уроки программирования

С. И. Смирнов

PASCAL DELPHI

часть I

С. И Смирнов

Уроки программирования

PASCAL

2011

Данный курс программирования предназначен для изучения языка программирования Pascal школьниками и студентами младших курсов ВУЗов.

Курс состоит из двух частей: изучение Turbo Pascal 7.0 и изучение Delphi.

Первая часть курса посвящена изучению языка Pascal: его операторов, основных конструкций, типов данных и т.д.

Вторая часть — программирование под Windows в Delphi.

Уроки программирования: Pascal — Delphi. — Красноярск, 2011

Занятие 1

Обсуждается понятие алгоритма и формы его представления. Рассматриваются основные элементы и правила построения блок-схем. Строится блок-схема простой линейной программы. Рассматривается упрощенная модель компилятора.

Мой дядя самых честных правил...
А. С. Пушкин, "Е. Онегин"

АЛГОРИТМЫ И СПОСОБЫ ИХ ОПИСАНИЯ

Слово «алгоритм» произошло от имени знаменитого математика IX века Аль-Хорезми, сформулировавшего правила арифметических действий над числами. В математике получило широкое распространение следующее определение алгоритма:

Алгоритм — это точное предписание конечной последовательности однозначных, понятных действий, направленных на достижение указанной цели, исходя из некоторых начальных данных.

Уточним особенности сформулированного определения:

1. **Определенность** (**детерминированность**) — однозначность результата процесса исполнения алгоритма при заданных исходных данных. Один и тот же алгоритм не может получать два разных результата при одних и тех же данных. Так, если на вход алгоритма сложения поступают два числа, то, сколько бы раз мы не предъявляли эти числа, ответ должен быть одним.
2. **Дискретность** определяемого алгоритмом процесса - расчлененность его на отдельные элементарные акты, возможность выполнения которых человеком или машиной не вызывает сомнения. Одно дело сказать: «Найди наиболее общий делитель чисел 1 345 672 и 164 930», и уж совсем иное дело - подробно показать, как это делается.
3. **Массовость**, или **повторяемость**, означает, что исходные данные для алгоритма можно выбирать из некоторого множества данных (потенциально бесконечного), т.е. алгоритм должен обеспечивать решение любой задачи из класса однотипных задач. Так, если вы описываете алгоритм решения некоторого класса алгебраических уравнений, то должны описать, как найти корни для любого уравнения из этого класса. В противном случае это не алгоритм, а лишь ваша догадка, что вы знаете, как решать такие уравнения.
4. **Понятность** алгоритма состоит в том, что он должен быть описан в виде последовательности команд, каждая из которых принадлежит системе команд, понятной для исполнителя этого алгоритма. Если исполнителем алгоритма является человек, то он не должен наделять команды, участвующие в записи алгоритма, никакими собственными

интерпретациями, ему следует выполнять в точности те действия, которые там указаны. Если исполнителем алгоритма является компьютер, то понятность в данном случае состоит в том, что программа, предлагаемая компьютеру для вычисления и записанная на каком-либо языке программирования, должна транслироваться на язык машинных команд соответствующим транслятором. Если такой транслятор не входит в математическое обеспечение вашего компьютера, то алгоритм, сколь бы хорош он ни был, так и останется простым текстом и не заставит машину совершать никаких действий.

5. **Конечность** (результативность) алгоритма означает, что исполнение алгоритма должно завершиться за конечное число шагов.

Несмотря на значительные достижения в разработке и распространении всевозможных алгоритмов в математике, попытки научного подхода к алгоритмам вплоть до XX века были малоуспешными. Причина — трудоемкость строгого, формального определения понятия алгоритм. Формулировка, приведенная выше, может быть названа определением лишь в интуитивном смысле. Она не является строгой. В ней нет указаний на то, что может быть объектами алгоритма, а понятия типа «точное предписание», «понятные действия» — расплывчаты.

Разработка алгоритма представляет «перевод мыслей разработчика на бумагу». Рассмотрим некоторые формы представления алгоритма.

Формы представления алгоритма

1. Словесный алгоритм.
2. Язык проектирования программ.

Это язык, не имеющий компьютерной реализации. Существует множество вариантов таких языков.

2.1. Пример: программный модуль «Чтение книги»:

```
Процедура Чтение_книги
  открыть книгу
  Пока страница не последняя
    читать страницу
    Если страница нечетная то перевернуть страницу
  Конец пока
  закрыть книгу
Конец процедуры
```

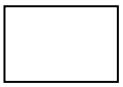
2.2. Самостоятельное составление алгоритма:

Составьте алгоритм ловли рыбы на удочку.

3. Блок-схема.

Является формой представления алгоритма с помощью графических символов. Графические символы, их размеры, а также правила построения блок-схем определены государственным стандартом. Рассмотрим некоторые:

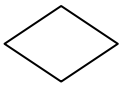
3.1. Процесс.



Выполнение операции или группы операций, в результате чего изменяется значение, форма представления или расположения данных.

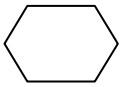
Внутри символа или в виде комментария на естественном языке или в виде формулы записываются действия, которые производятся при выполнении операции или группы операций.

3.2. Решение.



Выбор направления алгоритма или программы в зависимости от некоторых условий (развилка полная, неполная; выбор; цикл-до; цикл-пока).

3.3. Модификация.

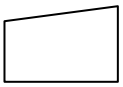


Выполнение операций, меняющих команды или группу команд, изменяющих программу (цикл с параметрами).

3.4. Предопределенный процесс.

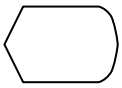
Использование ранее созданных и отдельно описанных алгоритмов или программ (процедур, функций, программных модулей).

3.5. Ручной ввод.



Ввод данных оператором в процессе обработки при помощи устройства, непосредственно сопряженного с компьютером (клавиатура).

3.6. Дисплей.



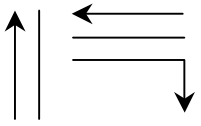
Ввод-вывод данных в случае, если непосредственно подключенное к процессу устройство воспроизводит данные и позволяет оператору вносить изменения в процессе их обработки.

3.7. Документ.

Ввод-вывод данных, носителем которых является бумага.

3.8. Линия потока. Указывают последовательности связей между символами.

- ♦ должны быть параллельны линиям внешней рамки блок-схемы;
- ♦ стрелка не ставится, если она направлена сверху вниз или справа налево;
- ♦ изменение направлений линий потока производится под прямым углом.



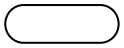
3.9. Соединитель.

Указание связи между прерванными линиями потока, связывающими символы на одной странице.

3.10. Межстрочный соединитель.

Указание связи между прерванными линиями потока, связывающими символы на разных страницах.

3.11. Пуск-останов.



Начало, конец, прерывание процесса обработки данных или выполнения программы.

3.12. Комментарий. Делается в виде выноски.

Размеры символов должны удовлетворять соотношению a/b=1.5.

Каждому символу присваивается порядковый номер.

Пример построения блок-схемы модуля «Чтение книги»

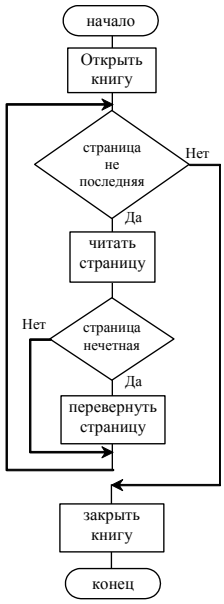


Рис. 1

После того как мы составили алгоритм программы в словесной реализации или в виде блок-схемы, можно переходить к этапу его перевода на какой-либо из языков программирования высокого уровня.

Упрощенная модель компилятора

Перевод программ, написанных на языках программирования высокого уровня, на язык машинных кодов, выполняемых компьютером, осуществляется специальными программами, которые называются *трансляторами*. По способу работы трансляторы делятся на *компиляторы* и *интерпретаторы*. Трансляторы языка Pascal от самых ранних реализаций до последней версии Borland Pascal 7.0 работают по компилирующему принципу. Для более глубокого понимания базовых конструкций языка Borland Pascal, кратко рассмотрим упрощенную модель компилятора, показанную на рисунке.

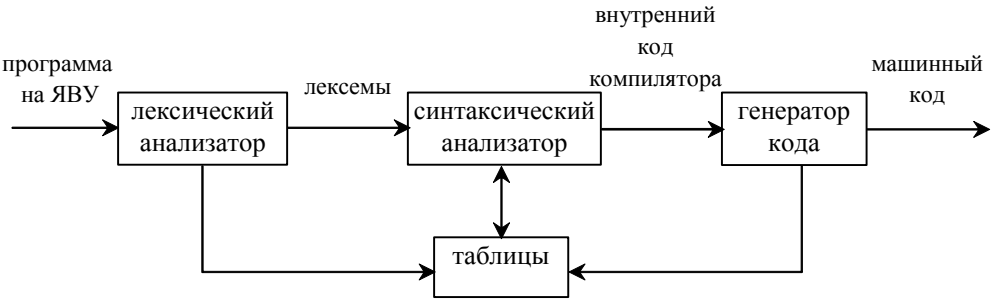


Рис. 2

Лексический анализатор

Исходная программа на языке высокого уровня (ЯВУ), представляет собой цепочку символов, образуемую последовательным сцеплением всех строк программы. Среди допустимых для языка символов всегда выделяют несколько, так называемых, *символов-разделителей*, благодаря которым предложения исходной программы разбиваются на отдельные слова. Такие слова в теории компиляции называются *лексемами*. Например, предложение (оператор)

```
for i:=1 to N do Writeln(i);
```

будет разбито на лексемы **`for, i, :=, 1, to, N, do, Writeln, (, i, ), ;`**. Здесь в качестве разделителя используется символ «пробел». Однако можно заметить, что между некоторыми лексемами пробел не стоит. Это связано с тем, что эти лексемы сами являются разделителями и поэтому для отделения их от других лексем специальные символы-разделители использовать не обязательно, хотя и допустимо. Например, то же самое предложение без изменения смысла можно было бы записать так

```
for i := 1 to N do Writeln ( i ) ;
```

Там, где может стоять один символ-разделитель, допускается любое количество символов-разделителей. Однако это правило не распространяется на лексемы-разделители, поскольку они несут определенную смысловую нагрузку.

Синтаксический анализатор

Синтаксический анализатор на основе синтаксических правил грамматики языка проверяет корректность записи предложений программы и переводит последовательность лексем в последовательность внутренних кодов компилятора. Эта последовательность уже отражает порядок действий, которые должен выполнить компьютер, но еще не является окончательным машинным кодом.

Генератор кода

Генератор кода осуществляет перевод внутреннего кода компилятора в окончательный машинный код компьютера.

Таблицы

В процессе работы все рассмотренные выше блоки компилятора обращаются к общему набору таблиц, куда помещается как постоянная для трансляции всех программ информация (например, таблица зарезервированных слов), так и

информация, индивидуальная для каждой программы (например, таблицы идентификаторов, литералов и др.).

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Каша-малаша»* Составить словесный алгоритм приготовления манной каши. Оформить его в виде блок-схемы.
2. *«Малевич»* Составить словесный алгоритм рисования квадрата. Оформить его в виде блок-схемы.
3. *«Звездочка»* Составить словесный алгоритм рисования пятиконечной звезды. Оформить его в виде блок-схемы.
4. *«Ловись, рыбка...»* Составить словесный алгоритм ловли рыбы. Оформить его в виде блок-схемы.
5. *«Кривая дракона»* Составить словесный алгоритм рисования «кривой дракона». Оформить его в виде блок-схемы.

Занятие 2

Изучается структура программы и некоторые типы данных в языке Pascal. Вводятся понятия оператора, выражения, операции и операнда.

Зарецкий тридцать два шага
Отмерил с точностью отменной...
А. С. Пушкин, "Е. Онегин"

СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ «PASCAL»

Структура программы на языке «Pascal» имеет следующий вид:

```
Program My_FirstProgram;
   {раздел описаний}
begin
   {раздел операторов}
end.
```

Слова **Program**, **begin** и **end** выделяют две части программы – раздел описаний и раздел операторов. Такая структура обязательна для любой программы, что является следствием жесткого требования языка: любой нестандартный идентификатор, используемый в исполняемых операторах, должен быть предварительно описан в разделе описаний. Требование предварительного описания идентификаторов диктуется повышением надежности создаваемых программ.

Описать идентификатор, значит, указать тип связанного с ним объекта программы (константы или переменной).

Переменная именует объект программы, который может изменять свое значение в ходе счета. При описании переменных за идентификатором ставятся двоеточие и имя типа. Несколько однотипных переменных можно объединять в список, разделяя их запятыми. В начале раздела описания переменных должно стоять зарезервированное слово **VAR** (VARiables – переменные).

В любом месте программы можно помещать комментарии, которые используются для пояснения текста программы. Комментарий необходимо выделить одним из следующих видов скобок:

```
{ Это комментарий }
(* Это – тоже комментарий *)
```

ТИПЫ ДАННЫХ В ЯЗЫКЕ PASCAL

Понятие типа — одно из фундаментальных понятий Turbo Pascal. Тип определяет способ внутреннего для компьютера представления данных, а также действия, которые разрешается над ними выполнять.

Рассмотрим некоторые типы данных:

- `Integer` — целочисленные данные, во внутреннем представлении занимают 2 байта; диапазон возможных значений – от –32768 до +32767; данные представляются точно;

- `Real`[1] — вещественные данные, занимают 6 байт; диапазон возможных значений модуля – от 2.9E-29 до 1.7E+38; точность представления данных – 11-12 значащих цифр;
- `Extended` – вещественный тип данных, занимает в памяти 10 байт; диапазон возможных значений модуля – от 3.4E-4932 до 1.1E+4932; точность представления данных – 19-20 значащих цифр;
- `Char` — символ, занимает 1 байт;
- `String` — строка символов, занимает MAX+1 байт, где MAX – максимальное число символов в строке;
- `Boolean` — логический тип, занимает 1 байт и имеет два значения `False` (ложь) и `True` (истина).

Тип константы определяется способом записи ее значения:

```
const
  c1 = 17;                {Integer}
  c2 = 3.14;              {Real}
  c3 = 'A';               {Char}
  c4 = '3.14';            {String}
  c5 = True;              {Boolean}
```

ПОНЯТИЕ ВЫРАЖЕНИЯ, ОПЕРАЦИИ И ОПЕРАНДА

Выражение в программировании служит для определения действий, которые в математике обычно описываются формулами. Выражения состоят из операций и операндов. По количеству операндов операции делятся на унарные и бинарные. Унарные операции имеют только один операнд, перед которым располагается символ операции. Например:

Выражение	Результат
`-7`	`-7`
`-(-9)`	`9`
`not` `False`	`True`

Большинство операций являются бинарными и содержат два операнда, между которыми ставится символ операции. Пример:

Выражение	Результат
`12+3`	`15`
`(7-4)*5+10`	`25`
`True` **`or`** `False`	`True`

По характеру выполняемых действий операции можно разделить на следующие группы[2]:

[1] На современных компьютерах данный тип использовать нецелесообразно, т. к. процессоры с сопроцессором сначала преобразуют тип `Real` в тип `Extended`, затем осуществляются вычисления и обратное преобразование в тип `Real`.

[2] Здесь перечислены не все операции.

1. Арифметические операции:
   - унарные: **+, –**
   - бинарные: **+, –, *, /, div, mod**
2. Операции отношения: **=, <>, <, >, <=, >=**
3. Булевские (логические) операции: **not, and, or, xor**
4. Строковая операция (конкатенация): **+**

Ниже описаны действия операций, а также типы операндов и результата.

Описание арифметических операций

Операция	Действие	Типы операндов	Тип результата
Унарные			
+	сохранение знака	целый, вещественный	целый, вещественный
–	отрицание знака	целый, вещественный	целый, вещественный
Бинарные			
+	сложение	целый, вещественный	целый, вещественный
–	вычитание	целый, вещественный	целый, вещественный
*****	умножение	целый, вещественный	целый, вещественный
/	деление	целый, вещественный	вещественный, вещественный
div	целочисленное деление	целый	целый
mod	остаток от целочисленного деления	целый	целый

Пример целочисленных арифметических операций:

Выражение	Результат
7 **div** 3	2
7 **mod** 3	1
23 **div** 5	4
23 **mod** 5	3

Описание операций отношения

Операция	Действие	Типы операндов	Тип результата
=	равно	совместимый простой, строковый	булевский
<>	не равно	совместимый простой, строковый	булевский
<	меньше	совместимый простой, строковый	булевский
>	больше	совместимый простой, строковый	булевский
<=	меньше или равно	совместимый простой, строковый	булевский

Операция	Действие	Типы операндов	Тип результата
>=	больше или равно	совместимый простой, строковый	булевский

Пример операций отношения:

Выражение	Результат
`4 = 3`	`False`
`False <> True`	`True`
`'ABC' < 'ABD'`	`True`
`'ABC' > 'D'`	`True`

Описание булевских (логических) операций

Операция	Действие	Типы операндов	Тип результата
Унарная			
not	логическое отрицание	булевский	булевский
Бинарные			
and	логическое **И**	булевский	булевский
or	логическое **ИЛИ**	булевский	булевский
xor	логическое исключающее **ИЛИ**	булевский	булевский

Булевские операции выполняются по правилам булевой алгебры, которые показаны ниже.

Операнды		Операции			
A	B	**not** A	A **and** B	A **or** B	A **xor** B
False	False	True	False	False	False
False	True	True	False	True	True
True	False	False	False	True	True
True	True	False	True	True	False

Описание строковых операций

Операция	Действие	Типы операндов	Тип результата
+	конкатенация (сцепление)	строковый, символьный	строковый

Пример строковых операций:

Выражение	Результат
`'Borland' + ' Pascal'`	`'Borland Pascal'`
`'ABC' + 'D'`	`'ABCD'`
`'X' + 'Z'`	`'XZ'`

Приоритет операций

Последовательность выполнения операций в выражении определяется их приоритетом, порядком расположения в выражении и использованием скобок. По приоритету все операции делятся на четыре группы. Операции первого (высшего) приоритета выполняются в первую очередь. Операции четвертого (низшего) приоритета выполняются в последнюю очередь. Операции с равным приоритетом выполняются слева направо, хотя иногда компилятор для генерации оптимального кода может переупорядочить операнды. Скобки служат для изменения обычного порядка обработки операций. Подвыражение, заключенное в скобки, сначала вычисляется как отдельный операнд, а затем его результат используется для выполнения операций, обрамляющих скобки.

Приоритет операций

Приоритет	Операции	Категории операций
Первый (высший)	+ - not	Унарные операции
Второй	* / div mod and	Бинарные операции типа умножения
Третий	+ - or xor	Бинарные операции типа сложения
Четвертый (низший)	= <> < > <= >=	Бинарные операции отношения

В Turbo Pascal приоритет операций отражен непосредственно в синтаксисе языка, что видно из нижеприведенных сложных выражений:

```
A*(((X-Y/Z)*3+5)/(B+C)-D*E)-14
(A<>B) and ((I<J) or (J<K) and (C = D)).
```

ПОНЯТИЕ ОПЕРАТОРА

Операторы предназначены для описания действий, которые будут выполнены при реализации алгоритма. Согласно синтаксису операторы языка Turbo Pascal разделяются на две группы:

- простые операторы;
- структурные операторы.

Простые операторы включают в себя: *оператор присваивания, оператор процедуры, оператор перехода*.

Структурные операторы включают в себя: *составной оператор, условный оператор, оператор цикла, оператор присоединения*.

Операторы отделяются друг от друга символом «точка с запятой» (;). Точка с запятой не является частью оператора, это **разделитель операторов**. Поэтому после последнего оператора программы и после последнего оператора в составном операторе, то есть перед ключевым словом **end**, точку с запятой ставить не обязательно. Если же в указанном случае символ «;» все же поставлен, то считается, что после него расположен пустой оператор и ошибкой не является.

ПРОСТЫЕ ОПЕРАТОРЫ

Простые операторы — это такие операторы, которые не содержат в себе других операторов.

Оператор присваивания

Оператор присваивания состоит из идентификатора переменной, символа присваивания «**:=**» и выражения. Например:

```
A:=B * C;
i:=k div 2;
F:= A*(((X-Y/Z)*3+5)/(B+C)-D*E)-14;
```

Выполнение оператора присваивания приводит к вычислению значения, определяемого *выражением*, и присваиванию этого значения переменной, идентифицируемой именем, стоящим слева от символа присваивания.

Тип выражения в правой части оператора, и тип переменной в левой части оператора обязательно должны быть совместимы по присваиванию.

Оператор процедуры

Оператор процедуры состоит из идентификатора, непосредственно за которым располагается в круглых скобках список фактических параметров. Оператор процедуры без параметров состоит только из идентификатора процедуры. Например:

```
Writeln('Привет, Вася!');
Writeln;
Readln(UserName);
Randomize;
```

Выполнение оператора процедуры приводит к активизации действий, описанных в ее теле.

Оператор перехода

Оператор перехода состоит из ключевого слова **Goto,** после которого указывается метка.

Выполнение оператора **Goto** приводит к передаче управления на оператор, перед которым стоит указанная в операторе **Goto** метка.

Следует заметить, что оператор **Goto** противоречит принципам структурного программирования, и его использование в программах настоятельно **не рекомендуется**.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Сколько будет – пятью пять?»* С клавиатуры вводится два числа *A* и *B*. Вывести на экран результаты суммы, разности, произведения и частного этих чисел.
2. *«Важен порядок...»* Посчитать и вывести результат на экран:

$$\frac{156+32}{56+2^3}-\frac{94}{51}.$$

3. *«Привет, Вася!»* После запуска программы на экран выводится сообщение: «Введите Ваше имя», и программа ждет ввода имени пользователя. После ввода имени пользователя на экран выводится приветствие «Привет, UserName», где UserName – имя, введенное пользователем. Далее на экран выводится сообщение: «Введите год Вашего рождения», и программа ждет ввода года рождения пользователя. После ввода года рождения пользователя на экран выводится сообщение «Вам Age лет», где Age – возраст пользователя.

