БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень

МЕТОДЫ ПРОГРАММИРОВАНИЯ

В двух частях

Часть 2

ТЕОРИЯ И ПРАКТИКА ПРОГРАММИРОВАНИЯ НА PASCAL

Рекомендовано
Учебно-методическим объединением
по естественно-научному образованию в качестве
учебно-методического пособия для студентов,
обучающихся по специальности «математика (по направлениям)»,
направление специальности «математика
(научно-педагогическая деятельность)»

Учебное электронное издание

Минск, БГУ, 2022

ISBN 978-985-881-166-2 (ч. 2)
ISBN 978-985-881-168-6

УДК 004.432Pascal(075.8)
ББК 32.973.26-018.1я73-1

Р е ц е н з е н т ы:
кафедра математического анализа дифференциальных уравнений
и их приложений Брестского государственного
университета им. А. С. Пушкина (заведующий кафедрой
кандидат физико-математических наук *Н. Н. Сендер*);
кандидат технических наук *О. Г. Смолякова*

Расолько, Г. А. Методы программирования [Электронный ресурс] : учеб.-метод. пособие. В 2 ч. Ч. 2. Теория и практика программирования на Pascal / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2022. – 1 электрон. опт. диск (CD-ROM). – ISBN 978-985-881-166-2.

Рассмотрено создание модулей пользователя, вопросы работы с файловой системой, программирование алгоритмов с использованием указателей, стандартные приемы работы с устройствами персонального компьютера.

Минимальные системные требования:
PC, Pentium 4 или выше;
RAM 1 Гб; Windows XP/7/10; Adobe Acrobat.

Оригинал-макет подготовлен в программе MicroSoft Word.

В авторской редакции

Ответственный за выпуск *Е. В. Кремень*. Дизайн обложки *В. П. Явуз*.
Технический редактор *В. П. Явуз*. Компьютерная верстка *Е. В. Кремень*.

Подписано к использованию 28.02.2022. Объем 1,78 МБ.

Белорусский государственный университет.
Управление редакционно-издательской работы.
Пр. Независимости, 4, 220030, Минск.
Телефон: (017) 259-70-70.
e-mail: urir@bsu.by
http://elib.bsu.by/

СОДЕРЖАНИЕ

ТЕМА 1
МОДУЛИ

Содержание темы

- Стандартные библиотечные модули.
- Модули пользователя.
- Подпрограммы в модулях.
- Комбинированный тип «запись».
- Обработка данных типа «запись».

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

СТАНДАРТНЫЕ БИБЛИОТЕЧНЫЕ МОДУЛИ

Pascal дает программисту довольно широкий набор «встроенных» процедур и функций для реализации действий, которые наиболее часто встречаются при написании программ.

Различают процедуры и функции таких видов: арифметические, преобразования типов, управления строками на экране, управления памятью для динамических переменных, обработки строк, файлов, выполнения действий по выводу графических объектов и др.

Система программирования Turbo Pascal имеет модульную структуру, когда все стандартные средства выделены в отдельные группы, которые расположены в физически обособленных библиотеках – стандартных модулях. Эти библиотеки обеспечивают неограниченное расширение программных возможностей. Каждый модуль объединяет логически отделенную именованную группу типов данных, констант, переменных, процедур и функций.

Представление программы как единой языковой конструкции явилось преградой для эффективной организации коллективных разработок сложных систем. Подпрограммы, рассмотренные нами раньше (процедуры и функции), были частью программы.

Решающим шагом на пути преобразования языка Turbo Pascal в язык, подходящий для крупных разработок производственного и коммерческого назначения на современном уровне технологии программирования, явилось

введение понятия модуля. За счет введения модулей удалось ослабить ограничения на суммарный объем (≤ 64) кб готовых программ.

В языке Turbo Pascal модуль (unit) по определению считается отдельной программной единицей. Если подпрограмма является структурным элементом Pascal-программы и не может существовать вне нее, то модуль представляет собой отдельно хранимую и независимо компилируемую программную единицу.

В общем виде *модуль* – это совокупность (коллекция) программных ресурсов, предназначенных для использования другими модулями и программами. Под ресурсами будем понимать константы, типы, переменные, подпрограммы.

Turbo Pascal включает 10 стандартных модулей для реального режима DOS. В библиотеке TURBO.TPL содержатся модули SYSTEM, OVERLAY, DOS, CRT, PRINTER. Остальные модули (GRAPH, STRINGS, WINDOS, TURBO3, GRAPH3) располагаются в отдельных файлах с расширением TPU.

Программные ресурсы, сосредоточенные в стандартных модулях, образуют мощные пакеты системных средств, обеспечивающих высокую эффективность и широкий спектр применения системы Turbo Pascal.

Модуль SYSTEM подключается автоматически, является сердцем Turbo Pascal, поскольку ресурсы, которые он содержит, обеспечивают работу всех остальных модулей системы. Этот модуль содержит все стандартные функции, как математические (*exp, ln, sin*, ...), так и другие: поддерживает динамическое распределение памяти, целочисленную арифметику и с плавающей точкой, объединяющей подпрограммы ввода-вывода и др.

Crt – управляет дисплеем, клавиатурой и звуком.

DOS и WINDOS - обслуживают прерывания, выполняют проверку состояния дисков, содержат специальные средства обработки файлов, совершают управление операционным окружением, таймером (т. е. обеспечивают работу с функциями операционной системы MS DOS и Windows).

Graph – содержит огромный пакет графических средств.

Graph3 – поддерживает использование стандартных графических средств версии Turbo Pascal 3.0 (графические процедуры и функции этого модуля имеют другие названия по сравнению с процедурами и функциями модуля Graph).

Turbo3 – обеспечивает совместимость с версией Turbo Pascal 3.0.

Printer – обеспечивает быстрый доступ к устройству печати (устарел).

Overlay – содержит средства организации программ, которые по очереди совместно используют общую часть памяти.

`Strings` – дает возможность программе использовать строки, которые используются в Windows-приложениях.

`Win` – является приложением к модулю `CRT`, обеспечивая новые возможности при работе с окнами.

МОДУЛИ ПОЛЬЗОВАТЕЛЯ

Модуль, кроме заголовка `Unit Unitname`, имеет три части – *интерфейсную*, *реализации* и *инициализации*.

В *интерфейсной части* модуля собраны описания объектов, которые доступны из других программ. Это видимые вне модуля объекты.

В *части реализации* содержатся рабочие объекты – скрытые, или невидимые.

В *части инициализации* описываются действия, которые будут выполнены при подключении модуля.

Общая структура модуля:

```
UNIT Unitname;
INTERFACE
    Описание видимых объектов
IMPLEMENTATION
    Описание скрытых объектов
BEGIN
    Операторы инициализации объектов модуля
END.
```

Описание типов, констант, переменных, заголовков процедур, функций

Реализация процедур и функций, которые описаны в интерфейсной части, и вспомогательных алгоритмов, типов, констант и переменных

Установка начальных значений переменных модуля перед его использованием

При описании модуля используются служебные слова: `unit, interface, implementation`.

Пример 1.

```
unit Calendar;
interface
type
       Days        = (Mon, Tue, Wed, Thu, Fri, Sat, Sun);
       WorkingDays = Mon .. Fri;
       Months      = (Jan, Feb, Mar,  Apr, May, June,
```

```
                  Jule, Aug, Sept, Oct, Nov, Desem);
      Summer      = June .. Aug;
      Autumn      = Sept .. Nov;
      Spring      = Mar  .. May;
      Dayno       =    1 ..31;
      Yearno      = 1900 ..2020;
      Date        = record
                     Day   : Dayno;
                     Month : Months;
                     Year  : Yearno
                    end;
          {здесь можно описать подпрограммы
           для получение дня недели
           по заданному дню, месяцу, году...}

 implementation

end.
```

Пример касается определений, связанных с датами и месяцами. Данный модуль, ввиду своей простоты, не содержит разделов реализации и инициализации.

Подключение модуля происходит в начале программы операторам `uses`.

Спецификация `uses` должна идти непосредственно после заголовка программы, так как она содержит определения данных и подпрограмм.

Если некоторый модуль подключается в модуль в раздел `interface` или `implementation`, тогда `uses` должен идти сразу после служебных слов `interface` или `implementation`.

ПОДПРОГРАММЫ В МОДУЛЯХ

Процедуры и функции могут использоваться в модулях наряду с другими Pascal-объектами. Считается, что заголовок подпрограммы является ее интерфейсом (видимый), а тело – реализацией (невидимо), так как заголовок содержит всю информацию, необходимую для вызова: ее имя, число и типы параметров, а для функций – и тип результата. А тело подпрограммы раскрывает алгоритм.

Значит, в интерфейсной части модуля представлены только заголовки процедур и функций, видимые (доступные) для других программ, а их

полное описание, которое содержится в разделе реализации, может иметь сокращенный заголовок, состоящий только из служебного слова `procedure` или `function`, имя подпрограммы и «;». (Но можно идентично повторить и полный заголовок).

Пример 2. В следующем модуле собраны средства для работы с комплексными числами, которые дальше можно расширить, например, для:

а) на работу с массивами: перемножить матрицы, умножить матрицу на вектор и т. д.

б) на подсчет тригонометрических функций от комплексного аргумента и т. д.

```
unit CmplVals;
interface
type
      Complex = record
                 Re, Im : Real
                end;
    {заголовки процедур, которые реализуют
      операции над комплексными числами}

procedure  InitC  (R, I : Real;      var C : Complex);
procedure  AddC   (C1, C2 : Complex; var R : Complex);
procedure  MultC  (C1, C2 : Complex; var R : Complex);
procedure  DivC   (C1, C2 : Complex; var R : Complex);
procedure  WriteC (C : Complex);
function   ModC   (C : Complex) : Real;

implementation
                 {полное описание процедур}
procedure  InitC;
begin
 with C do
     begin
      Re := R;      Im := I
     end
end;
…
end.
```

Таким образом, механизм модулей позволяет спрятать детали реализации тех или иных программных подсистем.

Как результат, изменение реализации какой-нибудь подпрограммы при условии, что интерфейс модуля при этом останется неизменным, никак не отобьется на программах, его использующих.

Модуль компилируется таким же образом, как и программа, но поскольку модуль не является непосредственно выполняемой единицей, то в результате его компиляции образуется дисковый файл с расширением «.TPU» (Turbo Pascal Unit), при этом имя TPU-файла должно совпадать с именем файла с исходным текстом модуля.

Поэтому имя модуля не может состоять более чем из восьми символов.

И, если мы хотим создать модуль unit, то нужно, чтобы у файла, в котором он будет сохранен, имя было таким же, как и у модуля unit.

Рассмотрим программу, которая использует модуль CmpVals:

```
program UsingComplex;
uses CmplVals;
  {чтобы получить доступ к интерфейсным объектам модуля,
    необходимо указать в программе имя нужного TPU-файла}
var C1, C2, C3 : Complex;
    {заданный в модуле тип Complex, доступен здесь так же,
      как будто он определен в программе}
begin
 InitC(1,  2, C1);
 InitC(3,  4, C2);
 MultC(C1, C2, C3);
 WriteC(C3);
 DivC(C1,  C2, C3);
 WriteC(C3)
end.
```

В связи с использованием модулей возникают следующие важные моменты:

1. Может случиться, что идентификаторы интерфейсной части модуля частично пересекаются с идентификаторами программы. В этом случае идентификаторы программы «экранируют» (затемняют) одноименные идентификаторы модуля, например:

```
program Pr;
uses A, B; …
```

В таком случае идентификаторы программы Pr затемняют идентификаторы модуля B, а те затемняют идентификаторы модуля A.

2. Но если все же следует обратиться к одноименному данному из модуля, то следует образовать составное имя по принципу Unitname.name, структура которого похожа на селектор поля записи, например:

```
X := A.X;
```

Переменной x программы Pr присваивается значение переменной x модуля A.

3. Возможны случаи косвенного использования модулей.

```
unit A;
interface
  …
end.
unit B;
interface
uses A;
  …
end.

program P;
uses B;  …
end.
```

Достаточно указать только те модули, которые непосредственно используются в программе.

4. Схема использования модулей может образовывать древовидную структуру любой сложности, но при этом недопустим явное или косвенное обращение модуля к самому себе.

5. Если в модуле существует раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы (или модуля), в которой используется данный модуль.

Отметим следующее:

1. Раздел unit содержит имя библиотечного модуля. Оно должно совпадать с именем дискового файла, где находится исходный текст модуля.

save *compile*

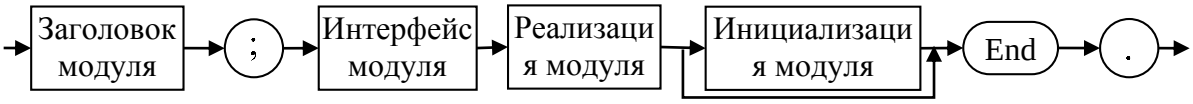
unit Stat ⇒ stat.pas (сохранили в файле) ⇒ stat.tpu

В *интерфейсной части* описываются константы, типы, переменные, процедуры и функции, которые являются глобальными, т. е. доступны основной программе (или модулю, который использует данный модуль).

В *секции реализации* определяются модули всех глобальных подпрограмм. Кроме того, в ней описываются константы, переменные, подпрограммы, которые уже локальные.

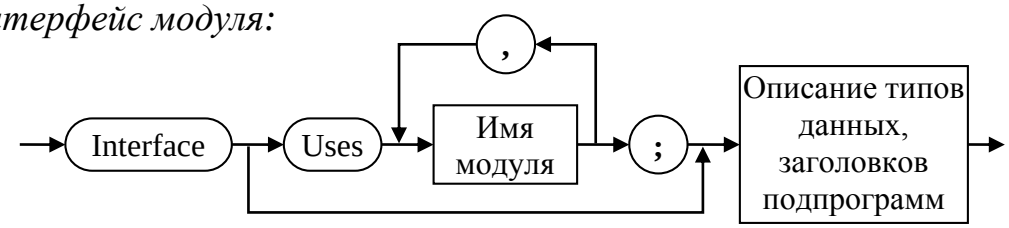
2. *Секция инициализации* – последняя секция модуля. Начинается словом `begin`, и далее располагаются операторы инициализации, если они есть.

