For*).

Если число требуемых повторений заранее известно, то используется оператор, называемый оператором цикла с параметром.

Оператор цикла с параметром имеет два варианта записи:

```
1) For <имя переменной> := <начальное значение>
To <конечное значение> Do
   <тело цикла>
2) For <имя переменной> := <начальное значение>
Downto <конечное значение> Do
   <тело цикла>
```

Имя переменной – параметр цикла, простая переменная целого типа; тело цикла – оператор(-ы). Цикл повторяется до тех пор, пока значение параметра лежит в интервале между начальным и конечным значениями. В первом варианте при каждом повторении цикла значение параметра увеличивается на единицу, во втором – уменьшается на единицу.

При первом обращении к оператору *For* вначале определяются начальное и конечное значения, а параметру цикла присваивается начальное значение. После этого циклически повторяются следующие действия:

1. Проверяется условие «параметр цикла <= конечному значению».

2. Если условие выполнено, то оператор продолжает работу (выполняется оператор в теле цикла); если условие не выполнено, то оператор завершает работу и управление в программе передается оператору, следующему за циклом.

3. Значение параметра изменяется, и работа повторяется с шага 1.

Если в теле цикла располагается более одного оператора, то они заключаются в операторные скобки *Begin ... End*.

Пример. Необходимо вывести на экран натуральные числа от 1 до 9 в обратном порядке.

```
Program z;
Var i:Integer;
Begin
  For i:=9 Downto 1 Do
  Writeln(i);
End.
```

Если число повторений заранее неизвестно, а задано лишь условие его повторения (или окончания), то используются операторы *While* и *Repeat*. Оператор *While* часто называют оператором цикла с предусловием, так как проверка условия выполнения цикла производится в самом начале оператора.

Общий вид:

```
While <условие> Do
  <тело цикла>;
```

Тело цикла – простой или составной оператор(-ы). Если операторов в теле цикла несколько, то оно заключается в операторные скобки *Begin ... End*.

Перед каждым выполнением тела цикла вычисляется значение выражения условия. Если результат – «истина», тело цикла выполняется и снова вычисляется выражение условия. Если результат – «ложь», происходят выход из цикла и переход к первому после *While* оператору.

Пример. Небходимо найти сумму десяти произвольных чисел.

```
Program z;
Const
  N=10;
Var k, x, s: Integer;
Begin
  k:=0; s:=0; {k- количество введенных чисел}
  While k < n Do
  Begin
    k:=k+1;
    Write('Введите число');
    Readln(x);
    s:=s+x;
  End;
  Writeln('Сумма чисел равна', s);
End.
```

Оператор цикла *Repeat* аналогичен оператору *While*, но отличается от него, во-первых, тем, что условие проверяется после очередного выполнения операторов тела цикла и таким образом гарантируется хотя бы однократное выполнение цикла. Во-вторых, тем, что критерием прекращения цикла является равенство выражения константе *True*. За это данный оператор часто называют циклом с постусловием, так как он прекращает свое действие, как только условие, записанное после слова *Until*, выполнится. Оператор цикла *Repeat* состоит из заголовка, тела и условия окончания.

Общий вид:

```
Repeat
  <оператор>
  . . . . .
  <оператор>
Until <условие окончания цикла>
```

Вначале выполняется тело цикла, затем проверяется условие выхода из цикла. В любом случае этот цикл выполняется хотя бы один раз. Если условие не выполняется, т.е. результатом выражения является *False*, то цикл активизируется еще раз. Если условие выполнено, то происходит выход из цикла. Использования операторных скобок в случае, если тело цикла состоит из нескольких операторов, не требуется.

Пример. Необходимо составить программу, которая вводит и суммирует целые числа. Если введено значение 999, то на экран выводится результат суммирования.

```
Program s;
Var x, s:Integer;
Begin
  S:=0;
  Repeat
    Write('Ввести число');
    Readln(x);
    If x<>999 Then s:=s+x;
  Until x=999;
  Writeln('Сумма введенных чисел', s);
End.
```

Массивы

Массив – упорядоченная последовательность данных одного типа, объединенных под одним именем. Например, результаты многократных замеров температуры воздуха в течение года удобно рассматривать как совокупность вещественных чисел, объединенных в один сложный объект – массив измерений. Проще всего представить себе массив в виде таблицы, где каждое значение находится в собственной ячейке.

Описать переменную-массив можно в разделе описания переменных. Синтаксис:

```
Var <имя массива>: Array [<диапазон индексов>]
Of <тип элементов>;
```

Примеры описания массивов:

```
Var
  S, BB : Array [1..40] Of Real;
  N : Array ['A'..'Z'] Of Integer;
  R : Array [-20..20] Of Word;
  T : Array [1..40] Of Real;
```

Как видим, при описании массива используют зарезервированные слова *Array* и *Of* (*массив, из*). За словом *Array* в квадратных скобках указывается тип-диапазон, с помощью которого компилятор определяет общее число элементов массива. Тип-диапазон задается левой и правой границами изменения индекса массива, так что массивы S, BB, R и T состоят из 40 элементов; массив N имеет индексы символьного типа. В качестве индексных типов в Turbo Delphi можно использовать любые порядковые типы, кроме *Longint*. За словом *Of* указывается тип элементов, образующих массив.

Таким образом можно описать тип массива в разделе описания типов, а затем объявлять переменные такого типа. Синтаксис:

```
Type <имя типа> = Array [<диапазон индексов>] Of
<тип элементов>;
```

Пример:

```
Type Arr = Array[1..20] Of Integer;
Var a, b: Arr;
```

Единственная операция, которую можно применить к массиву целиком, – это присваивание. Например:

```
a:=b;
```

Присваивать можно только массивы одинаковых типов. Так, нельзя будет выполнить не только присваивание *R:=T*, но и *T:=S* (из первого примера), хотя на вид их описание совершенно одинаково.

С массивами чаще всего работают поэлементно. Доступ к каждому элементу массива осуществляется с помощью *индекса*, служащего своеобразным именем элемента массива (если левая граница диапазона индексов массива равна единице, индекс элемента совпадает с его порядковым номером). Для доступа к элементу массива следует после имени массива указать его индекс в квадратных скобках, например *a[5]*. В правильно написанной программе индекс должен соответствовать указанному диапазону индексов (то есть *a[-2]* – неверно). Выход за границы массива вызовет ошибку на этапе компиляции. В качестве индекса может быть указано не только непосредственное значение, но и имя переменной индексного типа, например:

```
Var i: Integer;
  a: Array [-10..10] Of Real;
  ...
  i:=5;
  a[i]:=10.5;
```

Заполнять массивы и выводить их на экран проще всего с помощью циклов. Пример инициализации одномерного массива числами от 1 до 99 с помощью генератора псевдослучайных чисел:

```
Var i:Integer;
  a: Array[1..10] Of Integer;
Begin
  For i:=1 To 10 Do a[i]:=Random(100);
End;
```

Пример программы, использующей одномерные массивы:

```
Program Average;
{Программа создает массив из N случайных целых
чисел, равномерно распределенных в диапазоне от 0
до MAX_VALUE-1, подсчитывает среднее арифметическое
этих чисел,а также минимальное и максимальное из
них.}
Const
N = 1000; {Количество элементов массива}
MAX_VALUE = 100+1; {Диапазон значений случайных
чисел}
Var
  m: Array [1..N] of Integer;  {Массив чисел}
```

```
  i: Integer;                    {Индекс массива}
  max, min: Integer;   {Максимальное и минимальное
  число}
  s: Real;                       {Сумма чисел}
Begin
{Наполняем массив случайными числами:}
  For i := 1 To N Do
    m[i] := Random(MAX_VALUE);
{Задаем начальные значения переменных:}
  s := 0;
  max : = m[1];
  min := m[1];
{Цикл вычисления суммы всех случайных чисел и
поиска минимального и максимального:}
  For i := 1 To N Do
    Begin
      s := s + m [i];
      If m[i] < min Then
        min := m[i]
      Else
        If m[i] > max Then
          max := m[i]
    End;
{Вычисляем среднее значение и печатаем результат:}
  WriteLn('Мин = ', min, ' Макс = ', max,
  'Среднее = ', s/N)
End.
```

Двумерные и многомерные массивы

Представим себе таблицу, состоящую из нескольких строк, каждая из которых состоит из нескольких ячеек. Для точного определения положения каждой ячейки нам потребуется знать номера строки и столбца. Структура, аналогичная такой таблице, – это двумерный массив. Описать такой массив можно двумя способами:

```
Var a: Array[1..20] Of Array[1..30] Of Integer;
Var a: Array[1..20,1..30] Of Integer;
```

Доступ к элементам двумерного массива осуществляется по двум

индексам. Например, ячейка, находящаяся в пятой строке и шестом столбце, будет называться *a[5][6]*.

Проиллюстрируем работу с двумерными массивами на примере программы, которая задает массив из 10 × 10 элементов, инициализирует его случайными числами и считает сумму элементов выше главной диагонали.

При отсчете, начиная с левого верхнего угла таблицы, главной будем считать диагональ из левого верхнего угла в правый нижний. При этом получается, что элементы, лежащие на главной диагонали, будут иметь одинаковые индексы, а у элементов, лежащих выше главной диагонали, номер столбца будет превышать номер строки.

```
Var
i, j, s: Integer; {переменные циклов}
a: Array[1..10,1..10] Of Integer;

Begin
  For I := 1 To 10 Do
  Begin
    For j := 1 To 10 Do
    Begin
    a[i][j]:= Random(101);
    Write(a[i][j]:5);
    If j>i Then s:=s+a[i][j];
    End;
  End;
  Writeln();
  Writeln('sum=', s);
  Readln;
End.
```

3.2. Задания к работе

Вариант 1

Дана целочисленная прямоугольная матрица. Определите:

1) количество строк, не содержащих ни одного нулевого элемента;

2) максимальное из чисел, встречающихся в заданной матрице более одного раза.

Вариант 2

1. Дана целочисленная прямоугольная матрица. Определите количество столбцов, не содержащих ни одного нулевого элемента.

2. Характеристикой строки целочисленной матрицы назовём сумму её положительных чётных элементов. Переставляя строки заданной матрицы, расположите их в соответствии с ростом характеристик.