Пример построения блок-схемы программы «Привет, Вася!»

Словесный алгоритм:

Запускаем программу.

На экран выводится сообщение: «Введите Ваше имя», и программа ждет ввода имени пользователя (UserName).

После ввода имени пользователя на экран выводится «Привет, UserName», где UserName – имя, введенное пользователем.

На экран выводится сообщение: «Введите год Вашего рождения», и программа ждет ввода года рождения пользователя (UserBirth).

После ввода года рождения пользователя на экран выводится сообщение «Вам Age лет», где Age – возраст пользователя.

После нажатия клавиши Enter программа завершается.

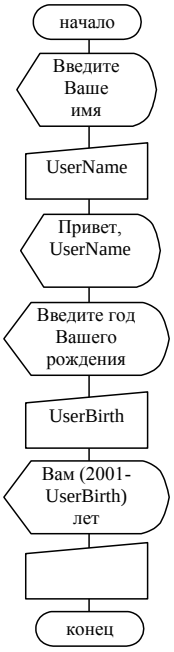


Рис. 3

Занятие 3

Вводятся понятия структурного оператора. Разбирается условная конструкция. Изучается условный оператор ***IF .. THEN .. ELSE.***

Идет направо — песнь заводит,
Налево — сказку говорит...
А. С. Пушкин

СТРУКТУРНЫЕ ОПЕРАТОРЫ

Структурные операторы включают в себя другие операторы и управляют последовательностью их выполнения.

Составной оператор

Составной оператор объединяет группу операторов в единое целое, после чего они могут считаться одним оператором.

Составной оператор состоит из последовательности объединяемых операторов, которые располагаются между ключевыми словами **`begin`** и **`end`**.

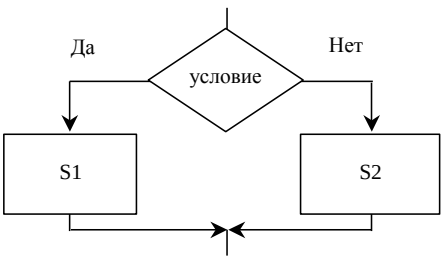
Составной оператор используется в тех случаях, когда синтаксис языка допускает в определенной точке программы указание только одного оператора, а по алгоритму в этом месте необходимо выполнить группу операторов. Как правило, составной оператор используется совместно со структурными операторами.

Зачастую, в процессе выполнения программы, на определенном ее этапе в зависимости от условий необходимо принять решение: в каком направлении идти дальше. Выбор направления зависит от того, выполняются эти условия или нет. Данная ситуация является стандартной алгоритмической структурой и носит название «Развилка».

УНИФИЦИРОВАННАЯ СТРУКТУРА «РАЗВИЛКА»

Структура включает в себя два варианта: развилка полная и развилка неполная.

Развилка полная. Используется в случае, когда выполнение программы может пойти двумя различными (альтернативными) путями. Внутри символа (или в виде комментария) записывается логическое условие, по которому осуществляется выбор требуемого направления выполнения алгоритма. В зависимости от значения логического условия истина (да, `TRUE`) или ложь (нет, `FALSE`) дальнейшее выполнение алгоритма идет либо по левой, либо по правой ветви. Символы ПРОЦЕСС S1 и



ПРОЦЕСС S2 могут обозначать унифицированные структуры, процедуры, функции и алгоритмы любой сложности.

Развилка неполная. Используется так же, как и развилка полная с тем отличием, что при выполнении одной из ветвей никаких изменений данных, поступивших на вход этой унифицированной структуры, не происходит.

УСЛОВНАЯ КОНСТРУКЦИЯ IF .. THEN .. ELSE

Алгоритмическая структура «Развилка» в языках программирования реализуется с помощью условного оператора.

Условный оператор позволяет проверить некоторое условие и в зависимости от результатов проверки выполнить то или иное действие. Таким образом, условный оператор — это средство ветвления вычислительного процесса.

Структура условного оператора имеет следующий вид:

```
if <условие> then <оператор1> else <оператор2>;
```

где **if, then, else** — зарезервированные слова (если, то, иначе);

<условие> — произвольное выражение логического типа;

<оператор1>, <оператор2> — любые операторы языка Турбо Паскаль.

Условный оператор работает по следующему алгоритму. Вначале вычисляется условное выражение <условие>. Если результат есть TRUE (истина), то выполняется <оператор1>, а <оператор2> пропускается; если результат есть FALSE (ложь), наоборот, <оператор1> пропускается, а выполняется <оператор2>. Например:

```
var
   x, y, max: Integer;
begin
   if x > max then
     y:= max
else
     y:= x;
```

При выполнении этого фрагмента переменная Y получит значение переменной X, если только это значение не превышает MAX, в противном случае Y станет равно MAX.

Часть **else** <оператор2> условного оператора может быть опущена. Тогда при значении TRUE условного выражения выполняется <оператор1>, в противном случае этот оператор пропускается:

```
var
 x, y, max: Integer;
begin
 if x > max then max:= x;
 y:= x;
```

В этом примере переменная Y всегда будет иметь значение переменной X, а в MAX запоминается максимальное значение X.

Поскольку любой из операторов <оператор1> и <оператор2> может быть любого типа, в том числе и условным, а в то же время не каждый из «вложенных» условных операторов может иметь часть **else** <оператор2>, то возникает неоднозначность трактовки условий. Эта неоднозначность в Турбо Паскале решается следующим образом: любая встретившаяся часть **else** соответствует ближайшей к ней «сверху» части **then** условного оператора. Например:

```
var
 a,b,c,d : Integer;
begin
 a:= 1; b:= 2; c:= 3; d:= 4;
 if a > b then
  if c < d then
   if c < 0 then
    c:= 0
   else
    a:= b; {a равно 1}
 if a > b then
  if c < d then
   if c < 0 then
    c:= 0
   else
  else
 else
a:= b;    {a равно 2}
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Пользователь вводит с клавиатуры любое число N. Если число N > 0, то на экран выводится сообщение: «N – положительное число»; если N < 0 — «N – отрицательное число».
2. Пользователь вводит с клавиатуры любое целое число N. Если число четное, то на экран выводится сообщение: «N – четное число»; если число нечетное, то сообщение: «N – нечетное число».
3. В программе «Привет, Вася!» при вводе года рождения проверять его «правильность». Если год рождения больше, чем текущий, или меньше, чем 1900, то выдается сообщение «Абсурдная дата рождения!». В противном случае — вычисляется год рождения пользователя.
4. Пользователь вводит с клавиатуры любой год. Определить и вывести на экран сообщение о том является ли этот год високосным. Необходимым условием того, что год високосный является его делимость на четыре. Если этот год еще делится нацело и на сто, но не делится на четыреста, то он не будет являться високосным.

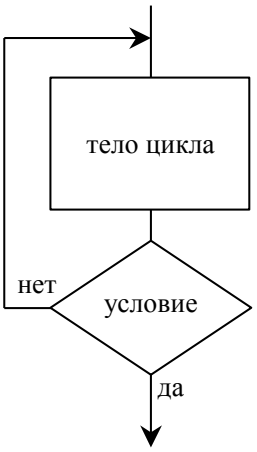
Занятие 4

Изучаются циклические конструкции: с постусловием, с предусловием и с параметром. Цель занятия: разобраться со всеми циклическими конструкциями и отработать их использование в программах.

... Он вновь посланье:
Второму, третьему письму
Ответа нет.
А. С. Пушкин, "Е. Онегин"

Если в программе требуется повторить один и тот же оператор (или группу операторов) несколько раз, то возникает необходимость использования циклической конструкции. В языке Turbo Pascal имеются три различных оператора повторений, с помощью которых можно запрограммировать повторяющиеся фрагменты программ: цикл с постусловием, цикл с предусловием и цикл с параметром.

Цикл с постусловием



Унифицированная алгоритмическая структура цикла с постпроверкой условия служит для организации циклов с заранее неизвестным числом повторений. Цикл данного типа в любом случае выполнится хотя бы один раз, поскольку проверка условия завершения цикла происходит после выполнения тела цикла.

Оператор цикла с постусловием на языке Turbo Pascal имеет вид:

```
repeat <тело цикла> until <условие>;
```

где `<тело цикла>` – произвольная последовательность операторов Turbo Pascal;

`<условие>` – выражение логического типа.

Операторы `<тело цикла>` выполняются хотя бы один раз, затем вычисляется `<условие>`: если его значение есть `False`, то цикл повторяется еще раз, в противном случае происходит выход из оператора повторения.

Рассмотрим пример использования оператора цикла с постусловием.

Пример 1:

```
Program Task1;
var
  A, k: Integer;
begin
  Write('Введите целое положительное число: ');
  Readln(A);
  repeat
    k:=A mod 10;
    A:=A div 10;
    Writeln(k);
```

```
  until A=0;
  Readln;
end.
```

Пользователь вводит натуральное число, а программа разбирает его на цифры и выводит в обратном порядке. Использование оператора цикла с постусловием обусловлено тем, что введенное число в любом случае будет содержать не менее одной цифры, а, следовательно, цикл должен выполниться до проверки условия выхода из него.

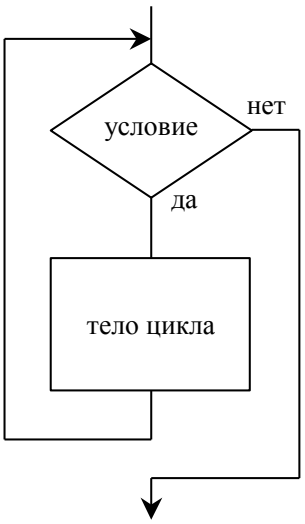
Пример 2:

```
Program Task2;
const
  Max = 9999;
var
  Num: LongInt;
begin
  repeat
    Write('Введите целое четырехзначное число: ');
    Readln(Num);
  until Num <= Max;
  Readln;
end.
```

Данная программа не позволяет пользователю ввести с клавиатуры число больше, чем величина константы `Max`, т. к. выход из цикла возможен лишь при условии `Num<=Max`.

LongInt – целочисленный тип. Его внутреннее представление – четыре байта; диапазон значений от **–2 147 483 648** до **+2 147 483 647**.

Цикл с предусловием



Унифицированная алгоритмическая структура цикла с предпроверкой условия также служит для организации циклов с заранее неизвестным числом повторений. Отличие от предыдущего цикла состоит в том, что тело данного цикла может быть ни разу не выполнено, т. к. условие проверяется перед его выполнением.

Оператор цикла с предусловием на языке Turbo Pascal имеет вид:

```
while <условие> do <тело цикла>
```

где `<тело цикла>` – простой или составной оператор Turbo Pascal;

`<условие>` – выражение логического типа.

При выполнении оператора **while** сначала вычисляется `<условие>`, и, в случае если оно оказывается истинным (`True`), то выполняется `<тело цикла>`.

После этого снова происходит проверка <условия> и процесс повторяется до тех пор, пока <условие> не примет ложное (False) значение.

Рассмотрим пример использования цикла с предусловием.

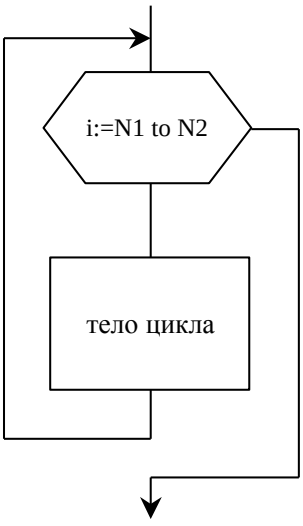
Пример 3:

```
Program Task3;
var
  N: Integer;
  Sum: Integer;
begin
  Write('Введите целое положительное число: ');
  Readln(N);
  Sum:=0;
  while N<>0 do
    begin
      Sum:=Sum+N;
      N:=N-1;
    end;
  Writeln('Сумма чисел от 0 до ', N, ': ', Sum);
  Readln;
end.
```

Пользователь вводит с клавиатуры натуральное число N, и в цикле происходит вычисление суммы всех чисел от 0 до N. Если пользователь ввел ноль, то цикл не выполняется.

Цикл с параметром

Если заранее известно, сколько повторений <тела цикла> необходимо сделать, то лучше всего использовать унифицированную структуру цикла с параметром. Данный тип отличается тем, что у него имеется специальный счетчик, который фиксирует количество повторов выполнения <тела цикла>. У счетчика задается начальное N1 и конечное N2 значение. При каждом очередном повторе его текущее значение изменяется на единицу в сторону N2. Когда счетчик достигает конечного значения N2 – выполняется последний повтор и происходит выход из цикла.



Оператор цикла с параметром на языке Turbo Pascal имеет вид:

```
for <счетчик>:=N1 to N2 do <тело цикла>
```

где <тело цикла> – простой или составной оператор Turbo Pascal;

<счетчик> – переменная порядкового типа.

N1, N2 – начальное и конечное значения – выражения порядкового типа.

Рассмотрим пример использования цикла с параметром.

Пример 4:

```
Program Task4;
var
  N: Integer;
  i: Integer;
begin
  Write('Введите целое положительное число: ');
  Readln(N);
  Writeln('Числа, делящиеся нацело на 3: ');
  for i:=1 to N do
    if i mod 3 = 0
      then Write(i: 5);
  Readln;
end.
```

Пользователь вводит с клавиатуры натуральное число N. Далее в цикле происходит проверка делимости каждого числа от 1 до N на три (если остаток от деления нацело равен нулю, то число делится на три) и вывод числа на экран, если такая делимость присутствует.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. «*Простое число*». Найти все простые числа от 0 до `N`, где `N` – натуральное число. Простое число – это число, которое делится только на себя и на единицу.
2. «*Счастливый билет*». Написать программу, которая находит и выводит на экран все «счастливые билеты» в диапазоне от 000 000 до `N`, где `N` – шестизначное натуральное число. Посчитать количество таких билетов.
3. «*Угадай число*». Компьютер загадывает число в диапазоне от 1 до 100. Игрок, пытаясь отгадать это число, вводит свой ответ. Если он ответит неправильно, то компьютер сообщает только о том: больше или меньше загаданное число, чем ответ игрока. Далее попытка угадывания повторяется. При правильном ответе – поздравление с победой.

Занятие 5

Повторение пройденного материала: условный оператор, циклические операторы. Цель занятия: закрепление понятия условной и циклической конструкции; отработка их использования на примерах различных программ; написание программы по готовой блок-схеме.

Об ней свежо воспоминанье...
А. С. Пушкин, "Медный всадник"

РЕШЕНИЕ КВАДРАТНЫХ УРАВНЕНИЙ

Квадратное уравнение имеет вид $A \cdot x^2 + B \cdot x + C = 0$. Задавая коэффициенты *A*, *B* и *C*, мы задаем квадратное уравнение. Если *A =0*, то данное уравнение не будет являться квадратным. Если уравнение квадратное и дискриминант *D≥0*, то его корни имеют значения: $X_{1,2} = \frac{-B \pm \sqrt{D}}{2A}$. На Рис. 4 приведена блок-схема алгоритма решения квадратного уравнения. По данной блок-схеме напишите программу на языке Pascal.

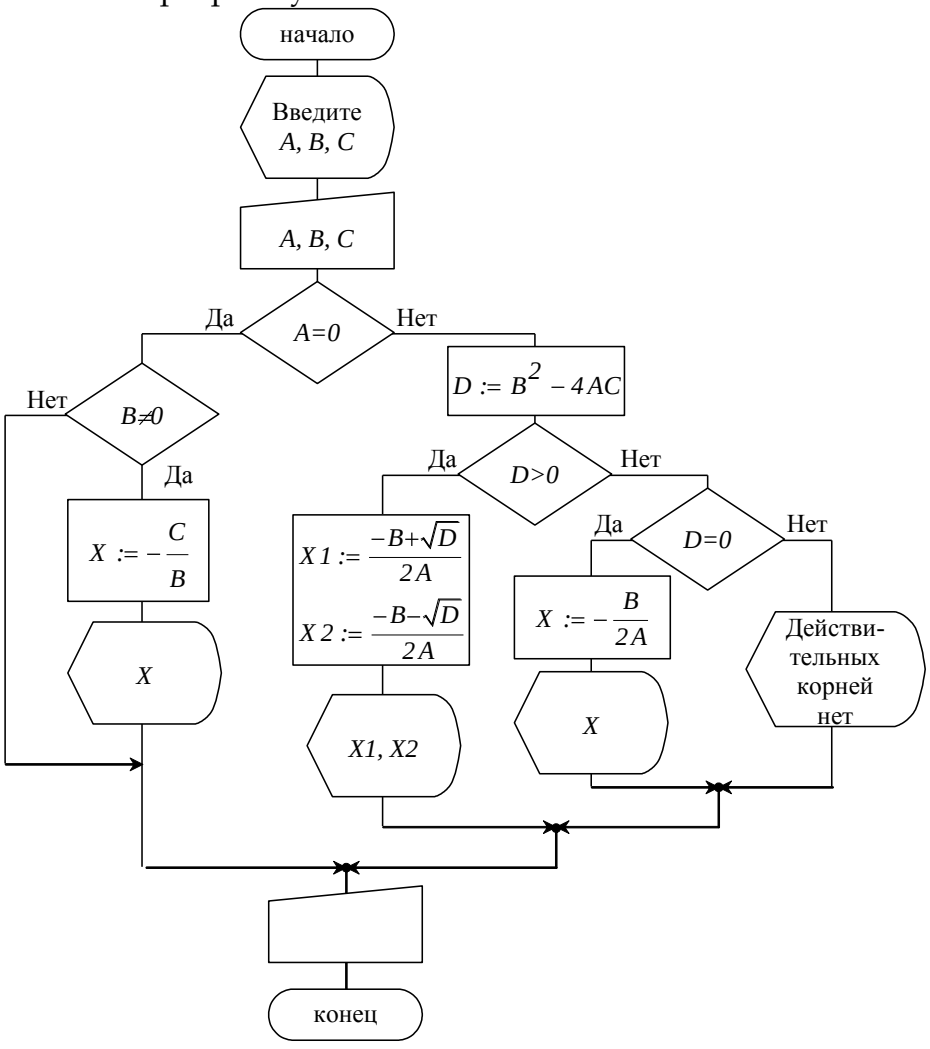


Рис. 4

ИГРА «УГАДАЙ ЧИСЛО»

Компьютер загадывает число в диапазоне от 1 до 100 и предлагает игроку отгадать это число. Если игрок не угадывает заветное число, то компьютер делает подсказку о величине правильного ответа: ***больше*** или ***меньше***, чем ответ игрока. Затем попытка отгадывания повторяется. Если число оказывается отгаданным, то «звучат» поздравления и игра заканчивается либо начинается снова.

На Рис. 5 приведена примерная блок-схема программы. Напишите программу на языке программирования Turbo Pascal по данной блок-схеме.

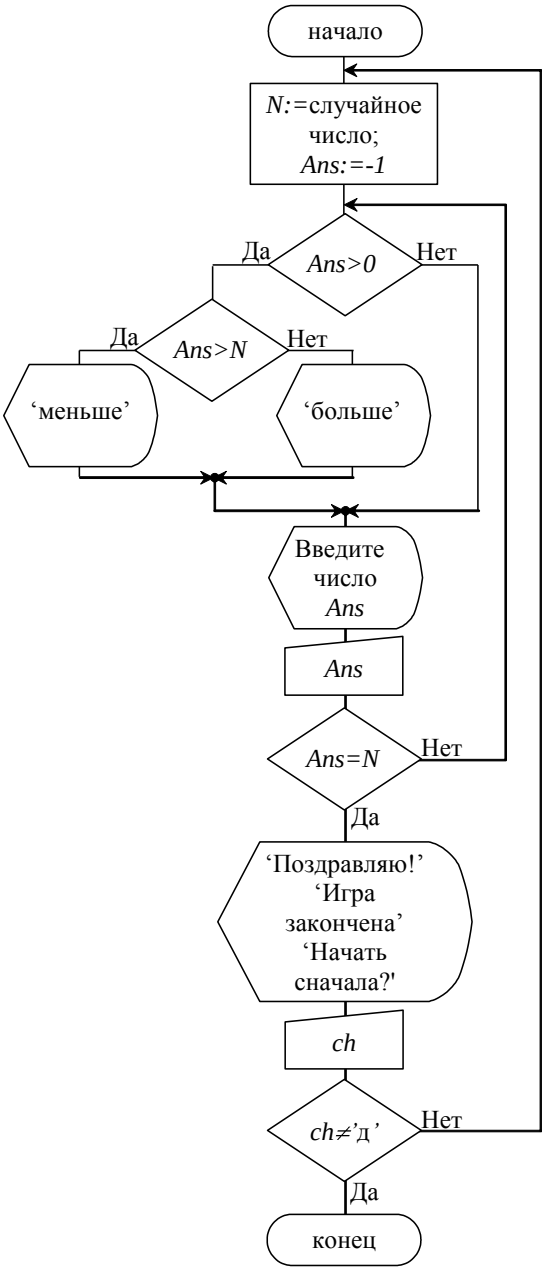


Рис. 5

Для получения случайного числа используйте встроенную функцию **Random**(<число>), где <число> – переменная целого типа. Функция генерирует случайное целое число в заданном диапазоне от [0...<число>).

Параметр <число> является необязательным. Если данный параметр опущен, то функция возвращает число действительного типа в диапазоне от [0...1).

Генератор случайных чисел инициализируется процедурой **Randomize**. Если не производить инициализацию, то при каждом запуске программы, функция **Random** будет возвращать одну и ту же последовательность чисел.

Пример:

```
var
  x: Real;
  y: Integer;
begin
  Randomize;
  ...
  x:=Random;                    {0<=x<1}
  y:=Random(100);               {0<=y<100}
  ...
end;
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Бермудский треугольник»* Пользователь вводит с клавиатуры три натуральных числа. Определить, можно ли построить треугольник, длины сторон которого относятся как эти натуральные числа.