Структурная диаграмма модуля:

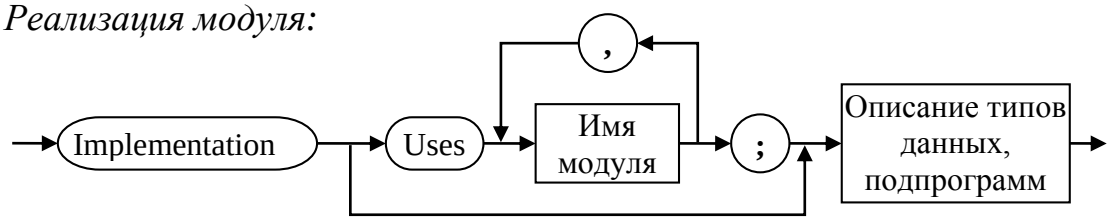


Заголовок модуля:

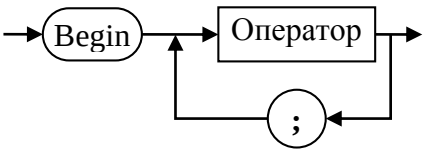
Интерфейс модуля:



Реализация модуля:



Инициализация модуля:



Встроенный компоновщик файлов ищет подключенный unit сначала в `TURBO.TPL` (`Turbo Pascal Library`), а потом в unit-директории, определенной в среде `Turbo`.

Свои модули можно помещать в `TURBO.TPL` специальной утилитой, но лучше построить свою `MyUnits` библиотеку.

Использование модуля:

- Открыть файл, содержащий текст программы, которая использует модуль `MyUnit.`

• В меню Options выбрать команду Directories и в диалоговом окне в поле UnitDirectories указать путь к модулю MyUnit.

• Далее через Run можно запустить на выполнение главную программу.

Модуль – это автономно компилируем программная единица.

В среде TP имеются средства, управляющие способом компиляции модулей. В меню COMPILE можно выбрать одну из следующих команд:

• COMPILE: при компиляции модуля или основной программы все заявленные модули должны быть предварительно откомпилированы и программе должны быть доступны все файлы *.Tpu. Путь к модулю задается в опции Unit Directories меню OPTIONS/Directories.

• MAKE: компилятор проверяет наличие tpu-файлов для каждого объявленного модуля. Если какой-либо из файлов не нашелся, система ищет одноименный файл с расширением pas (файл с исходным текстом) и его компилирует. Система перекомпилирует любой unit, который был модифицирован после последней компиляции. Если модуль не найден, то возникает ошибка компиляции.

• BUILD: существующие tpu-файлы игнорируются, система отыскивает и компилирует соответствующие pas-файлы для каждого объявленного модуля, подключаемого к данной программе. Программист должен обеспечить доступ к любому pas-файлу заявленного модуля unit.

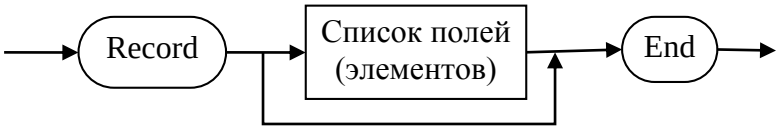
КОМБИНИРОВАННЫЙ ТИП «ЗАПИСЬ»

Кроме рассмотренных выше типов данных существует *комбинированный тип* данных, где объединяются разно-типовые элементы.

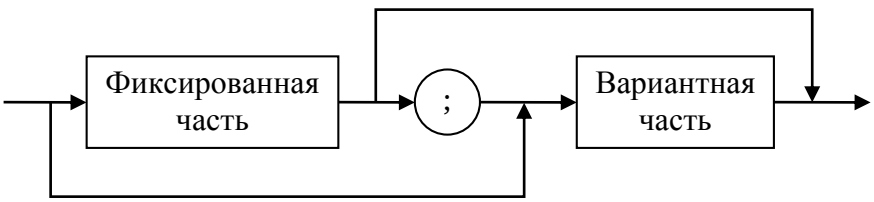
Запись – это объединение данных разных типов, посредством задания именованных полей. В отличие от массивов и множеств поля записей могут иметь различные типы.

Для определения записи применяют служебные слова record и end.

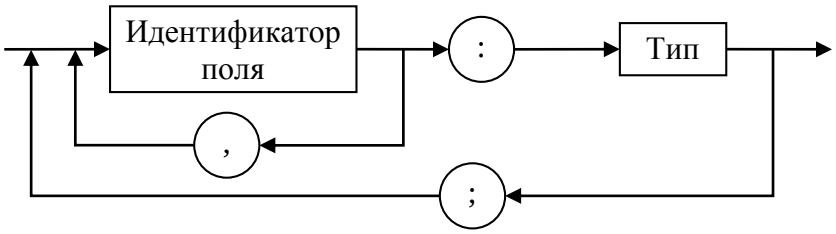
Запись:



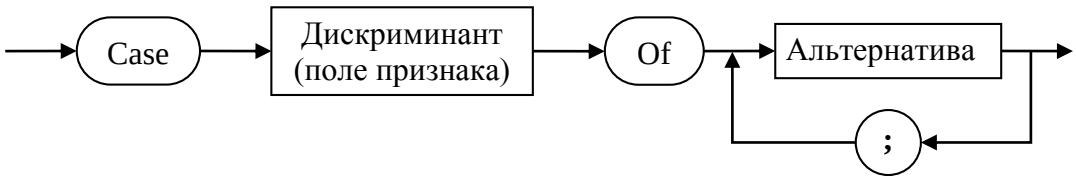
Список полей:



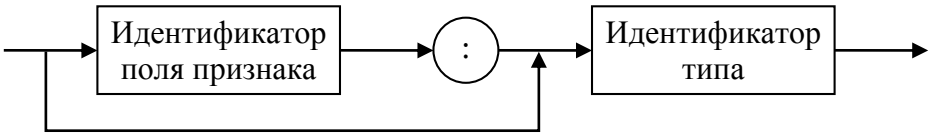
Фиксированная часть:



Вариантная часть:



Дискриминант:



Пример 1.

```
type
     Rec = record
              x, y : Real
           end;
var A : Rec;
```

Пример 2.

```
type
     Person = record
              FIO     : String;
                       {фамилия, имя, отчество ≤ 255 символов}
              Weight, Height: Word;
              Telephone    : Longint; {номер телефона}
              Marks        : array[1..4] of Byte;
              end;
```

Тип `Person` сейчас задает анкету из строки (`FIO`), двух чисел с дробной частью (`Weight, Height`), одного целого длинного числа (`Telephone`) и массива на четыре байта (`Marks`).

Если задано объявление

```
var  Somebody: Person;
```

то под именем `Kto_to` понимается данное с конкретными значениями.

Доступ к полям осуществляется по имени при помощи селектора записи согласно следующему формату:

```
имя_переменной.имя_поля
```

`Somebody.FIO` – это значение строки с фамилией.

`Somebody.Telephone` – это значение длинного целого с заданным телефоном.

При присвоении первоначальных значений данным типа «запись» делают так:

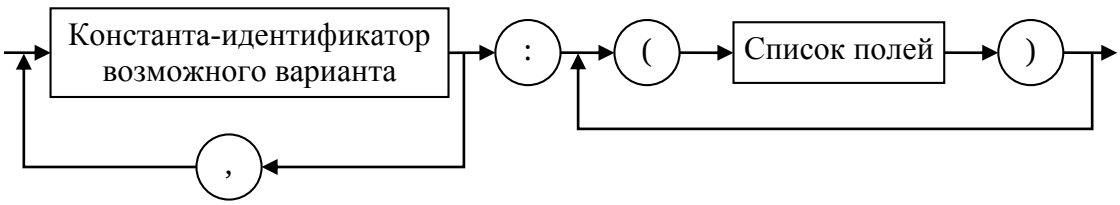
```
const
      Someone: Person =
                        (FIO    : 'XXX'; Weight    : 50.5;
                         Height : 158;   Telephone : 265539;
                         Marks  : (5,4,3,4));
```

Поле от своего значения отделяется символом «:». Порядок чередования полей должен соответствовать порядку их описания в типе, поля должны разделяться НЕ запятой, а «;», как это делается в описании типа «запись».

Дискриминант позволяет контролировать включение в запись тех или иных вариантов.

Если на схеме выбран путь через идентификатор поля признака далее к идентификатору типа, то поле в записи присутствует, иначе – отсутствует.

Альтернатива:



Константа-идентификатор возможного варианта принадлежит типу поля признака и дает значение поля признака.

Из определения типа запись видно, что запись содержит фиксированное количество компонент, которые называются *полями*. Количество полей в записи, а также их назначение известны заранее. В отличие от компонент массива (выбираемых по своим индексам) доступ к полям записи осуществляется через имена полей. Имя поля должно быть

уникальным только в пределах данной записи. Размер памяти, необходимый для записи, рассчитывается исходя из длин полей (если с вариантной частью, то максимальное поле переменной части). Функция `Sizeof(Z)` – возвращает длину записи `Z`.

Список полей может иметь только одну вариантную часть, которая всегда располагается после фиксированной, но вариантная часть сама может содержать и фиксированную часть и вариантную. Это видно из схемы.

Вернемся к примеру из предыдущей лекции и продолжим его разработку.

Пример 3. В отдельном модуле собрать средства для работы с комплексными числами, которые объявляются как запись с двумя вещественными полями.

В главной программе отладить описанные процедуры и функции.

Данный модуль расширить:

а) на работу с одно и двумерными массивами (инициализация массива, распечатка массива, реализация задач линейной алгебры, например, перемножить матрицы, умножить матрицу на вектор и т. д.);

б) на подсчет тригонометрических функций от комплексного аргумента и т. д.

```
unit CmplVals;
interface
type
      Complex = record
                 Re, Im : Real
                end;
procedure  InitC  (R, I : Real;      var C : Complex);
procedure  AddC   (C1, C2 : Complex; var R : Complex);
procedure  MultC  (C1, C2 : Complex; var R : Complex);
procedure  DivC   (C1, C2 : Complex; var R : Complex);
procedure  WriteC (C : Complex);
function   ModC   (C : Complex) : Real;

implementation
                 {полное описание процедур}
procedure  InitC;
  begin
     with C do
```

```
       begin
         Re := R;      Im := I
       end
  end;

procedure  AddC (C1, C2 : Complex; var R : Complex);
  begin
    with R do
      begin
        Re := C1.Re+C2.Re;
        Im := C1.Im+C2.Im
     end
  end;

procedure  MultC (C1, C2 : Complex; var R : Complex);
  begin
    with R do
      begin
        Re := C1.Re*C2.Re– C1.Im*C2.Im;
        Im := C1.Re*C2.Im+C2.Re*C1.Im
      end
  end;

function  ModC (C : Complex) : Real;
  begin
    ModC := C.Re*C.Re+C.Im*C.Im;
  end;

procedure  DivC (C1, C2 : Complex; var R : Complex);
  var Tmp : Real;
  begin
T    mp := ModC(C2);
     with R do
       begin
         Re := (C1.Re*C2.Re+C1.Im*C2.Im)/Tmp;
         Im := (C2.Re*C1.Im–C1.Re*C2.Im)/Tmp
       end
  end;

procedure  WriteC (C : Complex);
  begin
    with C do
      begin
```

```
        Write(Re); if Im=0 then Exit;
        if Im > 0 then Write ('+');
        Write(Im); Write ('i');
      end
  end;
end.
```

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 2
ФАЙЛЫ В ЯЗЫКЕ PASCAL

Содержание темы

- Файловые типы.
- Операции над файлами:
  - установочные и завершающие операции,
  - специальные операции.
- Операции ввода-вывода для файлов с типом.
- Последовательный и прямой доступ к файлу с типом.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ФАЙЛОВЫЕ ТИПЫ

Представители файла в Pascal-программе – переменные файловых типов. Для описания файловой переменной используются слова `text`, `file` или словосочетание `file of`. Например, описание переменной

```
var   f : file of Integer;
```

понимается как определение под именем `f` последовательности, расположенной на некотором внешнем запоминающем устройстве (например, на диске) и состоящей из неясного количества целых чисел, которые называются *компонентами* или *элементами файла*.

При работе с файлом с каждой переменной файлового типа согласуется «текущий указатель» файла, который можно понимать как скрытую переменную (т. е. не очевидно описанную вместе с файловой переменной). Текущий указатель обозначает («указывает» на) некоторый конкретный элемент файла.

Как правило, все действия с файлом (чтение из файла, запись в файл) выполняются поэлементно, причем в этих действиях участвует тот элемент файла, на который указывает текущий указатель.

На практике ввод-вывод выполняется целым блоком элементов через некоторый системный буфер ввода-вывода.

Будем понимать, что текущий указатель – это окно, которое перемещается по файлу и через которое мы «видим» элементы файла, доступные для обработки.

Если файловый тип задается в программе при помощи служебных слов `file of`, за которыми идет тип элементов файла (базовый тип), то базовый тип может быть любым типом, за исключением файлового и типа «объект» (который будет рассмотрен позже). Кроме того, в качестве базового типа не допускается запись, одним из полей которой является файл или объект.

ОПЕРАЦИИ НАД ФАЙЛАМИ

В отличие от переменных других типов язык Pascal не содержит встроенных операций над собственно файловыми переменными. Операции с файлами реализованы в виде стандартных подпрограмм, собранных в модуле `System`, который присоединяется автоматически.

Операции с файловыми переменными можно разбить на четыре основные группы:

- установочные и завершающие операции;
- специальные операции;
- собственно ввод-вывод;
- перемещение по файлу.

Установочные и завершающие операции

В эту группу входят следующие процедуры.

1. `Assign(f, Name)` связывает файл `Name` с файловой переменной `f`.

2. `Reset(f)` открывает доступ к существующему файлу `f`.

3. `Rewrite(f)` создает и открывает новый файл `f`.

4. `Flush(f)` переписывает данные из буфера в файл.

5. `Close(f)` закрывает файл.

Здесь `f` – переменная файлового типа.

Рассмотрим процедуры подробнее.

1. Процедура `Assign(f, Name)` имеет первый параметр – имя файловой переменной f, второй параметр – строковое выражение – полное имя файла. Процедура предназначена для установления связи между конкретным файлом (набором данных) на внешнем носителе и переменной файлового типа. Например:

```
Assign(f, 'd:\mydir\myfile.dat');
```

После выполнения этого оператора предполагается, что файловая переменная `f` будет связана с дисковым файлом `myfile.dat`, расположенном в каталоге `mydir` корневого каталога диска `d`. (Проверка на корректность не делается.) Эта процедура всегда предшествует другим процедурам работы с файлами.

Вместо набора данных (файла) может быть любое устройство ввода-вывода: клавиатура, печатающее устройство или дисплей. Их еще называют псевдофайлами MS DOS.