Вариант 3

Дана целочисленная прямоугольная матрица. Определите:

1) количество столбцов, содержащих хотя бы один нулевой элемент;

2) номер строки, в которой находится самая длинная серия одинаковых элементов.

Вариант 4

Дана целочисленная квадратная матрица. Определите:

1) произведение элементов только в тех строках, которые не содержат отрицательных элементов;

2) максимум среди сумм элементов диагоналей, параллельных главной диагонали матрицы.

Вариант 5

Дана целочисленная квадратная матрица. Определите:

1) сумму элементов в тех столбцах, которые не содержат отрицательных элементов;

2) минимум среди сумм модулей элементов диагоналей, параллельных побочной диагонали матрицы.

Вариант 6

Дана целочисленная прямоугольная матрица. Определите:

1) сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент;

2) номера строк и столбцов всех седловых точек матрицы.

Примечание. Матрица A имеет седловую точку A_{ij}, если A_{ij} является минимальным элементом в i-й строке и максимальным в j-м столбце.

Вариант 7

1. Для заданной матрицы размером 8×8 найдите такие k, что k-я строка матрицы совпадает с k-м столбцом.

2. Найдите сумму элементов в тех строках, которые содержат хотя бы один отрицательный элемент.

Вариант 8

1. Характеристикой столбца целочисленной матрицы назовём

сумму модулей его отрицательных нечётных элементов. Переставляя столбцы заданной матрицы, расположите их в соответствии с ростом характеристик.

2. Найдите сумму элементов в тех столбцах, которые содержат хотя бы один отрицательный элемент.

Вариант 9

1. Соседями элемента A_{ij} в матрице назовём элементы A_{kl} с $i-1 \le k \le i+1$, $j-1 \le l \le j+1$, $(k, l) \ne (i, j)$. Операция сглаживания матрицы даёт новую матрицу того же размера, каждый элемент которой получается как среднее арифметическое имеющихся соседей соответствующего элемента исходной матрицы. Постройте результат сглаживания заданной вещественной матрицы размером 10 × 10.

2. В сглаженной матрице найдите сумму модулей элементов, разложенных ниже главной диагонали.

Вариант 10

1. Элемент матрицы называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитайте количество локальных минимумов заданной матрицы размером 10 × 10.

2. Найдите сумму модулей элементов, расположенных выше главной диагонали.

Вариант 11

1. Коэффициенты системы линейных уравнений заданы в виде прямоугольной матрицы. С помощью допустимых преобразований приведите систему к треугольному виду.

2. Найдите количество строк, среднее арифметическое элементов которых меньше заданной величины.

Вариант 12

1. Уплотните заданную матрицу, удалив из неё строки и столбцы, заполненные нулями.

2. Найдите номер первой из строк, содержащих хотя бы один положительный элемент.

Вариант 13

Осуществите циклический сдвиг элементов прямоугольной матрицы на n элементов вправо или вниз в зависимости от введённого режима (n может быть больше количества элементов в строке или столбце).

Вариант 14

Осуществите циклический сдвиг элементов квадратной матрицы

размерности M × N вправо на *k* элементов таким образом: элементы 1-й строки сдвигаются в последний столбец сверху вниз, из него – в последнюю строку; для остальных элементов – аналогично.

Вариант 15

1. Дана целочисленная прямоугольная матрица. Определите номер первого из столбцов, содержащих хотя бы один нулевой элемент.

2. Характеристикой строки целочисленной матрицы назовём сумму её отрицательных чётных элементов. Переставляя строки заданной матрицы, расположите их в соответствии с убыванием характеристик.

3.3. Контрольные вопросы и задания

1. Перечислите основные типы операторов.
2. Приведите синтаксис и особенности работы условного оператора.
3. Приведите синтаксис и особенности работы циклических операторов.
4. Приведите синтаксис и особенности работы оператора выбора.
5. Приведите синтаксис объявления массива.
6. В чем заключаются особенности нумерации массивов?
7. Способы обращения к элементам массива.
8. Можно ли одним действием скопировать один массив в другой?

Лабораторная работа № 4. РАБОТА С СОСТАВНЫМИ ДАННЫМИ НЕОДНОРОДНОЙ СТРУКТУРЫ. ПОДПРОГРАММЫ

Цель работы: *изучить способы создания записей и методы работы с ними; освоить принципы работы с различными видами подпрограмм.*

4.1. Краткая теоретическая часть

Записи

Запись, в отличие от массивов, множеств и файлов, – составная структура данных. Если отдельно взятый массив, множество или файл всегда включают элементы одинакового типа, то записи могут объединять в единое целое любое число структур данных других типов: простых переменных, массивов, множеств, записей и файлов.

В языке Turbo Delphi различают фиксированные (обычные) и вариантные записи.

Фиксированные записи

Обычная фиксированная запись состоит из одного или нескольких полей, для каждого из которых при объявлении указывается имя (идентификатор) и тип.

```
Type
  ТипЗапись = Record
    Поле1 :Тип1;
    …
    ПолеN :ТипN;
    End;
```

Обращение к полям записей выполняется с помощью квалифицируемых (уточненных) идентификаторов, в которых указывается вся цепочка имен от идентификатора требуемого поля. Имена полей квалифицируемого идентификатора разделяются точками.

```
Type
  TRecord1 = Record
    field1 :Integer;
    field2 :Real;
    End;
  TRecord2 = Record
    field1 :String;
    field2 :TRecord1;
    End;
Var
  rec1 :TRecord1;
  rec2 :TRecord2;
Begin
  rec1.field1 := 10;
  rec1.field2 := 3.14;
  rec2.field1 := 'поле1';
  rec2.field2.field1 := 8;
  rec2.field2.field2 := 6.5;
  {копирование полей записи rec1 в запись rec2.
field2}
  rec2.field2 := rec1;
End.
```

При заполнении информацией структур данных типа «запись» необходимо помнить, что из текстовых файлов (клавиатуре также соот-

ветствует текстовый файл *Input*) допускается вводить данные только некоторых стандартных типов, поэтому в процедурах *Read* и *ReadLn* могут располагаться идентификаторы только самых внутренних полей, которые имеют допустимый для ввода из файла тип.

Вариантные записи

В некоторых случаях целесообразно использовать вариантные записи, в которых так же, как и в фиксированных, описываются поля для всех возможных случаев, однако, альтернативные группы полей разграничиваются более наглядно и память выделяется только под конкретный, необходимый в данный момент вариант.

Вариантные записи включают две части: обычную *фиксированную* запись и *вариантную*, состоящую из *поля признака* и одной или нескольких вариантных компонент, которые включаются в конкретный элемент типа «запись» по альтернативному принципу в зависимости от значения поля признака.

```
Type
  TRecord = Record
    {фиксированная часть}
    field1 :String;
    {вариантная часть}
    Case variant :Integer Of
      1: (field2 :Real;
          field3 :Byte);
      2: (field4 :Integer;
          field5 :Char);
    End;
Var
  Rec :TRecord;
Begin
  {заполнение фиксированной части}
  rec.field1 := 'поле1';
  {заполнение вариантной части}
  rec.variant := 2;
  rec.field4 := 82;
  rec.field5 := 'A';
End.
```

В заключение отметим одно неудобство применения вариантных

записей: в вариантных компонентах таких записей не допускается использование одинаковых идентификаторов.

Оператор присоединения

Для упрощения работы с записями и придания программе большей наглядности в Turbo Pascal имеется специальный оператор присоединения *with*.

```
With ПеременнаяТипаЗапись[, …] Do Оператор
```

Если оператор *with* не использовать, то при обращении к полям записей необходимо указывать полное квалифицируемое имя поля, состоящее из цепочки идентификаторов, разделенных точками.

```
Type
  TRecord1 = Record
    field1 :Integer;
    field2 :Real;
    End;
  TRecord2 = Record
    field1 :String;
    field2 :TRecord1;
    End;
Var
  rec2 :TRecord2;
Begin
  With rec2 Do
    Begin
    field1 := 'поле1';
    With field2 Do
      Begin
      field1 := 19;
      field2 := 3.14;
      End;
    End;
  {компактный вариант}
  With rec2, field2 Do
    Begin
    field1 := 19;
    field2 := 3.14;
    End;
End.
```

Подпрограммы

Подпрограммой называют независимую часть программы, предназначенную для решения некой подзадачи. Подпрограмма взаимодействует с основной программой через механизм параметров – так называют входные и выходные данные, с которыми работает подпрограмма. Однажды написанная подпрограмма, выполненная с теми или иными значениями параметров, может решать некоторый класс задач.

Использование подпрограмм в чем-то похоже на расчеты с использованием математических или физических формул. Так, имея общие формулы решения квадратного уравнения, можно подставить вместо коэффициентов *a*, *b* и *c* любые числовые значения и решить любое конкретное уравнение. Аналогично можно написать подпрограмму с входными параметрами *a*, *b*, *c* и выходными параметрами *x1*, *x2* (найденные корни уравнения), а затем, применяя нужное число раз эту подпрограмму (однократное использование подпрограммы называется ее вызовом), найти корни любого количества квадратных уравнений.

Использование подпрограмм позволяет решить следующие задачи:

• уменьшение размеров кода и экономия памяти за счет возможности неоднократного вызова одной и той же подпрограммы в рамках одной программы;

• лучшее структурирование программы за счет разбиения задачи на более простые подзадачи;

• эффективное повторное использование однажды написанного кода.

Все подпрограммы в языке Turbo Delphi делятся на две группы:

• процедуры;

• функции.

Главное отличие функции от процедуры – то, что функция может возвращать под своим именем какое-либо значение. Процедура этого делать не может.

Общий вид подпрограммы-процедуры:

```
Procedure Имя (Список формальных параметров);
  раздел описаний процедуры;
Begin
  {Тело процедуры}
End;
```

Функции – это подпрограммы, которые могут возвращать под своим именем результирующее значение.

Описание функции состоит из двух частей: заголовка и блока.

```
Function имя функции (формальные параметры):тип
  результата;
  раздел описаний функции
Begin
  исполняемая часть функции
End;
```

Имя функции однозначно идентифицирует данную функцию и используется для ее вызова из основной программы.

Параметры – это необязательная часть функции, содержащая список переменных, которые передаются в функцию из основной программы.