Занятие 6

*Изучается унифицированная алгоритмическая структура «Выбор», оператор выбора **CASE**. Отрабатывается использование оператора выбора в программах.*

Он знак подаст: и все хлопочут;
Он пьет: все пьют и все кричат;
Он засмеется: все хохочут;
Нахмурит брови: все молчат;
А.С. Пушкин, "Евгений Онегин"

В унифицированной структуре «Развилка» условие может принимать только два значения: `True` (истина) и `False` (ложь). Поэтому и количество направлений движения в алгоритме возможно тоже два. Для осуществления алгоритма с более широким диапазоном выбора можно использовать вложенные условные операторы:

```
if <условие_1> then <оператор_1>
else
  if <условие_2> then <оператор_2>
  else
    if <условие_3> then <оператор_3>
    else
      if <условие_4> then <оператор_4>
      else
      ................
```

Пример 1: Перевод цифрового балла в строковую оценку.

```
..............
if i=5 then Writeln('отлично')
else
  if i=4 then Writeln('хорошо')
  else
    if i=3 then Writeln('удовлетворительно')
    else
      if (i=2) then Writeln('неудовлетворительно')
      else
        if (i=1) then Writeln('плохо')
        else
          Writeln('ошибочная оценка');
          ..............
```

Конструкция получается достаточно громоздкой, особенно если приходится проверять не пять, а гораздо больше условий. В таких случаях гораздо удобней использовать структуру выбора.

Унифицированная структура выбора

Операция выбора заключается в выборе одного варианта из нескольких предложенных. В зависимости от того, какое значение имеет ключ выбора, выполняется соответствующее этому ключу действие. Другими словами, если ключ `i` принимает значение `C1`, то выполнится действие `S1`, а если `C2` – то выполняется действие `S2` и т. д. С помощью структурной схемы это можно представить так:

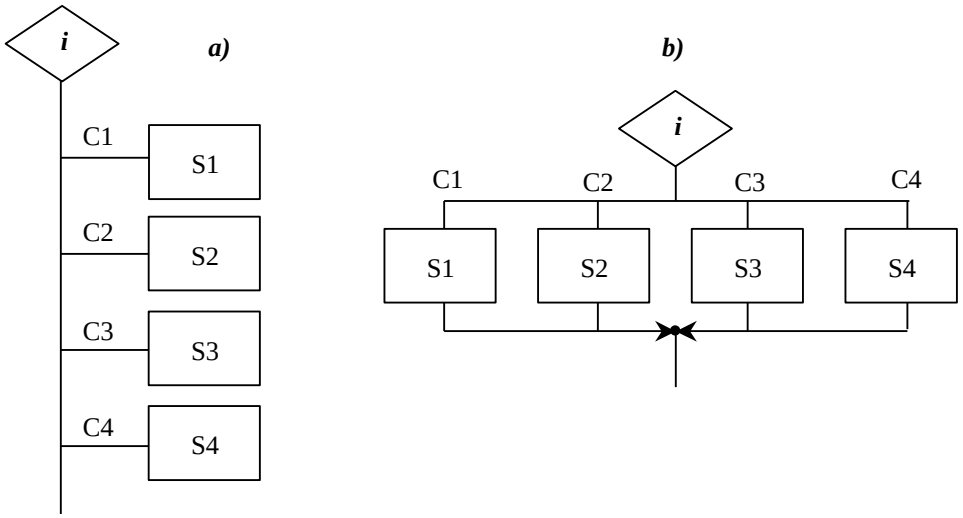


Рис. 6

Оператор выбора CASE

Оператор выбора **`case`** является обобщением оператора **`if`** — он дает возможность выполнить одно из нескольких действий в зависимости от значения ключа выбора. В качестве ключа выбора используется выражение, которое располагается между ключевыми словами **`case`** и **`of`**. Результатом этого выражения может быть только значение порядкового типа, общее количество элементов которого не превышает 65 535.

С помощью этого оператора можно выбрать вариант из любого количества вариантов. Структура этого оператора на языке Pascal будет выглядеть так:

```
case i of
  C1: <оператор1>;
  C2: <оператор2>;
  ..................
  CN: <операторN>;
else
  <оператор>;
end;
```

В этой структуре:
`i` – выражение порядкового типа, значение которого вычисляется;
`C1`, `C2`, ..., `CN` – константы, с которыми сравнивается значение выражения `i`;

<оператор1>, <оператор2>, <операторN> – операторы простые или составные, из которых выполняется тот, с константой которого совпадает значение выражения i;

<оператор> – оператор, который выполняется, если значение выражения i не совпадает ни с одной из констант C1, C2, ..., CN.

Ветвь оператора **else** является необязательной. Если она отсутствует, и значение выражения i не совпадет ни с одной из перечисленных констант, весь оператор рассматривается как пустой. В отличие от оператора **if** перед словом **else** точку с запятой можно ставить.

Используя оператор **case**, первый пример можно переписать:

Пример 2:

```
case i of
  5: Writeln('отлично');
  4: Writeln('хорошо');
  3: Writeln('удовлетворительно')
  2: Writeln('неудовлетворительно');
  1: Writeln('плохо');
else
  Writeln('ошибочная оценка');
end;
```

Если для нескольких констант нужно выполнять один и тот же оператор, их можно перечислить через запятую (или даже указать диапазон, если возможно), сопроводив их одним оператором.

Пример 3:

```
case i of
  1, 2: Writeln('Незачет');
  3..5: Writeln('Зачет');
else
  Writeln('ошибочная оценка');
end;
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«День недели»*. По номеру дня недели вывести его название. При вводе нуля – программа заканчивается.
2. *«Больше ясности»*. Пользователь вводит с клавиатуры время и дату в цифровом формате, например: время – 20 30; дата – 26 10 2001. Написать программу вывода на экран времени и даты в «текстово-цифровом» формате, например: 20 часов 30 минут 26 октября 2001 года.
3. *«Какой Вы сказали размер – сороковой?»*. Написать программу перевода арабских цифр от 1 до 10 в римские цифры.
4. *«Кукушка, кукушка, сколько мне...»*. Пользователь вводит с клавиатуры дату, например: 7 07 2001. Посчитать и вывести на экран: сколько дней осталось до конца года, каким по счету этот день является от начала года.

Занятие 7

*Повторение пройденного материала: оператор выбора **CASE**. Написание программы по готовой блок-схеме. Цель занятия: закрепление понятия структуры выбора; отработка ее использования на примерах различных программ.*

...И о былом воспоминать?
А. С. Пушкин, "Е. Онегин"

Калькулятор

Написать программу, выполняющую арифметические действия: сложение, вычитание, умножение, деление вещественное; функции взятия модуля, квадратного корня, возведения в квадрат, взятия **sin**, **cos**, **tg**, **ln** аргумента. Используйте блок-схему, представленную на **Рис. 7**. Описание математических функций приведено ниже.

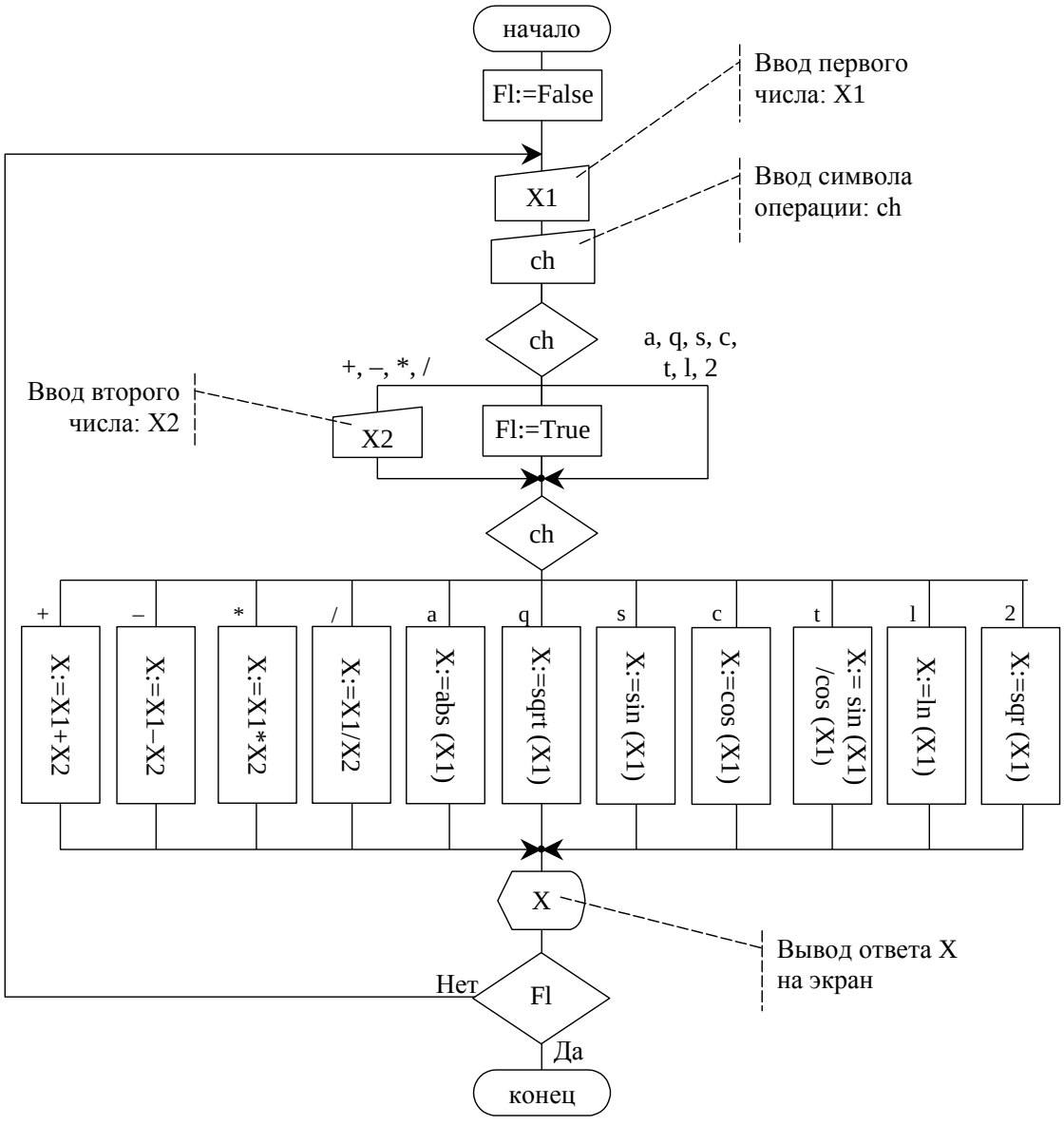


Рис. 7

Описание программы «Калькулятор»

В программе используются переменные: X, X1, X2 — вещественного типа; ch — символьного типа; Fl — логического типа.

Переменные X1 и X2 являются операндами выражения, ch содержит символ операции, а в переменную X помещается ответ. Если вычисляется функция (abs(), sqrt(), sqr(), sin(), cos(), ln()), а не выражение (сложение, вычитание, умножение, деление), вторая переменная X2 не используется. Логическая переменная введена для выхода из программы — если введен любой символ, не являющийся символом операции или не обозначающий функцию, осуществляется выход из цикла и завершение программы.

В блок-схеме используется цикл с постусловием для повторения программы и две универсальные структуры выбора: одна – для ввода второго операнда X2, другая – для выполнения соответствующего оператора.

НЕКОТОРЫЕ МАТЕМАТИЧЕСКИЕ ФУНКЦИИ И ПРОЦЕДУРЫ TURBO PASCAL 7.0

В системную библиотеку Turbo Pascal 7.0 кроме математических операций **+**, **–**, *****, **/**, **div** и **mod** встроены процедуры и функции, которые помогают осуществлять некоторые математические преобразования. Ниже приводится описание некоторых из них.

Математические функции

функция	действие	замечания
Abs(X)	Возвращает абсолютную величину параметра	Параметр X – выражение вещественного или целочисленного типа. Результат того же типа, что и X является абсолютной величиной X.
Cos(X)	Возвращается косинус параметра	Параметр X - выражение вещественного типа. Результат - косинус числа X, где X – угол в радианах
Sin(X)	Возвращает синус параметра	Параметр X - выражение вещественного типа. Возвращает синус угла X в радианах
ArcTan(X)	Возвращает арктангенс параметра	В языке Turbo Pascal нет встроенной функции Tan, но тангенс может быть вычислен с помощью выражения: Sin(X) / Cos(X)
Exp(X)	Возвращает экспоненту параметра	Возвращаемое значение: Значение e возведенное в степень X, где e – основание натуральных логарифмов
Ln(X)	Возвращает натуральный логарифм аргумента	Возвращает натуральный логарифм вещественного выражения X
Sqr(X)	Возвращает квадрат параметра	Число X или вещественное, или целочисленное выражение. Результат, того же самого типа, что и X, является квадратом числа X
Sqrt(X)	Возвращает квадратный корень аргумента	X – выражение вещественного типа. Результатом будет квадратный корень из X
Odd(X)	Проверяет параметр на	Значение функции Odd(X) равно True, если X –

	нечетность	нечетное число

Математические процедуры

процедура	действие	замечания
Inc(X, N)	Увеличивает значение переменной	Параметр X – переменная перечислимого типа или переменная типа PChar, если допускается расширенный синтаксис, а N – выражение целочисленного типа. Значение X увеличивается на 1, если параметр N не определен, или на N, если праметр N определен, то есть Inc(X) соответствует X:=X+1, а Inc(X, N) соответствует X:=X+N. С помощью Inc генерируется более оптимизированный код, особенно полезный в плотном цикле
Dec(X, N)	Уменьшает значение переменной	Параметр X – переменная перечислимого типа или переменная типа PChar, если допускается расширенный синтаксис, а N – выражение целочисленного типа. Значение X уменьшается на 1, если параметр N не определен, или на N, если праметр N определен, то есть Dec(X) соответствует X:=X–1, а Dec(X, N) соответствует X:=X–N. С помощью Dec генерируется более оптимизированный код, особенно полезный в плотном цикле

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Помни о былом...»* Модернизировать программу «Калькулятор» так, чтобы результат последнего вычисления оставался в переменной `X1` и мог использоваться последующих вычислениях, тем самым появилась возможность вычислять значения выражений, состоящих из нескольких операций.

Занятие 8

Изучаются структурированные типы данных — массивы: одномерные, многомерные, а так же строковый тип данных. Вводится понятие типизированной константы.

...Под цифрами, в порядке, строй за строем,
Не позволять им в сторону брести,
Как войску, в пух рассыпанному боем!
А. С. Пушкин, "Домик в Коломне", 1830

Зачастую в программе приходится использовать несколько переменных одного и того же типа для работы с одинаковыми данными. Например, нам нужно найти сумму и произведение всех цифр шестизначного числа. Для этого можно завести шесть переменных (`Num1, Num2, Num3, Num4, Num5, Num6`), в которых будут храниться цифры данного числа `N`. Далее можно проводить различные манипуляции с этими переменными, в частности, – найти сумму и произведение.

```
Program Task_8_1;
var
  N, N1: LongInt;
  Num1, Num2, Num3, Num4, Num5, Num6: Byte;
  Sum: Integer;
  Multi: LongInt;
begin
  Write('Введите целое шестизначное число: ');
  Readln(N);
  N1:=N;
  Num6:=N1 mod 10;
  N1:=N1 div 10;
  Num5:=N1 mod 10;
  N1:=N1 div 10;
  Num4:=N1 mod 10;
  N1:=N1 div 10;
  Num3:=N1 mod 10;
  N1:=N1 div 10;
  Num2:=N1 mod 10;
  N1:=N1 div 10;
  Num1:=N1 mod 10;
  N1:=N1 div 10;
  Sum:=Num1 + Num2 + Num3 + Num4 + Num5 + Num6;
  Multi:= Num1 * Num2 * Num3 * Num4 * Num5 * Num6;
  Writeln('Сумма всех цифр в числе ', N, ': ', Sum);
  Writeln('Произведение цифр в числе ', N, ': ', Multi);
  Readln;
end.
```

Данный подход не является особо оптимальным и гибким, так как разобрать на цифры можно только не более чем шестизначное число. Кроме того, количество «однородных» переменных такого плана во многих задачах может не

ограничиваться только шестью и быть в десятки раз больше. Как результат – увеличение кода программы, прежде всего из-за повторения одинаковых действий с каждой из этих переменных.

Существует более удобный и простой инструмент для работы с составными данными однородной структуры — массивы.

СТРУКТУРИРОВАННЫЕ ТИПЫ ДАННЫХ: МАССИВ

Массив объединяет элементы одного типа данных. Более формально массив можно определить следующим образом:

Массив[3] — это структура данных, которая представляет собой однородную, фиксированную по размеру и конфигурации совокупность элементов простой или составной структуры, упорядоченных по номерам.

В качестве иллюстрации можно представить себе шкаф, имеющий множество пронумерованных ящиков. Доступ к содержимому конкретного ящика (элементу данных) осуществляется после выбора ящика по его номеру (индексу). Индексная переменная, служащая для указания отдельного элемента массива, должна быть простого типа (например, типа **`Byte`**).

Используя массив, Вы вместо множества маленьких шкафов, каждый из которых имеет только один ящик, применяете один большой шкаф, в котором имеется огромное количество ящиков. Кроме того, существует еще одно отличие с точки зрения реализации — элементы массива в памяти хранятся по соседству, в то время как одиночные элементы простого типа не гарантируют такого расположения данных в памяти.

Наряду с одномерными массивами (шкафами с ящиками в один ряд) в Turbo Pascal используются также двухмерные массивы данных (матрицы), которые можно представить как шкаф с множеством ящиков, положение которых определяется двумя координатами — по горизонтали и по вертикали. В двухмерных массивах данных координата по горизонтали соответствует номеру строки, а координата по вертикали — номеру ряда. Тоже можно сказать и о трехмерных массивах.

Массивы большей размерности на практике встречаются редко.

Размер массива Turbo Pascal ограничивается только объемом рабочей памяти компьютера.

Одномерные массивы

В математике и информатике массив называется одномерным, если для получения доступа к его элементам достаточно одной индексной переменной. Так, чтобы найти в шкафу с одним рядом ящиков нужный Вам ящик, достаточно знать его номер и точку начала отсчета.

Схематично одномерный массив можно представить как последовательность нумерованных ячеек:

[3] В литературе, наряду с термином «массив», часто можно встретить термины «матрица», «таблица» или «вектор». Суть всех этих терминов одна и та же.

Рис. 8

В Turbo Pascal одномерный массив в разделе описания переменных объявляется следующим образом:

```
var
  <Имя_массива>: array[N1..N2] of <Тип_данных>;
```

где <Имя_массива> – правильный идентификатор;
N1, N2 – начальный и конечный индекс элементов массива;
<Тип_данных> – любой тип Turbo Pascal;
array, of – зарезервированные слова (*массив*, *из*).

Начальный и конечный индекс определяют размер массива — количество элементов, входящих в него. Квадратные скобки, обрамляющие список, — требование синтаксиса.

Например, массив из 10 целых чисел можно описать следующим образом:

```
var
 Vector: array[1..10] of Integer;
```

Другой способ создания переменной-массива заключается в описании пользовательского типа-массива в разделе описания типов **type** и последующего описания переменной этого типа в разделе **var**:

```
type
  TVector: array[1..10] of Integer;
var
  Vector: TVector;
```

Доступ к каждому элементу-ячейке массива можно получить путем указания значения индекса в квадратных скобках после идентификатора имени массива:

```
Vector[5]:=404;
```

Здесь пятому элементу массива Vector присваивается значение 404.

В качестве индекса элемента можно использовать переменную порядкового типа. Например, для заполнения массива из 10 элементов можно использовать оператор цикла с параметром:

```
for i:=1 to 10 do
  begin
    Write('Введите ', i, 'элемент массива - ');
    Readln(Vector[i]);
```

```
end;
```

С использованием массива предыдущая программа становится более универсальной, так как теперь можно вводить число с любым количеством цифр. Кроме того, описание одного массива любого размера менее утомительно, чем описание множества переменных по отдельности. Уменьшение же размера программы достигается за счет использования цикла.