2-3. Процедуры `Reset (f)` и `Rewrite (f)` предназначены для открытия файла, где `f` — файловая переменная, назначенная предварительно оператором `Assign` файлу. Под открытием в данном случае понимается поиск файла на внешнем носителе; создание специальных системных буферов для обмена с ним (операционная система ставит в соответствие каждому открываемому файлу скрытый от нас обработчик файлов со своим номером); установка текущего указателя файла на его начало. Все элементы файла с типом условно нумеруются: 0, 1, 2, ...

Процедура `Reset(f)` предполагает, что файл, который открывается, уже существует, в противном случае возникает ошибка.

Процедура `Rewrite(f)` допускает, что файл, который открывается, может не существовать, тогда она создает заданный файл. Если же файл существует, тогда `Rewrite` очищает его.

В обоих случаях, если файл `f` был открыт, он предварительно закрывается. Но связь с существующим файлом не нарушается (она настроена через `Assign`).

4. Процедура `Flush(f)` завершает обмен с файлом без его закрытия.

Обмены с файлами всегда реализуются через некоторый буфер в оперативной памяти, поэтому в процессе записи в файл последние элементы, которые записываются, могут еще «остаться» в буфере. Процедура `Flush` вызывает принудительное сбрасывание этих элементов в файл.

5. Процедура `Close (f)` завершает действие с файлом `f`:

а) переписывает информацию из буфера в файл (если файл для вывода);

б) устраняет внутренние буферы, созданные при открытии этого файла;

в) устраняет связь файла с файловой переменной `f` (освобождает обработчик файла для других работ).

После этого файловую переменную можно связать с помощью `Assign` с каким-либо другим файлом или устройством.

Заметим, что при окончании работы программы происходит автоматическое закрытие всех открытых в программе файлов. Однако хорошим правилом является явное закрытие файлов после окончания работы с ними, ибо может потеряться часть информации, которая осталась в буфере.

6. Процедура `Append(f)` – открывает существующий тестовый файл `f` для добавления.

Если файл отсутствует на диске, то возникает ошибка ввода-вывода.

Специальные операции

Эта группа операций предназначена для действий с элементами файловой системы `MS DOS` – каталогами и именами файлов, позволяя создавать и ликвидировать файлы, работать с атрибутами файлов и так далее.

Подробнее об этих операциях можно прочесть в фирменных руководствах по языку. Рассмотрим только 2 следующие:

- `Erase(f)` уничтожение файла на диске, который был связан с файловой переменной f. Если файл не существует, то возникает ошибка ввода-вывода;
- `Rename(f, name)` – переименование файла.

Эти процедуры работают с неоткрытыми файлами: требуется выполнить только процедуру `Assign`, но не выполнять `Reset` или `Rewrite`.

`Rename(f, newname)` переименовывает неоткрытый файл `f`. Новое имя задается строкой `newname`. При этом нельзя изменять имя диска и путь к файлу, изменяется только собственное имя физического файла.

Операции ввода-вывода для файлов с типом

В эту группу входят две операции (процедуры) `Read` и `Write`, которые реализуют чтение информации из файла и запись информации в файл соответственно.

Обмен данными происходит через буфер ввода-вывода, размер которого устанавливается автоматически, исходя из размера элементов файла.

Как бы файл с типом ни был открыт (`Reset` или `Rewrite`), в модуле `System` описана переменная `FileMode`, значение которой становится равной 2 (значение 0 означает только чтение, значение 1 – только запись). Оно показывает, что из файла можно и читать, и в файл можно записывать данные.

В отличие от многих других процедур, Read и Write могут вызываться с разным числом параметров. Формат операторов:

Read(f, v1, ..., vn) и

Write(f, v1, ..., vn).

Список v1, ..., vn может состоять из нескольких или из одной переменной базового типа файла.

Процедура Read предназначена для чтения значений из файла. Первый параметр – имя файловой переменной, к которой была применена одна из операций открытия (Reset или Rewrite). Далее должны идти переменные, в которых будут помещены значения из файла. Тип этих переменных должен совпадать с базовым типом файла.

Выполнение процедуры Read происходит следующим образом. Начиная с текущей позиции указателя файла, последовательно будут читаться значения, находящиеся в файле, и присваиваться очередной переменной из тех, которые перечислены в списке ввода. После каждого действия по чтению данного из файла, указатель файла будет перемещаться на следующий элемент. Если в процессе выполнения процедуры Read текущий указатель файла будет установлен на позицию последнего элемента файла, и он будет прочитан, тогда возникнет ситуация «*конец файла*».

Возникновение ситуации «конец файла» можно проверить при помощи встроенной логической функции EoF(F) с параметром F – файловой переменной. Функция EoF(F) возвращает логическое значение true, если достигнут конец файла, и false – в противном случае. При обращении к файлу для чтения, у которого EoF имеет значение true, система дает фатальную ошибку 100 (считывание после конца файла).

При записи в файл истинность функции EoF (f) означает, что очередная операция записи поместит информацию в конец данного файла.

Процедура Write позволяет записывать информацию в файл. Первым параметром этой процедуры должна быть файловая переменная, открытая процедурой Reset или Rewrite. Далее должен идти список переменных, тип которых совпадает с базовым типом файловой переменной. При записи в файл записывается внутреннее представление очередного элемента. В файле элементы занимают один и тот же объем памяти в соответствии со своим типом.

Последовательный и прямой доступ к файлу с типом

Данная группа операций позволяет произвольно изменять последовательность выполнения операций чтения и записи. Сюда входят следующие процедуры.

- `Seek(f, N)` устанавливает указатель на элемент с номером `N`, где `N` имеет тип `Longint`.
- `Truncate(f)` уничтожает все элементы файла, начиная с места текущего указателя.

Можно пользоваться и двумя дополнительными функциями:

- `FileSize(f)` возвращает размер файла (количество элементов);
- `FilePos(f)` возвращает номер элемента, на который установлен текущий указатель.

Процедура `Seek` позволяет явно изменить значение текущего указателя, установив его на элемент файла с заданным номером (нумерация начинается с нуля). Это фактически прямой доступ к элементу с заданным номером. После выполнения процедуры `Seek` дальнейшие операции чтения или записи будут проводиться, начиная с установленной позиции указателя.

Примеры.

`Seek(f, 0)` → установка указателя на начало файла.

`Seek(f, FilePos(f)+1)` → пропуск одного элемента.

`Seek(f, FileSize(F))` → установка текущего указателя непосредственно за последним элементом файла.

Рассмотрим типовые алгоритмы для работы с файлами.

1. Общая структура фрагмента программы, предназначенной для чтения из файла с целью последующей обработки данных:

```
Assign(f, '…');
Reset(f);
while not eof(f) do
   begin
    Read(f, a);
    …
   end;
Close(f);
```

2. Общая структура фрагмента программы, предназначенной для записи в файл:

```
Assign(f, '…');
Rewrite(f);
 {Организовать цикл на количество записываемых элементов}
    begin
       {Получение данного для записи}
       Write(f, a);
    end;
Close(f);
```

3. Общая структура фрагмента программы, предназначенной для добавления в файл:

```
Reset(f);
Seek(f, FileSize(f));
 {Организовать цикл на количество записываемых элементов}
     begin
       {Получение данного для записи}
       Write(f, a);
     end;
Close(f);
```

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 3
АЛГОРИТМЫ РАБОТЫ С ТИПИЗИРОВАННЫМИ ФАЙЛАМИ

Содержание темы

- Алгоритмы работы с типизированными файлами:
  - обработка ошибок ввода-вывода,
  - создание телефонного справочника.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

НЕКОТОРЫЕ АЛГОРИТМЫ РАБОТЫ С ТИПИЗИРОВАННЫМИ ФАЙЛАМИ

Задание. Прочитав информацию из текстового файла, создать типизированный файл для следующей задачи.

Задача. В типизированном файле существует информация о студентах: фамилия, имя, отчество (30 символов), возраст (2 цифры). Обновить данные, увеличив возраст всех студентов на единицу.

Алгоритм. Нецелесообразной будем считать следующую последовательность действий: читаем очередную запись, корректируем возраст, оператором `Seek(f, FilePos(f)–1)` перемещаемся на один элемент назад и записываем откорректированную запись. Ведь может случиться так, что возникнет непредвиденное аварийное завершение программы, и наш файл будет фактически испорчен. Поэтому на основе исходного файла создадим новый файл с откорректированной информацией. Если все хорошо закончится, уничтожим исходный файл, и новый переименуют на исходный файл.

```
program Correct;
type
     Zap = record
            FIO : String[30];
            Age : 0..99;
           end;
var
```

```
      f, fRes : file of Zap;
      Z       : Zap;
begin
 Assign(f,    'C:\1_курс\Baza.dat' );
 Assign(fRes, 'C:\1_курс\Baza1.dat');
 Reset(f);   Rewrite(fRes);
 while not eof(f) do
      begin
       Read(f, Z);
       Inc(Z.Age);
       Write(fRes, Z);
      end;
 Close(f);        Close(fRes);
 Writeln('Ok');   Readln;
 Assign(f, 'C:\1_курс\Baza.dat');
 Erase(f);
 Writeln('Ok');   Readln;
 Assign(f, 'C:\1_курс\Baza1.dat');
 Rename(f, 'C:\1_курс\Baza.dat');
 Close(f);
 Writeln('Ok');
 Readln;
end.
```

Обработка ошибок ввода-вывода

Компилятор языка Pascal позволяет генерировать исполняемый код в двух режимах: с проверкой корректности ввода-вывода и без нее. По умолчанию включена директива компиляции режима проверки {$I +} ({$I-} – режим проверки отключен).

При включенном режиме проверки любая ошибка ввода-вывода будет фатальной: программа прервется и выдаст номер ошибки. Возможные номера ошибок находятся в диапазоне от 2 до 200. Расшифровка кодов приводится в специальных таблицах и в строке – состояния завершения программы.

Если отключить режим проверки, тогда при возникновении ошибки программа уже не будет прерываться, а продолжит работу со следующего оператора. При этом код ошибки сохранится в предопределенной системной переменной `InOutRes`, но результат операции ввода-вывода, которая вызвала ошибку, будет неопределенным.

Однако при опросе кода завершения операции обмена лучше пользоваться специальной функцией Pascal `IOResult`. При успешном

выполнении операций ввода-вывода обращение к IOResult дает 0, а ненулевое результат свидетельствует об ошибке.

Если программист захочет предусмотреть личную реакцию на ошибочные ситуации, тогда можно опросить функцию IOResult, которая вернет значение типа Integer – код (статус) последней выполненной операции ввода-вывода. Вызов функции IOResult очищает внутренний флаг ошибок (т. е. сбрасывает его в 0), и поэтому еще один вызов функции к одной и той же операции ввода-вывода даст некорректный результат.

Использование этой функции возможно только тогда, когда отключена стандартная проверка операций ввода-вывода: {$I-}. Рассмотрим следующий пример.

```
var
     Code : Integer; f : file;
      …
Assign(f, 'd:\myfile. dat');
{$I–}              {отключаем автоматический контроль,
                    иначе система остановит работу
                    при возникновении ошибки}
Reset(f);          {открыть существующий файл}
Code := IOResult;  {получили код результатат предыдущей
                    операции}
{$I+}              {включаем автоматический контроль}
if Code<>0 then
  begin
       Write('ошибка при открытии файла: ');
       case  Code   of
          2 : Write('файл не найден ');
          3 : Write('путь не найден ');
          4 : Write('слишком много открытых файлов');
          5 : Write('доступ к файлу запрещен');
          6 : Write('испорчена файловая переменная');
         12 : Write('некорректный код доступа к файлу');
        102 : Write('файлу не дано имя')
          else
       end; {case}
        Halt;
    end;
```

Приведенный выше фрагмент кода можно использовать при написании специальной функции для анализа любой операции ввода-вывода, и в дальнейшем операции ввода-вывода проводить с обязательной

диагностикой ошибок. Рассмотрим фрагмент программы проверки наличия файла с данными.

Пример.

```
Assign(f, 'NoFile.dat');
{$I-}   Reset(f);                {попытка открыть файл}
{$I+}
if IOResult <> 0 then
   Writeln('файл не найден или не читается')
   else
       begin
        Read(f, …);              {можно работать}
           …
        Close(f)
       end;
```

Аналогично можно построить функцию анализа существования файла:

```
function  FileExists
         (FName : String; var Cod : Integer) : Boolean;
var
      f : file;                 {тип файла не существенен}
begin
 Assign(f, FName);
 {$I-}
  Reset(f);
 {$I+}
 Cod := IOResult;
 if Cod = 0 then
   begin
    FileExists := true;
    Close(f);
   end
 else  FileExists := false
end;
```

Создание телефонного справочника

Рассмотрим типовую задачу по работе с файлами. Напишем небольшую базу данных на примере работы с телефонным справочником, содержащим сведения: фамилия, имя, отчество абонента (60 символов) и номер телефона (6 символов).

Опишем модуль `unit`, который объединит следующие описания и процедуры.

```
unit BookPhone;
interface

type
    RecBook = record
       FIO   : String [60];
       Phone : String [6];
    end;
var
     BookFile : file of RecBook;
     Work     : RecBook;

procedure Output_Rec;
                              {вывод на экран одной записи}
procedure Output_All_Rec;
                  {вывод по одной записи всего справочника}
procedure Add_Rec;
             {добавление записи в файл в текущую позицию}
procedure Update_Rec(Nam : Longint);
              {замена записи в файле с номером Nam}

implementation

procedure Output_Rec;
 begin
  Read(BookFile, Work);
  with Work do
       Writeln('ФИО :  ', FIO ,' – тел.– :', Phone);
 end;

 procedure Output_All_Rec;
  begin
   Seek(BookFile, 0);
   Writeln('**телефонный справочник**');
   while not EOF(BookFile) do
        Output_Rec;
  end;

 procedure Add_Rec;
  begin
   with Work do
      begin
```

```
       Write('Введи фамилию ');
       Readln(FIO);
       Write('Введи телефон ');
       Readln(Phone);
      end;    {with}
   Write(BookFile, Work);
  end;

 procedure UpDate_Rec;
                  {нумерация в файле начинается с нуля}
  begin
   Seek(BookFile, Nam);
   Writeln('—Новое значение--');
   Add_Rec;
  end;
end.
```

Алгоритм главной программы.