Тип результата показывает, какой тип будет иметь результат выполнения функции. Тип результата может быть любым, за исключением файлового.

Само по себе написание подпрограммы еще не вызывает выполнения никаких действий. Для того чтобы процедура сработала, ее нужно вызвать, записав в нужной точке программы имя процедуры со списком фактических параметров, которые будут подставлены на место формальных.

При каждом вызове подпрограммы значения фактических параметров подставляются на место формальных и с ними производятся вычисления, предусмотренные операторами подпрограммы. Указанные требования называют согласованием параметров и описывают следующим образом: формальные и фактические параметры должны быть согласованы между собой по количеству, типу и порядку следования. Это означает, что количество формальных и фактических параметров должно быть одинаковым, при этом при вызове процедуры каждый фактический параметр должен иметь тот же тип и занимать в списке то же место, что и соответствующий ему формальный параметр.

Для досрочного выхода из подпрограммы и передачи управления основной программы достаточно вызвать процедуру *Exit*. Подпрограммы допускается вызывать из других подпрограмм.

Передача параметров может производиться двумя способами – по значению и по ссылке. Параметры, передаваемые по значению, называют параметрами-значениями, передаваемые по ссылке – параметрами-

переменными. Последние отличаются тем, что в заголовке подпрограммы перед ними ставится служебное слово *var*.

При первом способе (передача по значению) значения фактических параметров копируются в соответствующие формальные параметры. При изменении этих значений в ходе выполнения подпрограммы исходные данные (фактические параметры) измениться не могут. Таким способом передают данные только из вызывающего блока в подпрограмму (т.е. входные параметры). При этом в качестве фактических параметров можно использовать и константы, и переменные, и выражения.

При втором способе (передача по ссылке) все изменения, происходящие в теле подпрограммы с формальными параметрами, приводят к немедленным аналогичным изменениям соответствующих им фактических параметров. Изменения происходят с переменными вызывающего блока, поэтому по ссылке передаются выходные параметры. При вызове соответствующие им фактические параметры могут быть только переменными.

Выбор способа передачи параметров при создании подпрограммы происходит в соответствии с вышеизложенным: входные параметры нужно передавать по значению, а выходные – по ссылке. Практически это сводится к расстановке в заголовке подпрограммы описателя *var* при всех параметрах, которые обозначают результат работы подпрограммы. Однако в связи с тем, что функция возвращает только один результат, в ее заголовке использовать параметры-переменные не рекомендуется.

4.2. Задания к работе

Вариант 1

Опишите запись с именем STUDENT, содержащую следующие поля:

1) фамилия и инициалы;

2) номер группы;

3) успеваемость (массив из пяти элементов).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;

2) вывод на дисплей фамилий и номеров групп для всех студентов, включенных в массив, если средний балл студента больше 4,0;

3) если таких студентов нет, вывести соответствующее сообщение.

Вариант 2

Опишите структуру с именем STUDENT, содержащую следующие поля:

1) фамилия и инициалы;

2) номер группы;

3) успеваемость (массив из пяти элементов).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по возрастанию среднего балла;

2) вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;

3) если таких студентов нет, вывести соответствующее сообщение.

Вариант 3

Опишите структуру с именем STUDENT, содержащую следующие поля:

1) фамилия и инициалы;

2) номер группы;

3) успеваемость (массив из пяти элементов).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из десяти структур типа STUDENT; записи должны быть упорядочены по алфавиту;

2) вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;

3) если таких студентов нет, вывести соответствующее сообщение.

Вариант 4

Опишите структуру с именем AEROFLOT, содержащую следующие поля:

1) название пункта назначения рейса;

2) номер рейса;

3) тип самолета.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;

2) вывод на экран номеров рейсов и типов самолетов, вылетающих

в пункт назначения, название которого совпало с названием, введенным с клавиатуры;

3) Если таких рейсов нет, выдать на экран соответствующее сообщение.

Вариант 5

Опишите структуру с именем AEROFLOT, содержащую следующие поля:

1) название пункта назначения рейса;

2) номер рейса;

3) тип самолета.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из семи элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

2) вывод на экран пунктов назначения и номеров рейсов самолета, тип которого введен с клавиатуры;

3) если таких рейсов нет, выдать на экран соответствующее сообщение.

Вариант 6

Опишите структуру с именем WORKER, содержащую следующие поля:

1) фамилия и инициалы работника;

2) название занимаемой должности;

3) год поступления на работу.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из десяти элементов типа WORKER; записи должны быть размещены по алфавиту;

2) вывод на экран фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;

3) если таких работников нет, выдать на экран соответствующее сообщение.

Вариант 7

Опишите структуру с именем TRAIN, содержащую следующие поля:

1) название пункта назначения;

2) номер поезда;

3) время отправления.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

2) вывод на экран информации о поездах, отправляющихся в указанное время, введенное с клавиатуры;

3) если таких поездов нет, выдать на экран соответствующее сообщение.

Вариант 8

Опишите структуру с именем TRAIN, содержащую следующие поля:

1) название пункта назначения;

2) номер поезда;

3) время отправления.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из шести элементов типа TRAIN; записи должны быть упорядочены по времени отправления поезда;

2) вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры;

3) если таких поездов нет, выдать на экран соответствующее сообщение.

Вариант 9

Опишите структуру с именем TRAIN, содержащую следующие поля:

1) название пункта назначения;

2) номер поезда;

3) время отправления.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа TRAIN; записи должны быть упорядочены по номерам поезда;

2) вывод на экран информации о поезде, номер которого введен с клавиатуры;

3) если таких поездов нет, выдать на экран соответствующее сообщение.

Вариант 10

Опишите структуру с именем MARSH, содержащую следующие поля:

1) название начального пункта маршрута;

2) название конечного пункта маршрута;

3) номер маршрута.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми эле-

ментов типа MARSH; записи должны быть упорядочены по номерам маршрутов;

2) вывод на экран информации о маршруте, номер которого введен с клавиатуры;

3) если таких маршрутов нет, выдать на экран соответствующее сообщение.

Вариант 11

Опишите структуру с именем MARSH, содержащую следующие поля:

1) название начального пункта маршрута;

2) название конечного пункта маршрута;

3) номер маршрута.

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;

2) вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры;

3) если таких маршрутов нет, выдать на экран соответствующее сообщение.

Вариант 12

Опишите структуру с именем NOTE, содержащую следующие поля:

1) фамилия, имя;

2) номер телефона;

3) дата рождения (массив из трех чисел).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE; записи должны быть упорядочены по датам рождения;

2) вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;

3) если такого нет, выдать на экран соответствующее сообщение.

Вариант 13

Опишите структуру с именем NOTE, содержащую следующие поля:

1) фамилия, имя;

2) номер телефона;

3) дата рождения (массив из трех чисел).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE; записи должны быть размещены по алфавиту;

2) вывод на экран информации о людях, чьи дни рождения приходятся на месяц, название которого введено с клавиатуры;

3) если таких нет, выдать на экран соответствующее сообщение.

Вариант 14

Опишите структуру с именем NOTE, содержащую следующие поля:

1) фамилия, имя;

2) номер телефона;

3) дата рождения (массив из трех чисел).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа NOTE; записи должны быть упорядочены по трем первым цифрам номера телефона;

2) вывод на экран информации о человеке, чья фамилия введена с клавиатуры;

3) если такого нет, выдать на экран соответствующее сообщение.

Вариант 15

Опишите структуру с именем ZNAK, содержащую следующие поля:

1) фамилия, имя;

2) знак зодиака;

3) дата рождения (массив из трех чисел).

Напишите программу, выполняющую следующие действия:

1) ввод с клавиатуры данных в массив, состоящий из восьми элементов типа ZNAK; записи должны быть упорядочены по датам рождения;

2) вывод на экран информации о человеке, чья фамилия введена с клавиатуры;

3) если такого нет, выдать на экран соответствующее сообщение.

4.3. Контрольные вопросы и задания

1. Что такое запись?
2. Напишите синтаксис создания записи.
3. Перечислите способы инициализации записи.
4. Перечислите и раскройте способы обращения к полям записи.
5. Опишите синтаксис и работу оператора *With*.
6. Что такое подпрограмма?
7. Опишите синтаксис процедуры и синтаксис функций.
8. В чем основное отличие функции от процедуры?
9. Перечислите и опишите особенности способов передачи параметров в подпрограммы.
10. Что такое фактические и формальные параметры? Назовите основные отличия одних от других.

Лабораторная работа № 5. РЕКУРСИЯ

Цель работы: *изучить принцип и механизм работы рекурсивных функций.*

5.1. Краткая теоретическая часть

Рекурсия – принцип организации программного кода, при котором подпрограмма вызывает себя прямо либо косвенно, путем цепочки вызовов других функций.

Прямой (непосредственной) рекурсией является вызов функции внутри тела этой функции:

```
Integer a()
{.....a().....}
```

Косвенной является рекурсия, осуществляющая рекурсивный вызов функции посредством цепочки вызова других функций. Все функции, входящие в цепочку, тоже считаются рекурсивными, например:

```
a(){.....b().....}
b(){.....c().....}
c(){.....a().....}.
```

Все функции *a*, *b*, *c* – рекурсивные, так как при вызове одной из них осуществляется вызов других и самой себя.

Если функция вызывает себя, в стеке создается копия значений ее параметров, как и при вызове обычной функции, после чего управление передается первому исполняемому оператору функции. При повторном вызове этот процесс повторяется. Ясно, что для завершения вычислений каждая рекурсивная функция должна содержать хотя бы одну нерекурсивную ветвь алгоритма, заканчивающуюся оператором возврата. При завершении функции соответствующая часть стека освобождается, и управление передается вызывающей функции, выполнение которой продолжается с точки, следующей за рекурсивным вызовом.

При написании рекурсивных функций следует использовать оператор условия, чтобы заставить функцию вернуться без рекурсивного вызова. Если это не сделать, то, однажды вызвав функцию, выйти из нее будет невозможно.

Достоинство рекурсии – компактная запись, недостатки – расход времени и памяти на повторные вызовы функции и передачу ей копий параметров и, главное, опасность переполнения стека.