```
Program Task_8_2;
var
  N, N1: LongInt;
  i: Integer;
  Num: array[1..10] of Byte;
  Sum: Integer;
  Multi: LongInt;
begin
  Write('Введите целое число: ');
  Readln(N);
  N1:=N;
  i:=0;
  repeat
    Inc(i);
    Num[i]:=N1 mod 10;
    N1:=N1 div 10;
  until N1 = 0;
  Sum:=0; Multi:=1;
  repeat
    Sum:=Sum + Num[i];
    Multi:=Multi * Num[i];
    Dec(i);
  until i = 0;
  Writeln('Сумма всех цифр в числе ', N, ': ', Sum);
  Writeln('Произведение цифр в числе ', N, ': ', Multi);
  Readln;
end.
```

Многомерные массивы

Массивы могут иметь и более одного измерения. В таких случаях говорят о многомерных массивах. Многомерные массивы широко используются в математике, статистике и некоторых других прикладных науках.

Схематично двухмерный массив можно представить как совокупность индексированных ячеек (**Рис. 9**). Причем индекс ячейки определяется номером строки и номером столбца.

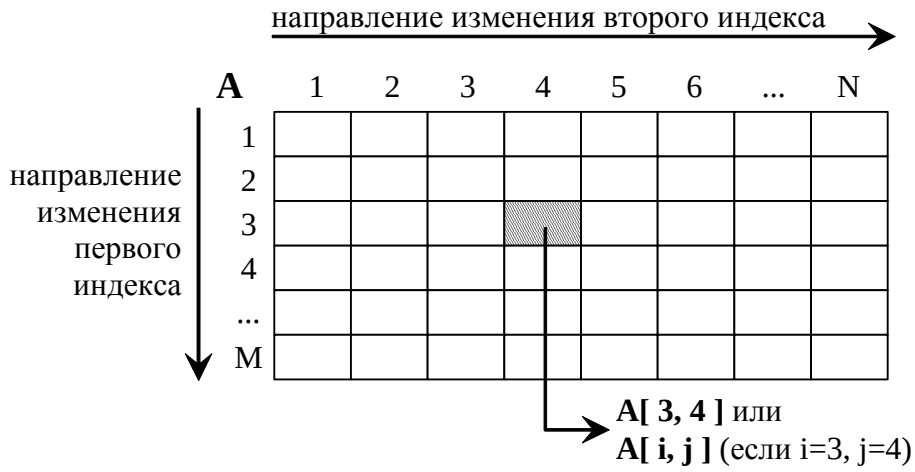


Рис. 9

Объявление двухмерного массива аналогично объявлению одномерного массива, с той лишь разницей, что указывается не один диапазон значений номеров элементов, а два.

Выглядит объявление двухмерного массива следующим образом:

```
var
  Matrix: array[1..25, 1..80] of Char;
```

Либо можно описать массив в разделе описания типов:

```
type
  TMatrix: array[1..25, 1..80] of Char;
var
  Matrix: TMatrix;
```

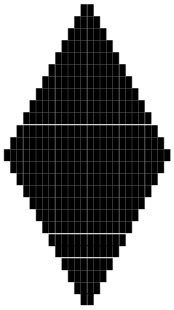
Переменной Matrix соответствует некоторая матрица или двухмерный массив, причем каждому элементу данного массива может быть поставлена в соответствие некоторая позиция на экране Вашего компьютера[4]. Таким образом, каждый элемент массива может быть использован для хранения одного символа, отображенного в соответствующей позиции экрана. Напишем простую программу, которая по определенному закону заполняет массив, а затем выводит его на экран.

```
Program ScreenMatrix;
var
  i, j: Byte;
  Matrix: array[1..25, 1..80] of Char;
begin
  {заполнение первой строки матрицы символов}
  for j:=1 to 80 do
    if (j=40) then Matrix[1, j]:='■'
      else Matrix[1, j]:=' ';
  {заполнение первой четверти матрицы символов}
```

[4] На экране компьютера помещается 80 столбцов (с номерами от 1-го до 80-го) и 25 строк (с номерами от 1-ой до 25-ой).

```
    for i:=2 to 13 do
      for j:=2 to 40 do
        if (Matrix[i-1, j+1]='') or (Matrix[i-1, j]='')
           then
                Matrix[i, j]:=''
           else Matrix[i, j]:=' ';
    {заполнение второй четверти матрицы отражением}
    for i:=1 to 13 do
      for j:=1 to 40 do
        Matrix[i, 81-j]:=Matrix[i, j];
    {заполнение остальных строк матрицы отражением}
    for i:=1 to 13 do
      for j:=1 to 80 do
        Matrix[26-i, j]:=Matrix[i, j];
    {вывод матрицы на экран}
    for i:=1 to 25 do
      for j:=1 to 80 do
        if (j=80) and (i=25)
           then
                Readln
           else Write(Matrix[i, j]);
  end.
```

Так как мы рисуем симметричную фигуру, то достаточно это сделать только в одной четверти. Все остальные четверти можно заполнить простым копированием соответствующих элементов матрицы с отражением. Пример копирования:



```
Matrix[26-i, j]:=Matrix[i, j];
```

Все необходимые пояснения есть в тексте программы.

Результатом работы программы `ScreenMatrix` является ромб, представленный на Рис. 10.

Рис. 10

СТРОКОВЫЙ ТИП ДАННЫХ

Тип **`String`** (строка) в Turbo Pascal широко используется для обработки текстов. Он во многом похож на одномерный массив символов **`array`**`[0..N]` **`of`** `Char`, однако, в отличие от последнего, количество символов в строке-переменной может меняться от 0 до N, где N — максимальное количество символов в строке. Значение N определяется объявлением типа **`String[N]`** и может быть любой константой порядкового типа, но не больше 255. Turbo Pascal разрешает не указывать N, в этом случае длина строки принимается максимально возможной, а именно N=255.

Строка в Turbo Pascal трактуется как цепочка символов. К любому символу в строке можно обратиться точно так же, как к элементу одномерного массива **`array`**`[0..N]` **`of`** `Char`, например:

```
  var
```

```
  St: String;
begin
  ......
  if St[5] = 'A' then ...
end.
```

Самый первый байт в строке имеет индекс 0 и содержит текущую длину строки. Первый значащий символ строки занимает второй байт и имеет индекс 1.

К строкам можно применять операцию «+» – сцепление, например:

```
St1:='Turbo ';
St2:='Pascal';
St1:=St1 + St2;
```

В результате переменная `St1` примет значение: 'Turbo Pascal'.

Если длина сцепленной строки превысит максимально допустимую длину N, то «лишние» символы отбрасываются.

ТИПИЗИРОВАННЫЕ КОНСТАНТЫ

В Turbo Pascal допускается использование типизированных констант. Они задаются в разделе объявления констант следующим образом:

```
<идентификатор>: <тип> = <значение>,
```

где `<идентификатор>` – идентификатор константы;
`<тип>` – тип константы;
`<значение>` – значение константы.

Типизированным константам можно присваивать другие значения в ходе выполнения программы, поэтому фактически они представляют собой переменные с начальными значениями.

Типизированные константы могут быть любого типа, кроме файлов.

Константы-массивы

В качестве начального значения типизированной константы – массива используется список констант, отделенных друг от друга запятыми; список заключается в круглые скобки, например:

```
const
  vector: array[1..5] of Byte = (0, 1, 0, 3, 8);
```

При объявлении многомерных констант-массивов множество констант, соответствующих каждому измерению, заключается в дополнительные круглые скобки и отделяется от соседнего множества запятыми. В результате образуются вложенные структуры множеств, причем глубина вложения должна соответствовать количеству измерений (размерности) массива. Самые внутренние множества констант связываются с изменением самого правого индекса массива.

```
const
  Matrix: array[1..3, 1..5] of Byte =
```

```
        (( 0,  1,  2,  3,  4)
( 5,  6,  7,  8,  9)
(10, 11, 12, 13, 14));
```

Количество переменных в списке констант должно строго соответствовать объявленной длине массива по каждому измерению.

При объявлении массива символов можно использовать то обстоятельство, что все символьные массивы и строки в Turbo Pascal хранятся в упакованном формате, поэтому в качестве значения массива – константы типа `Char` допускается задание символьной строки соответствующей длины. Два следующих объявления идентичны:

```
const
  Digit1: array[0..9] of Char =
    ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9');
  Digit2: array[0..9] of Char = '0123456789';
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Сколько нас?»* Пользователь вводит натуральное число. Посчитать сколько раз каждая из цифр повторяется в этом числе.
2. *«А нас?»* Пользователь вводит с клавиатуры строку символов. Посчитать сколько раз каждый символ повторяется в данной строке.
3. *«Лучше один раз увидеть...»* Ко второму заданию построить гистограмму.
4. *«Нам лишнего не нужно...»* Пользователь вводит с клавиатуры строку символов. Получить аналогичную строку, в которой отсутствуют все символы пробелов, и вывести ее на экран.
5. *«А я маленький – ниже стремени...»* Пользователь вводит с клавиатуры строку символов. Преобразовать все строчные литеры в прописные литеры.
6. *«Оля – ялО»* Пользователь вводит с клавиатуры строку символов. Вывести на экран инвертированную строку.
7. *«У нас все записано...»* Пользователь вводит с клавиатуры номер дня недели. Вывести на экран название этого дня. **Указание**: используйте массив строк – названий дней недели.
8. *«Перестройка...»* Дан двухмерный массив (матрица) размерности ***m*n***, элементами которого являются целые числа. Выполнить «зеркальное отображение» элементов матрицы относительно вертикальной оси симметрии (поменять местами элементы первого столбца с последним, второго с предпоследним и так далее).
9. *«Шило на мыло...»* Дана квадратная матрица порядка ***n***, элементами которой являются целые числа. Поменять местами элементы соответствующих строк и столбцов.
10. *«Как голова кружится...»* Дана квадратная матрица порядка ***n***, элементами которой являются целые числа. Вывести значения элементов на печать, выполнив обход матрицы по «спирали».
11. *«Кис-кис...»* Используя типизированную константу в виде двухмерного массива символов (или одномерного массива строк), нарисуйте на

экране в текстовом режиме рисунок представленный на **Рис. 11**. Учтите, что размер символа на экране по высоте в два раза больше, чем по ширине. Символ '█' имеет код #219.

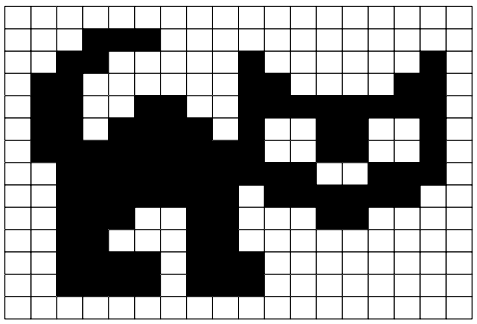


Рис. 11

Занятие 9

Закрепление темы «Массивы». Изучаются методы сортировки массивов и поиска элемента в массиве.

Ровняясь, строятся полки...
А. С. Пушкин, "Полтава"

При работе с массивами данных чаще всего приходится осуществлять либо сортировку, либо поиск элемента в данном массиве. Рассмотрим некоторые способы сортировки и поиска.

СОРТИРОВКА МАССИВОВ

При решении задачи сортировки обычно выдвигается требование минимального использования дополнительной памяти, из которого вытекает недопустимость применения дополнительных массивов.

Для оценки быстродействия алгоритмов различных методов сортировки, как правило, используются два показателя:

- количество присваиваний;
- количество сравнений.

Все методы сортировки можно разделить на две большие группы:

- прямые методы сортировки;
- улучшенные методы сортировки.

Прямые методы сортировки по принципу, лежащему в основе метода, в свою очередь разделяются на три подгруппы:

1. сортировка вставкой (включением);
2. сортировка выбором (выделением);
3. сортировка обменом («пузырьковая» сортировка).

Улучшенные методы сортировки основываются на тех же принципах, что и прямые, но используют некоторые оригинальные идеи для ускорения процесса сортировки. Прямые методы на практике используются довольно редко, так как имеют относительно низкое быстродействие. Однако они хорошо показывают суть основанных на них улучшенных методов. Кроме того, в некоторых случаях (как правило, при небольшой длине массива или особом исходном расположении элементов массива) некоторые из прямых методов могут даже превзойти улучшенные методы.

Сортировка вставкой

Массив разделяется не две части: отсортированную и неотсортированную. Элементы из неотсортированной части поочередно выбираются и вставляются в отсортированную часть так, чтобы не нарушить в ней упорядоченность элементов. В начале работы алгоритма в качестве отсортированной части массива принимают только один первый элемент, а в качестве неотсортированной части — все остальные элементы.

Таким образом, алгоритм будет состоять из $n-1$ повторений (n – размерность массива) четырех действий:

1. взятие очередного i-го неотсортированного элемента и сохранение его в дополнительной переменной;

2. поиск позиции *j* в отсортированной части массива, в которой присутствие взятого элемента не нарушит упорядоченности элементов;
3. сдвиг элементов массива от ***i***–1-го до *j*-го вправо чтобы освободить найденную позицию вставки;
4. вставка взятого элемента в найденную *j*-ю позицию.

Для реализации данного метода можно предложить несколько алгоритмов, которые будут отличаться способом поиска позиции вставки. Рассмотрим схему реализации одного из возможных алгоритмов.

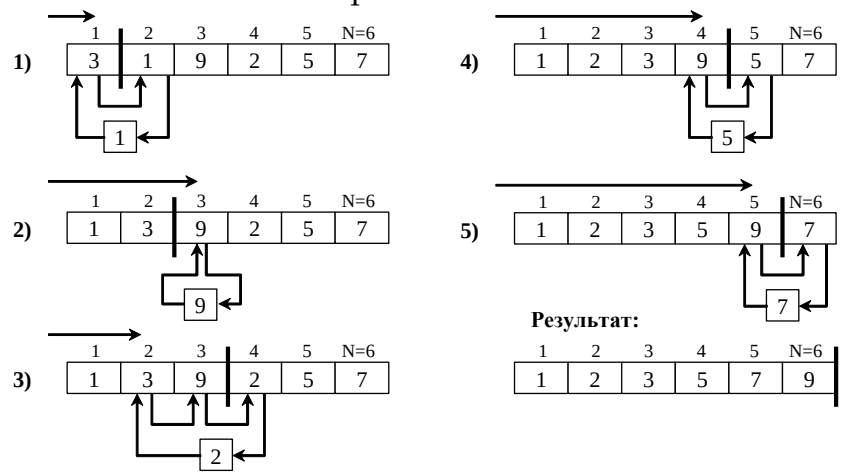


Рис. 12. Схема сортировки массива методом вставки.

Программа, реализующая рассмотренный алгоритм, будет иметь следующий вид:

```
Program InsertionSort;
const
  N = 20; {длина массива}
var
  Vector: array[1..N] of Real;
  B: Real;
  i, j, k: Integer;
begin
  Writeln('Введите элементы массива:');
  for i:=1 to N do Read(Vector[i]);
  Readln;
  {================================}
  for i:=2 to N do
  begin
    B:=Vector[i];  {Взятие неотсортированного элемента}
    {Цикл поиска позиции вставки}
    j:=1;
    while (B > Vector[j]) do j:=j+1;
    {цикл сдвига эл-в для освобождения позиции вставки}
    for k:=i-1 downto 1 do Vector[k+1]:=Vector[k];
    {вставка взятого элемента в найденную позицию}
    Vector[j]:=B;
  end;
  {================================}
  Writeln('Отсортированный массив: ');
```

```
    for i:=1 to N do Write(Vector[i]:8:3);
    Writeln;
  end.
```

Сортировка выбором

Находим (выбираем) в массиве элемент с минимальным значением на интервале от 1-го элемента до ***n***-го (последнего) элемента и меняем его местами с первым элементом. На втором шаге находим элемент с минимальным значением на интервале от 2-го до ***n***-го элемента и меняем его местами со вторым элементом. И так далее для всех элементов до ***n***–1-го. Схема алгоритма прямого выбора представлена на рисунке:

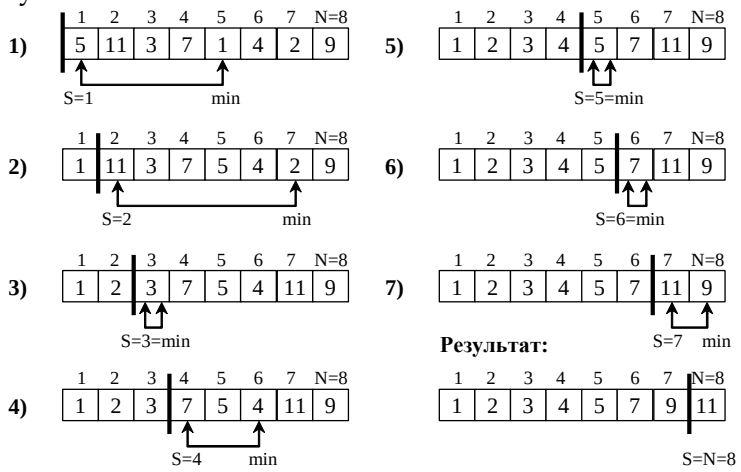


Рис. 13. Схема алгоритма прямого выбора.

Программа, реализующая метод выбора, будет иметь следующий вид:

```
Program SelectionSort;
const
  N = 20;  {длина массива}
var
  Vector: array[1..N] of Real;
  Min: Real;
  Imin, S: Integer;
  i: Integer;
begin
  Writeln('Введите элементы массива: ');
  for i:=1 to N do Read(Vector[i]); Readln;
  {=====================================}
  for S:=1 to N-1 do
    begin
      {поиск минимального элемента в диапазоне}
      Min:=Vector[S];
      Imin:=S;
      for i:=S+1 to N do
        if Vector[i] < Min then
          begin
            Min:=Vector[i];
            Imin:=i;
          end;
      {обмен местами минимального и S-го элементов}
```

```
      Vector[Imin]:=Vector[S];
      Vector[S]:=Min;
    end;
  {=====================================}
  Writeln('Отсортированный массив: ');
  for i:=1 to N do Write(Vector[i]:8:2);
end.
```

Сортировка обменом (метод «пузырька»)

Слева направо поочередно сравниваются два соседних элемента, и если их взаиморасположение не соответствует заданному условию упорядоченности, то они меняются местами. Далее берутся два следующих соседних элемента и так далее до конца массива.

После одного такого прохода на последней ***n***-ой позиции массива будет стоять максимальный элемент («всплыл» первый пузырек). Поскольку максимальный элемент уже стоит на своей последней позиции, то второй проход обменов выполняется до ***n*–1**-го элемента. И так далее. Всего требуется ***n***–1 проход.

Рассмотрим схему алгоритма сортировки методом прямого обмена по неубыванию.

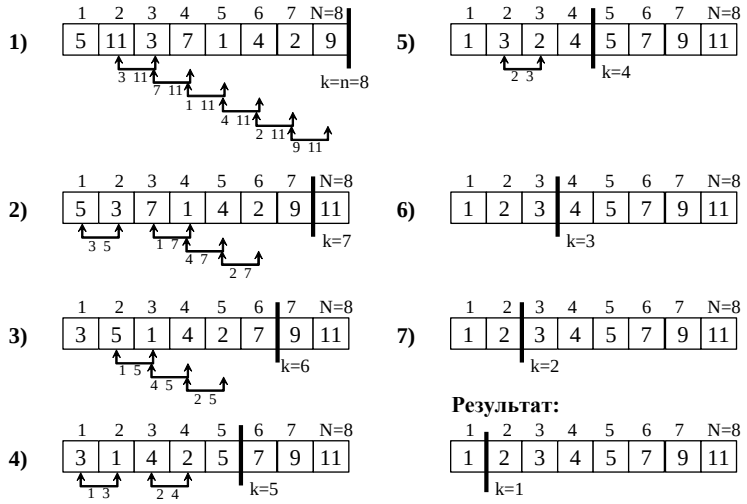


Рис. 14. Схема алгоритма сортировки методом прямого обмена по неубыванию.

Программа, реализующая метод обмена («пузырька»), будет иметь следующий вид:

```
Program BubbleSort;
const
  N = 20;
var
  Vector: array[1..N] of Real;
  B: Real;
  i, k: Integer;
begin
  Writeln('Введите элементы массива:');
  for i:=1 to N do Read(Vector[i]); Readln;
  {=====================================}
  {"Всплывание" очередного максимального элемента}
```

```
    for k:=N downto 2 do
      for i:=1 to k-1 do
        if Vector[i]>Vector[i+1] then
          begin
            B:=Vector[i];
            Vector[i]:= Vector[i+1];
            Vector[i+1]:=B
          end;
    {=====================================}
    Writeln('Отсортированный массив:');
    for i:=1 to N do Write(Vector[i]:8:2);
  end.
```

Двоичный поиск

Этот метод поиска называют еще бинарным или поиском делением пополам.

Алгоритм двоичного поиска допустимо использовать для нахождения заданного элемента только в упорядоченных массивах. Рассмотрим его на примере массива, упорядоченного по неубыванию (`Vector[i] <= Vector[i+1]`).

Исходный массив делится пополам и для сравнения выбирается средний элемент. Если он совпадает с искомым, то поиск заканчивается. Если же средний элемент меньше искомого, то все элементы левее его также будут меньше искомого. Следовательно, их можно исключить из зоны дальнейшего поиска, оставив только правую часть массива. Аналогично, если средний элемент больше искомого, то отбрасывается правая часть, а остается левая.

На втором этапе выполняются аналогичные действия над оставшейся половиной массива. В результате после второго этапа остается ¼ часть массива.

И так далее, пока или элемент будет найден, или длина зоны поиска станет равной нулю. В последнем случае искомый элемент найден не будет.

Рассмотрим схему двоичного поиска.

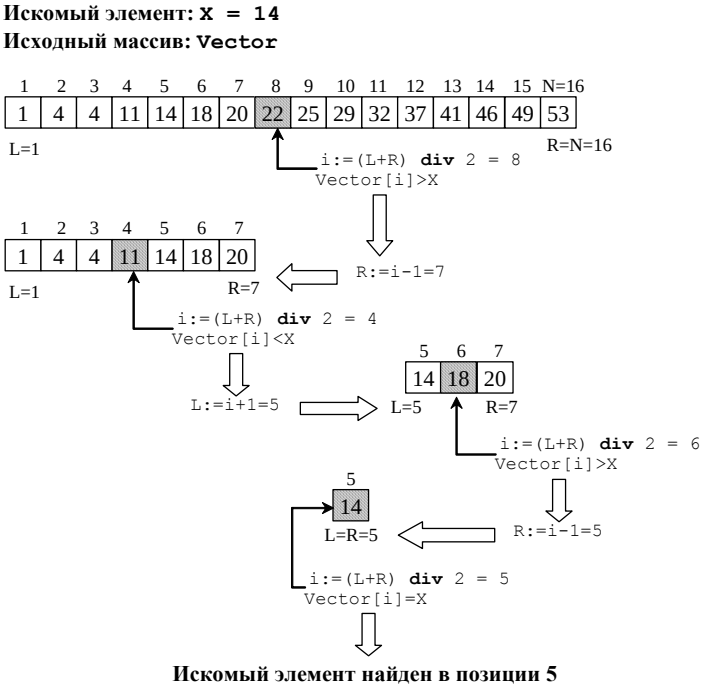


Рис. 15. Схема двоичного поиска.