1. Открыть доступ к файлу (`Assign, Rewrite/Reset`).
2. Создать файл (в цикле; `Add_Rec`).
3. Распечатать (`Output_All_Rec`).
4. Проверить, будет ли корректировка (Y/N)?
5. Для корректировки ввести номер заменяемой записи `UpDate_Rec(N)`.
6. Повторять п. 5, пока не будет “отказ от корректировки“.
7. Распечатать (`Output_All_Rec`).
8. Закрыть доступ к файлу (`Close`).
9. Если надо (Y/N), то уничтожить файл.

```
program Example;
uses BookPhone;
var
        ind : Byte;
        ch  : Char;
begin                    {собственно программа}
 Assign(BookFile, 'Phone.dat');
 Rewrite(BookFile);
 Writeln('Создание 5 записей файла');
 for  Ind := 1   to   5   do
     Add_Rec;                        {Создание }
 Write(‘ Создание 5 записей закончено ',
       ‘нажмите любую клавишу');
 Readln;
```

```
 Output_All_Rec;          {вывод всех записей на экран}
 Write('Введите номер изменяемой записи (1 ≤ n ≤ 5)');
 Readln(Ind);
 UpDate_Rec(Ind-1);  {номер 1 соответствует второй записи}
 Write('Для продолжения нажмите любую клавишу');
 Readln;
 Output_All_Rec;
 Close(BookFile);
 Write('Удалить файл? (Y/N)');
 Readln(Ch);
 case ch of
     'Y', 'y', 'Y', 'y' : Erase(BookFile);
 end;
end.
```

Программа `BookFile` может быть дополнена процедурой `Add_Rec_to_end`, которая присоединяет запись в конец файла, и процедурой `Find_FIO` для вывода на экран абонентов, находящихся в справочнике, по заданной фамилии. Приведем процедуру `Add_Rec_to_end`:

```
 procedure Add_Rec_to_end;
  begin
   Seek(BookFile, FileSize(BookFile));
   Add_Rec;
  end;
```

Если записи располагаются не в отсортированном по фамилиям порядке, поиск необходимо осуществлять последовательно по всему файлу. Это требует значительных затрат машинного времени и не оптимально. Нужно иметь программу для упорядочения записей по алфавиту.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 4
РАБОТА С ТИПИЗИРОВАННЫМИ ФАЙЛАМИ

Содержание темы

- Слияние двух отсортированных последовательностей:
  - интуитивный алгоритм,
  - первое улучшение алгоритма,
  - второе улучшение алгоритма.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

Слияние двух отсортированных последовательностей

Тривиальный алгоритм слияния, когда в конец одной последовательности дописывается вторая, а потом происходит ее сортировка, отбросим как нецелесообразный, так как при таком подходе требуется много неоправданных затрат на сортировку. Рассмотрим следующую задачу.

Задача. Пусть некий типизированный файл '`f1.dat`' содержит первую последовательность элементов, отсортированных по возрастанию, а '`f2.dat`' – другую. Получить '`f.dat`' – объединенную последовательность с ненарушенным порядком возрастания, причем файлы разрешается читать только один раз.

Интуитивный алгоритм

Пусть

A={2, 4, 6, 8, 17, 20, 22, 30, 80, 82, 84, 86, 90}
B={1, 3, 4, 9, 15, 28}

Будем читать по одному элементу из каждой последовательности: a=2; b=1. Меньший из них пишется в C, и если это a, тогда читается новый элемент из A в a; иначе – из B в b. Так делаем до тех пор, пока одна из двух последовательностей не иссякнет.

На нашем примере первой закончится последовательность B. Тогда элементы другой последовательности, которые остались, просто дописываем в C.

Первый этап:

C_1={1, 2, 3, 4, 4, 6, 8, 9, 15, 17, 20, 22, 28, 30}

Второй этап:

C=C_1∪{80, 82, 84, 86, 90}

Значит, программа должна анализировать, будет ли второй этап и какую последовательность еще нужно добавить к C (из A или из B).

Здесь сталкиваемся с проблемой, являющейся результатом специфики работы функции `Eof` в Turbo Pascal. Логическая функция `Eof(f)` возвращает значение `True` при чтении последнего элемента файла. Поэтому последний прочтенный элемент не будет анализироваться в цикле и придется этот анализ выполнять после выхода из цикла.

Задание. Этот алгоритм запрограммируйте самостоятельно как на примере файлов, так и на примере массивов.

Первое улучшение алгоритма

Есть первое улучшение этого алгоритма – добавление фиктивного элемента в каждую последовательность, а именно:

A′=A∪{$b_{\text{последний}}$}

B′=B∪{$a_{\text{последний}}$}

Поскольку $b_{\text{последний}}$ и $a_{\text{последний}}$ – наибольшие элементы своих последовательностей, то ситуация улучшится и нужно будет записать только элементы из исходной последовательности, которые остались необработанными. Получим следующую программу.

Программа 1.

```
program Var2;
type
     fil = file of Integer;
var
     f1, f2, f : fil;

procedure Sozd(nam1, nam2 : String);
 var
         f : fil;
         t : text;
         a : Integer;
 begin
  Assign(t, nam1);
  Assign(f, nam2);
  Reset(t);
  Rewrite(f);
  while not eof(t) do
```

```
     begin
      Read(t, a);
      Write(f, a);
     end;
 Close(f);
 Close(t);
end;
procedure Show(var f : fil; st : String);
 var
     a : Integer;
 begin
  Writeln(st);
  Seek(f, 0);
  while not eof(f) do
       begin
       Read(f, a);
       Write(a : 4);
       end;
 Writeln;
 end;
 procedure Union(var f1, f2, f : fil);
  var
      a, b : Integer;
  procedure Astatok(var f3, f : fil; a : Integer);
    begin
    Write(f, a);
    while not eof(f3) do
         begin
          Read(f3, a);
          Write(f, a);
         end;
    end;
  begin
   Seek(f1, filesize(f1)-1);
   Read(f1, a);
   Seek(f2, filesize(f2)-1);
   Read(f2, b);
   Write(f1, b);
   Write(f2, a);
   Seek(f1, 0);
   Read(f1, a);
   Seek(f2, 0);
```

```
    Read(f2, b);
    while not eof(f1) and not eof(f2) do
        begin
         if a>b then
          begin
            Write(f,  b);
            Read(f2, b)
          end
         else
          begin
            Write(f,  a);
            Read(f1, a)
          end;
        end;
   if eof(f1) then Astatok(f2, f, b)
              else Astatok(f1, f, a);
   Seek(f, Filesize (f)-1);
   Truncate(f);
   Seek(f1, Filesize (f1)-1);
   Truncate(f1);
   Seek(f2, Filesize (f2)-1);
   Truncate(f2);
   end;
begin
Sozd('f1.txt', 'f1.dat');
Sozd('f2.txt', 'f2.dat');
Assign(f1, 'f1.dat');
Assign(f2, 'f2.dat');
Assign(f,  'f.dat');
Reset(f1);
Reset(f2);
Rewrite(f);
Show(f1, '*****  1 file  ********');
Show(f2, '*****  2 file  ********');
Union(f1, f2, f);
Show(f,  '*****  3 file  ********');
Close(f1);
Close(f2);
Close(f3)
end.
```

Второе улучшение алгоритма

Есть еще одно улучшение предыдущего алгоритма – добавление двух фиктивных элементов в каждую последовательность, а именно:

$A' = A \cup \{b_{последний}\} \cup \{b_{последний}\}$

$B' = B \cup \{a_{последний}\} \cup \{a_{последний}\}$

Поскольку $b_{последний}$ и $a_{последний}$ есть наибольшие элементы своих последовательностей, то положение улучшится, и не надо будет анализировать ситуацию, в которой из исходных последовательностей остались необработанные элементы.

Рассмотрим это улучшение на следующих данных.

Первый файл:

1 3 4 9

Второй файл:

0 4 6 8 10 12 14 16 22

Результат:

0 1 3 4 4 6 8 9 10 12 14 16 22

Программа 2.

```
program Var3;
type
     Fil = file of Integer;
var
     f1, f2, f : fil;
procedure Sozd(nam1, nam2 : String);
var
         f : fil;
         t : text;
         a : Integer;
begin
 Assign(t, nam1);
 Assign(f, nam2);
 Reset(t);
 Rewrite(f);
 while not eof(t) do
      begin
       Read(t, a);
       Write(f, a);
      end;
 Close(f);
 Close(t);
end;
```

```
procedure Druk(var f : fil; st : String);
 var
     a : Integer;
 begin
 Writeln(st);
 Seek(f, 0);
 while not eof(f) do
      begin
       Read(f, a);
       Write(a : 4);
      end;
 Writeln;
 end;
procedure Union(var f1, f2, f : fil);
var
     a, b : Integer;
begin
 Seek(f1, filesize(f1)-1);
 Read(f1, a);
 Seek(f2, filesize(f2)-1);
 Read(f2, b);
 Write(f1, b, b);
 Write(f2, a, a);
 Seek(f1, 0);
 Seek(f2, 0);
 Read(f1, a);
 Read(f2, b);
 while not eof(f1) and not eof(f2) do
     begin
      if a=b then
        begin
         Write(f, a, b);
         Read(f2, b);
         Read(f1, a)
        end
       else
        if a>b then
         begin
          Write(f, b);
          Read(f2, b)
         end
        else
         begin
```

```
          Write(f, a);
          Read(f1, a)
         end;
       end;
       if a = b then Seek(f, filesize(f)-2)
                else Seek(f, filesize(f)-1);
       Truncate(f);
       Seek(f1, filesize(f1)-2);
       Truncate(f1);
       Seek(f2, filesize(f2)-2);
       Truncate(f2);
 end;
begin
 sozd('f2.txt', 'f1.dat');
 sozd('f1.txt', 'f2.dat');
 Assign(f1, 'f1.dat');
 Reset(f1);
 Assign(f2, 'f2.dat');
 Reset(f2);
 Assign(f, 'f.dat');
 Rewrite(f);
 Druk(f1, '*****  1 file  ********');
 Druk(f2, '*****  2 file  ********');
 Union(f1, f2, f);
 Druk(f,  '*****  3 file  ********');
 Close(f);
 Close(f1);
 Close(f2)
end.
```

Задание. В файле действительных чисел записана информация, которая составлена из трех частей, отсортированных по убыванию. Алгоритмом «слияния» получить файл отсортированных по убыванию чисел.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Каков главный результат для вас лично при изучении темы?

ТЕМА 5
ТЕКСТОВЫЕ ФАЙЛЫ. ФАЙЛЫ БЕЗ ТИПА

Содержание темы

- Текстовые файлы:
  - процедуры для работы с текстовыми файлами,
  - функции для работы с текстовыми файлами,
  - практика работы с текстовыми файлами.
- Файлы без типа.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ТЕКСТОВЫЕ ФАЙЛЫ

При записи в текстовый файл данных (чисел, строк, логических значений и т. д.) они превращаются в символьный вид.

При чтении из текстового файла данные автоматически преобразуются из текстового представления во внутреннее.

Текстовые файлы можно обрабатывать только последовательно.

Ввод и вывод нельзя выполнять одновременно для одного и того же текстового файла.

Представителем текстового файла в программе является переменная файлового типа, которая должна быть описана так:

```
var
    Texinf : text;
```

Чтобы наглядно показать различие в представлении информации для двух видов файлов – текстовых и файлов с типом, обратимся к записям с типом `String[11]`.

Представление строк в текстовом файле следующее:

Первая строка файла:	`МАМА#13#10`
Вторая строка файла:	`ПОШЛА#13#10`
Третья строка файла:	`В МАГАЗИН#13#10`

Общая длина фразы для текстового файла будет 25 байт (6 байт – первая строка, 7 – вторая, 11 – третья, 1 – символ конца файла).

Представление строк в типизированных файле с типом `String[11]`:

Первая запись:	`#4МАМА#0#0#0#0#0#0#0`
Вторая запись:	`#5ПОШЛА#0#0#0#0#0#0`
Третья запись:	`#9В МАГАЗИН#0#0`

Здесь на месте символа #0 может стоять любой другой.

Общая длина фразы для типизированных файлов будет в 36 байт.

Для выполнения работ над строками в режиме записи или чтения предпочтение отдается текстовым файлам. Чтобы создать текстовый файл, лучше всего пользоваться любым текстовым редактором.

Когда вы разрабатываете программу, которая требует ввода трех и более чисел, то рекомендуется создать вспомогательные текстовые файлы с тестовыми вариантами наборов данных, и не тратить время на ввод данных с клавиатуры на каждый прогон программы. Но, конечно же, возможны и другие способы.

Процедуры для работы с текстовыми файлами

Процедуры `Assign`, `Close`, `Flush`, функция `EoF` одинаковы для типизированных и текстовых файлов.

`Append(F)` – открывает существующий файл `F` для добавления.

`Rewrite(F)` – создает новый текстовый файл, к которому можно только добавлять строки. Если файл с таким именем уже существует на диске, то он уничтожается и создается новый текстовый файл.

`Reset(F)` – можно применять только к существующему файлу, после чего из этого файла можно читать только последовательно.

Если новый текстовый файл закрывается, к нему автоматически добавляется маркер конца файла.

Второе обращение к `Reset(F)` установит текущий указатель снова в начало файла `F`.

Чтение информации с файлов типа `Text` нами было подробно рассмотрено, когда рассматривался стандартный файл `Input`.

`Read([F,] V1[,V2,…,Vn])` – читает информацию из файла `F` (когда `F` присутствует, иначе – с клавиатуры) в заданные переменные.

Замечание. Здесь и далее в квадратных скобках пишется информация, которая может как присутствовать – тогда скобки надо опустить, так и отсутствовать вместе со скобками.

Процедура чтения зависит от типа текущей переменной.

В символьную переменную читается очередной символ из файла, включая символы-разделители (`CR ≡ #13`, `LF ≡ #10`, `^Z ≡ #26`).

При чтении в переменную арифметического типа пропускаются пробелы и символы табуляции, и считывается значение арифметической константы до появления пробела, символа табуляции, маркера конца строки или файла. (Символ-ограничитель не считывается.)

При чтении данных в строковую переменную передается или столько символов, какова ее длина в объявлении переменной, или до появления маркера конца строки или файла. (Маркер в строку не заносится.)

`Readln` – отличается от `Read` тем, что, после считывания данных в перечисленные переменные, пропускаются все символы, которые остались в данной строке, в том числе и маркер конца строки, и происходит переход на новую строку.

`Write([F,] V1 [,V2,…,Vn])` – записывает информацию в файл `F` (если он задан, иначе – на экран) из заданных переменных или выражений. Процедура записи зависит от типа данных в списке вывода. Для символьных, арифметических и строковых данных выводится их значение, а для `Boolean` – строка `true` или `false`. Используется и форматированный вывод.