Программист разрабатывает программу, сводя исходную задачу к более простым. Среди этих задач может оказаться и первоначальная, но в упрощенной форме. Например, для вычисления F(N) может понадобиться вычислить F(N-1). Иными словами, частью алгоритма вычисления функции будет вычисление этой же функции.

Пример рекурсивной функции вычисления факториала

```
Function factorial(N: Integer) : Longint;
Begin
  If N= 0 Then
  Factorial := 1
  Else Factorial := factorial(N-1) * N
End;
```

Пример рекурсивной процедуры, возводящей число в степень

```
Procedure Power (X: Real; N: Integer; Var Y:
Real);
Begin
  If N=0 Then
    Y:= 1
  Else Begin Power(X, N-1,Y);
    Y:= Y*X;   End;
End;
```

5.2. Задание к работе

Необходимо реализовать алгоритм сортировки числовых массивов данных методом Хоара.

5.3. Контрольные вопросы и задания

1. Что такое рекурсия?
2. Какие виды рекурсии Вы знаете?
3. Приведите пример рекурсивного алгоритма.
4. Какие проблемы могут возникнуть из-за некорректной организации рекурсивной подпрограммы?

Лабораторная работа № 6. УКАЗАТЕЛИ

Цель работы: *изучить приемы работы с указателями в Turbo Delphi.*

6.1. Краткая теоретическая часть

Основные понятия

Указатель – это какой-либо адрес в памяти компьютера. Это может быть адрес переменной, записи данных либо процедуры или функции. Обычно программисту не важно, где расположен элемент в памяти. Можно просто ссылаться на него по имени, и Turbo Delphi знает, где его нужно искать. Именно это происходит, когда программист описывает переменную. Например, если программа включает в себя следующий код, то компилятору указывается необходимость зарезервировать область в памяти, на которую будут ссылаться по имени *SomeNumber*.

```
Var SomeNumber: Integer;
```

Программисту не нужно беспокоиться о том, где *SomeNumber* находится в памяти. Именно для этого задается имя. Адрес размещения *SomeNumber* в памяти можно найти с помощью операции *@*. *@SomeNumber* – это адрес целочисленной переменной, который можно присвоить переменной-указателю, то есть переменной, содержащей адрес данных или кода в памяти.

Ссылочный тип

Для того чтобы хранить указатели, требуется переменная-указатель, а для ее создания необходим ссылочный тип (или тип "указатель"), простейший вид которого – стандартный тип с именем *Pointer*. Переменная типа *Pointer* – это общий (нетипизированный) указатель, то есть просто адрес. Он не содержит информации о том, на что он указывает. Таким образом, чтобы использовать тот же пример *SomeNumber*, можно присвоить его адрес переменной-указателю:

```
Var
SomeNumber: Integer;
SomeAddress: Pointer;
Begin
  SomeNumber := 17;
{присвоить SomeNumber значение}
  SomeAddress := @SomeNumber;
{присвоить SomeAddress адрес}
```

```
  SomeAddress := Addr(SomeNumber);
{другой способ получения адреса}
End.
```

Типизированные указатели

Обычно определяют ссылочные типы, которые указывают на конкретный вид элемента, например целое значение или запись данных. Можно извлечь преимущество из того факта, что указателю известно, на что он указывает. Для того чтобы определить типизированный указатель, можно описать новый тип, определенный символом каре (^), за которым следуют один или более идентификаторов. Например, чтобы определить указатель на *Integer*, программист может сделать следующее:

```
Type PIneger = ^Integer;
```

Теперь можно описать переменные типа *PInteger*. Если программист не собирается часто использовать ссылочный тип, то можно просто описать переменные как указатели на уже определенный тип. Например, если *PInteger* определен как ^*Integer*, то следующие описания переменной эквивалентны:

```
Var
  X: ^Integer:
  Y: PInteger;
```

Разыменование указателей

Ранее было изложено, как можно присваивать указателям значения, но если нет возможности получить значения обратно, польза от этого невелика. Разыменовав типизированный указатель, можно интерпретировать его так, как если бы это была переменная типа, на которую он указывает. Для того чтобы разыменовать указатель, необходимо поместить символ каре (^) после идентификатора указателя. Ниже показаны некоторые примеры разыменования указателя:

```
Type PInteger = ^Integer;
Var
SomeNumber: Integer;
{ присвоить SomeNumber 17 }
SomeAddress := @SomeNumber;
{ SomeAddress указывает на SomeNumber }
Writeln(SomeNumber);
```

```
{ напечатать 17 }
Writeln(SomeAddress);
{ не допускается; указатели печатать нельзя }
Writeln(SomeAddress^);
{ напечатать 17 }
AnotherAddress := SomeAddress;
{ также указывает на SomeNumber }
AnotehrAddress^ := 99;
{ новое значение для SomeNumber }
Writeln(SomeNumber);
{ напечатать 99 }
End.
```

Наиболее важными строками в приведенном выше примере являются:

```
AnotherAddress := SomeAddress;
{также указывает на SomeNumber }
AnotehrAddress^ := 99;
{ новое значение для SomeNumber }
```

Первый оператор присваивает адрес переменной *AnotherAddress*; он сообщает ей, куда нужно указывать. Второй оператор присваивает новое значение элементу, на который указывает *AnotherAddress*.

Использование указателей

Turbo Delphi предусматривает две пары процедур для выделения и освобождения памяти, распределяемой для динамических переменных. Чаще всего используют процедуры *New* и *Dispose*, которые отвечают большинству потребностей. Процедуры *GetMem* и *FreeMem* выполняют те же функции, но на более низком уровне.

Один из наиболее важных моментов использования указателей – распределение динамических переменных в динамически распределяемой области памяти. Turbo Delphi предусматривает два способа выделения для указателя памяти: процедура *New* и процедура *GetMem*.

New – это очень простая процедура. После описания переменной-указателя программист может вызвать процедуру *New* для выделения пространства в динамически распределяемой памяти для указываемого переменной элемента.

Пример

```
Var
IntPointer: ^Integer;
StringPointer: ^String;
Begin
  New(IntPointer);  {выделяет в динамически распределяемой области два байта}
  New(StringPointer);  {выделяет в динамически распределяемой области 256 байт}
  .
  .
End.
```

После вызова процедуры *New* переменная-указатель указывает на память, выделенную в динамически распределяемой памяти. В данном примере *IntPointer* указывает на двухбайтовую область, выделенную процедурой *New*, а *IntPointer*^ – это допустимая целочисленная переменная (хотя это целочисленное значение еще не определено). Аналогично *StringPointer* указывает на выделенный для строки 256-байтовый блок, а его разыменование дает доступную для использования строковую переменную.

Кроме выделения памяти для конкретной динамической переменной вы можете использовать *New* как функцию, возвращающую указатель конкретного типа. Например, если *PInteger* – это тип, определенный как ^*Integer*, а *IntPointer* имеет тип *PInteger*, то следующие два оператора эквивалентны:

```
New(IntPointer);
IntPointer := New(PInteger);
```

Это особенно полезно в случаях, когда может потребоваться присваивать переменной-указателю элементы различных типов. Иногда желательно распределять динамическую переменную, не присваивая явно указатель конкретной переменной. Вероятно, можно это сделать только создав для процедуры и функции параметр:

```
SomeProcedure(New(PointerType));
```

В этом случае *SomeProcedure* будет добавлять передаваемый параметр к некоторому списку. В противном случае распределяемая память будет потеряна. Библиотеки Turbo Delphi широко используют этот метод для присваивания динамических объектов спискам.

Память, распределенная для переменных с помощью *New*, после завершения работы с ними должна освобождаться. Это позволит использовать динамически распределяемую память для других переменных. Для того чтобы освободить память, выделенную для динамической переменной, программист должен использовать процедуру *Dispose*.

Иногда нежелательно выделять память тем способом, как это делает *New*. Может потребоваться выделить больше или меньше памяти, чем это делает *New* по умолчанию, либо до начала выполнения программист может просто не знать, сколько памяти нужно использовать. Turbo Delphi выполняет такое распределение с помощью процедуры *GetMem*. Процедура *GetMem* воспринимает два параметра: переменную-указатель, для которой необходимо распределить память, и число распределяемых байт.

Аналогично тому, как требуется освобождать память, выделенную с помощью *New*, нужно освобождать память, распределенную с помощью процедуры *GetMem*. Это можно сделать с помощью процедуры *FreeMem*. Аналогично тому, как каждому вызову *New* должен соответствовать парный вызов *Dispose*, каждому вызову процедуры *GetMem* должен соответствовать вызов *FreeMem*. Как и *GetMem*, процедура *FreeMem* воспринимает два параметра: освобождаемую переменную и объем освобождаемой памяти. Важно, чтобы ее объем точно совпадал с объемом выделенной памяти. *New* и *Dispose*, основываясь на типе указателя, всегда знают, сколько байт нужно выделять или освобождать. Но в случае *GetMem* и *FreeMem* объем выделяемой памяти находится всецело под контролем программиста. Если освободить меньше байт, чем было выделено, то оставшиеся байты теряются (происходит «утечка» динамически распределяемой памяти). Если освободить большее число байт, чем было выделено, то можно освободить память, распределенную для другой переменной, что может привести к порче данных. В защищенном режиме освобождение большего объема памяти, чем было выделено, вызовет ошибку по нарушению защиты (*GP*).

В Turbo Delphi определены две функции, возвращающие важную информацию о динамически распределяемой области памяти: *MemAvail* и *MaxAvail*. Функция *MemAvail* возвращает общее число байт, доступных для распределения в динамической памяти. Перед выделением большого объема в динамически распределяемой памяти полезно убедиться, что такой объем памяти доступен. Функция *MaxAvail* возвращает

размер наибольшего доступного блока непрерывной памяти в динамически распределяемой области. Первоначально при запуске программы *MaxAvail* равно *MemAvail*, поскольку вся динамически распределяемая область памяти доступна и непрерывна. После распределения нескольких блоков памяти пространство в динамически распределяемой области, скорее всего, станет фрагментированным. Это означает, что между частями свободного пространства имеются распределенные блоки. Функция *MaxAvail* возвращает размер наибольшего свободного блока.