Программа, реализующая один из возможных вариантов двоичного поиска, имеет следующий вид:

```
Program BinSearch;
const
  N = 20;   {длина массива}
var
  Vector: array[1..N] of Real;  {исходный массив}
  X: Real;  {искомый элемент}
  L, R: Integer;  {текущие границы зоны поиска}
  i: Integer;
begin
  Writeln('Введите элементы массива:');
  for i:=1 to N do Read(Vector[i]); Readln;
  Write('Введите искомый элемент: ');
  Readln(X);
  {=====================================}
  L:=1; R:=N;
  while (L<=R) do
    begin
      i:=(L+R) div 2;
      if Vector[i] = X then Break
      else
        if Vector[i]<X then L:=i+1
        else R:=i-1;
    end;
  if Vector[i] = X then
    Writeln('Искомый элемент найден на позиции ', i:3)
  else
    Writeln('Искомый элемент не найден');
  {=====================================}
end.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Где же, ты, где...»* Пользователь вводит неупорядоченный список фамилий учеников класса (5–10 фамилий). Отсортировать список по алфавиту (по возрастанию, по убыванию) и осуществить поиск нужной фамилии в списке.

Занятие 10

Изучаются процедуры и функции. Цель занятия: на простых примерах добиться понимания целесообразности использования процедур и функций.

Опять его на свет пустили для исполненья...
А. С. Пушкин, "Анджело"

При создании программы для решения достаточно сложной задачи используют ее деление на более простые подзадачи. Каждую подзадачу разбивают на еще более мелкие подзадачи и так далее, до легко программируемых подзадач.

В Turbo Pascal имеются различные средства для деления программы на части. На верхнем уровне деления (больших задач) — это модули (о них мы поговорим на следующем занятии), на нижнем уровне (элементарных подзадач) — это чаще всего процедуры и функции.

ПРОЦЕДУРЫ И ФУНКЦИИ

Процедуры и функции являются важным средством в большинстве языков программирования. С их помощью можно скомпоновать группу операторов для выполнения некоторого единого действия.

Процедуру или функцию можно вызывать из различных мест программы, она может возвращать вычисленные результаты, и ей можно передавать информацию, которую она использует для выполнения вычислений.

Для того чтобы процедура или функция начала работу, ее нужно вызвать (активизировать).

Рассмотрим пример без использования процедур и функций. Здесь производится последовательное выравнивание строки по левой стороне экрана, по центру и по правой стороне.

```
Program Task1;
var
  St1, St2: String;
  i: Integer;
begin
  St2:='Строка выравнивается по левой стороне экрана';
  Write(St2);
  Readln;
  {================================================}
  St1:='Строка выравнивается по центру экрана';
  St2:='';
  for i:=1 to (80-Length(St1)) div 2 do St2:=St2+' ';
  St2:=St2+St1;
  Write(St2);
  Readln;
  {================================================}
  St1:='Строка выравнивается по правой стороне экрана';
  St2:='';
  for i:=1 to 80-Length(St1) do St2:=St2+' ';
```

```
  St2:=St2+St1;
  Write(St2);
  Readln;
end.
```

Если выравнивание нужно будет выполнять достаточно часто, то всегда придется повторять один и тот же код при выводе каждой новой строки. Для того чтобы избежать столь «утомительных» повторений, нужно создать процедуры, осуществляющие соответствующие выравнивания и вызывать их в ходе выполнения программы.

Процедуры

Примеры встроенных в Turbo Pascal процедур нам уже известны. Вот некоторые из них: `Write()`, `Writln()`, `Read()`, `Readln()`, `Randomize`. При вызове каждой из этих процедур осуществляется сложная последовательность действий для получения необходимого результата. Можно конечно эту последовательность действий описывать каждый раз в программе, но гораздо проще идти по пути использования процедур. При этом программа разделяется на структурные блоки и становится более понятной и простой.

Описание процедуры имеет вид:

```
Procedure Имя(список формальных параметров);
  {раздел описания меток, констант, типов, переменных}
begin
  {раздел операторов}
end;
```

где `Имя` — любой допустимый идентификатор процедуры.

Список формальных параметров необязателен и может отсутствовать. Если же он есть, то в нем должны быть перечислены имена формальных параметров и их типы, например:

```
Procedure InputData(a: Real; b: Integer; c: Char);
```

Параметры в списке отделяются друг от друга точкой с запятой. Несколько следующих подряд однотипных параметров можно объединять в подсписки, перечисляя их через запятую.

```
Procedure InputData(a: Real; b: Integer; c1, c2: Char);
```

Любой из формальных параметров процедуры может быть либо параметром-значением, либо параметром-переменной, либо, наконец, параметром-константой.

Если необходимо объявить параметр-переменную, то перед ним необходимо поставить зарезервированное слово **var**, если параметр-константу — **const**, а если параметр-значение, то ставить ничего не нужно.

```
Procedure MyProc(var a: Real; b: Integer; const c: String);
```

Здесь: a – параметр-переменная;
b – параметр-значение;
c – параметр-константа.

Определение формального параметра тем или иным способом существенно, в основном, только для вызывающей программы: если формальный параметр объявлен как параметр-переменная, то при вызове подпрограммы ему должен соответствовать фактический параметр в виде переменной нужного типа; если формальный параметр объявлен как параметр-значение или параметр-константа, то при вызове ему может соответствовать произвольное выражение.

Размещение процедуры осуществляется в разделе описаний программы или процедуры, в которой она будет использоваться. Активизируется процедура в основной программе по имени. Если у процедуры имеются параметры, то они передаются в круглых скобках сразу после имени.

Рассмотрим создание и использование процедур на предыдущем примере выравнивания строк на экране. В разделе описаний программы Task2 создаем процедуры выравнивания строки. Сама строка будет передаваться в процедуру через формальный параметр St. Каждая процедура производит свои манипуляции с этой строкой и выводит результат на экран.

```
Program Task2;
var
  St1, St2: String;
  i: Integer;
  {==============Описание процедур================}
  {выравнивание по левому краю}
  Procedure WriteL(St: String);
  begin
    Writeln('   ', St);
  end;
  {--------------------------}
  {выравнивание по центру экрана}
  Procedure WriteM(St: String);
  var
    St2: String;
  begin
    St2:='';
    for i:=1 to (80-Length(St)) div 2 do St2:=St2+' ';
    St2:=St2+St;
    Writeln(St2);
  end;
  {--------------------------}
  {выравнивание по правому краю}
  Procedure WriteR(St: String);
  var
    St2: String;
  begin
    St2:='';
    for i:=3 to 80-Length(St) do St2:=St2+' ';
    St2:=St2+St;
    Writeln(St2);
  end;
```

```
  {===============Основная программа================}
begin
  WriteR('Директору пасеки № 13');
  WriteR('У. С. Пасечникову от');
  WriteR('В. Т. Пчелкина');
  WriteM('Заявление');
  WriteL('   Прошу разрешить мне участвовать в соревно-');
  WriteL('ваниях по сбору гречишного меда на приз');
  WriteL('газеты «Красный ПЧЕЛОВОД».');
  WriteR('1.04.2001');
  WriteR('Пчелкин Вася');
  Readln;
end.
```

Функции

Функция отличается от процедуры только тем, что результат ее работы возвращается в виде значения этой функции, и, следовательно, вызов функции может использоваться наряду с другими операндами в выражениях.

Примеры известных нам встроенных в Turbo Pascal функций: `Abs()`, `Sin()`, `Cos()`, `Ln()`, `Random()` и так далее.

Описание функции имеет вид:

```
Function Имя(список формальных параметров): Тип_результата;
  {раздел описания меток, констант, типов, переменных}
begin
  {раздел операторов}
end;
```

где `Имя` — любой допустимый идентификатор процедуры;

`Тип_результата` — тип результата, который возвращает функция.

Пример функции нахождения максимального из двух чисел:

```
Function Max(a, b: Integer):Integer;
begin
  if a>b then
    Max:=a
  else
    Max:=b;
end;
```

Вызов этой функции можно осуществить в программе следующим образом:

```
begin
  ...........
  Write('Введите два целых числа: ');
  Readln(x, y);
  Writeln('Максимальное из двух чисел: ', Max(x, y));
  ...........
end.
```

Кодирование информации

Кодирование данных чаще всего преследует цель защиты информации от несанкционированного доступа к ней. Методы и алгоритмы кодирования могут быть самыми разнообразными. Разберем простой пример кодирования строки.

Пусть нам дана последовательность из N символов:

abcdefgh

Переставим символы в каждой из последовательных пар:

badcfehg

Следующим шагом поменяем местами крайние символы в каждой тройке:

dabefchg

Последние два символа не переставляются, т. к. они не входят в тройку. Далее переставляем крайние символы в каждой четверке, затем – в каждой пятерке и т. д. до тех пор, пока не обменяются первый и последний символы.

eabdgchf
gabdechf
cabdeghf
habdegcf

В результате получаем строку:

fabdegch

Этот примитивный метод кодировки текстовых данных реализуем в программе.

```
Program StringCoder;
VAR
  Stg: String;
{=== Описание процедур и функций ===}
{процедура перестановки местами двух символов}
Procedure ChangeCh(var St: String; Ch1, Ch2: Byte);
var
  TmpCh: Char;
begin
   TmpCh:=St[Ch1];
   St[Ch1]:=St[Ch2];
   St[Ch2]:=TmpCh;
end;
{функция кодирования строки}
Function Coder(St: String): String;
var
  i, j, N: Integer;
begin
  N:=Length(St);
  for i:=1 to N-1 do
    for j:=1 to (N div (i+1)) do
      ChangeCh(St, j*(i+1)-i, j*(i+1));
  Coder:=St;
end;
{функция декодирования строки}
Function DeCoder(St: String) : String;
var
  i, j, N: Integer;
begin
```

```
  N:=Length(St);
  for i:=N-1 downto 1 do
    for j:=(N div (i+1)) downto 1 do
      ChangeCh(St, j*(i+1)-i, j*(i+1));
  DeCoder:=St;
end;
{======= Главная процедура ======}
BEGIN
  Write(' > ');
  Readln(Stg);              {ввод строки}
  Stg:=Coder(Stg);          {кодирование строки}
  Writeln(Stg);
  Readln;
  Stg:=DeCoder(Stg);        {декодирование строки}
  Writeln(Stg);
  Readln;
END.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

«Чистота — залог здоровья...» Сознайте процедуру очистки экрана.

«Третьего не дано...» Напишите функцию нахождения максимального и функцию нахождения минимального числа из трех данных чисел.

«С ног на голову...» Сознайте процедуру переворота строки.

«Порядок тот же» Напишите функцию переворота натурального числа.

«От сих пор до сих пор...» Напишите функцию получения случайного числа из заданного диапазона [a..b].

«ВЫШЕ – ниже» Напишите функцию преобразования строчных букв в прописные и функцию обратного преобразования.

«*Ранжир*» Сознайте процедуру сортировки одномерного массива целых чисел.

«*Сумматор*» Напишите функцию суммирования натуральных чисел от 1 до N.

«*Факториал*» Напишите функцию нахождения факториала N!.

«*Переводчик*» Сознайте процедуру преобразования целого числа в строку и строки в число.

«*Поменяемся*?» Написать программу кодирования-декодирования строки символов, используя метод замены символов, например

‘а’ → ‘к’

‘б’ → ‘ж’

‘в’ → ‘я’

‘г’ → ‘о’

…… и т. д.

Занятие 11

Рассматривается модульное программирование: создание модулей пользовательских процедур и функций, их использование в программах.

Все для мечтательницы нежной
В единый образ облеклись,
В одном Онегине слились.
А. С. Пушкин, "Е. Онегин"

В программе `StringCoder` из предыдущего занятия используются: процедура перестановки местами двух символов в строке, и две функции кодирования и декодирования строки. Эта программа существенно упростится, если поместить данные процедуры в отдельный модуль и просто подключить его к основной программе.

Разберем, что собой представляет модуль и как его подключить к программе.

Модули

Прогрессивным подходом в программировании считается использование модулей или блоков. Набор процедур и функций, объединенных в один блок (UNIT), может компилироваться независимо от главной программы. Благодаря этому время компиляции для больших программ может существенно сократиться, что очень важно при отладке программ, когда приходится их часто и многократно компилировать. Если попытаться дать модулю (UNIT) более формальное определение, то оно будет выглядеть приблизительно так:

> Модуль (UNIT) — программная единица, текст которой компилируется независимо (автономно). Она включает определения констант, типов данных, переменных, процедур и функций, доступных для использования в вызывающих программах. Однако внутренняя структура модуля (тексты программ и т.п.) скрыта от пользователя.

Структура модуля

Модуль имеет следующую структуру:

```
Unit <Имя>;
Interface
  <интерфейсная часть>
Implementation
  <исполняемая часть>
Begin
  <инициирующая часть>
End.
```

Здесь **Unit** – зарезервированное слово (единица); начинает заголовок модуля;
<Имя> – имя модуля (любой правильный идентификатор);

Interface – зарезервированное слово (интерфейс); начинает интерфейсную часть модуля;
Implementation – зарезервированное слово (выполнение); начинает исполняемую часть;
Begin – зарезервированное слово; начинает инициирующую часть модуля; конструкция **Begin** <инициирующая часть> необязательна;
End – зарезервированное слово – признак конца модуля.

Заголовок модуля состоит из зарезервированного слова **Unit** и следующего за ним имени модуля. Имя модуля должно совпадать с именем дискового файла, в котором помещается текст модуля. Если, например, модуль имеет заголовок

```
Unit MyCoder;
```

то исходный текст соответствующего модуля должен размещаться в файле MyCoder.pas.

Подключение модуля

Имя модуля служит для его связи с другими модулями и основной программой. Подключение модуля к программе осуществляется специальным предложением в самом начале раздела описаний

```
Uses <Список модулей>;
```

Здесь **Uses** – зарезервированное слово (использует);
<Список модулей> – список модулей, с которыми устанавливается связь; элементами списка являются имена модулей, отделяемых друг от друга запятыми, например:

```
Uses CRT, MyCoder, Graph;
```

Модули могут использоваться в других модулях. В этом случае их описание помещается сразу после зарезервированного слова **Interface**, либо сразу за словом **Implementation**, либо и там, и там.

Интерфейсная часть

Открывается зарезервированным словом **Interface**. В этой части содержится объявление всех глобальных типов, констант, переменных и подпрограмм (процедур и функций), которые должны стать доступными основной программе. При объявлении глобальных подпрограмм в интерфейсной части указывается только их заголовок, например:

```
Unit MyCoder;
Interface
Function Coder(St: String): String;
Function DeCoder(St: String): String;
```

Если теперь в основной программе написать предложение

```
Uses MyCoder;
```

то в программе станут доступными функции `Coder` и `DeCoder`.

Исполняемая часть

Начинается зарезервированным словом **Implementation** и содержит описания программ, объявленных в интерфейсной части.

Описанию подпрограммы должен предшествовать заголовок, в котором можно опускать список формальных переменных (и тип результата для функции), так как они уже описаны в интерфейсной части. Но если заголовок приводится в полном виде, т. е. со списком формальных параметров и объявлением результата, он должен совпадать с заголовком, объявленным в интерфейсной части. Рассмотрим пример модуля, который можно создать на основе процедур и функций программы `StringCoder` кодирования текстовой строки из предыдущего занятия:

```
Unit MyCoder;
Interface
{интерфейсная часть}

Function Coder(St: String): String;
Function DeCoder(St: String): String;

Implementation
{исполняемая часть}

{процедура перестановки местами двух символов}
Procedure ChangeCh(var St: String; Ch1, Ch2: Byte);
var
  TmpCh: Char;
begin
   TmpCh:=St[Ch1];
   St[Ch1]:=St[Ch2];
   St[Ch2]:=TmpCh;
end;
{функция кодирования строки}
Function Coder;
var
  i, j, N: Integer;
begin
  N:=Length(St);
  for i:=1 to N-1 do
    for j:=1 to (N div (i+1)) do
      ChangeCh(St, j*(i+1)-i, j*(i+1));
  Coder:=St;
end;
{функция декодирования строки}
Function DeCoder;
var
  i, j, N: Integer;
begin
  N:=Length(St);
```

```
  for i:=N-1 downto 1 do
    for j:=(N div (i+1)) downto 1 do
      ChangeCh(St, j*(i+1)-i, j*(i+1));
  DeCoder:=St;
end;

end.
```

Компиляция модулей

Для того чтобы откомпилировать модуль, необходимо в главном меню для опции **Compile/Destination** (компиляция/расположение) установить значение **Disk** и выполнить компиляцию с помощью комбинации клавиш [Alt+F9]. Turbo Pascal распознает в начале текста модуля заголовок **Unit** и автоматически создает файл с расширением **.tpu** вместо **.exe** (как для обычных программ). Выдаваемое при этом сообщение

Can not run a unit...

просто информирует Вас о том, что модуль самостоятельно не выполняется.

Откомпилированный файл с расширением **.tpu** необходимо поместить в каталог **/Units** программы Turbo Pascal или в любой другой каталог, указанный в пункте **Unit directories** в разделе главного меню **Options/Directories**. После этого данный модуль можно вызывать из любой программы, объявлением его имени в разделе **Uses**.

Изменим программу кодирования текстовой строки **StringCoder** так, чтобы функции кодирования вызывались из модуля **MyCoder**.

```
Program StringCoder;
USES MyCoder;
VAR
  Stg: String;
BEGIN
  Write(' > ');           {ввод строки}
  Readln(Stg);            {ввод строки}
  Stg:=Coder(Stg);        {кодирование строки}
  Writeln(Stg);
  Readln;
  Stg:=DeCoder(Stg);      {декодирование строки}
  Writeln(Stg);
  Readln;
END.
```

Как видим, программа значительно уменьшилась в размерах и стала понятней. Это оказывается очень важным как при создании больших проектов, так и при написании скромных модульно оформленных программ.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Подвиньтесь, пожалуйста...»* Добавьте в модуль MyCoder функцию кодирования и функцию декодирования строки методом сдвига кода символа на фиксированное число позиций.

2. *«Маска»* Добавьте в модуль `MyCoder` функцию кодирования и функцию декодирования строки применением к коду символа логической операции **`XOR`** с фиксированной маской.

Занятие 12

Изучается структурированный тип данных – файл: назначение, открытие, закрытие файлов, общие средства работы с файлами. Рассматривается пример работы с текстовыми файлами.

Читал, читал, а все без толку...
А. С. Пушкин, "Е. Онегин"

ФАЙЛЫ

У понятия файл есть две стороны.

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. Файл в таком понимании называют ***логическим файлом,*** то есть существующим только в нашем логическом представлении при написании программы. В программах логические файлы представляются файловыми переменными определенного типа.

Структура физического файла

Структура физического файла представляет собой простую последовательность байт памяти носителя информации — жесткого магнитного диска (ЖМД) или гибкого магнитного диска (ГМД).

байт	байт	байт	...	байт	байт	байт

Структура логического файла

Структура логического файла — это способ восприятия файла в программе. 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. Образное представление некоторых из "шаблонов" языка Turbo Pascal показано на следующих рисунках.

```
File of Byte
```

байт	байт	байт	...	байт	Eof

```
File of Char
```

код символа	код символа	код символа	...	код символа	Eof

```
File of Integer
```

целое со знаком	целое со знаком	целое со знаком	...	целое со знаком	Eof

Логическая структура файла в принципе очень похожа на структуру массива. Различия между массивом и файлом заключаются в следующем.

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 объявлении.

У файла количество элементов в процессе работы программы может изменяться, и 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 не известно. 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 Pascal

Файлы в Turbo Pascal классифицируются по двум признакам:

1. по типу (логической структуре):
   - типизированные файлы;
   - текстовые файлы;
   - нетипизированные файлы;
2. по методу доступа к элементам файла:
   - последовательного доступа;
   - прямого доступа.

Допустимость применения методов доступа к каждой разновидности файлов по типу покажем такой схемой:

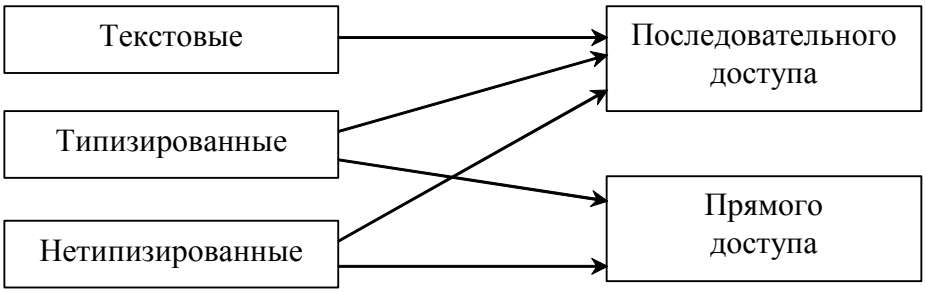


Рис. 16

Назначение, открытие и закрытие файлов

Для работы с каким-либо физическим файлом, находящимся на ЖМД или ГМД, необходимо первоначально связать его с 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 параметром — строковая константа или идентификатор строковой переменной, значением которых должно быть имя физического файла, указанное согласно правилам записи идентификаторов в MS-DOS.

```
Assign(F, 'MyFile.dat');
```

В приведенном примере выполняется связывание логического файла F с физическим файлом `MyFile.dat`, при условии, что он находится в текущем каталоге активного диска MS-DOS. Если же требуется, чтобы действие процедуры **`Assign`** не зависело от текущих устройств MS-DOS, то записывается полное имя файла с указанием диска, пути каталогов и имени файла, например

```
Name:='C:\Temp\MyFile.dat';
Assign(F, Name);
```

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(F);
ReWrite(F);
Close(F);
```

Процедура **`ReSet`** открывает существующий физический файл, который был связан с файловой переменной F. Если F — текстовый файл, то он будет доступен только для чтения при последовательном доступе к элементам, если F — типизованный 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. Ее можно подавить выключением директивы компилятора {$I-}. При такой установке директивы можно проанализировать результат завершения операции открытия файла с помощью функции **`IOResult`**, которая возвращает значение 0, если операция завершилась успешно, и ненулевой код ошибки в противном случае.

Процедура **`ReWrite`** создает новый физический файл, имя которого связано с файловой переменной F. Если такой физический файл уже существует, то он удаляется, и 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(F)
```

Функция **`Eof(F)`** возвращает значение `True`, если указатель текущей позиции в файле F находится за последним элементом файла, или, если файл пуст. В противном случае она возвращает значение `False`.