`Writeln` – отличается от `Write` тем, что после вывода в файл всех данных записывается маркер конца строки.

Функции для работы с текстовыми файлами

`Eof[(F)] : Boolean` – возвращает `true`, если следующим за последним прочитанным символом является маркер конца файла `F`.

Если файловая переменная `F` отсутствует, тогда подразумевается стандартный файл ввода.

`Eoln[(F)] : Boolean` – возвращает `true`, если следующим за последним прочитанным символом является маркер конца строки (или `Eof`).

При отсутствии аргумента подразумевается стандартный файл ввода.

Если `Eof(F)` – `true`, то и `Eoln(F)` – `true`.

`SeekEoln[(F)] : Boolean` проводит поиск конца текущей строки.

Эта функция пропускает все символы-разделители (пробелы и символы табуляции) в строке и устанавливает текущий указатель файла `F` или на конце строки и тогда функция возвращает `true`, или на первом значащем символе и тогда функция возвращает `false`.

`SeekEof[(F)] : Boolean` возвращает `true`, когда указатель файла `F` находится на маркере конца файла.

Функция пропускает все символы-разделители и также символы концов строк (т. е. переходит со строки на строку в поисках или конца файла, или первого значащего символа).

Практика работы с текстовыми файлами

Далее рассмотрим некоторые фрагменты из программ.

Задача 1. Написать процедуру, которая выбирает режим добавления в существующий текстовый файл или создания файла, если набор данных не нашелся при открытии.

```
procedure OpenWrite(var  f: text; FileName : String);
 begin
  Assign(f, FileName);
  {$I-}
  Append(f);
  {$I+}              {попытка открыть файл для добавления}
  if IOResult <> 0 then
    Rewrite(f);
       {если файл не может быть открыт, тогда создать его}
  end;
```

Задача 2. Написать программу работы с текстовыми файлами данных произвольного формата, если файл содержит N столбцов чисел, но еще может содержать строки, которые являются комментариями.

Алгоритм. Пусть в файле находится следующая таблица:

X	Y	Z
Вариант 1		
1.0	125.25	2.56
Вариант 2		
0.5	–1.5	–0.75
...		

Из этой таблицы следует ввести данные, пропуская строки комментариев.

Тогда программа будет такой:

```
const
     N = 3;                        {пусть есть N столбцов}
var
     f : text;
     i : Byte;
     D : array[1..N] of Real;   {для ввода одной строки}
begin
 Assign(f, 'Myfile.dat');
 Reset(f);
```

```
 {$I-}
 while not SeekEof(f) do
          {просматриваем до конца файла, пропуская
                        пробелы и символы табуляции}

      begin
       Read(f, D[1]);              {попытка читать число}
       if IOResult = 0 then
            {если нашли первое число, то там и остальные}
         begin
           for  i := 2  to  N  do
                  Read(f, D[i]);
              …  {обработка чисел}
           for  i := 1  to  N  do
                  Write(D[i]:4);
           Writeln;
         end;
         Readln(f);       {переход на следующую строку}
       end;
 {$I+}
  Close(f);
end.
```

ФАЙЛЫ БЕЗ ТИПА

Файл без типа состоит из элементов одинакового размера, структура которых не имеет значения или неизвестна. Файлы без типа применяются для высокоскоростных программ обмена между оперативной и внешней памятью. Нетипизированный файл объявляется так:

```
var    f : file;
```

Для работы с такими файлами используют следующие процедуры и функции:

`Assign, Rewrite, Reset, Blockread, Blockwrite, Seek, Eof, FileSize, FilePos.`

Обмен с типизированным файлам осуществляется так: вводится-выводится значение данного в форме его внутреннего представления. Вспомним, что при обмене через текстовый файл идет преобразование последовательности символов во внутреннее представление данного при вводе и наоборот – при выводе.

Нетипизированный файл рассматривается в языке Pascal как совокупность байт-блоков.

Любой файл, который был подготовлен как файл `text` или файл с типом, можно открыть и начать работу с ним, как с нетипизированным набором данных.

Для таких файлов не нужно тратить время на преобразование данных или поиск управляющих последовательностей. Достаточно лишь считать содержимое файла в определенную память.

Для нетипизированных файлов самым важным параметром является длина блока в байтах, и этот параметр или присутствует в процедурах открытия файла, чтения, записи, или берется по умолчанию. Однако суммарный объем разового обмена не должен превышать 64 кб.

Рассмотрим подробнее перечисленные выше процедуры:

`Rewrite(F [, Size])` – создает новый файл `F`: если файл существует, то он уничтожается и создается новый. Второй параметр `Size` определяет размер блока в файле; если `Size` отсутствует, то считается размер блока в 128 байт.

`Reset(F [, Size])` – открывает существующий файл `F`. В файл можно записывать информацию и читать из него. Второй параметр – как в предыдущей процедуре.

`Blockread(f, Buf, count [,Result])` – читает из файла `F` count-записей (блоков) в буфер ввода `Buf`. Если есть четвертый параметр `Result`, тогда он получает значение реального числа прочитанных записей (блоков). Если читается последняя порция count-записей (блоков) из файла, тогда она может быть либо пустой, либо неполной. Если же четвертый параметр отсутствует и количество востребованных (`count`) и реально прочитанных записей (блоков) не совпадает, тогда возникает ошибка ввода-вывода.

Если четвертый параметр присутствует, тогда ненормальное завершение ввода-вывода фиксируется четвертым параметром.

`Blockwrite(f, Buf, count [, Result])` – записывает в файл `count`-записей из буфера ввода `Buf`. Если указан четвертый параметр `Result`, тогда он получает реальное число записанных записей (блоков). Если, например, на диске не хватает места, тогда количество записанных элементов может не совпадать с заданным в `count`.

Если четвертый параметр не указан и количество требуемых и реально выведенных записей не совпадает, тогда возникает ошибка ввода-вывода.

Пример. Рассмотрим использование файлов без типа для ускорения обмена данных между внешней и оперативной памятью.

```
program Pr_Best;
var
     A, B, C : array[1..2000] of Real;
                    {компилятор распределяет их в цепочку
                     один за другим (специфика Pascal)}
     f       : file;
     Rez     : Integer;
begin
      ...                {заполняем массивы}
 Assign(f, 'ABC.dat');
 Rewrite(f, Sizeof(A));
 Blockwrite(f, A, 3, Rez);
                       {или три раза: из A, из B, из C}
 if Rez = 3 then Writeln('Okey')
            else Writeln('Error');
 Close(f); …
 end.
```

Если нужно быстро сбросить информацию из файла в память, тогда возможно следующее использование файла без типа:

```
 … Reset(f, Sizeof(A));
 Blockread(f, A, 3, Rez);
 if Rez = 3 then Writeln('Okey')
            else Writeln('Error');
                        {дальше с массивами A, B, C
                         можно работать по отдельности}
 Close(f);…
```

Сравните описанный здесь вариант обмена данных между внешней и оперативной памятью с обменом при помощи типизированных файлов.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?

ТЕМА 6
СПЕЦИАЛЬНЫЕ СРЕДСТВА TURBO PASCAL. МОДУЛЬ SYSTEM

Содержание темы

- Указатели и динамические структуры данных.
  - создание динамических переменных,
  - операции,
  - доступ к переменной по указателю.
- Действия над динамическими переменными.
- Нетипизированные указатели.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

УКАЗАТЕЛИ И ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Создание динамических переменных

При выполнении программы каждая объявленная в ней переменная получает свой адрес в оперативной памяти. Программисту не нужно заботиться о механизме определения адресов, это делается автоматически при трансляции.

В языке Pascal существуют два способа распределения памяти для переменных: статический и динамический.

Всем переменным, объявленным в программе или в интерфейсной части модуля, выделяются в определенном месте оперативной памяти (сегменте данных) фиксированные участки оперативной памяти в соответствии с их типом. Это статическое распределение памяти.

При динамическом распределении памяти есть возможность создавать новые, не объявленные заранее переменные и размещать их на свободные участки в динамической области оперативной памяти. Это достигается путем использования указателей.

Указатель – это элемент данных, представляющий собой ссылку (адрес) на начало определенного блока оперативной памяти (ячейку), где может содержаться значение данных заявленного типа. Сам указатель занимает 4 байта.

В IBM-совместимых компьютерах память условно разделена на так называемые *сегменты*. Адрес каждого байта составляется из адреса сегмента и адреса смещения относительно начала сегмента.

Компилятор Pascal формирует *сегмент кода*, в котором хранится программа в виде машинных команд, *сегмент данных*, в котором выделяется память под глобальные переменные программы и *сегмент стека*, предназначенный для размещения локальных переменных во время выполнения программы.

Адрес хранится в виде адреса сегмента, уменьшенного в 16 раз (два байта), и адреса смещения (два байта) относительно начала сегмента.

Переменные, которые размещаются в динамической области оперативной памяти (в куче – она же Heap) при помощи указателя, называются *динамическими переменными.*

Указатель может принимать значения, равные всем адресам оперативной памяти, в которые возможна запись данных. Указатель может быть равен стандартному значению `nil` (пусто), тогда говорят, что соответствующая переменная в оперативной памяти отсутствует, или же, что указатель не получил значение адреса в оперативной памяти.

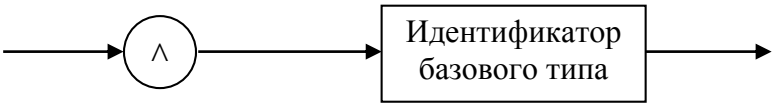
Указатель объявляется при помощи специального символа каре «^», за которым записывается идентификатор типа динамической переменной в соответствии со следующими форматами:

```
type имя_ссылочного типа = тип;
var имя_переменной: имя_ссылочного типа;
```

или

```
var имя_переменной:  ^тип;
```

Ссылочный тип:



К статическим переменным обращение осуществляется по имени, к динамическим – по адресу.

Указатели нужно проинициализировать – присвоить значение.

Получить адрес статического данного можно либо с помощью функции

`Addr(x) : Pointer`,

где `x` – любая переменная, имя процедуры или функции, либо используя унарную операцию @.

Пример 1.

```
type
        NameStr  = String [30];
        ArrayInt = array [1 .. 20] of Integer;
var
        RealP    : ^Real;
                     {указатель на значение типа Real,
                      требующего 6 байт памяти}
        NameStrP : ^NameStr;
        ArrIntP  : ^ArrayInt;
```

Turbo Pascal допускает и такое объявление:

```
type
         PtrType  = ^BaseType;
         BaseType = record
                     x, y : Real;
                     next : PtrType;
                    end;
```

Для других определений здесь была бы ошибка, так как при описании типа `PtrType` идет использование типа `BaseType`, который еще не описан, но для типа «указатель» это возможно.

К статическим переменным обращение осуществляется по имени, к динамическим – по адресу.

Указатели нужно проинициализировать – присвоить значение.

Получить адрес статического данного можно либо с помощью функции

`Addr(x) : Pointer`,

где `x` – любая переменная, имя процедуры или функции,
либо используя унарную операцию @.

Пример 2.

```
var
       i     : Integer;
       A     : array [1..10] of Integer;
       P1    : ^Integer;
      Begin
            …
      i := 7;
      P1 := Addr(i);
      {указатель на целое получил адрес переменной i}
```

```
  …
P1 := @A[i];
      {указатель на i-е целое в массиве A}
      {можно P1 := Addr(A[i])            };
…
End.
```

Операции

Над значениями ссылочных типов допускаются две операции сравнения: на равенство и неравенство:

```
if P1 <> nil then …
 {проверка, получил ли указатель адрес в памяти}
if P1 = nil then …
 {проверка, ссылается ли этот указатель на nil}
```

Сравнение указателей между собой (`P1 = P2`) производить не рекомендуется ввиду неоднозначного хранения значения адреса в виде адреса сегмента и адреса смещения.

При сравнении указателей на «<>» или «=» типы, с которыми связаны указатели, должны быть совместимыми. При присваивании значения одного указателя другому типы, с которыми связаны указатели, должны быть совместимыми по присваиванию.

```
var
     P, P1 : ^Integer;
     Pr    : ^Real;
```

то

```
     P1    := nil;  {можно}
     P     := P1;   {можно}
     Pr    := P;    {так нельзя – не совпадает
                        базовый тип}
```

Доступ к переменной по указателю

Обратимся снова к ситуации

```
P1:=@i;
```

Чтобы обратиться к переменной `i`, есть две возможности:

- использовать идентификатор `i`;
- использовать адрес этой переменной, который находится в `P1`.

В последнем случае – при косвенном доступе к переменной через указатель – используют конструкцию, которая называется *разыменование.*

Разыменование означает: для того чтобы по указателю на переменную получить доступ к самой переменной, надо после переменной-указателя поставить знак «^».

После выполнения оператора `P1 := @i` следующие операторы полностью эквивалентны:

`i := i+2` и `P1^ := P1^ + 2.`

Поскольку `P1` – указатель на тип `Integer`, то `P1^` – переменная целого типа. А `P1^` нужно понимать так: это переменная, на которую ссылается указатель `P1`.

В результате использования «указателя на указатель» возможно многократное разыменование.

Пример 3.

```
type   P  =  ^Integer;
var
       P1  : ^Integer;
       PP1 : ^P;
       i   : Integer;
   …
PP1 := @P1;   …   P1 := @i;   …   i := 2;
Write(PP1^^);                    {напечатается 2}.
```

Заметим, что следующая ситуация потенциально опасна:

```
P1 := nil; P1^ := 2;
```

потому что константа `nil` указывает, что не существует никакой динамической переменной.

ДЕЙСТВИЯ НАД ДИНАМИЧЕСКИМИ ПЕРЕМЕННЫМИ

Основные действия над динамическими переменными – это их создание и уничтожение.

Процедура

`New(P)`

предназначена для создания динамической переменной, т. е. для отведения памяти в куче, которую будет использовать динамическая переменная.

При этом динамической переменной отводится блок памяти, который соответствует размеру типа, на который ссылается указатель P (или чуть больше, так как должно быть кратное 8 байтам число), а указателю P выделяется адрес начала блока памяти (адрес динамической переменной).

Если в ходе вычислительного процесса динамическая переменная становится ненужной, ее следует уничтожить процедурой

```
Dispose(P).
```

Эта процедура освобождает память, занятую динамической переменной, и делает значение ее указателя P неопределенным.

При использовании процедуры New надо иметь в виду, что возможен недостаток (исчерпание) памяти для размещения динамической переменной.

В этой ситуации указатель P не получает значения, а программа продолжает выполнение, и никаких сообщений не выдает.