Общие проблемы использования указателей

Указатели позволяют делать в Turbo Delphi некоторые важные вещи, но есть пара моментов, которые при работе с указателями нужно отслеживать. При использовании указателей допускаются следующие общие ошибки:

- разыменование неинициализированных указателей;
- потери динамически распределяемой памяти ("утечки").

Один из общих источников ошибок при работе с указателями – разыменование указателя, который еще не был инициализирован. Как и в случае других переменных Turbo Delphi, значение переменной-указателя не будет определено, пока ей не присвоено значение, так что она сможет указывать на какой-то адрес в памяти. Перед использованием указателей им всегда нужно присваивать значения. Если программист разыменовывает указатель, которому еще не присвоено значение, то считанные из него данные могут представлять собой случайные биты, а присваивание значения указываемому элементу может «затереть» другие данные, вашу программу или даже операционную систему. Это звучит несколько пугающе, но при определенной дисциплине такие вещи легко отслеживаются.

Для того чтобы избежать разыменования указателей, которые не указывают на что-либо значащее, нужен некоторый способ информирования о том, что указатель недопустим. В Turbo Delphi предусмотрено зарезервированное слово *Nil*, которое вы можете использовать в качестве содержательного значения указателей, которые в данный момент ни на что не указывают. Указатель *Nil* является допустимым, но ни с чем не связанным. Перед разыменованием указателя вы должны убедиться, что он отличен от *Nil* (не пуст).

При использовании динамически распределяемых переменных ча-

сто возникает общая проблема, называемая утечкой динамической памяти – это ситуация, когда пространство выделяется в динамически распределяемой памяти и затем теряется, т.е. по каким-то причинам указатель не указывает больше на распределенную область, поэтому нет возможности освободить пространство. Общая причина утечек памяти – переприсваивание динамических переменных без освобождения предыдущих.

Пример

```
Var IntPointer: ^Integer;
Begin
  New(IntPointer);
  New(IntPointer);
End.
```

При первом вызове *New* в динамически распределяемой памяти выделяется 8 байт, и на них устанавливается указатель *IntPointer*. Второй вызов *New* выделяет другие 8 байт, и *IntPointer* устанавливается на них. Теперь нет указателя, ссылающегося на первые 8 байт, поэтому нет возможности их освободить. В программе эти байты будут потеряны.

Естественно, утечка памяти может быть не такой очевидной, как в указанном примере. Выделение памяти почти никогда не происходит в последовательных операторах, но может выполняться в отдельных процедурах или далеко отстоящих друг от друга частях программы. В любом случае лучший способ отслеживания динамических переменных – это установка их в *Nil* при освобождении. Тогда при попытке распределить их снова пользователь может убедиться, что они имеют значение *Nil*:

```
Var IntPointer: ^Integer;
Begin
  New(IntPointer);
  .
  .
  .
  Dispose(IntPointer);
  IntPointer := Nil;
  .
  .
  .
  If IntPointer = Nil Then New(IntPointer);
End.
```

6.2. Задание к работе

Необходимо реализовать сортировку массива записей из лабораторной работы № 4 по одному из строковых полей с использованием дополнительной структуры данных (массива указателей) без перестановки самих элементов массива, а путем перенастройки указателей.

6.3. Контрольные вопросы и задания

1. Что такое указатель?
2. Воспроизведите синтаксис объявления указателя.
3. Какие типы указателей вы знаете?
4. Что представляет собой операция разыменования?
5. Какие существуют способы выделения памяти?
6. Опишите принцип действия и синтаксис функций *MemAvail* и *MaxAvail*.
7. Какие существуют проблемы, связанные с использованием указателей?

Лабораторная работа № 7. ФАЙЛЫ

Цель работы: *изучить особенности работы с файлами в Turbo Delphi.*

7.1. Краткая теоретическая часть

Понятие физического и логического файлов

У понятия «файл» есть два значения. С одной стороны, *файл* – это именованная область внешней памяти, содержащая какую-либо информацию. Файл в таком понимании называют *физическим файлом*, то есть существующим физически на некотором материальном носителе информации. С другой стороны, *файл* – это одна из многих структур данных, используемых в программировании. Файлы в таком понимании называют *логическими*, то есть существующими только в нашем логическом представлении при написании програм-

мы, в которой они представляются файловыми переменными определенного типа.

Структура физического файла представляет собой простую последовательность байт памяти носителя информации. Структура логического файла – это способ восприятия файла в программе (рис. 7.1). Образно говоря, это «шаблон» («окно»), через который мы смотрим на физическую структуру файла. В языках программирования таким «шаблонам» соответствуют типы данных, допустимые в качестве компонент файлов.

`File of` *`Тип`*`, n = SizeOf(`*`Тип`*`)`

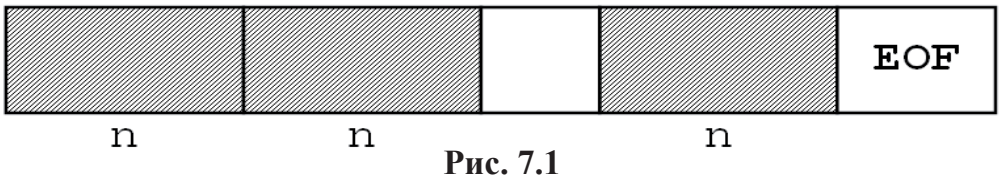

Рис. 7.1

Логическая структура файла в принципе очень похожа на структуру массива, но есть и различия.

У массива количество элементов фиксируется в момент распределения памяти, и он целиком располагается в оперативной памяти. Нумерация элементов массива выполняется соответственно нижней и верхней границам, указанным при его обновлении.

У файла количество элементов в процессе работы программы может изменяться; он располагается на внешних носителях информации. Нумерация элементов файла выполняется слева направо, начиная от нуля (кроме текстовых файлов). Количество элементов файла в каждый момент времени неизвестно. Зато известно, что в конце файла располагается специальный символ конца файла *EOF*, в качестве которого используется управляющий символ ASCII с кодом 26 (Ctrl + Z). Кроме того, определить длину файла и выполнить другие часто требуемые операции можно с помощью стандартных процедур и функций, предназначенных для работы с файлами.

Классификация файлов в Turbo Delphi

Файлы в Turbo Delphi классифицируются по двум признакам (рис 7.2): по типу (логической структуре) и по методу доступа к элементам фай-

ла. Допустимость применения методов доступа к каждой разновидности файлов по типу показана стрелками (см. рис. 7.2).

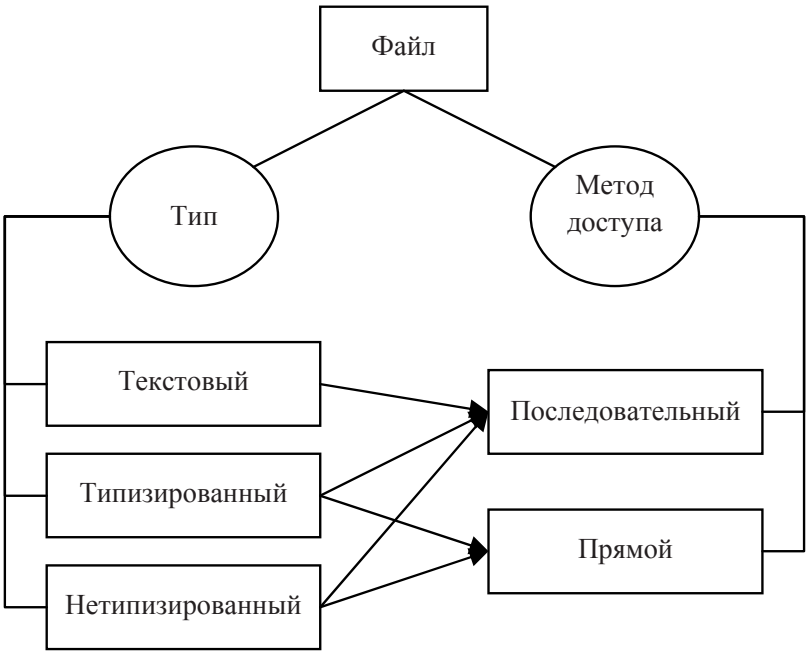

Рис. 7.2. Классификация файлов в Turbo Delphi

Назначение, открытие и закрытие файлов

Для работы с каким-либо физическим файлом, находящимся на диске, необходимо первоначально связать его файловой переменной (логическим файлом), с помощью которой будет осуществляться доступ к этому физическому файлу. Связывание логического и физического файлов выполняется процедурой *Assign*, которая может использоваться только для закрытого файла. Первым параметром этой процедуры является файловая переменная, а вторым – строковая константа или идентификатор строковой переменной, значением которых должно быть имя физического файла, указанное согласно правилам записи идентификаторов в *MS-DOS*:

```
Assign(ФайловаяПеременная, ИмяФайла);
```

Перед выполнением каких-либо операций чтения и записи в файлах, эти файлы должны быть открыты. Открытие файлов выполняется процедурами *Reset* и *Rewrite*, а закрытие – процедурой *Close*.

```
Reset(ФайловаяПеременная);
Rewrite(ФайловаяПеременная);
Close(ФайловаяПеременная);
```

Процедура *Reset* открывает существующий физический файл, который был связан с файловой переменной. Если открыт текстовый файл,

то он будет доступен только для чтения при последовательном доступе к элементам, если открыт типизированный файл, то он будет открыт и для чтения, и для записи как при последовательном доступе, так и при прямом. При открытии указатель текущей позиции файла устанавливается в его начало.

Если физический файл с указанным именем отсутствует, то возникает ошибка времени исполнения, которую можно подавить выключением директивы компилятора {$I-}. При такой установке директивы можно проанализировать результат завершения операции открытия файла с помощью функции *IOResult*, которая возвращает значение ноль, если операция завершилась успешно, и ненулевой код ошибки в противном случае.

Процедура *Rewrite* создает новый физический файл, имя которого связано с файловой переменной. Если такой физический файл уже существует, то он удаляется, а на его месте создается новый пустой файл. При открытии указатель текущей позиции в файле устанавливается в его начало.

Еще одной функцией, используемой практически во всех программах, является функция *Eof*.

```
Eof(ФайловаяПеременная)
```

Функция возвращает значение *True*, если указатель текущей позиции находится за последним элементом файла или если файл пуст. В противном случае она возвращает значение *False*.