Некоторые процедуры для работы с файлами

процедура	действие	замечания
ChDir(S: String)	Изменяет текущий каталог	Текущий каталог изменяется на каталог, определенный параметром S. Если в S содержится указатель на дисковод, то текущий диск также изменяется.
MkDir(S: String)	Создает подкаталог	Создает новый подкаталог с именем, определенным строкой S. Последняя часть строки не может быть именем существующего файла.
RmDir(S: String)	Удаляет пустой каталог	Удаляет подкаталог с путем, указанным в S. Если путь не существует, каталог не пуст или это текущий каталог, происходит ошибка ввода/вывода.
GetDir(D: Byte; Var S: String)	Возвращает текущий каталог заданного диска	Параметр D: 0 - Текущий диск 1 - Диск A 2 - Диск B 3 - Диск C И так далее... Не выполняет проверку ошибок. Если диск, заданный параметром D недопустим, то в строке S возвращается X:\, как будто это корневой каталог недопустимого диска.
Rename(var F; NewName)	Переименовывает внешний файл	Параметр F – переменная любого файлового типа. Параметр NewName – типа String или PChar, если включен расширенный синтаксис. Внешний файл, связанный с переменной F переименовывается на NewName. Дальнейшие операции на F происходят уже с внешним файлом с новым именем.
Erase(var F)	Стирает внешний файл с диска	Параметр F – файловая переменная любого файлового типа. Внешний файл, связанный с переменной F удаляется. Никогда не используйте Erase на открытом файле!

Для всех процедур в режиме **{$I-}** функция **IOResult** вернет 0, если операция была успешна, иначе, она возвращает отличный от нуля код ошибки.

Текстовые файлы

В текстовых файлах помимо признака конца файла **Eof** используется еще признак конца строки **Eoln**. Признак **Eoln** представляет собой последовательность из двух символов кода ASCII — символа с кодом 13 ("возврат каретки") и символа с кодом 10 ("перевод строки").

Текстовый файл можно образно представить как страницу книги, в конце каждой строки которой стоит **Eoln**.

код символа	код символа	...	Eoln		
код символа	код символа	...	код символа	код символа	Eoln

код символа	код символа	...	код символа	Eoln	
код символа	код символа	...	Eoln		
код символа	код символа	...	код символа	код символа	Eof

Напомним, что стандартные файлы ввода-вывода Input (ввод с клавиатуры) и Output (вывод на дисплей) являются текстовыми. Использование процедур **Read**, **Readln**, **Write**, **Writeln** при стандартном вводе-выводе нам известно и для текстовых файлов будет практически таким же. Отличие состоит в том, что первым параметром этих процедур должна быть указана файловая переменная.

```
Read (F, A, B);
Write (G, 'A-', A, 'B-', B);
Readln (F, C, D);
Writeln (G, 'C=', C, 'D=', D) ;
```

Процедуры и функции для работы с текстовыми файлами

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

процедура	действие	замечания
Append(Var F: Text)	Открывает существующий файл для продолжения записи в файл	Параметр F – переменная текстового файла, которая должна быть связана с внешним файлом при помощи вызова процедуры Assign. Append открывает существующий внешний файл с именем, определенным в файловой переменной F. Если внешний файл с данным именем не существует, то происходит ошибка ввода/вывода. Если F уже открыт, то он закрывается и вновь открывается. Текущая позиция файла устанавливается на конец файла. Если в последнем 128-байтовом блоке файла присутствует символ Ctrl+Z (символ, с кодом 26), то текущая позиция файла устанавливается, так чтобы перезаписать первый встретившийся Ctrl+Z в блоке. Таким образом, к файлу, который завершается символом Ctrl+Z может быть добавлен текст. Если F было назначено, пустое имя, например Assign(F, ''), то после обращения к Append, F относится к стандартному устройству вывода (номер дескриптора = 1). После обращения к Append, F становится файлом только для чтения, и указатель позиции файла устанавливается на его конец.

Flush(Var F: Text)	Очищает буфер текстового файла открытого на вывод	Параметр F – переменная текстового файла. Если текстовый файл был открыт на вывод с использованием процедур ReWrite или Append, то вызов Flush очистит буфер файла. Это гарантирует, что все символы, записанные в это время в файл, будут насильно записаны на диск. Вызов Flush не имеет никакого эффекта для файлов, открытых на ввод.
Read([Var F: Text;] V1 [, V2,...,Vn])	Для текстовых файлов, считывает одно или большее количество значений в одну или большее количество переменных	Read считывает все символы (но не включая) до следующего маркера конца строки или пока Eof(F) станет равным True. Read не переходит к следующей строке после чтения. Если полученная в результате строка длиннее, чем максимальная длина строковой переменной, то она усекается. После первого Read, каждые последующие вызовы Read будут видеть маркер конца строки и возвращать строку нулевой длины. Используйте несколько обращений к ReadLn, чтобы считать несколько строковых значений.
ReadLn([var F: Text;] V1 [, V2, ...,Vn])	Выполняется процедура Read, затем выполняется переход на следующую строку файла.	После выполнения Read, ReadLn переходит на начало следующей строки файла.
Write([Var F: Text;] P1 [,P2,...,Pn])	Записывает одну или большее количество переменных в файл	Параметр F (если определен) – переменная текстового файла. Каждый параметр P – параметр записи, который включает выражение, значение которого должно быть записано в файл. Параметр записи может также содержать спецификаторы ширины поля и количества знаков после десятичной точки. Каждое выражение вывода должно иметь тип Char, Integer, Real, String, Packed String или Boolean.
WriteLn([Var F: Text;] P1 [, P2, ...,Pn])	Выполняет процедуру Write, затем записывает маркер конца строки в файл	Процедура WriteLn является расширением процедуры Write, поскольку она определена только для текстовых файлов. После вызова процедуры Write, WriteLn записывает маркер конца строки (CR/LF) в файл. Обращение типа WriteLn(F) записывает маркер конца строки в файл F. Вызов WriteLn без параметров соответствует вызову WriteLn(Output). Ограничения: Файл должен быть открыт на запись.

SetTextBuf(Var F: Text; Var Buf [; Size: Word])	Назначает буфер ввода/вывода для текстового файла	Процедуру SetTextBuf нельзя применять к открытому файлу, хотя ее и можно вызывать сразу после выполнения Reset, ReWrite и Append. Если вы вызываете SetTextBuf для открытого файла во время операций ввода/вывода, то это может вызвать потерю данных из-за смены буфера. Borland Pascal не гарантирует, что буфер будет существовать во время всей операций ввода/вывода в файл. Обычная ошибка состоит в том, что используют локальную переменную как буфер, а затем используют файл вне процедуры, в которой был объявлен буфер.

функция	действие	замечания
SeekEof[(Var F: Text)]: Boolean	Возвращает состояние конца файла	Может использоваться только для текстовых файлов. Файл должен быть открытый.
SeekEoln[(Var F: Text)]: Boolean	Возвращает состояние конца строки в файле	Может использоваться только для текстовых файлов. Файл должен быть открыт.

«Алекс – Юстасу»

Рассмотрим пример программы построчного чтения из текстового файла, кодирования (декодирования) с помощью функций модуля MyCoder и запись в новый текстовый файл результата.

```
Program Alex_Ust;
USES MyCoder;
VAR
  Stg: String;
  FileName1, FileName2: String;
  F, G: Text;
  k: Byte;
{==========================}
BEGIN
  Writeln('  Кодирование - 1');
  Writeln('Декодирование - 2');
  Readln(k);
  case k of
    1: Writeln('************* Кодирование *************');
    2: Writeln('************ Декодирование ************');
  end;
  Write('Введите имя входного файла: ');
  Readln(FileName1);
  Write('Введите имя выходного файла: ');
  Readln(FileName2);
  Assign(F, FileName1+'.txt');
  Assign(G, FileName2+'.txt');
  {$I-}
  ReSet(F);
  ReWrite(G);
```

```
    {$I+}
    if IOResult<>0 then
      begin
        Writeln('Ошибка...');
        Writeln('Файла '+ FileName1+'.txt'+' не существует');
        Halt;
      end;
    repeat
      Readln(F, Stg);
      case k of
        1: Stg:=Coder(Stg);
        2: Stg:=DeCoder(Stg);
      end;
      Writeln(G, Stg);
      Writeln(Stg);
    until Eof(F);
    Close(F);
    Close(G);
END.
```

Переменные `FileName1` и `FileName2` являются именами входного и выходного файла без расширения; `F`, `G` — файловые переменные; `k` — селектор операции (1 – кодирование, 2 – декодирование); `Stg` — обрабатываемая строка.

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

3. *«Зеркало»* Дан произвольный текстовый файл. Создать другой текстовый файл, в котором все строчки являются зеркальным отражением соответствующих строк исходного файла.

«...семь, восемь...» Дан произвольный текстовый файл. Посчитать количество строк в этом файле.

«Поплотней, пожалуйста...» Дан произвольный текстовый файл. Создать на его основе другой текстовый файл, в котором удалены все пустые строки.

«Кто больше?» Дан произвольный текстовый файл. Посчитать сколько раз в его тексте встречается каждый символ.

Занятие 13

Изучаются структурированные типы данных: записи и нетипизированный файл. Закрепляется тема работы с файлами на примере создания BMP-файла.

Как рано мог уж он тревожить
Сердца кокеток записных...
А. С. Пушкин, "Е. Онегин"

Записи

Запись — это структура данных, состоящая из фиксированного числа компонентов, называемых полями записи. В отличие от массива, компоненты (поля) записи могут быть различного типа. Чтобы можно было ссылаться на тот или иной компонент записи, поля именуются.

Структура объявления типа записи такова:

```
<Имя типа> = record
               <список полей>
             end;
```

где <Имя типа> – правильный идентификатор;

<список полей> – список полей; представляет собой последовательность разделов записи, между которыми ставится точка с запятой;

record, end – зарезервированные слова.

Каждый раздел записи состоит из одного или нескольких идентификаторов полей, отделяемых друг от друга запятыми. За идентификатором (идентификаторами) ставится двоеточие и описание типа поля (полей), например:

```
type
  TPerson = record
              SurName: String[25];      {фамилия}
              Name: String[15];         {имя}
              Patronymic: String[25];   {отчество}
              BirthYear: Word;          {год рождения}
              BirthMonth: Byte;         {месяц рождения}
              BirthDay: Byte;           {день рождения}
              City: String[25];         {город проживания}
              Street: String[35];       {улица}
              House, Flat: Integer;     {номер дома,
                                               квартиры}
              Phon: LongInt;            {номер телефона}
            end;
const
  N = 25;
var
  Friend: TPerson;
  Person: array[1..N] of TPerson;
```

В этом примере в разделе **type** описан тип TPerson являющийся записью с полями, в которых можно хранить сведения о человеке: фамилия, имя, отчество; год, месяц, день рождения; место проживания; телефон. В разделе **var** описываются переменная Friend и переменная-массив Person типа TPerson. В переменной Friend или каждом элементе массива Person можно хранить данные о конкретном человеке, например, ученике класса.

Значения переменных типа записи можно присваивать другим переменным того же типа:

```
Person[10]:= Friend;
```

К каждому из компонентов записи можно получить доступ, если использовать составное имя:

```
<Имя переменной>.<Имя поля>
```

Например, чтобы заполнить поле, в котором хранится имя первого человека в массиве Person или номер телефона в переменной Friend, нужно обратиться к соответствующему полю следующим образом:

```
Person[1].Name:='Александр';
Friend.Phon:=231201;
```

Чтобы упростить доступ к полям записи, используется оператор присоединения **WITH**. Структура оператора присоединения:

```
with <переменная> do <оператор>;
```

Здесь **with, do** – ключевые слова (с, делать);
<переменная> – имя переменной типа запись, за которым, возможно, следует список вложенных полей;
<оператор> – любой оператор Turbo Pascal.

Например:

```
with Person[i] do
  begin
    SurName:='Ильин';
    Name:='Александр';
    Patronymic:='Константинович';
    BirthYear:=1970;
    BirthMonth:=6;
    BirthDay: =7;
    City: ='Красноярск';
    Street:='Марковского';
    House:=49;
    Flat:=12;
    Phon:=231201;
  end;
```

Здесь, для доступа к полю записи достаточно обратиться к нему по имени, так как имя соответствующей переменной `Person[i]` указано в конструкции **`WITH`**.

Закрепим работу с записями на примере создания графического BMP файла. Здесь необходимо использовать нетипизированную файловую переменную, так как информация в разных частях файла имеет различную структуру и, соответственно, различный тип.

Для начала разберемся с нетипизированными файлами.

НЕТИПИЗИРОВАННЫЕ ФАЙЛЫ

Нетипизированные файлы объявляются, как файловые переменные типа **`FILE`**, и отличаются тем, что для них не указан тип компонентов. Отсутствие типа делает эти файлы, с одной стороны, совместимыми с любыми другими файлами, а с другой — позволяет организовать высокоскоростной обмен данными между диском и памятью.

При инициации нетипизированного файла процедурами **`RESET`** или **`REWRITE`** можно указать длину записи нетипизированного файла в байтах. Например, так:

```
var
  F: File;
begin
...........
Assign(F, 'MyFile.dat');
ReSet(F, 512);
...........
end.
```

Длина записи нетипизированного файла указывается вторым параметром при обращении к процедурам **`RESET`** или **`REWRITE`**, в качестве которого может использоваться выражение типа `WORD`, Если длина записи не указана, она принимается равной 128 байтам.

Turbo Pascal не накладывает каких-либо ограничений на длину записи нетипизированного файла, за исключением требования положительности и ограничения максимальной длины `65535` байтами (емкость целого типа `WORD`). Однако для обеспечения максимальной скорости обмена данными следует задавать длину, которая была бы кратна длине физического сектора дискового носителя информации (512 байт). Более того, фактически пространство на диске выделяется любому файлу порциями — кластерами, которые в зависимости от типа диска могут занимать 2 и более смежных секторов. Как правило, кластер может быть прочитан или записан за один оборот диска, поэтому наивысшую скорость обмена данными можно получить, если указать длину записи, равную длине кластера.

При работе с нетипизированными файлами могут применяться все процедуры и функции, доступные типизированным файлам, за исключением **`READ`** и **`WRITE`**, которые заменяются соответственно высокоскоростными

процедурами **BLOCKREAD** и **BLOCKWRITE**. Для вызова этих процедур используются следующие предложения:

```
BlockRead(F, Buf, Count, Result);
BlockWrite(F, Buf, Count, Result);
```

Здесь `F` – нетипизированная файловая переменная;

`Buf` — буфер: имя любой переменной, которая будет участвовать в обмене данными с дисками;

`Count` — количество записей, которые должны быть прочитаны или записаны за одно обращение к диску;

`Result` — необязательный параметр, содержащий при выходе из процедуры количество фактически обработанных записей.

Если при чтении указана переменная `Buf` недостаточной длины или если в процессе записи на диск не окажется нужного свободного пространства, возникнет ошибка ввода—вывода, которую можно заблокировать, указав необязательный параметр `Result` (переменная типа **WORD**).

После завершения процедуры указатель смещается на `Result` записей. Процедурами **SEEK**, **FILEPOS** и **FILESIZE** можно обеспечить доступ к любой записи нетипизированного файла.

процедура	действие	замечания
Seek(Var F; N: Longint)	Перемещает текущий указатель позиции файла на определенный компонент	`F` – переменная любого файлового типа за исключением текстового, и `N` – выражение типа `Longint`. Указатель позиции файла F перемещается на номер компонента `N`. Номер первого компонента файла равен нулю. Чтобы расширить файл, вы можете передвинуть указатель на один компонент за последний компонент в файле. То есть, оператор `Seek(F, FileSize(F))` перемещает текущий указатель позиции файла на конец файла.
FilePos(Var F): Longint	Возвращает текущую позицию указателя файла	`F` – переменная любого файлового типа за исключением текстового. Если указатель текущей позиции файла находится в начале файла, то `FilePos(F)` возвращает нуль. Если указатель текущей позиции файла находится в конце файла, то есть, если `Eof(F)=True`, то значение `FilePos(F)` равно значению `FileSize(F)`.
FileSize(Var F): Longint	Возвращает текущий размер файла	`F` – переменная любого файлового типа за исключением текстового. `FileSize(F)` возвращает число компонентов в `F`. Если файл пустой, то `FileSize(F)` возвращает нуль.

В режиме {$I–} функция IOResult вернет нуль, если операция была успешна, иначе она вернет отличный от нуля код ошибки.

Размер памяти, занимаемый любой переменной, можно определить с помощью функции **SIZEOF**.

функция	действие	замечания

SizeOf(X): Integer;	Возвращает число байт, занимаемых параметром X.	Когда применяется к объектному типу, который имеет виртуальную таблицу методов (VMT), SizeOf возвращает размер, сохраненный в VMT.

СОЗДАНИЕ BMP ФАЙЛА

Структуру BMP файла можно разделить на три части:

- заголовок файла, который идентифицирует его как растровый графический и хранит другую информацию о содержании файла;
- информация об изображении: содержит такие характеристики растрового изображения, как ширина, высота, разрешение, сжатие данных, палитра и так далее;
- массив данных изображения: содержит пикселы, которые различаются по формату в зависимости от типа растрового изображения.

Исходя из структуры BMP файла, можно создать цельные типы данных на основе записей для каждой из его частей:

- tagBitMapFileHeader — заголовок файла;
- tagBitMapInfoHeader — информация об изображении;
- tagRGBTriple — формат пиксела изображения в массиве данных.

Запись заголовка файла состоит из пяти полей:

1. bfType — идентификатор BMP файла (тип Word); должен содержать два символа 'B' и 'M', что означает BitMap;
2. bfSize — размер файла в байтах (тип LongInt);
3. bfReserved1 — не документировано и не используется (тип Word); должно быть равно нулю;
4. bfReserved2 — не документировано и не используется (тип Word); должно быть равно нулю;
5. bfOffBits — специфицирует байтовое смещение до начала растрового изображения (тип LongInt);

Запись информации об изображении содержит одиннадцать полей:

1. biSize — специфицирует собственный размер структуры в байтах (тип LongInt);
2. biWidth — содержит ширину изображения в пикселах (тип Longint);
3. biHeight — содержит высоту изображения в пикселах (тип Longint);
4. biPlanes — должен быть равен единице, т. к. BMP-файлы, какого бы типа они ни были, хранятся в независимом от устройства формате с одной цветовой плоскостью (тип Word);
5. biBitCount — содержит число битов на пиксел (тип Word);
6. biCompression — показывает, хранится ли данное растровое изображение в сжатом виде, а также метод его упаковки (тип LongInt);

7. biSizeImage — содержит размер растрового изображения в байтах. Может быть нулевым, если изображение не сжато (тип LongInt);
8. biXPelsPerMeter — указывает предпочтительное разрешение по горизонтали в пикселах на метр (тип Longint);
9. biYPelsPerMeter — указывает предпочтительное разрешение по вертикали в пикселах на метр (тип Longint);
10. biClrUsed — обычно содержит число цветов, используемое в растровом изображении (тип LongInt);
11. biClrImportant — содержит число важных цветов изображения (тип LongInt).

Запись формата пиксела изображения в массиве данных содержит информацию о цвете данного пиксела. Цветовое значение представляет собой трехбайтовую композицию интенсивностей красного, зеленого и синего цветов:

rgbtBlue — содержит относительную интенсивность синего цвета от 0 до 255 (тип Byte);
rgbtGreen — содержит относительную интенсивность зеленого цвета от 0 до 255 (тип Byte);
rgbtRed — содержит относительную интенсивность красного цвета от 0 до 255 (тип Byte);

Опишем эти структуры в разделе описания типов. Введем константы Width – ширина изображения в пикселах, Height – высота изображения в пикселах. Опишем переменные BitMapFileHeader и BitMapInfoHeader как соответствующие типы tagBitMapFileHeader и BitMapInfoHeader, а RGBTriple – как массив типа tagRGBTriple (строка пикселов). Будем заполнять (и записывать в файл) данные построчно. Назовем нетипизированную файловую переменную именем FBitMap, а переменную, содержащую название файла – FileName.

Опишем процедуру установки нужного цвета SetColor. В качестве параметров процедуры выступают компоненты основных цветов, входящих в данный цвет. Процедура установки цвета заполняет соответствующие поля записи tagRGBTriple.