Значит, надо контролировать текущее состояние динамической памяти перед каждым обращением к New.

Функция

```
MaxAvail : Longint
```

возвращает размер (в байтах) наибольшего непрерывного блока в динамической области оперативной памяти, где может быть размещена динамическая переменная.

Функция

```
MemAvail : Longint
```

возвращает суммарный размер (в байтах) свободной области динамической памяти.

Пример 4.

```
type
       Person  = record … end;
       PPerson = ^Person;
var P : PPerson;
     …
 if MaxAvail >=Sizeof (Person)
 then New(P);
 {можно разместить и так: P:=New(PPerson)};
     …
  Dispose (P);
```

В приведенном фрагменте программы оператор New(P) выделяет память под указатель P, который ссылается на переменную типа Person, а адрес начала выделенной памяти хранится в P. Отметим, что к New можно обращаться как к функции: P:=New(PPerson).

Оператор `Dispose (P)` освобождает память, которая связана с указателем `P`, и уничтожает динамическую переменную, и значение указателя `P` становится неопределенным.

Существуют и другие подпрограммы для работы с динамическими переменными.

Процедура

```
GetMem(P, size)
```

создает новую динамическую переменную размером `size` байт, устанавливая значение указателя `P` на начало выделенной ей динамической области (`size` <= 65521).

Процедура

```
FreeMem(P, size)
```

уничтожает динамическую переменную, освобождая `size` байт. После выполнения процедуры значение `P` становится неопределенным.

Процедура

```
Mark(P)
```

запоминает адрес начала распределения данных в динамической памяти, а процедура

```
Release(P)
```

освобождает данные в динамической памяти, начиная с адреса `P`.

НЕТИПИЗИРОВАННЫЕ УКАЗАТЕЛИ

Стандартный ссылочный тип `Pointer` позволяет не конкретизировать свой базовый тип и совместим со всеми ссылочными типами.

Например, описания

```
var
    {следующие указатели могут указывать на на:}
        P0 : ^Word;      {переменную типа Word    }
        P1 : ^Integer;   {переменную типа Integer }
        P2 : ^Real;      {переменную типа Real    }
        P3 : Pointer;    {на любую переменную     }
```

допускают присваивания значений вида `P3:= P1` или `P3:= P2`.

Значения типа `Pointer` называются нетипизированными (бестиповыми) указателями.

Что адресуется таким указателем – известно лишь программисту.

Если нужно «посмотреть» на динамическую переменную, на которую ссылается P3, как на определенную типизированную, используют механизм приведения типа:

```
Integer (P3^) : = P1^;
Real (P3^)    : = P2^;
Word (P3^)    : = P0^;
```

Указатели дают большие возможности для довольно гибкой работы с памятью, включая адресную арифметику.

Turbo Pascal допускает описание типизированных констант ссылочного типа. Начальным значением таких констант может быть только `nil`. Например:

```
type     Ar   = array [0..5] of Char;
         Ptr  = ^Ar;
const    PP  : Ptr = nil;
var
     A, B  : Ptr;
    {A, B – указатели на переменную - массив символов}
…
```

Для динамических переменных `A^, B^` допустимы все те же операции, что и над обычными переменными-массивами.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?

ТЕМА 7
ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ С УКАЗАТЕЛЯМИ

Содержание темы

- Программирование алгоритмов с использованием указателей.
- Разные варианты размещения матрицы в Heap.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ С ИСПОЛЬЗОВАНИЕМ УКАЗАТЕЛЕЙ

При разработке программ, выполняющих одинаковые преобразования над большим количеством однотипных данных, используют массивы.

В программе при объявлении каждого статического массива всегда необходимо указывать его размеры.

Это требование делает программы зависимыми от конкретных размеров массивов, что снижает потребительские свойства программ. В таких программах используются динамические переменные-массивы.

Рассмотрим различные подходы при работе с массивами на примере следующих задач.

Задача 1. Выполнить транспонирование матрицы A_{nn} на месте ее размещения.

Задача 2. Упорядочить строки матрицы так, чтобы элементы k-го столбца (k – задается при выполнении программы) образовывали невозрастающую последовательность.

Решение этих задач зависит от того, на каком этапе задаются размеры матрицы и какие это размеры.

Возможны, например, такие ситуации.

1. Известны размеры n, m матрицы A_{nn} (задача 1) и A_{nm} (задача 2), такие, что матрицы поместятся в сегменте данных (в статической памяти).

Эта ситуация не требует распределения матриц в `Heap`, и их можно объявить, например, так:

1)
```
const
       n=50; {задача 1}
var
       A : array [1..n, 1..n] of Integer;
```
2)
```
const
       n=50; m=30;{задача 2}
var
       A : array [1..n, 1..m] of Integer;
```

Заметим, что при этом действия компилятора следующие: в памяти матрица распределится цепочкой по строкам:

$$A: a_{11}, a_{12}, ..., a_{1n}, a_{21}, a_{22}, ..., a_{2n}, ..., a_{n1}, a_{n2}, ..., a_{nn}.$$

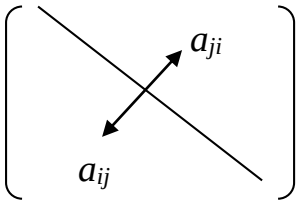
Используем это свойство потом в одном из решений.

2. Если известны размеры *n*, *m* матрицы A_{nn} (задача 1) и A_{nm} (задача 2), такие, что Sizeof($A_{n\times n}$) ≥ 64 кб и Sizeof($A_{n\times m}$) ≥ 64 кб, тогда матрицы необходимо распределить в динамической области частями.

Если размеры *n*, *m* матрицы A_{nn} (задача 1) и A_{nm} (задача 2) задаются во время выполнения программы, тогда матрицы необходимо распределить в динамической области.

Сначала рассмотрим более подробно решение задачи 1.

Схема транспонирования матрицы на своем месте следующая



Отсюда получается такая последовательность действий.

1. Размеры матрицы будем задавать при выполнении программы, учитывая, что Sizeof($A_{n\times m}$) < 64 кб.

2. Матрицу *A* рассматривать как одномерный массив *V*, образованный строками одна за другой.

3. Местоположение элемента a_{ij}, $i=\overline{1,n}$, $j=\overline{1,n}$, будем вычислять относительно начала массива по формуле $k := (i-1)*n+j$.

4. Соответственно местоположение элемента a_{ji}, $i=\overline{1,n}$, $j=\overline{1,n}$, будем вычислять относительно начала массива по формуле $l := (j-1)*n+i$.

5. В виду того, что V содержит элементы матрицы A, для обмена элементов a_{ij} и a_{ji} нужно выполнить обмен элементов V_k и V_l.

Алгоритм.

1. Объявить матрицу как одномерный массив.
2. Задать значения ее элементов.
3. Напечатать исходную матрицу.
4. Транспонировать матрицу.
5. Напечатать полученную матрицу.

```
program Transp_ukaz;        {Транспонирование}
uses CRT;
type
       TVect  = array [1..1] of Real;
       TPVect = ^TVect;
var
       MasV : TPVect;
       i, n : Byte;
{$R-}           {отключили контроль выхода за пределы
             индексов объявленного массива         }

  procedure Init(var MasV : TPVect; n : Byte);
  var
      i : Integer;
  begin
   for i := 1 to n * n do
            MasV^[i]:= Random*100-50;
  end;

 procedure Show(MasV : TPVect; n : Byte);
  var
     i, j : Integer;
  begin
   Writeln;
   Writeln('MATP');
   for i := 1 to n do
       begin
       for j := 1 to n do
           Write (MasV^[(i - 1) * n + j]:6:0);
       Writeln;
       end;
  end;
```

```
 procedure Transp(MasV:TPVect; n:Byte);
  var
       i, j, k, l : Integer;
       r          : Real;
      begin
   for i := 1 to n do
       for j := i + 1 to n do
           begin
 {1}         k       := (i - 1) * n + j;
 {2}         l       := (j - 1) * n + i;
             r       := MasV^[k];
             MasV^[k]:=MasV^[l];
             MasV^[l]:=r;
           end;
  end;
begin
  ClrScr;
  Writeln('n – ?');
  Readln(n);
  if MaxAvail < Sizeof(Real)* Sqr(n) then Halt;
  GetMem(MasV, Sizeof(Real)* Sqr(n));
  Init(MasV, n);
  Show(MasV, n);
  Transp(MasV, n);
  Show(MasV, n);
  Readln;
  FreeMem(MasV, Sizeof(Real) * Sqr(n));
end.
```

Целесообразно было бы определить функцию

```
       function Ord(i, j : Integer) : Integer;
        begin
         Ord := (i - 1) * n + j;
        end;
```

и операторы {1}, {2} заменить вызовом функции `Ord` от соответствующих аргументов: `Ord(i,j)` и `Ord(j,i)`.

Далее рассмотрим решение задачи 2.

Алгоритм.

1. Объявить матрицу.
2. Задать значения ее элементов.
3. Напечатать исходную матрицу.
4. Упорядочить по условию задачи.
5. Напечатать полученную матрицу.

Будем использовать *сортировку выбором*. Находим среди элементов $\{a_{1k}, ..., a_{nk}\}$ наибольший, например a_{pk}. Переставляем *p*-ю строку с первой. Среди оставшихся элементов $\{a_{2k}, ..., a_{nk}\}$, опять находим наибольший. Переставляем строку, в которой он находится, со 2-й строкой.

Продолжаем так и далее, пока не будут сравниваться $a_{n-1,k}$ и a_{nk}. На этом процесс остановится.

Получилась следующая последовательность действий.

1. Образуем цикл по $i = \overline{1, n-1}$.
   2. Устанавливаем $p := i$.
   3. Образуем цикл по $j = \overline{i+1, n}$.
      4. Если $a_{pk} < a_{jk}$, тогда $p := j$.
   5. Конец цикла по j.
   6. Переставляем *i*-ю и *p*-ю строки.
7. Конец цикла по *i*.

Чтобы написать программную реализацию этого алгоритма, нужно определиться, как будет размещена матрица в памяти. А это зависит не только от ее размера, но и от того, когда (на какой момент) задаются *n* и *m*.

Пусть *n* и *m* – константы и Sizeof(A_{nm}) < 64 кб. Матрицу распределяем в сегменте данных.

```
program Primer_statika;
const
       n=10;     m=10;     k=5;
type
       TItem = Integer;
       TMatr = array[1..n,1..m] of  TItem;
var
       A : TMatr;

procedure Show(A:TMatr; n, m: Byte; Str:String);
{Процедура печати матрицы}
var
     i, j : Byte;
begin
 Writeln(Str:40);
 for I := 1 to n do
     begin
```

```
      Writeln;
      for j := 1 to m do
         Write (A[i, j] :5);
     end;
 Writeln;
end;

procedure Init(var A:TMatr; n, m: Byte);
{Процедура инициализации матрицы}
 var
     i, j : Byte;
 begin
  for i := 1 to n do
     for j := 1 to m do
         A[i, j] := Random(101);
 end;
procedure Sort(var A:TMatr; n, m: Byte);
{Процедура сортировки}
  procedure Swap(i, p: Byte);
   {Процедура обмена строк матрицы}
   var
        j : Byte;
        r : TItem;
   begin
        for j:=1 to m do
            begin
             r       := A[i, j];
             A[i, j] := A[p, j];
             A[p, j] := r
            end;
   end;
var
        i, j, p : Byte;
begin
 for  i := 1 to n-1 do
    begin
     p := i;
     for j := i+1 to  n  do
         if A[p, k] < A[j, k]  then  p := j;
     if p<>i then Swap(i, p);
    end;
 end;
```

```
begin
 Randomize;
 Init(A, n, m);
 Show(A, n, m, 'A – исходная');
 Sort(A, n, m);
 Show(A, n, m, 'A – упорядоченная');
end.
```

При решении задачи 2 рассмотрим ситуацию когда *m*, *n* задаются во время выполнения программы. В этом случае можно использовать разные варианты размещения матрицы в Heap. Рассмотрим их подробнее.

Разные варианты размещения матрицы в Heap

Вариант 1

Рассмотрим ситуацию, когда количество столбцов матрицы *m* – константа, а количество строк становится известным во время выполнения программы (вводится).

Если еще раз проанализировать действия компилятора, то можно заметить, что при выборке элемента a_{ij} матрицы A_{nm} формируется индекс $k := (i-1)*m+j$, по которому в последовательности (по строкам) элементов матрицы и разыскивается нужный элемент.

В следующей программе используем это обстоятельство.

Зададим тип матрицы как бы из одной строки.

При распределении в `Heap` запросим место на *n* строк процедурой `GetMem`.

```
program Primer1;
const
         m = 10;      k = 5;
 type
       TItem  = Integer;
       TMatr  = array [1..1, 1..m] of TItem;      {*}
       TPmatr = ^TMatr;                           {*}
{$R-}
 var
       A : TPMatr;                                {*}
       n : Byte;
procedure Show(A:TPMatr; n, m: Byte; Str:String); {*}
var
     i, j : Byte;
```

```
 begin
  Writeln(Str:40);
  for i := 1 to n do
     begin
      Writeln;
      for j := 1 to m do
         Write (A^[i, j] :5);                    {*}
     end;
  Writeln;
 end;
procedure Init(A:TPMatr; n, m: Byte);            {*}
var
     i, j : Byte;
 begin
  for i := 1 to n do
     for j := 1 to m do
         A^[i, j] := Random(101);                {*}
 end;
procedure Sort(A:TPMatr; n, m: Byte);            {*}
  procedure Swap(i, p: Byte);
   var
        j : Byte;
        r : TItem;
    begin
        for j:=1 to m do
            begin
                r       := A^[i, j];             {*}
                A^[i, j]:= A^[p, j];             {*}
                A^[p, j]:=r                      {*}
            end;
    end;
 var
        i, j, p : Byte;
begin
 for i := 1 to n-1 do
    begin
     p := i;
     for j := i + 1 to n do
         if A^[p, k] < A^[j, k] then p := j;     {*}
         if p <> i then Swap(i, p);
    end;
```

```
 end;
begin
 ClrScr;
 Write ('задай n=');
 Readln (n);
 GetMem (A, n * Sizeof(TMatr));                    {*}
 Randomize;
 Init(A, n, m);
 Show(A, n, m, 'A – исходная');
 Sort(A, n, m);
 Show(A, n, m, 'A – упорядоченная');
 FreeMem (A, n * Sizeof(TMatr));                   {*}
end.
```

Замечание. В данном и последующем вариантах программ существенно следующее: размеры матриц такие, что размер матрицы Sizeof(A_{nm}) < 64 кб.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМЫ 8–9
АЛГОРИТМЫ С УКАЗАТЕЛЯМИ

Содержание тем

- Разные варианты размещения матрицы в Heap.
- Проблема потерянных ссылок.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

Вариант 2

Рассмотрим ситуацию, когда количество столбцов *m* неизвестно и становится известным во время выполнения программы (вводится), а количество строк - константа.