Типизированные файлы

Все элементы типизированного файла должны быть одного типа. Типизированные файлы могут быть любого типа, кроме файлового и опирающегося на файловый:

```
Var ФайловаяПеременная :File Of Тип;
```

Типизированные файлы допускают как последовательный, так и прямой доступ. Работая с прямым доступом, следует помнить, что элементы типизированных файлов всегда нумеруются, начиная с нуля.

Чтение из типизированных файлов выполняется только процедурой *Read*, а запись – только процедурой *Write*:

```
Write(ФайловаяПеременная, СписокПеременных);
Read(ФайловаяПеременная, СписокПеременных);
```

При этом единицей чтения/записи может быть только переменная того же типа, что и тип файла

При считывании в каждую переменную из списка процедуры *Read*, указатель текущей позиции в файле перемещается на следующий элемент. Если указатель текущей позиции файла находится за последним элементом, то есть в конце файла, то выполнение процедуры *Read* приводит к ошибке времени выполнения.

Выполняя запись в файл, следует помнить, что при записи каждой переменной указатель текущей позиции в файле, так же как и при чтении, перемещается на следующий элемент. Если указатель текущей позиции файла находится за последним элементом, то есть в конце файла, то при выполнении процедуры *Write* файл расширяется.

Для работы с прямым доступом предназначены следующие процедуры и функции:

• *FilePos* – возвращает номер текущей позиции в файле (от нуля);

• *FileSize* – возвращает текущий размер файла;

• *Seek* – перемещает указатель текущей позиции в файле на элемент с заданным номером (от нуля);

• *Truncate* – усекает размер файла до текущей позиции. Все элементы, расположенные после текущей позиции в файле, удаляются, и текущая позиция становится его концом.

Текстовые файлы

Особый вид файлов представляют собой текстовые файлы, которые в Turbo Delphi являются разновидностью файлов типа *File Of Char*. Для описания текстовых файлов используют предопределенный тип *Text*.

```
Var ФайловаяПеременная :Text;
```

В текстовых файлах помимо признака конца файла *EOF* используют признак конца строки *EOLn*, который представляет собой последовательность из двух символов кода ASCII – символа с кодом 13 («возврат каретки») и символа с кодом 10 («перевод строки»). Текстовый файл можно представить как страницу книги, в конце каждой строки которой стоит *EOLn*.

Напомним, что стандартные файлы ввода-вывода *Input* (ввод с клавиатуры) и *Output* (вывод на дисплей) – текстовые, поэтому использование процедур стандартного ввода-вывода *Read*, *ReadLn*, *Write*, *WriteLn* для текстовых файлов будет практически таким же. Отличие состоит в

том, что первым параметром этих процедур должна быть указана файловая переменная:

```
Read(ФайловаяПеременная, СписокПеременных);
Write(ФайловаяПеременная, СписокПеременных);
ReadLn(ФайловаяПеременная, СписокПеременных);
WriteLn(ФайловаяПеременная, СписокПеременных);
```

Для текстовых файлов дополнительно к общим допускается использование следующих процедур и функций:

• *Append* – открывает существующий файл для добавления элементов в конец файла;

• *Flush* – сбрасывает для текстового файла буфер вывода;

• *SeekEof* – возвращает для текстового файла состояние *EOF*;

• *SeekEoln* – возвращает для текстового файла состояние *EOLn*;

• *SetTextBuf* – назначает для текстового файла буфер ввода-вывода.

Нетипизированные файлы

При объявлении нетипизированного файла указывается только ключевое слово *file*:

```
Var ФайловаяПеременная :File;
```

Нетипизированные файловые переменные предназначены для низкоуровневой работы с файлами. С их помощью можно обратиться к файлу любого типа и логической структуры, аналогично тому, как выполняется обращение к символьному файлу посредством файловой переменной типа *Byte*. Отличие состоит в том, что нетипизированный файл не имеет жестко установленной единицы чтения/записи, как типизированные файлы. В нетипизированных файлах за одно обращение считывается/ записывается число байт, приблизительно равное величине буфера ввода/вывода, что способствует увеличению скорости работы с файлами. В качестве буфера ввода/вывода нетипизированных файлов может выступать любая переменная.

Для работы с нетипизированными файлами можно применять все процедуры и функции, что и для типизированных. Исключение составляет то, что вместо процедур *Read* и *Write* используют процедуры *BlockRead* и *BlockWrite*, а процедуры *Reset* и *Rewrite* могут иметь вто-

рой параметр типа *Word*, который определяет размер записи, используемой при передаче данных. Если этот параметр опущен, то размер записи принимается по умолчанию равным 128 байтам.

7.2. Задания к работе

Задание 1

Разработайте программу копирования файла, используя нетипизированный файл.

Задание 2

Вариант 1

Напишите программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке.

Вариант 2

Напишите программу, которая считывает текст из файла и выводит на экран только те предложения, которые содержат выведенное с клавиатуры слово.

Вариант 3

Напишите программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа.

Вариант 4

Напишите программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв.

Вариант 5

Напишите программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова.

Вариант 6

Напишите программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых.

Вариант 7

Напишите программу, которая считывает текст из файла и определяет, сколько в нём слов, состоящих из четырёх и менее букв.

Вариант 8

Напишите программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключённые в кавычки.

Вариант 9

Напишите программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов.

Вариант 10

Напишите программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся или оканчивающиеся на гласные буквы.

Вариант 11

Напишите программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел.

Вариант 12

Напишите программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы.

Вариант 13

Напишите программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную.

Вариант 14

Напишите программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова “ноль”, “один”, и т.д., начиная каждое предложение с новой строки.

Вариант 15

Напишите программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте.

7.3. Контрольные вопросы и задания

1. Дайте определения логического и физического файлов.
2. Воспроизведите классификацию файлов в Turbo Delphi по методу доступа и типу.
3. Назначение и принцип работы процедуры *Assign.*
4. Назначение и принцип работы процедур *Reset* и *Rewrite.*
5. Назначение и принцип работы функции *Eof.*
6. В чем различие типизированных и нетипизированных файлов?
7. В чем особенность работы с текстовыми файлами?
8. В чем особенность работы с нетипизированными файлами?

Лабораторная работа № 8. МНОГОМОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Цель работы: *изучить основы многомодульного программирования на языке Turbo Delphi*.

8.1. Краткая теоретическая часть

Применение готовых и разработанных программистом модулей позволяет эффективно решать задачу повторного использования однажды написанного кода.

Модулями называют заранее скомпилированные библиотеки подпрограмм, которые программист может использовать для создания новых программ.

Подробно рассмотрим общую структуру модуля:

```
Unit ИмяМодуля;
```

Модуль открывается заголовком, именующим его. По этому имени модуль может быть подключен из программы оператором *uses ИмяМодуля;*. Имена составляются по обычным для языка правилам.

Ключевым словом *Interface* открывается интерфейсная часть, в которой объявляются константы, типы данных, переменные, процедуры и функции модуля. Тела общих процедур и функций находятся в разделе реализации.

Раздел интерфейса является общим. В нем можно определить то, что будет видимо и доступно для любой другой программы (или модуля), использующей данный модуль. В интерфейсной части может находиться раздел *Uses*, если модуль подключает другие модули. В таком случае ключевое слово *uses* должно следовать сразу за словом *Interface*.

В разделе *Implementation* реализации модуля находятся тела процедур и функций, объявленных в интерфейсной части. Раздел реализации является частным. Все объявления, сделанные здесь, могут быть видимы только внутри данного раздела модуля. При этом все константы, типы, переменные, процедуры и функции, объявленные в интерфейсной части, видимы и в разделе реализации.

В разделе реализации могут находиться собственные дополнительные объявления модуля, невидимые любыми программами, использующими его.

Раздел *Uses* может находиться в части реализации сразу после зарезервированного слова *Implementation*.

Заголовки процедур или функций в разделе реализации должны соответствовать их объявлениям в разделе интерфейса.

Наконец, главная программа модуля, ограниченная операторными скобками *Finalization ... End*., обычно пуста. Тем не менее в ней можно давать начальные значения данным модуля, открывать используемые им файлы, если таковые есть, и т.п.

В качестве примера создадим простейший модуль для работы с координатами точек на плоскости и вызовем его подпрограммы при написании новой программы:

```
Unit points;
Interface
  Type point = Array [1..2] Of Real;
  Procedure put (Var p:point;x,y:Real);
  Function distance (p1,p2:point):Real;
  Function corner (p1:point):Integer;
Implementation
  Procedure put (Var p:point;x,y:Real);
  Begin
    p[1]:=x; p[2]:=y;
  End;
  Function distance (p1,p2:point):Real;
  Begin
    distance:=sqrt(sqr(p1[1]-p2[1])+sqr(p1[2]-
p2[2]));
  End;
  Function corner (p1:point):Integer;
  Begin
    If p1[1]>0 Then Begin
      If p1[2]>0 Then corner:=1
      Else
      If p1[2]<0 Then corner:=4
      Else corner:=0;
    End
    Else If p1[1]<0 Then Begin
      If p1[2]>0 Then corner:=2
      Else If p1[2]<0 Then corner:=3
      Else corner:=0;
    End
    Else corner:=0;
```

```
  End;
Finalization
End.
```

Наш модуль определяет тип данных *Point* («точка») как массив из двух вещественных чисел. Процедура *put* позволяет задать для точки значения *x*- и *y*-координаты, функция *distance* возвращает расстояние между двумя точками, а функция *corner* – номер координатной четверти, в которой находится точка, или 0, если точка лежит на одной из осей координат. Разумеется, реальные модули могут включать сотни функций, если предметная область, которую они моделируют, достаточно сложна. Теперь напишем небольшую тестовую программу, использующую наш модуль:

```
Uses points;
Var a,b:Point;
Begin
  put (a,1,1);
  put(b,0,0);
  Writeln('Расстояние от A до B=',
    distance(a,b):8:3);
  Writeln ('Номер четверти для A=',
    corner(a));
End.
```

Оператор *Uses*, подключающий модуль, указан в первой строке программы. Во время компиляции этой программы в текущем каталоге должен присутствовать файл *points.tpu*, содержащий созданный ранее модуль *points*.