В основной программе связываем файловую переменную FBitMap с файлом FileName с помощью процедуры Assign. Создаем этот файл на жестком диске процедурой ReWrite, и начинаем заполнять его (BlockWrite): сначала заголовок файла, затем – заголовок изображения, затем – построчная запись в файл данных изображения[5]. Результатом работы программы является BMP-файл, содержащий картинку (Рис. 17).

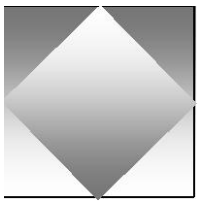


Рис. 17

Полный текст данной программы приведен ниже.

```
Program BMP_Format;
TYPE
  {заголовок файла}
  tagBitMapFileHeader = record
```

[5] В файле BMP-формата изображение хранится в перевернутом виде: сначала записывается последняя строка, затем – предпоследняя, и т. д. до первой строки.

```
    bfType: Word;      {'BM'}
    bfSize: LongInt;   {размер файла (байты)}
    bfReserved1: Word; {0}
    bfReserved2: Word; {0}
    bfOffBits: LongInt;{смещение в байт. до начала изобр.}
  end;
  {заголовок изображения}
  tagBitMapInfoHeader = record
    biSize: LongInt; {размер структуры tagBitMapInfoHeader}
    biWidth: Longint;
    biHeight: Longint;
    biPlanes: Word;   {число цветовых плоскостей 1}
    biBitCount: Word;         {бит на пиксел 24}
    biCompression: LongInt;   {0}
    biSizeImage: LongInt;     {0 (только для сжатых)}
    biXPelsPerMeter: Longint;{96}
    biYPelsPerMeter: Longint;{96}
    biClrUsed: LongInt;       {0 только для палитры}
    biClrImportant: LongInt; {0 цветов точно переданных}
  end;
  {формат пиксела}
  tagRGBTriple = record
    rgbtBlue: Byte;
    rgbtGreen: Byte;
    rgbtRed: Byte;
  end;
CONST
  Width  = 400;
  Height = 400;
VAR
  FBitMap: File;
  FileName: String;
  BitMapFileHeader: tagBitMapFileHeader;
  BitMapInfoHeader: tagBitMapInfoHeader;
  RGBTriple: array[1..Width] of tagRGBTriple;
  i, j: Integer;
{процедура установки цвета}
Procedure SetColor(clRed, clGreen, clBlue: Byte;
                                        k: Integer);
begin
  RGBTriple[k].rgbtBlue:=clRed;
  RGBTriple[k].rgbtGreen:=clGreen;
  RGBTriple[k].rgbtRed:=clBlue;
end;

BEGIN
  Write('Введите имя BMP файла: ');
  Readln(FileName);
  FileName:=FileNAme+'.bmp';
  Assign(FBitMap, FileName);
  {$I-}
  ReWrite(FBitMap, 1);
  {$I+}
```

```
if IOResult<>0 then
  begin
    Writeln('Ошибка создания файла '+ FileName);
    Halt;
  end;
with BitMapFileHeader do
begin
  bfType:=$4D42;
  bfOffBits:=SizeOf(tagBitMapFileHeader) +
      SizeOf(tagBitMapInfoHeader);
  bfSize:=bfOffBits + SizeOf(tagRGBTriple)*Height;
  bfReserved1:=0;
  bfReserved2:=0;
end;

{запись в файл заголовка файла}
BlockWrite(FBitMap, BitMapFileHeader,
      SizeOf(BitMapFileHeader));

with BitMapInfoHeader do
  begin
    biSize:=SizeOf(BitMapInfoHeader);
    biWidth:=Width;
    biHeight:=Height;
    biPlanes:=1;
    biBitCount:=24;
    biCompression:=0;
    biSizeImage:=0;
    biXPelsPerMeter:=96;
    biYPelsPerMeter:=96;
    biClrUsed:=0;
    biClrImportant:=0;
  end;

{запись в файл заголовка изображения}
BlockWrite(FBitMap, BitMapInfoHeader,
            SizeOf(BitMapInfoHeader));
{заполнение строки}
for i:=1 to Height do
  begin
    for j:=1 to Width do

      if i < Height div 2 then
        if (j>(Width div 2)-i) and (j<(Width div 2)+i)
          then
            SetColor(25+Round(230/Height*i), 0 , 0, j)
          else
            SetColor(255-Round(230/Height*i), 0 , 0, j)
      else
        if  (j>(Width div 2 - Height+i)) and
            (j<(Width div 2 + Height-i))
          then
            SetColor(25+Round(230/Height*i), 0, 0, j)
```

```
            else
              SetColor(255-Round(230/Height*i), 0 , 0, j);
        {запись строки в файл}
        BlockWrite(FBitMap, RGBTriple, SizeOf(RGBTriple));
    end;
  Close(FBitMap);
  Writeln('>>> Файл '+FileName+' успешно создан...');
END.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. *«Андреевский флаг»* Создать BMP-файл: на белом фоне по диагонали прямоугольника крест на крест проведены две синие полосы. Размер изображения 200 на 150 пикселов.
2. *«Сижу за решеткой...»* Создать BMP-файл: синяя решетка на голубом фоне. Размер изображения 200 на 200 пикселов.
3. *«Сам по себе...»* Создать BMP-файл: белая кошка на синем фоне. Размер изображения 180 на 140 пикселов (Рис. 18). Усложнение: сделать градиентную заливку фона (от голубого цвета сверху до темно-синего цвета внизу).

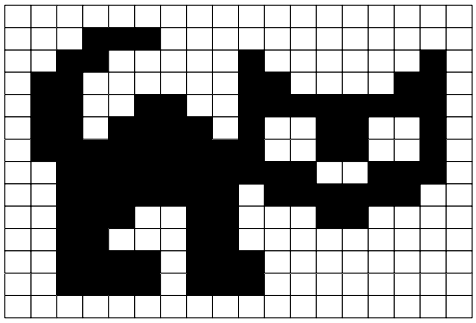


Рис. 18

Занятие 14

Интегрированная среда разработки. Палитра компонентов. Инспектор объектов. Написание первой программы на Delphi.

ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ

После запуска Delphi 6 Вы увидите на экране несколько окон, относящихся к его среде разработки программ (**рис. 19**).

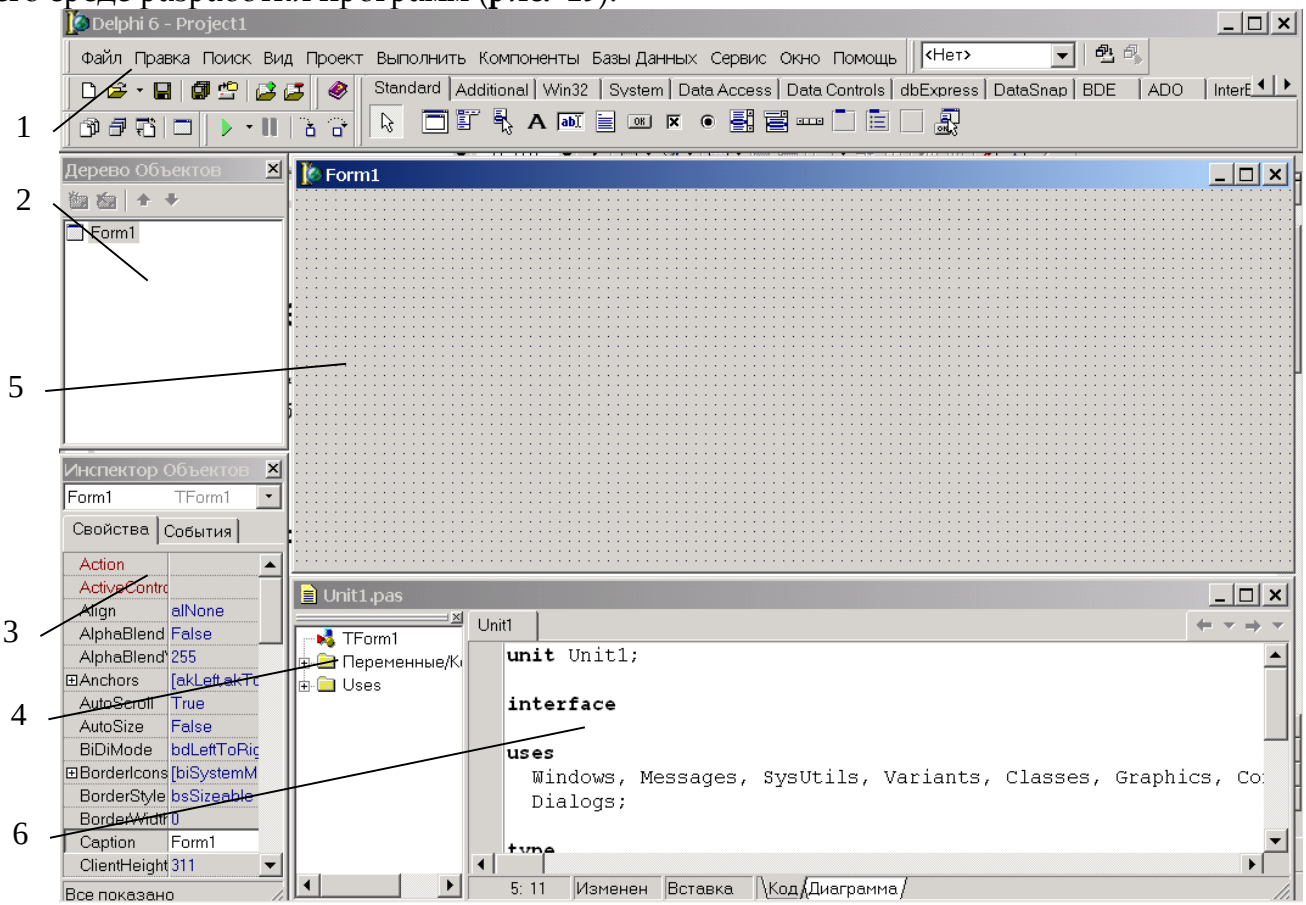


рис. 19

Наиболее важные окна Delphi: 1 – главное окно; 2 – окно Дерева объектов; 3 – окно Инспектора объектов; 4 – окно браузера; 5 – окно формы; 6 – окно кода программы.

Главное окно

Осуществляет основные функции управления проектом создаваемой программы. Это окно всегда присутствует на экране и занимает его верхнюю часть.

В главном окне располагается: 1 – главное меню Delphi, 2 – набор пиктографических командных кнопок и 3 – палитра компонентов (**Ошибка! Источник ссылки не найден.**).

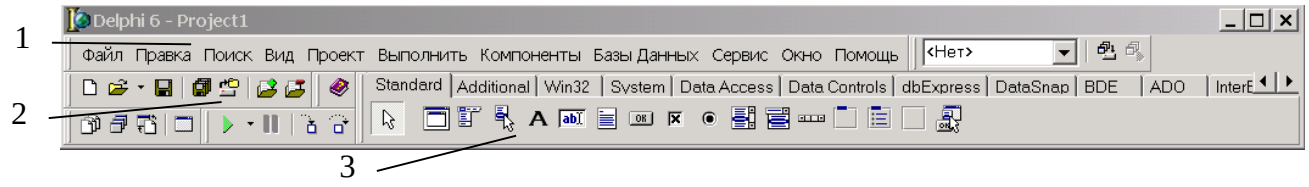


рис. 20

Главное меню содержит все необходимые средства для управления проектом. Все опции главного меню представляют собой опции-заголовки, открывающие доступ к выпадающим меню второго уровня.

Пиктографические кнопки открывают быстрый доступ к наиболее важным опциям главного меню.

Палитра компонентов занимает правую часть окна и имеет закладки, обеспечивающие быстрый поиск нужного компонента.

Компонент — функциональный элемент, содержащий определенные свойства и размещенный программистом в окне формы.

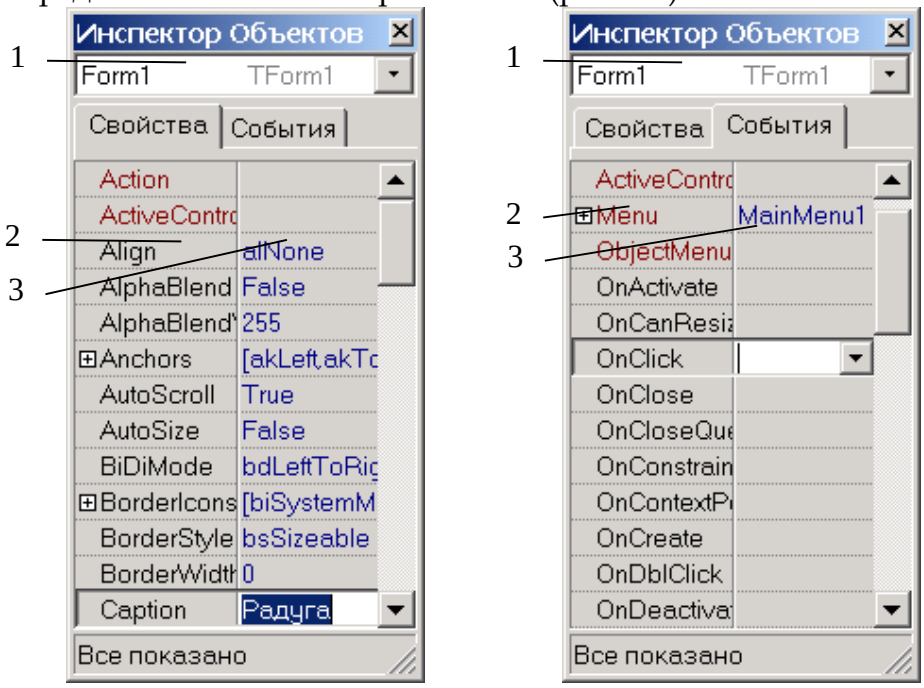
С помощью компонентов создается каркас программы, во всяком случае – ее видимые на экране внешние проявления: окна, кнопки, списки выбора и т. д.

Дерево объектов

Предназначено для наглядного отображения связей между отдельными компонентами, размещенными на активной форме или в активном модуле данных.

Инспектор объектов

Любой размещенный на форме компонент характеризуется некоторым набором параметров: положением, размером, цветом и т. д. Для изменения этих параметров предназначен инспектор объектов (рис. 21).



a). **b).**

рис. 21

В верхней части окна располагается Список компонентов формы (1).

Это окно содержит две страницы — Свойства (Properties) и События (Events). Страница Свойства служат для установки нужных свойств компонента, страница События позволяет определить реакцию компонента на то или иное событие.

Каждая страница Инспектора объектов представляет собой таблицу из двух колонок, левая колонка которой содержит название свойства или события (2), а правая — конкретное значение свойства или имя подпрограммы, обрабатывающей соответствующее событие (3).

Окно формы

Представляет собой проект Windows-окна будущей программы. Вначале оно пусто. Программист, как из деталей конструктора, собирает требуемый вид будущего окна из компонентов, находящихся в палитре компонентов главного окна.

Окно кода программы

Предназначено для создания и редактирования текста программы. Текст программы в системе Delphi пишется на языке программирования Object Pascal.

Первоначально окно кода содержит минимальный исходный текст, обеспечивающий нормальное функционирование пустой формы в качестве Windows-окна. В ходе работы над проектом программист вносит в него необходимые дополнения, чтобы придать программе нужную функциональность.

Сразу после открытия нового проекта в нем будут такие строки:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Основной код программы вставляется между {$R *.dfm} и **end.**

ИЗМЕНЯЕМ ПРИВЫЧНОЕ

Создадим простой пример: Windows-окно которое будет изменять свой цвет в зависимости от положения на нем курсора мышки.

Windows-окно будет иметь строку главного меню с тремя пунктами: Файл, Размер и О программе. В пункте меню Файл будет один подпункт Выход; в пункте Размер – два подпункта: Больше и Меньше; при выборе пункта меню О программе будет появляться окно сообщения.

Для того чтобы изменить название в заголовке окна нужно для формы в Инспекторе объектов изменить свойство Caption. По умолчанию название имеет значение Form1. Заменим это значение на слово Радуга.

Для создания главного меню поместим на форму компонент MainMenu из раздела Standard палитры компонентов. По умолчанию он будет иметь название MainMenu1. Откроем Дизайнер меню, дважды щелкнув на компоненте MainMenu1 мышкой. В Дизайнере меню создаем три пункта главного меню, задавая их названия в свойствах Caption. В первом создаем подпункт Выход, а во втором два подпункта: Больше и Меньше. После этого Дизайнер меню можно закрыть. На форме появилось главное меню.

Каждому объекту, находящемуся на форме, соответствует определенный набор событий. Все эти события указаны в Инспекторе объектов на странице События. Каждое событие можно обработать, назначив ему соответствующую процедуру обработки. Имя этой процедуры обработки указывается во втором столбце страницы События. Для того чтобы шаблон процедуры был автоматически вставлен в текст программы нужно дважды щелкнуть мышкой в этом месте или на соответствующем компоненте формы. Так, если щелкнуть на пункте меню Выход, то в тексте программы появится шаблон процедуры обработки события выбора данного пункта меню.

```
procedure TForm1.N2Click(Sender: TObject);
begin

end;
```

Это только шаблон процедуры. Наполнить ее содержанием – задача программиста. В данном случае для завершения программы достаточно вызвать только одну процедуру закрытия формы:

```
Close;
```

При нажатии на пункте главного меню О программе должно появиться окно сообщения, которое можно вызвать следующим образом:

```
Application.MessageBox('Это моя первая программа на
                        Delphi...', 'О программе', 0);
```

В процедурах изменения размеров формы нужно при каждом нажатии изменять ширину и высоту формы на определенную величину (например, 5 пикселов) пока они не достигнут максимального или минимального значения.

Процедура обработки события перемещения мышки над формой имеет вид:

```
procedure TForm1.FormMouseMove(Sender: TObject; Shift:
TShiftState; X, Y: Integer);
var
  R, G, B: Byte;
begin
  R:=Round(X*255/ClientWidth);
  G:=Round(Y*255/ClientHeight);
  B:=255;
  Form1.Color:=RGB(R, G, B);
end;
```

Окончательный вид программы:

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms,
  Dialogs, StdCtrls, Menus;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    procedure FormMouseMove(Sender: TObject;
              Shift: TShiftState; X, Y: Integer);
    procedure N3Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;


implementation
```

```
{$R *.dfm}

procedure TForm1.FormMouseMove(Sender: TObject; Shift:
TShiftState; X,
  Y: Integer);
var
  R, G, B: Byte;
begin
   R:=Round(X*255/ClientWidth);
   G:=Round(Y*255/ClientHeight);
   B:=255;
   Form1.Color:=RGB(R, G, B);
end;

procedure TForm1.N3Click(Sender: TObject);
begin
  Application.MessageBox(('Это моя первая программа на
                        Delphi...', 'О программе', 0);
end;

procedure TForm1.N2Click(Sender: TObject);
begin
  Close;
end;

procedure TForm1.N5Click(Sender: TObject);
begin
  Width:=Width+10;
  Height:=Height+10;
end;

procedure TForm1.N6Click(Sender: TObject);
begin
  Width:=Width-10;
  Height:=Height-10;
end;

end.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Добавить кнопку выхода на форму.

Занятие 15

Игра «Угадай число».

ИГРА «УГАДАЙ ЧИСЛО»

Вид формы:

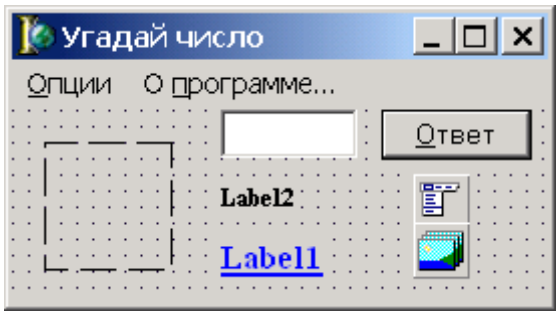


Рис. 22

Полный текст программы:

```
unit Ugaday;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Menus, ImgList, ExtCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    MainMenu1: TMainMenu;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N4: TMenuItem;
    Image1: TImage;
    ImageList1: TImageList;
    N5: TMenuItem;
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure N4Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
```

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

var
  Form1: TForm1;
  X, Y: Integer;
  Attempts: Integer;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
begin
  Randomize;
  Attempts:=0;
  X:=Random(100)+1;
  Label1.Caption:='';
  Label2.Caption:='';
  ImageList1.GetBitmap(0, Image1.Picture.Bitmap);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   try
      Edit1.SetFocus;
      Edit1.SelectAll;
      Y:=StrTont(Edit1.Text);
      Inc(Attempts);
      Label2.Caption:='Попытка № '+IntToStr(Attempts);
      if Y<X then
        begin
          Label1.Caption:='Больше';
          Image1.Picture.Bitmap.Assign(nil);
          ImageList1.GetBitmap(1, Image1.Picture.Bitmap);
        end
      else
        if Y>X then
          begin
            Label1.Caption:='Меньше';
            Image1.Picture.Bitmap.Assign(nil);
            ImageList1.GetBitmap(2, Image1.Picture.Bitmap);
          end
        else
          begin
            Label1.Caption:='ПОЗДРАВЛЯЮ!';
            Edit1.Enabled:=False;
            Image1.Hint:='Новая игра?';
            Image1.Picture.Bitmap.Assign(nil);
            ImageList1.GetBitmap(3, Image1.Picture.Bitmap);
```

```
            end;
      except
      end;

  end;

  procedure TForm1.N4Click(Sender: TObject);
  begin
    Close;
  end;

  procedure TForm1.N2Click(Sender: TObject);
  begin
    Attempts:=0;
    X:=Random(100)+1;
    Edit1.Enabled:=True;
    Edit1.Text:='';
    Label1.Caption:='';
    Label2.Caption:='';
    Edit1.SetFocus;
    Image1.Hint:='Я загадал число от 1 до 100';
    Image1.Picture.Bitmap.Assign(nil);
    ImageList1.GetBitmap(0, Image1.Picture.Bitmap);
  end;

  procedure TForm1.N5Click(Sender: TObject);
  begin
     Application.MessageBox(
        'Игра "Угадай число".' + #10#13 +
        'Компьютер загадывает число от 1 до 100.' + #10#13 +
        'Игрок должен отгадать это число.',
        'О программе...', 0);
  end;

  end.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Занятие 16

Создается программа просмотра bmp-файлов. Изучаются функции поиска файлов на диске FindFirst и FindNext, класс TStringList, компоненты TImage и TButton.

Поиск файла на диске

Для поиска файла на диске существуют процедуры и функции модуля SysUnits:

```
function FindFirst(const Path: string; Attr: Integer;
                                  var F: TSearchRec):Integer;
function FindNext(var F: TSearchRec):Integer;
procedure FindClose(var F: TSearchRec);
```

Функция FindFirst находит файл с набором атрибутов Attr в каталоге и по маске, определенных константой Path. Найденное имя записывается в переменную F. Если указанный файл найден, то функция возвращает 0, иначе возвращается код ошибки Windows. Константа Path представляет собой полный путь с маской файла (например, 'C:\WINDOWS*.ini'). Повторный поиск файла производится с помощью функции FindNext. По окончанию поиска необходимо высвободить память, выделенную при вызове функции FindFirst, с помощью процедуры FindClose.

Параметр Attr при обращении к FindFirst содержит двоичные разряды (биты), уточняющие, к каким именно файлам разрешен доступ. Вот как объявляются файловые атрибуты в модуле SysUnits:

```
const
  faReadOnly  = $00000001 //только чтение
  faHidden    = $00000002 //скрытый файл
  faSysFile   = $00000004 //системный файл
  faVolumeID  = $00000008 //идентификатор тома
  faDirectory = $00000010 //имя подкаталога
  faArchive   = $00000020 //архивный файл
  faAnyFile   = $0000003F //любой файл
```

Комбинацией бит в этом байте можно указывать самые разные варианты, например $00000006 – выбрать все скрытые и/или системные файлы.

Результат работы процедуры FindFirst возвращается в переменной типа TSearchRec. Этот тип определяется следующим образом:

```
type
  TSearchRec = record
    Time: Integer;              //время и дата создания
    Size: Integer;              //длина в байтах
    Attr: Integer;              //атрибуты файла (см. выше)
    Name: TFileName;            //имя и расширение
    ExcludeAttr: Integer;       //
```

```
    FindHandle: THandle;        //
    FindDate: TWin32FindDate;   //дополнительная информация
  end;
```

КЛАСС: СПИСОК СТРОК TSTRINGLIST

Является полнофункциональным классом общего назначения. Создается объект типа `TStringList` встроенным методом `Create`:

```
slFileName:=TStringList.Create;
```

После того, как объект `slFileName` оказывается ненужным или при завершении программы, необходимо его уничтожить методом `Free`, освободив выделенную под него оперативную память:

```
slFileName.Free;
```

Ниже приводятся некоторые свойства и методы класса `TStringList`.

Свойства класса:

Свойство	Описание
`Sorted: Boolean;`	Указывает, проводить автоматическую сортировку или нет.
`Strings[Index: Integer]: String;`	Содержит строку с индексом `Index`.

МЕТОДЫ КЛАССА:

Метод	Описание
`function Add(const S: String): Integer;`	Помещает новую запись в конец набора в том случае, если список строк не сортирован. В сортированном списке помещает ее на соответствующую позицию. Возвращает ее индекс.
`procedure Clear;`	Удаляет все строки из набора и все ссылки на связанные с ними объекты.
`procedure Delete(Index: Integer);`	Удаляет строку с номером `Index` и ссылку на связанный с ней объект.
`function Find(const S: String; var Index: Integer): Boolean;`	Ищет строку в отсортированном списке. В случае удачного поиска возвращает `True`, а в переменной `Index` – индекс строки. Используется только для сортированных наборов.
`procedure Sort;`	Производит сортировку по возрастанию для несортированного набора. Сортированный набор (`Sorted = True`) сортируется автоматически.

Отображение картинок

Для этих целей можно использовать компонент `TImage` со страницы `Additional` палитры компонентов. Это компонент служит для размещения на форме одного из трех поддерживаемых Delphi типов изображений: растровой картинки, пиктограммы или метафайла. Любой из этих типов изображения содержится в центральном свойстве компонента – `Picture`.

Создание галереи просмотра картинок

На форму поместите компонент `TImage` и две кнопки `TButton` (рис. 23).

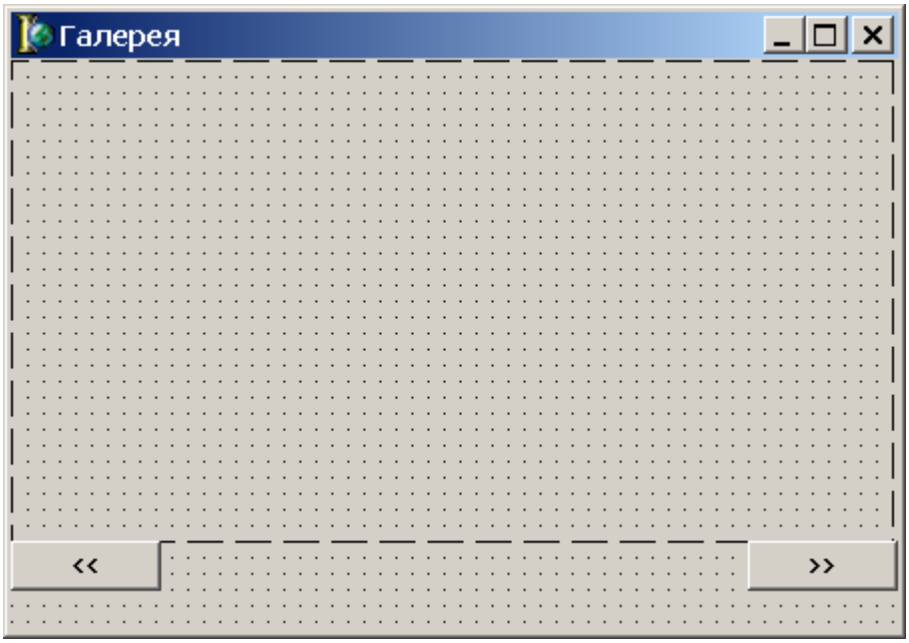


рис. 23

У формы установите следующие свойства:

Caption	Галерея
AutoSize	True

У кнопок `TButton` установите свойства (для левой и правой соответственно):

Caption	<< или >>
Name	bnLeft или bnRight

Создание процедуры открытия картинки

Создадим свою процедуру открытия BMP-файла. В ней по номеру картинки `N` в списке строк `TStringList` находим имя файла и открываем его.

```
procedure TForm1.OpenBMP(N: Integer);
```

```
begin
  try
    Image1.Left:=0;
    Image1.Top:=0;
    Image1.Picture.LoadFromFile(SL.Strings[N]);
    if Image1.Picture.Width > 2*bnLeft.Width then
      Image1.Width:=Image1.Picture.Width
    else
      Image1.Width:=2*bnLeft.Width;
    Image1.Height:=Image1.Picture.Height;
    bnLeft.Top:=Image1.Height;
    bnRight.Top:=bnLeft.Top;
    bnLeft.Left:=0;
    bnRight.Left:=Image1.Width-bnRight.Width;
  except
    Application.MessageBox(PChar('Файла '+SL.Strings[N]+
                                 ' не существует...'),'Ошибка',0);
  end;
end;
```

ОБРАБОТКА СОБЫТИЙ

Создаем процедуры обработки событий создания формы `Form1` и нажатия на кнопки `bnLeft` и `bnRight`.

В процедуре `FormCreate` осуществляем поиск всех `*.bmp` файлов в текущем каталоге, запоминаем их имена в списке строк `SL` и открываем первую картинку для просмотра.

В процедуре `FormDestroy`, выполняемой при уничтожении формы, освобождается выделенная под переменную `SL` память.

Процедуры `bnLeftClick` и `bnRightClick` осуществляют последовательный показ найденных в текущем каталоге `*.bmp` файлов.

ПОЛНЫЙ ТЕКСТ ПРОГРАММЫ

```
unit Unit1;
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Image1: TImage;
    bnLeft: TButton;
    bnRight: TButton;
    procedure OpenBMP(N: Integer);
    procedure FormCreate(Sender: TObject);
    procedure bnLeftClick(Sender: TObject);
    procedure bnRightClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
```

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

var
  Form1: TForm1;
  N: Integer;
  SL: TStringList;
  F: TSearchRec;

implementation

{$R *.dfm}

procedure TForm1.OpenBMP(N: Integer);
begin
  try
    Image1.Left:=0;
    Image1.Top:=0;
    Image1.Picture.LoadFromFile(SL.Strings[N]);
    if Image1.Picture.Width > 2*bnLeft.Width then
      Image1.Width:=Image1.Picture.Width
    else
      Image1.Width:=2*bnLeft.Width;
    Image1.Height:=Image1.Picture.Height;
    bnLeft.Top:=Image1.Height;
    bnRight.Top:=bnLeft.Top;
    bnLeft.Left:=0;
    bnRight.Left:=Image1.Width-bnRight.Width;
  except
    Application.MessageBox(PChar('Файла '+SL.Strings[N]+
                           ' не существует...'),'Ошибка',0);
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  N:=0;
  SL:=TStringList.Create;
  if FindFirst('*.bmp', faAnyFile, F)=0 then
    begin
      SL.Add(F.Name);
      while FindNext(F)=0 do SL.Add(F.Name);
      FindClose(F);
    end
  else
    begin
      FindClose(F);
      Close;
    end;
  OpenBMP(N);
```

```
end;

procedure TForm1.FormDestroy(Sender: TObject);
begin
  SL.Free;
end;

procedure TForm1.bnLeftClick(Sender: TObject);
begin
  if N = 0 then N:=SL.Count-1 else Dec(N);
  OpenBMP(N);
end;

procedure TForm1.bnRightClick(Sender: TObject);
begin
  if N = SL.Count-1 then N:=0 else Inc(N);
  OpenBMP(N);
end;

end.
```

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Написать программу поиска в заданном каталоге всех текстовых файлов.

Занятие 17

Изучается рисование на канве.

ЛЕТАЮЩАЯ ТАРЕЛКА

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, Menus, ComCtrls, StdCtrls, Spin, ImgList, MMSystem;

type
  TForm1 = class(TForm)
    MainMenu: TMainMenu;
    nGame: TMenuItem;
    nNewGame: TMenuItem;
    N2: TMenuItem;
    nExit: TMenuItem;
    Timer: TTimer;
    StatusBar: TStatusBar;
    procedure nExitClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FormPaint(Sender: TObject);
    procedure DrawImage;
    procedure TimerTimer(Sender: TObject);
    procedure FormKeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure FormKeyUp(Sender: TObject; var Key: Word;
      Shift: TShiftState);
    procedure nNewGameClick(Sender: TObject);
    procedure NewGame;
  private
    { Private declarations }
    BkGraund1, BkGraund2: TBitmap; //фон
    ImageFly: TBitmap;  //тарелка
    Fly: array[0..6] of TBitmap;
    Bounds, OldBounds: TRect;
  public
    { Public declarations }
  end;

const
  BkFileName = 'Bk.bmp';
  gx = 0;
  gy = 200;
  m = 1;

var
  Form1: TForm1;
  x, y: Extended;
  vx, vy: Extended;
  ax, ay: Extended;
  dt: Extended;
  Start, CanStart: Boolean;
```

```
  SetDown: Boolean;

implementation

{$R *.DFM}

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

procedure TForm1.NewGame;
begin
  Bounds.Left:=Random(ClientWidth-ImageFly.Width);
  Bounds.Top:=Random(100);
  Bounds.Right:=Bounds.Left+ImageFly.Width;
  Bounds.Bottom:=Bounds.Top+ImageFly.Height;
  vx:=0;
  vy:=0;
  dt:=Timer.Interval/1000;
  x:=Bounds.Left;
  y:=Bounds.Top;
  ax:=gx;
  ay:=gy;
  StatusBar.Panels[5].Text:='';
  ImageFly.Assign(Fly[0]);
  Start:=False;
  CanStart:=True;
  SetDown:=False;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  Randomize;
  BkGraund1:=TBitmap.Create;
  BkGraund2:=TBitmap.Create;
  ImageFly:=TBitmap.Create;
  for i:=0 to 6 do
    begin
      Fly[i]:=TBitmap.Create;
      Fly[i].LoadFromFile('Fly'+IntToStr(i)+'.bmp');
      Fly[i].Transparent:=True;
    end;
  BkGraund1.LoadFromFile(BkFileName);
  ImageFly.Assign(Fly[0]);
  Parent:=nil;
  BkGraund2.Assign(BkGraund1);
  with BkGraund1 do
    begin
      Left:=0;
      Top:=0;
      ClientWidth:=Width;
      ClientHeight:=Height+StatusBar.Height;
    end;
  OldBounds:=Bounds;
end;

function Max(a, b: Integer): Integer;
begin
  if b < a then
    Result:=a
  else
```

```
    Result:=b;
end;

function Min(a, b: Integer): Integer;
begin
  if b > a then
    Result:=a
  else
    Result:=b;
end;

function MoveImage(Image: TBitmap): TRect;
begin
   x:=x+vx*dt+ax*dt*dt/2;
   y:=y+vy*dt+ay*dt*dt/2;
   vx:=vx+ax*dt;
   vy:=vy+ay*dt;
   Result.Left:=Round(x);
   Result.Right:=Result.Left+Image.Width;
   Result.Top:=Round(y);
   Result.Bottom:=Result.Top+Image.Height;
end;

procedure TForm1.DrawImage;
var
  NewBounds: TRect;
  xx, yy, fire: Integer;
const
  MaxV = 100;
begin
  Bounds:=MoveImage(ImageFly);
  fire:=ImageFly.Height-ImageFly.Width;
  if Bounds.Bottom-fire>=ClientHeight-StatusBar.Height
    then
      begin
        SetDown:=True;
        Bounds.Top:=ClientHeight-StatusBar.Height-ImageFly.Width;
        Bounds.Bottom:=ClientHeight-StatusBar.Height;
        y:=Bounds.Top;
        Timer.Enabled:=False;
        if Sqrt(vx*vx+vy*vy)<=MaxV
          then
            begin
              StatusBar.Panels[5].Text:='Успешная посадка';
              ImageFly.Assign(Fly[0]);
              CanStart:=True;
            end
          else
            begin
              StatusBar.Panels[5].Text:='Авария';
              ImageFly.Assign(Fly[5]);
              CanStart:=False;
            end;
        vx:=0; vy:=0; ax:=0; ay:=gy;
      end;
  xx:=Round((Bounds.Left+ImageFly.Width div 2)/10) ;
  yy:=Round((ClientHeight-StatusBar.Height-Bounds.Bottom)/10);
  StatusBar.Panels[1].Text:=IntToStr(xx);
  StatusBar.Panels[3].Text:=IntToStr(yy);

  NewBounds.Left:=Min(OldBounds.Left, Bounds.Left);
  NewBounds.Top:=Min(OldBounds.Top, Bounds.Top);
  NewBounds.Right:=Max(OldBounds.Right, Bounds.Right);
  NewBounds.Bottom:=Max(OldBounds.Bottom, Bounds.Bottom);
```

```
  BkGraund1.Canvas.CopyRect(OldBounds, BkGraund2.Canvas, OldBounds);
  BkGraund1.Canvas.Draw(Bounds.Left, Bounds.Top, ImageFly);
  Canvas.CopyRect(NewBounds, BkGraund1.Canvas, NewBounds);
  OldBounds:=Bounds;
  if SetDown then
    if not CanStart then
      begin
        PlaySound('damage.wav', 0, SND_NOSTOP);
        ImageFly.Assign(Fly[6]);
        BkGraund1.Canvas.CopyRect(OldBounds, BkGraund2.Canvas, OldBounds);
        BkGraund1.Canvas.Draw(Bounds.Left, Bounds.Top, ImageFly);
        Canvas.CopyRect(NewBounds, BkGraund1.Canvas, NewBounds);
      end
    else
      PlaySound('Fanfare.wav', 0, SND_NOSTOP);
end;

procedure TForm1.FormDestroy(Sender: TObject);
var
  i: Integer;
begin
  BkGraund1.Free;
  BkGraund2.Free;
  ImageFly.Free;
  for i:=0 to 4 do Fly[i].Free;
end;

procedure TForm1.FormPaint(Sender: TObject);
begin
   Canvas.Draw(0, 0, BkGraund1);
end;

procedure TForm1.TimerTimer(Sender: TObject);
begin
  DrawImage;
end;

procedure TForm1.FormKeyDown(Sender: TObject; var Key: Word; Shift:
TShiftState);
const
  sh = gy;
begin
  if not SetDown or Start then
  case Key of
      VK_UP: begin
               ay:=ay-sh;
               PlaySound('Fire1.wav', 0, SND_ASYNC);
               ImageFly.Assign(Fly[1]);
               if (not Timer.Enabled) and Start and CanStart
                 then
                   begin
                     Start:=False;
                     SetDown:=False;
                     Timer.Enabled:=True;
                     ay:=ay-sh;
                     StatusBar.Panels[5].Text:='';
                   end;
             end;
    VK_DOWN: begin
               ay:=ay+sh;
               ImageFly.Assign(Fly[4]);
               PlaySound('Fire1.wav', 0, SND_ASYNC);
             end;
```

```
    VK_LEFT: begin
               ax:=ax-sh div 2;
               ImageFly.Assign(Fly[2]);
               PlaySound('Fire1.wav', 0, SND_ASYNC);
             end;
   VK_RIGHT: begin
               ax:=ax+sh div 2;
               ImageFly.Assign(Fly[3]);
               PlaySound('Fire1.wav', 0, SND_ASYNC);
             end;
   VK_SPACE: begin
               vx:=0;
             end;
  VK_RETURN: begin
               Start:=True;
             end;

  end;
end;

procedure TForm1.FormKeyUp(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
//  if CanStart then
  case Key of
      VK_UP: begin
               ay:=gy;
               ImageFly.Assign(Fly[0]);
               PlaySound('Nil.wav', 0, SND_ASYNC);
             end;
    VK_DOWN: begin
               ay:=gy;
               ImageFly.Assign(Fly[0]);
               PlaySound('Nil.wav', 0, SND_ASYNC);
             end;
    VK_LEFT: begin
               ax:=gx;
               ImageFly.Assign(Fly[0]);
               PlaySound('Nil.wav', 0, SND_ASYNC);
             end;
   VK_RIGHT: begin
               ax:=gx;
               ImageFly.Assign(Fly[0]);
               PlaySound('Nil.wav', 0, SND_ASYNC);
             end;
  end;
end;

procedure TForm1.nNewGameClick(Sender: TObject);
begin
  Timer.Enabled:=True;
  NewGame;

end;

end.
```

Оглавление:

ЗАНЯТИЕ 1.....5

Алгоритмы и способы их описания.....5
Упрощенная модель компилятора.....8
Задания для самостоятельной работы.....10

ЗАНЯТИЕ 2.....11

Структура программы на языке «Pascal».....11
Типы данных в языке Pascal.....11
Понятие выражения, операции и операнда.....12
Понятие оператора.....15
Простые операторы.....16
Задания для самостоятельной работы.....16

ЗАНЯТИЕ 3.....18

Структурные операторы.....18
Унифицированная структура «Развилка».....18
Условная конструкция if .. then .. else.....19
Задания для самостоятельной работы.....20

ЗАНЯТИЕ 4.....21

Цикл с постусловием.....21
Цикл с предусловием.....22
Цикл с параметром.....23
Задания для самостоятельной работы.....24

ЗАНЯТИЕ 5.....25

Решение квадратных уравнений.....25
Игра «Угадай число».....26
Задания для самостоятельной работы.....27

ЗАНЯТИЕ 6.....28

Унифицированная структура выбора.....29
Оператор выбора CASE.....29
Задания для самостоятельной работы.....30

ЗАНЯТИЕ 7.....31

Калькулятор.....31
Некоторые математические функции и процедуры Turbo Pascal 7.0.....32
Задания для самостоятельной работы.....33

ЗАНЯТИЕ 8.....34

Структурированные типы данных: массив.....35
Строковый тип данных.....39
Типизированные константы.....40
Задания для самостоятельной работы.....41

ЗАНЯТИЕ 9.....43

Сортировка массивов 43
Двоичный поиск 47
Задания для самостоятельной работы 48
ЗАНЯТИЕ 10 49
Процедуры и функции 49
Кодирование информации 53
Задания для самостоятельной работы 54
ЗАНЯТИЕ 11 55
Модули 55
Задания для самостоятельной работы 58
ЗАНЯТИЕ 12 60
Файлы 60
«Алекс – Юстасу» 66
Задания для самостоятельной работы 67
ЗАНЯТИЕ 13 68
Записи 68
Нетипизированные файлы 70
Создание BMP файла 72
Задания для самостоятельной работы 76
ЗАНЯТИЕ 14 77
Интегрированная среда разработки 77
Изменяем привычное 80
Задания для самостоятельной работы 82
ЗАНЯТИЕ 15 83
Игра «Угадай число» 83
Задания для самостоятельной работы 85
ЗАНЯТИЕ 16 86
Поиск файла на диске 86
Класс: список строк TStringList 87
Отображение картинок 88
Создание галереи просмотра картинок 88
Создание процедуры открытия картинки 88
Обработка событий 89
Полный текст программы 89
Задания для самостоятельной работы 91
ЗАНЯТИЕ 17 92
Летающая тарелка 92