В этом случае нельзя объявить матрицу двумерным массивом.

Разместим матрицу в одномерном массиве (как при решении задачи 1) и сами будем по индексам (`i, j`) отыскивать нужный элемент.

Для этого используем функцию `function Ord(i,j)`.

Мы существенно будем использовать текст предыдущей программы, все обращения к `A^[i,j]` заменив на `A^[Ord(i, j)]`. Измененные операторы обозначим `{!}`.

```
program Primer2;
 const
       k : Integer =  5;                              {!}
       n           = 10;                              {!}
                     {k – должна быть типизированной
                      иначе компиляция дает ошибку }
 type
         TItem = Integer;
{$R-}
         TMas  = array [1 .. n*1] of TItem;           {!}
         TPmas = ^TMas;
 var
      A : TPMas;
      m : Integer;                                    {!}
function Ord (i, j : Byte) : Word;                    {!}
```

```
 begin
  Ord := (I - 1) * m + j;
 end;
procedure Show(A:TPMas; n, m: Byte; Str:String);   {!}
 var
     i, j : Byte;
 begin
  Writeln(Str:40);
  for i := 1 to n do
     begin
      Writeln;
      for j := 1 to m do
         Write (A^[Ord(i,j)] :5);                  {!}
     end;
  Writeln;
 end;
procedure Init(A: TPMas; n, m: Byte);              {!}
 var
     i, j : Byte;
 begin
  for i := 1 to n do
     for j := 1 to m do
         A^[Ord(i,j)] := Random (101);             {!}
 end;
procedure Sort(A:TPMas; n, m: Byte);               {!}
var
      i, j, p : Byte;
  procedure Swap(i, p: Byte);
   var
        j : Byte;
        r : TItem;
   begin
        for j:=1 to m do
            begin
                r            := A^[Ord(i,j)];      {!}
                A^[Ord(i,j)] := A^[Ord(p,j)];      {!}
                A^[Ord(p,j)] := r                  {!}
            end;
   end;
begin
 for i := 1 to n - 1 do
```

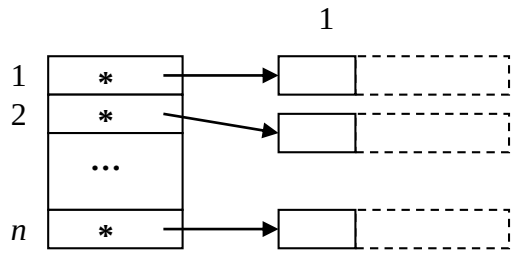
```
      begin
      p := i;
      for j := i + 1 to n do
        if A^[Ord(p,k)] < A^[Ord(j,k)] then p := j; {!}
          if p <> i then Swap(i, p);
      end;
 end;
begin
      Write('input m=');
      Readln(m);
      if MemAvail < m * Sizeof(TMas) then
         begin
         Writeln ('Heap – ?!');
         Halt;
         end;
      GetMem (A, m * Sizeof(TMas));                  {!}
      Randomize;
      Init(A, n, m);
      Show(A, n, m, 'A – исходная');
      Sort(A, n, m);
      Show(A, n, m, 'A – упорядоченная');
      FreeMem (A, m * Sizeof(TMas));                 {!}
end.
```

Вариант 3

Рассмотрим ситуацию, когда количество столбцов m неизвестно и становится известным во время выполнения программы (вводится), а количество строк – константа, размеры всей матрицы Sizeof(A_{nm}) > 64 кб, однако строку можно распределить в сегменте.

Количество элементов в строке сразу неизвестно, и будем считать, что в строке есть один элемент.

Заведем массив указателей на каждый из строк матрицы по следующей схеме.



На схеме символом * отмечены данные, значениями которых являются указатели на строки матрицы.

Тогда, когда станет известно количество элементов в строке, нужно будет с каждым указателем на строку `A[i]` связать процедурой `GetMem` в динамической памяти выделенное под строку место.

Обращение к элементу a_{ij} будет реализовываться так:

$$a_{ij} \Rightarrow A[i]^[j].$$

В такой интерпретации матрицы при сортировке строк по фиксированному столбцу, если понадобится поменять местами какие-то две строчки, мы поменяем местами значения ссылок (адреса) на них.

Поэтому выберем другой метод сортировки, а именно: будем искать те элементы в зафиксированном столбце, которые не удовлетворяют условию упорядочения, и менять соответствующие строки местами.

```
program Primer3;
 const
       k : Integer =  5;
       n           = 10;
type
         TItem = Integer;
         TRad  = array[1..1]  of  TItem;                    {•}
         PTRad = ^TRad;
{$R-}
         TMas  = array[1..n]  of  PTRad;                    {•}
 var
         A    : TMas;                                       {•}
         m, i : Word;                                       {•}
procedure Show(A:TMas; n, m: Word; Str:String);             {•}
var
     i, j : Word;
 begin
   Writeln(Str:40);
   for i := 1 to n do
     begin
      Writeln;
      for j := 1 to m do
         Write (A[i]^[j] :5);                               {•}
     end;
   Writeln;
 end;
procedure Init(var A:TMas; n, m: Word);                     {•}
```

```
var
     i, j : Word;
 begin
  for i := 1 to n do
     for j := 1 to m do
         A[i]^[j] := Random(101);                              {•}
 end;

procedure Sort(var A:TMas; n, m: Word);                        {•}
var
       i, j, p : Word;
       r       : PTRad;                                        {•}
 begin
  for i := 1 to n - 1 do
    begin
     p:=i;
     for j := i + 1 to n do
         if A[p]^[k] < A[j]^[k] then p:=j;                     {•}
     if p <> i then
       begin
        r    := A[i];                                         {•}
        A[i] := A[p];                                         {•}
        A[p] := r
       end;                                                   {•}
    end;
 end;
begin
 Write('input m=');                                           {•}
 Readln(m);                                                   {•}
    {Размещение матрицы по строкам }
 for  i := 1  to  n  do                                       {•}
     GetMem(A[i], m * Sizeof(TRad));                          {•}
 Randomize;
 Init(A, n, m);
 Show(A, n, m, 'A – исходная');
 Sort(A, n, m);
 Show(A, n, m, 'A – упорядоченная');
 for  i := 1  to  n  do                                      {•}
     FreeMem(A[i], m * Sizeof(TRad));                        {•}
end.
```

Вариант 4

Самостоятельно рассмотрите ситуацию, когда количество строк неизвестно и становится известным во время выполнения программы (вводится), а число столбцов – константа, но Sizeof(A_{nm}) > 64 кб.

Здесь можно реализовать задачу так, чтобы работать с транспонированной матрицей, и тогда можно применить предыдущий алгоритм.

Вариант 5

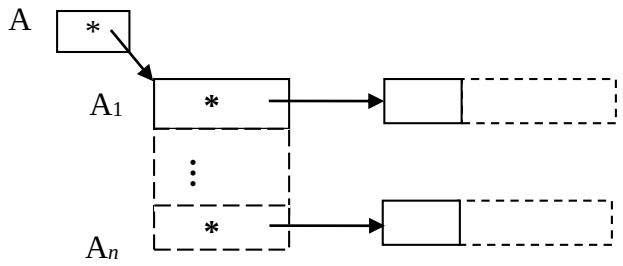
Рассмотрим ситуацию, когда размеры матрицы неизвестны и становятся известными во время выполнения программы (вводятся).

Решение задачи дальше зависит от того, поместится ли матрица целиком в сегмент (Sizeof(A_{nm}) < 64 кб). При положительном ответе решение задачи можно реализовать, представив матрицу как одномерный массив сначала из одного элемента, а затем, если размеры матрицы станут известными, распределить в динамической памяти всю матрицу. Этот алгоритм мы уже рассматривали в варианте 1.

Предлагаем еще один вариант решения задачи, который будет подходить и в случае, когда Sizeof(A_{nm}) < 64 кб и Sizeof(A_{nm}) > 64 кб. Здесь мы будем использовать подход, изложенный в варианте 3.

Заведем «генеральный» указатель на массив указателей из одного элемента на строку матрицы, которая также состоит из одного элемента.

Тогда схема представления матрицы будет следующей:



и обращение к элементу a_{ij} будет реализовываться так:

$$a_{ij} \Rightarrow \texttt{A\^{}[i]\^{}[j].}$$

Как и прежде, мы существенно будем использовать текст предыдущей программы. Измененные операторы обозначим комментарием `{~}`.

```
program  Primer5;
const
         k : Integer =  5;
type
         TItem  = Integer;
         TRad   = array[1..1]  of  TItem;
         TPRad  = ^TRad;                                  {~}
{$R-}
         TMas  = array[1..1]  of TPRad;                   {~}
         TPMas = ^ TMas;                                  {~}
var
         A       : TPMas;                                 {~}
         n, m, i : Word;                                  {~}

procedure Show(A: TPMas; n, m: Word; Str:String);         {~}
var
     i, j : Word;
 begin
  Writeln(Str:40);
  for i := 1 to n do
    begin
     Writeln;
     for j := 1 to m do
         Write (A^[i]^[j] :5);                            {~}
    end;
  Writeln;
 end;

procedure Init(A: TPMas; n, m: Word);                     {~}
 var
     i, j : Word;
 begin
  for i := 1 to n do
     for j := 1 to m do
         A^[i]^[j] := Random (101);                       {~}
 end;

procedure Sort(A: TPMas; n, m: Word);                     {~}
 var
       i, j, p : Word;
```

```
      r       : TPRad;                                    {~}
 begin
  for i := 1 to n - 1 do
    begin
     p := i;
     for j := i + 1 to n do
         if A^[p]^[k] < A^[j]^[k] then p := j;           {~}
         if p<>i then
           begin
            r    := A^[i];                               {~}
            A^[i]:= A^[p];                               {~}
            A^[p]:= r
           end                                           {~}
     end;
 end;
begin
 Writeln('n, m - ??');
 Readln(n, m);
 GetMem(A, n * Sizeof(TMas));                            {~}
 for  i := 1  to  n  do                                  {~}
     GetMem(A^[i], m * Sizeof(TPRad));                   {~}
 Randomize;
 Init(A, n, m);
 Show(A, n, m, 'A – исходная');
 Sort(A, n, m);
 Show(A, n, m, 'A – упорядоченная');
 for  i := 1  to  n  do                                  {~}
    FreeMem(A^[i], m * Sizeof(TPRad));                   {~}
 FreeMem(A, n * Sizeof(TMas))                            {~}
end.
```

Проблема потерянных ссылок

При работе с указателями желательно учитывать следующее:

1. Процедуры `Mark(P)` и `Release(P)` фактически должны вызываться в программе один раз, иначе могут быть не спрогнозированные ситуации.

2. Нужно осторожно пользоваться ссылками, объявленными в подпрограммах, т. к. локальные переменные появляются во время вызова подпрограммы, распределяясь в сегменте стека, и исчезают после завершения работы подпрограммы.

3. Нужно стремиться освобождать выделенные области сразу же после того, как необходимость в них отпадает, иначе «загрязнение»

памяти ненужными динамическими переменными может привести к быстрому ее исчерпанию.

Проанализируем следующую ситуацию:

```
program LostReference;
type
      TPPerson = ^TPerson;
      TPerson  = record …{информационная часть записи}
                 end;
var
      Memory : Longint;
 procedure GetPerson;
  var  P : TPPerson;
       {локальная переменная – указатель на запись}
  begin
   P := New(TPPerson)
  end;
begin
 Memory := MemAvail;
 Writeln(MemAvail);
 GetPerson;
 Writeln (MemAvail);
 if  Memory = MemAvail  then  Writeln ('OK');
end.
```

Вызов New в процедуре GetPerson приведет к выделению памяти для динамической переменной типа Person, указатель на эту переменную присваивается переменной P, которая является локальной. Значит, после выхода из подпрограммы она «потеряется», но память, отведенная в подпрограмме, продолжает существовать, поскольку освободить ее можно только явно, с помощью процедуры Dispose, но доступа к переменной уже нет, так как ссылка, которая показывала на переменную, потерялась!

Можно запрограммировать вывод общего объема свободной памяти до и после работы подпрограммы. В данном случае он показал бы, что утеряна какая-то часть памяти.

4. После выполнения New(P) второе New(P) с этим же показателем можно выполнять только тогда, когда после первого New(P) будет освобождена память в Heap, связанная с P.

Последовательность операторов New(P); New(P); приведет к ошибке, так как первая ссылка на динамическую переменную, под которую была отведена память процедурой New(P), потеряется при втором выделении

памяти под другую динамическую переменную, но связанную опять же с указателем P.

Нужно придерживаться такого правила: при выходе из блока необходимо или освободить (уничтожить) все созданные в них динамические переменные, или сохранить каким-то образом ссылки на них (например, присвоив эти ссылки глобальным переменным).

5. Бывает и другая ситуация, когда некоторая область памяти освобождена, а в программе остался указатель на эту область.

Рассмотрим следующий пример. Что будет напечатано следующей программой?

```
var
      P : ^Integer;
 procedure X1_Proc;
  var
      i : Integer;
        {локальная переменная размещается в
         стековой памяти, и после завершения процедуры
         эта память снова считается незанятой    }
 begin
  i := 12345;
  P := @i;
  Writeln (P^);
 end;
 procedure X2_Proc;
  var
      j : Integer;
  begin
   j := 7777;
   Writeln (P^);
  end;
begin
 X1_Proc;                {напечатается 12345}
 X2_Proc;                {напечатается 7777 }
   {только такая последовательность вызовов процедур}
end.
```

Замечание. Глобальная ссылочная переменная P в процедуре X1_Proc устанавливается на локальную переменную i, которая была в стеке. После завершения процедуры X1_Proc переменная i исчезает (место в стеке освобождается); указатель P «завис». Если же мы вызвали процедуру

`X2_Proc`, то локальная переменная `j` (того же типа!) будет распределена на то место в стеке, где находилась ранее переменная `i`. Теперь указатель `P` ссылается на `j`, что и подтверждает результат работы программы.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 10
РАБОТА С ДИНАМИЧЕСКИМИ СВЯЗНЫМИ СПИСКАМИ

Содержание темы

- Введение в связные динамические структуры данных.
- Алгоритмы работы с линейными списками.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ВВЕДЕНИЕ В СВЯЗНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Рассмотрим динамические структуры данных, количество элементов которых сразу неизвестно, и порядок следования их обусловлен самой совокупностью данных, например список студентов, очередь в магазине и т. д.