Модули *System* и *SysInit* используются автоматически каждым приложением и не могут быть перечислены в предложении использований. Другие стандартные модули библиотеки типа *SysUtils* должны быть явно включены в предложение использований.

При использовании модулей, для избегания проблем со ссылками модуля необходимо обратиться к источнику файла модуля:

```
Uses MyUnit in "myunit.pas";
```

Если такая явная ссылка появляется в проектном файле, другие исходные файлы могут обратиться к модулю с простыми использованиями предложения, который не должен соответствовать случаю:

```
Uses MyUnit;
```

Для того чтобы использовать любой из модулей (написанных программистом или готовых), его достаточно подключить оператором *Uses*.

8.2. Задание к работе

В модуль поместите функции сортировки символьных и числовых массивов. Подключите созданный модуль и примените эти функции к вводимым с клавиатуры данным.

8.3. Контрольные вопросы и задания

1. Что такое модуль?
2. Синтаксис создания и подключения модулей.

Лабораторная работа № 9. ОСНОВЫ ОБЪЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ (ООП)

Цель работы: *знакомство с основами ООП, изучение синтаксических правил описания классов, создания и использования объектов.*

9.1. Краткая теоретическая часть

Класс – это тип данных, определяемый пользователем, включающий в себя данные, процедуры и функции для работы с ними.

Данные класса называют *полями*, а процедуры и функции – «*методами*». Поля и методы называют *элементами* класса.

Класс объявляется с областью видимости «программа» или модуль, до того как будут объявлены переменные этого типа, которые называются *объектами*.

Синтаксис объявления класса:

```
Type
<имя класса > = Class (<имя класса родителя>)
 Public   {доступно всем}
 < поля, методы, свойства, события>
Protected {доступно только потомкам}
 < поля, методы, свойства, события >
Private {только  внутри класса}
<поля, методы, свойства, события.>
End;
```

Имя класса может быть любым допустимым идентификатором. Имя класса родителя может не указываться, если данный класс – наследник только класса *TObject*. Доступ к элементам класса определяется спецификаторами доступа *Public, Private, Protected.*

Public (открытый) – объявленные в этом разделе элементы доступны для внешнего использования.

Private (закрытый) – содержит объявление полей и методов, доступных только внутри данного класса.

Protected (защищенный) – содержит объявления, доступные только для потомков объявляемого класса.

Исходя из принципа инкапсуляции, поля данных следует объявлять в разделе *Private* либо *Protected*, доступ к ним должен осуществляться через свойства, включающие методы чтения и записи полей.

Обычно имена полей совпадают с именами соответствующих свойств, но с добавлением префикса *F*.

Свойство объявляется следующим образом:

```
Property <имя свойства> : <тип>
Read <имя поля или метода чтения>
Write <имя поля или метода чтения>;
```

Если в разделах *Read* или *Write* указано имя поля, предполагается прямое чтение или запись данных.

Если в разделе *Read* указано имя метода, то чтение будет осуществляться только функцией с этим именем.

Функция чтения – это функция без параметров, возвращающая значение типа, объявленного для свойства. Обычно имя этой функции состоит из префикса *Get* и имени свойства.

Если в разделе *Write* значится имя метода, то запись будет осуществляться только процедурой с этим именем.

Процедура записи – это процедура с одним параметром, типа, объявленного для свойства. Имя этой процедуры строится из префикса *Set* и имени свойства.

Если раздел *Write* при описании свойства опустить, то данное свойство будет свойством только для чтения.

Метод, создающий и инициализирующий объект, носит название *конструктор;* создание объекта заключается в выделении для него об-

ласти в динамической памяти. Объявление конструктора предваряется ключевым словом *Constructor*, в качестве его имени обычно используют *Create***,** а в остальном объявление не отличается от объявления обычного метода.

```
Constructor Create;
```

Методы, уничтожающие объекты и освобождающие занимаемую ими память, называют *деструкторами*. Объявление деструктора предваряется ключевым словом *Destructor*, в качестве имени деструктора обычно задают имя *Destroy*:

```
Destructor Destroy();
```

9.2. Задание к работе

Опишите класс *Complex*, содержащий закрытые поля, хранящие *Re* и *Im* части числа В классе должны содержаться методы: *Constructor*, *Destructor*, вывода на экран, расчета суммы двух комплексных чисел. Создайте объект данного класса; продемонстрируйте работу методов класса.

9.3. Контрольные вопросы и задания

1. Синтаксис объявления класса.
2. Спецификаторы доступа.
3. Понятие инкапсуляции.
4. Назначение конструкторов и деструкторов.

Лабораторная работа № 10. ВИЗУАЛЬНЫЕ КОМПОНЕНТЫ TURBO DELPHI

Цель работы: *изучение основных визуальных компонент ввода и отображения текстовой информации в Turbo Delphi.*

10.1. Краткая теоретическая часть

Библиотека Turbo Delphi содержит множество визуальных компонентов, позволяющих отображать, вводить с клавиатуры и редактировать текстовую информацию (таблица).

Визуальные компоненты Turbo Delphi

Компонент	Основное свойство
Label	Метка. Позволяет отображать текст, неизменяемый пользователем; основное свойство Caption
StaticText	Отличается от компонента Label только возможностью задания стиля бордюра
Panel	Контейнер для размещения и группировки компонент; может быть использован для отображения информации; основное свойство Caption
Edit	Отображение, ввод и редактирование однострочных текстов; основное свойство – Text
Memo	Отображение, ввод и редактирование многострочных текстов; основное свойство – Lines
RichEdit	Отображение, ввод и редактирование многострочных текстов; основное свойство – Lines
ListBox	Список с возможностью выбора пунктов; основное свойство – Items
ComboBox	Компонент объединяет в себе возможности ListBox и Edit
StringGrid	Отображение текстовой информации в виде двумерной таблицы строк; основное свойство Cells

Доступ к свойствам и методам объектов можно осуществлять на шаге проектирования с помощью инспектора объектов или программно. Для этого необходимо указать имя свойства объекта после имени самого объекта, разделив их точкой без пробелов:

```
<имя объекта>.<имя свойства>
Label1.Caption:='Ok';
```

Для хранения и отображения строковой информации в виде таблицы используют компонент *StringGrid*, расположенный на странице *Additional*.

Основные свойства этого компонента:

• *Cells[ACol, A Row:Integer]:String* – строка, содержащаяся в ячейке с соответствующими индексами;

• *Cols[indexs: Integer]:TStrings* – список строк, содержащихся в столбце с соответствующим индексом;

• *Rows[indexs: Integer]:TStrings* – список строк, содержащихся в строке с соответствующим индексом;

• *ColCount*, *RowCount* определяют число столбцов и строк, соответственно;

• *FixedCols* и *FixedRows* – число фиксированных, непрокручевыемых столбцов и строк; *FixedColor* задает их цвет.

Свойство *ScrollBars,* установленное в значение *True,* определяет автоматическое добавление полос прокрутки в случае, если таблица не помещается в заданные размеры.

Свойство *Options* включает в себя множество свойств таблицы (наличие или отсутствие линий, разделяющих ячейки, возможность пользователя изменять размеры строк и столбцов при помощи мыши, перемещать столбцы и строки), его подсвойство *goEditing* дает возможность пользователю редактировать данные. Важное событие этого компонента – событие *OnSelectCells*, которое происходит при выборе пользователем ячейки. В обработчик этого события передаются целые параметры *ACol*, *ARow* – столбец и строка – и булевский параметр *CanSelect* – допустимость выбора, если задать его значение *False*, выделение ячейки будет запрещено.

Примеры использования параметров:

```
Label1.Caption:='выбрана ячейка №' + IntToStr
(ARow)+IntToStr(ACol); - номер выбранной ячейки
Label1.Caption:=StrindGrid1.Cells[ACol,ARow]; -
текст из выделенной ячейки.
```

Определенные действия пользователя – перемещение курсора мыши, нажатие кнопок – вызывают соответствующие *события*. Например, при щелчке кнопкой мыши возникает событие *OnClick*. Для каждого компонента определено множество событий, в конкретных экземплярах этого компонента могут быть написаны обработчики тех событий, на которые должен реагировать данный объект. Для этого необходимо выбрать в инспекторе объектов на странице *Events* нужное событие, двойной щелчок по которому приведет к автоматическому добавлению в программный модуль необходимых строк кода, останется только запрограммировать необходимые действия.

10.2. Задание к работе

Разместите на главной форме проекта компонент *StringGrid*, один или несколько управляющих компонентов, напишите обработчик события *OnClick* кнопки (либо другого события любого управляющего элемента), заполняющего таблицу следующим образом:

1) элемент равен произведению квадратов своих индексов, если

сумма индексов – четное число, и разности квадратов индексов, если сумма индексов нечетное число;

2) элементы главной диагонали равны сумме своих индексов, остальные элементы равны сумме квадратов своих индексов;

3) элемент равен квадрату разности своих индексов;

4) элементы на главной диагонали равны сумме остальных элементов строки, которые заполняются случайным образом;

5) элементы заполняются случайным образом; элементы на главной диагонали равны сумме остальных элементов столбца;

6) элементы главной диагонали заполняются случайным образом, элементы над главной диагональю равны сумме квадратов индексов и элемента главной диагонали, элементы под главной диагональю равны разности суммы квадратов индексов и элемента главной диагонали;

7) элемент равен сумме своих индексов, если это число четное, и их разности, если сумма нечетная;

8) элементы главной диагонали равны квадрату суммы своих индексов, остальные элементы равны сумме своих индексов;

9) элемент равен произведению квадратов своих индексов, если сумма соответствующих индексов – четное число, и разности квадратов индексов, если сумма индексов нечетная;

10) элементы главной диагонали заполняются случайным образом, остальные равны разности квадратов суммы своих индексов и элемента главной диагонали;

11) если сумма индексов элемента – четное число, то элемент заполняется случайным образом, в противном случае он равен разности квадратов индексов;

12) элемент равен сумме квадратов своих индексов, если сумма индексов – четное число, и разности квадратов своих индексов, если сумма нечетная;

13) элемент равен разности квадратов своих индексов.

10.3. Контрольные вопросы и задания

1. Основные компоненты ввода и отображения текстовой информации в Turbo Delphi.

2. Способы доступа к свойствам и методам объектов.

3. Основные свойства компонента *TStringGrid*.

4. События.

Лабораторная работа № 11. РАБОТА С ГРАФИКОЙ В TURBO DELPHI

Цель работы: *знакомство с основными возможностями построения графических изображений в системе визуального объектно-ориентированного программирования Turbo Delphi.*

11.1. Краткая теоретическая часть

Многие компоненты Turbo Delphi имеют свойство *Canvas* («холст», «канва»), представляющее собой область, на которой можно отображать готовые изображения или строить свои. Это свойство имеют формы, компоненты *Image*, *PaintBox*, *BitMap* и другие. Канва содержит свойства и методы, позволяющие отображать графику. Каждая точка канвы имеет координаты *X* и *Y*. Начало системы координат – верхний левый угол.

Основные свойства и методы канвы, используемые при рисовании:

• *Pixelx* – двумерный массив, хранящий цвета соответствующих точек; *Pixels[x,y]* соответствует цвету пикселя с координатами *x*, *y*. Для описания цвета используется тип *TColor*, в Turbo Delphi определены константы типа *TColor*, которые можно использовать при задании цвета:

```
Canvas.Pixelx[10,20]:=clBlack.
```

• *Pen* – перо, имеет ряд подсвойств: *Color* – цвет, *Width* – ширина линии в пикселях, *Style* – вид линии;

• *Brush* – кисть, определяет цвет и стиль заливки замкнутых фигур, свойства *Color* и *Style*;

• *MoveTo(x,y)* – перемещает перо в точку с координатами *x*, *y*;

• *LineTo(x,y)* – проводит линию из точки, в которой находится перо, в точку с координатами *x*, *y*;

• *Rectangle(x1, y1, x2, y2)* – рисует прямоугольник. *[x1, y1]* – координаты левого верхнего угла, *[x2, y2]* – правого нижнего;

• *TextOut(x: Integer, y: Integer, str: String)* – выводит строку *str*, начиная с точки *[x, y]*;

```
Form1.Canvas. TextOut(x,y,str);
```

Форма имеет свойства *ClientWidth*, *ClientHeight*, хранящие ширину и длину клиентской области, соответственно.

Для корректного отображения графика функции *F(x)*, если известен диапазон изменения аргумента *Xmin*, *Xmax* и диапазон значений функции *Ymin*, *Ymax*, необходимо ввести дополнительные переменные

```
X1, Y1 : Integer;
```

в которые следует записывать координаты пикселей, соответствующих переменным *X* и *Y*, где *Y* – значение функции:

```
x:=Xmin+X1*(Xmax-Xmin)/ClientWidth;
y1:=trunk(ClientHeight-(Y-Ymin) *ClientHeight /
(Ymax-Ymin));
```

11.2. Задания к работе

На главной форме проекта отобразите график функции *F(x)*, подпишите оси, отметьте рассматриваемый отрезок по *X*, максимальное и минимальное значения по *Y*. Напишите обработчик события *ReSize* формы, позволяющий сохранять пропорции изображения при изменении размеров формы.

1) $Y(x) = 3\sin(2x) - 2x$;
2) $Y(x) = 2\cos(2x) + 5x$;
3) $Y(x) = x^2 + 5x$;
4) $Y(x) = x2^x - 7$;
5) $Y(x) = \ln(3x) - 2x$;
6) $Y(x) = 3x^3 - 2x^2$;
7) $Y(x) = 2 - (1 - x)\sin(2x - 1)$;
8) $Y(x) = 2x + \sin^2(3x)$;
9) $Y(x) = \frac{\cos(x)}{x}$;
10) $Y(x) = 2x + \ln(x)$;
11) $Y(x) = -3 + \cos(2x)$;
12) $Y(x) = (4 + \sin(x))^3$;
13) $Y(x) = x + 2x^3 - x^2$.

11.3. Контрольные вопросы и задания

1. Какое свойство формы и других объектов позволяет отображать графику?
2. Какое свойство канвы определяет цвет и стиль заливки замкнутых фигур?
3. Каким образом можно задать цвет и стиль отображаемой линии?
4. Каким образом отобразить текст, не используя дополнительные компоненты?

Лабораторная работа № 12. ПОЛИМОРФИЗМ

Цель работы: *знакомство с основами ООП, изучение механизмов наследования, полиморфизма как свойств ООП; изучение принципа создания простейшей анимации с помощью Turbo Delphi, знакомство с компонентом Timer.*

12.1. Краткая теоретическая часть

Класс – это тип данных, определяемый пользователем, включающий в себя данные, процедуры и функции для работы с ними.

Новый класс может быть порожден от уже описанного, для этого при его объявлении необходимо указать имя класса-родителя:

```
TChildClass = Class (TParentClass)
```

Порожденный класс наследует все поля, методы и свойства родительского класса и может добавлять новые.

Все классы *Object Pascal* порождены от класса *TObject*, который не имеет полей и свойств, а содержит методы общего назначения, обеспечивающие жизненный цикл любого объекта (от создания до уничтожения).

Следующие два объявления идентичны:

```
TMyClass = Class
TMyClass = Class (TObject)
```

Для изменения метода необходимо перекрыть его в потомке, т.е. объявить в потомке одноименный метод, в котором будет решаться необходимая задача.

Методы могут быть одного из трех видов: статические, виртуальные и абстрактные.

По умолчанию все методы – статические. Переопределение такого метода в классе-наследнике приводит к отмене родительского для класса наследника.

Полиморфизм – это свойство ООП, позволяющее использовать одно и тоже имя для решения схожих, но разных по реализации задач.

При объявлении виртуальных и динамических методов после завершающей точки с запятой добавляются слова *Virtual* или *Dynamic*, соответственно.

```
Type
MyClass = Class
  Procedure Show; Virtual;
. . .
```

Встретив такое объявление, компилятор создаст таблицу *DMT* или *VMT*, в которой будут храниться адреса соответствующих методов.

Для того, чтобы перегрузить в классе-наследнике такой метод, после его объявления ставят ключевое слово *override*:

```
Type
MyClass1 = Class(MyClass)
  Procedure Show; Override;
. . .
```

Если переопределить метод, объявленный в классе-родителе с директивой *Virtual* или *Dynamic,* то при обращении по этому имени будет вызываться метод, соответствующий классу указанного при этом объекта.

Метод, объявленный виртуальным или динамическим в классе-родителе, остается таким и в наследниках, в том числе и в наследниках наследников.

Таблица динамических методов DMT содержит адреса только тех методов, которые объявлены в данном классе как *Dynamic*, таблица виртуальных методов VMT содержит адреса виртуальных методов данного класса и всех его родителей. Следовательно, она значительно большего размера, но обеспечивает более быстрый поиск нужного метода, в то время как DMT экономит память, но требует больших временных затрат, что необходимо учитывать при проектировании.

Виртуальный или динамический метод, реализация которого неопределена в классе, в котором он объявлен, называют абстрактным.

Создание простейшей анимации

Для задания частоты смены кадров следует использовать компонент *Timer* – это невизуальный компонент, расположенный на странице *System*, позволяющий задавать в приложении интервалы времени.

Свойства компонента *Timer*:

• *Interval* – интервал времени в миллисекундах. Через заданный промежуток времени от предыдущего срабатывания вызывается событие *OnTimer*;

• *Enabled* – доступность запускает (*true*) или останавливает (*false*) таймер.

Запускать и останавливать таймер следует при помощи какого-либо управляющего компонента, например в обработчике события *OnClick* компонента *TButton*:

```
Timer1.Enabled := not Timer1.Enabled;
```

В обработчике события *OnTimer* рекомендуется стирать предыдущий кадр, для чего необходимо нарисовать объект со старыми координатами, используя режим пера *pmNotXor*, далее выполнить пересчет координат и прорисовать второй кадр.

12.2. Задания к работе

Опишите иерархию классов, содержащую виртуальный метод отображения графического объекта, создайте приложение с использованием компонента *Timer*, реализующее движение графического объекта с изменением его формы и/или объема (используйте не менее двух кадров функций вывода).

12.3. Контрольные вопросы и задания

1. Поясните принцип наследования ООП.
2. Поясните принцип полиморфизма ООП.
3. Перечислите виды методов. Опишите назначения таблиц *DMT*, *VMT*.
4. *Timer*: назначение и свойства.

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. *Архангельский, А. Я.* Программирование в Delphi для Windows. Версии 2006, 2007, Turbo Delphi / А. Я. Архангельский. – М. : Бином-Пресс, 2007. – 1248 с. – ISBN 978-5-9518-0202-6.
2. *Кауфман, В. Ш.* Языки программирования. Концепции и принципы / В. Ш. Кауфман. – М. : ДМК-Пресс 2010. – 464 с. – ISBN 978-5-94074-622-6.
3. *Буч, Г.* Объектно-ориентированное программирование с примерами применения / Г. Буч. – М. : Кондорд, 1992. – 519 с. – ISBN 5-87737-002-2.
4. *Фаронов, В. В.* Turbo pascal 7.0: учеб. пособие для вузов / В. В. Фаронов. – М. : КноРус, 2009. – 368 с. – ISBN 978-5-390-00218-6.
5. *Он же.* Delphi. Программирование на языке высокого уровня / В. В. Фаронов. – СПб. : Питер, 2008. – 639 с. – ISBN 978-5-8046-0008-3.
6. *Серебряков, В. А.* Лекции по конструированию компиляторов [Электронный ресурс] / В. А. Серебряков. – М. : CodeNet, 1993. – Режим доступа: http:// www.codenet.ru.

ОГЛАВЛЕНИЕ

ЯЗЫКИ ПРОГРАММИРОВАНИЯ
И МЕТОДЫ ТРАНСЛЯЦИИ

Методические указания к лабораторным занятиям

Составители:
ДУХАНОВ Алексей Валентинович
МЕДВЕДЕВА Ольга Николаевна
ШИШКИНА Мария Викторовна

Ответственный за выпуск – зав. кафедрой профессор С.М. Аракелян

Подписано в печать 14.03.11.
Формат 60x84/16. Усл. печ. л. 3,95. Тираж 100 экз.
Заказ
Издательство
Владимирского государственного университета.
600000, Владимир, ул. Горького, 87.