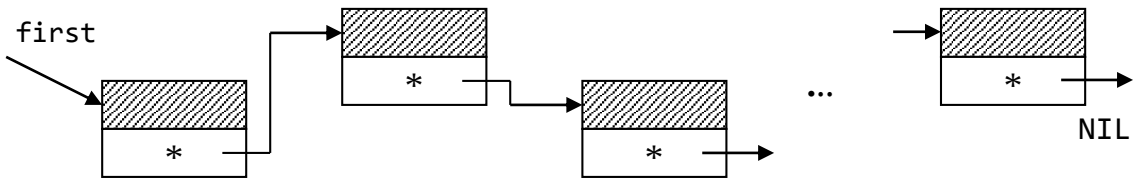
Элементы таких данных последовательно друг за другом можно распределять в `Heap`, связывая их воедино. При таком подходе у элемента должны быть как информационная часть (в дальнейшем на схеме выделена штриховкой), так и ссылочная (на схеме отмечена символом *), которая и поможет включить элемент в последовательность.

Для последовательности линейного вида, количество элементов которых заранее неизвестно, можно применять связную форму хранения их в динамической памяти, например, в виде списка.

Списком называют такую структуру данных, каждый элемент которой содержит ссылку, связывающую его со следующим элементом – звеном (узлом) списка. Звено списка делится на части: информационную и ссылочную.

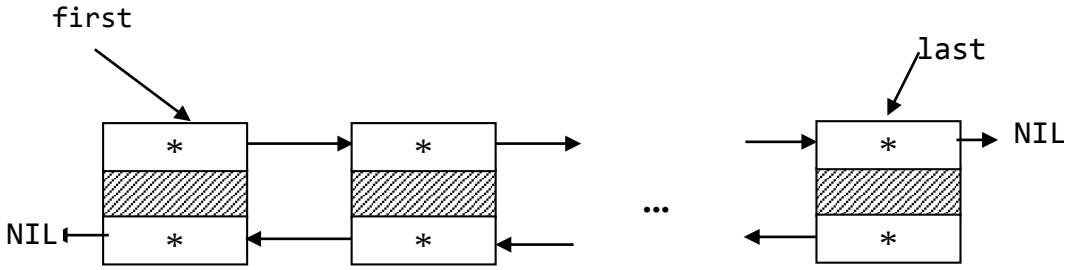
Пусть на первый элемент будет указывать указатель `first`, а на последний – `last`.

Списки бывают разные, в том числе и *однонаправленные линейные*:



По такому списку проход может быть только от `first` (с начала) в конец.

Существуют и *двунаправленные линейные* списки:



На первое звено списка должна быть ссылка, например `first`. А остальные связаны друг с другом собственными связями. Последнее звено в своей ссылочной части ссылается в `nil`.

Описание звена списка можно делать в соответствии со следующим фрагментом кода:

```
type
        type_inf =
                record …
                end;
        Ptr_zveno  = ^zveno;
        zveno      = record
                       inf  : type_inf;
                       next : Ptr_zveno
                     end;
```

Чтобы добавить элемент в список, нужно в `Heap` под звено запросить память, потом это звено включить в список (первым, в середину, последним), настроив соответствующие связи при помощи ссылочных частей звена.

АЛГОРИТМЫ РАБОТЫ С ЛИНЕЙНЫМИ СПИСКАМИ

Ресурсы для работы со списками лучше объединить в отдельном модуле.

Для работы со списками желательно предусмотреть следующие действия.

1. Размещение элемента списка в динамической памяти.

Можно пользоваться как процедурами New, так и GetMem. например,

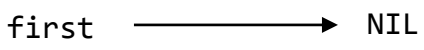
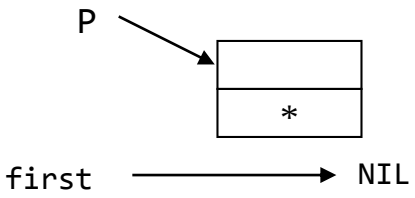
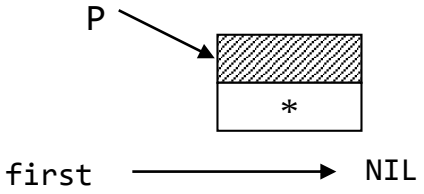
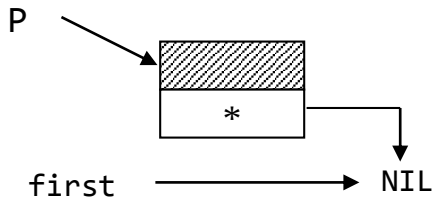
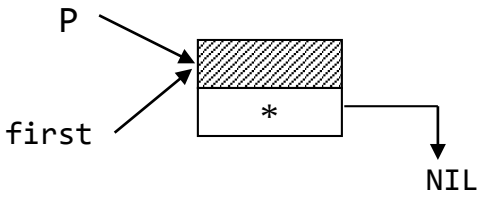
```
var    P : Ptr_zveno;
       …
begin
      New(P); …
```

2. Инициализация информационной части звена.

Заполнить информационную часть – значит присвоить значение соответствующим полям записи P^.

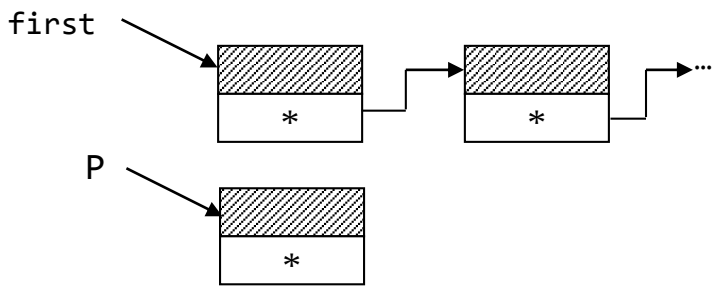
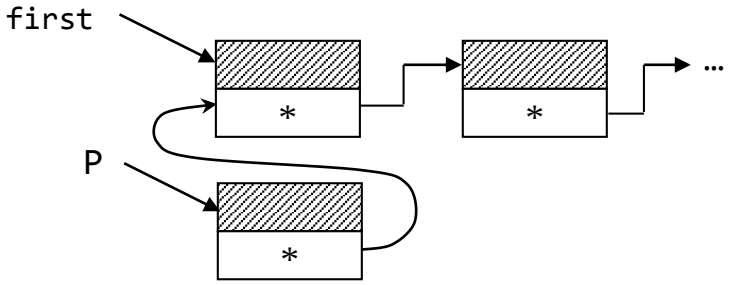
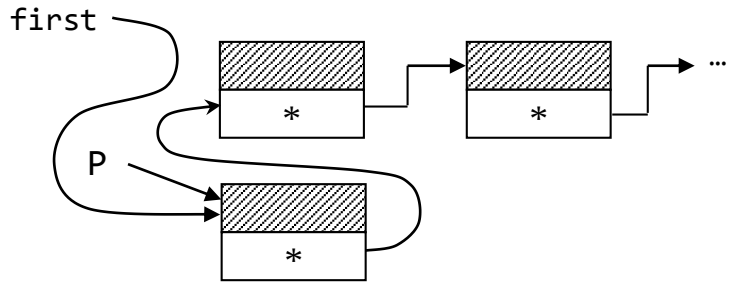
3. Включение звена в нужное место списка.

Создание нового списка

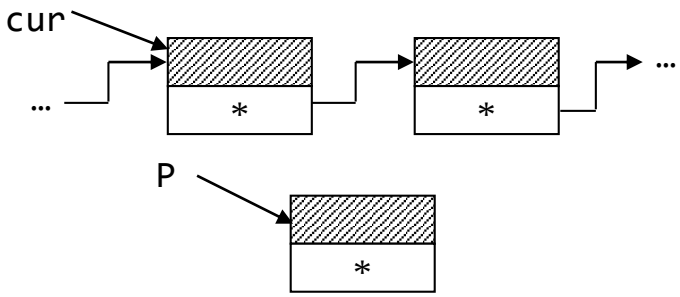
Действие	Схема
Первоначально, когда список пустой, указатель first устанавливается в nil. `first:=nil`	
Затем в динамической области создается новое звено – значение типа Ptr_zveno. `P := New(Ptr_zveno)`	
Информационная часть звена наполняется информацией при помощи описанной заранее процедуры Init_inf. `Init_inf(P)`	
Ссылочная часть нового звена перенаправляется на first (фактически на nil). `P^.next := first`	
Указатель на начало списка first перенаправляется на P. `first := P`	

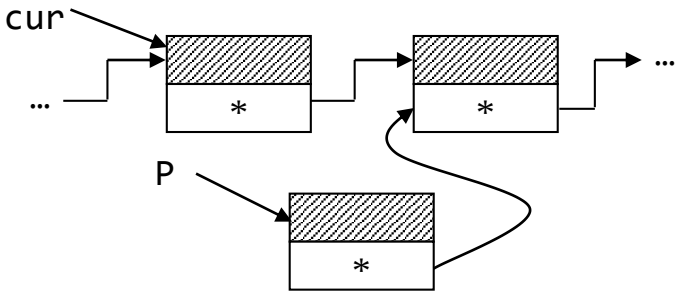
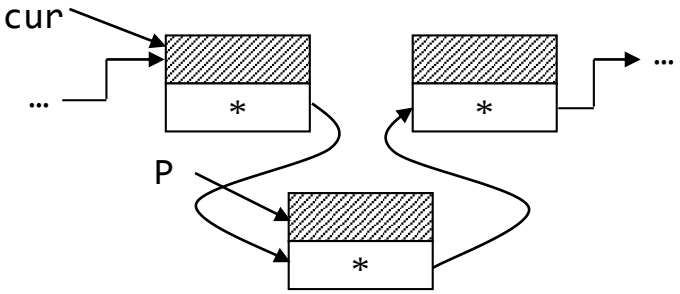
Рассмотрим далее, как в список добавить первый элемент, в середину списка и в конец списка.

Добавление в начало списка

Действие	Схема
Добавление в начало списка нового звена, на которое ссылается указатель P. Исходное состояние	
Ссылочная часть нового звена перенаправляется на first. `P^.next := first`	
Указатель на начало списка first перенаправляется на P. `first := P`	

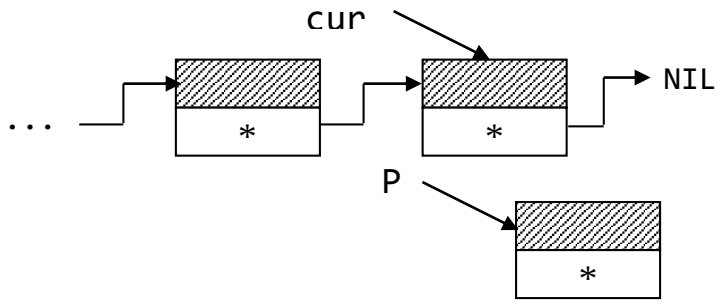
Добавление в середину списка

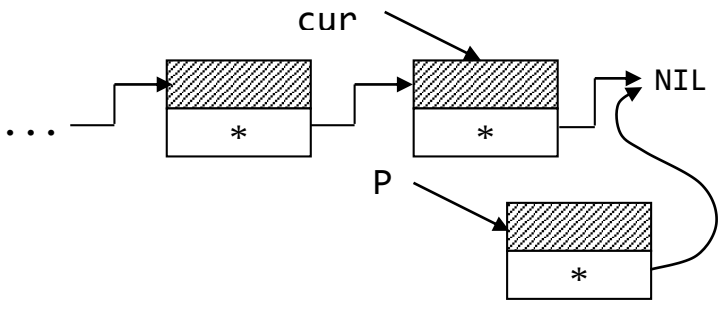
Действие	Схема
Добавление нового звена, на которое ссылается указатель P, в середину списка после звена, на которое ссылается указатель cur. Исходное состояние	

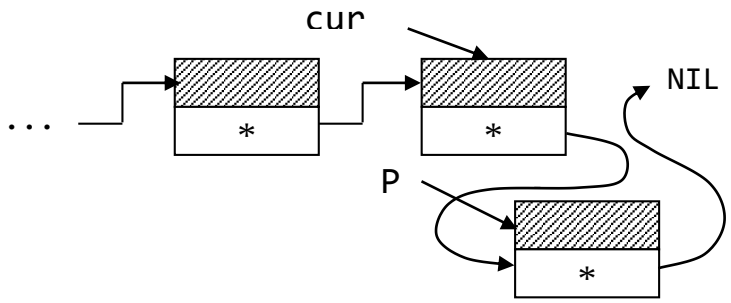
Действие	Схема
Ссылочная часть нового звена перенаправляется на `cur^.next`. `P^.next := cur^.next`	
Ссылочная часть звена, на которое ссылается указатель `cur`, перенаправляется на `P`. `cur^.next := P`	

Замечание. Возможно добавление нового звена, на которое ссылается указатель `P`, в середину списка перед звеном, на которое ссылается указатель `cur`. Разработайте такой алгоритм самостоятельно.

Добавление в конец списка

Действие	Схема
Добавление нового звена, на которое ссылается указатель `P`, в конец списка, после последнего звена, на которое ссылается указатель `cur`. Исходное состояние	

Ссылочная часть нового звена перенаправляется на `cur^.next` (фактически на `nil`).
`P^.next :=`
`cur^.next`
	

Действие	Схема
Ссылочная часть звена, на которое ссылается указатель `cur`, перенаправляется на `P`. `cur^.next := P`	

4. Поиск элемента списка, на который наложены определенные условия, можно выполнять по аналогии со следующим фрагментом кода.

```
Cur := first;
while  cur <> nil  do
       begin
            {по информационной части ищем
             нужное звено. Обрабатываем его}
        cur := cur^.next;
            end;
```

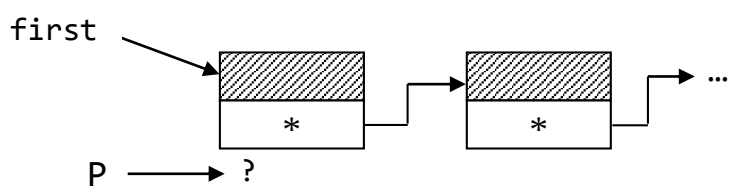
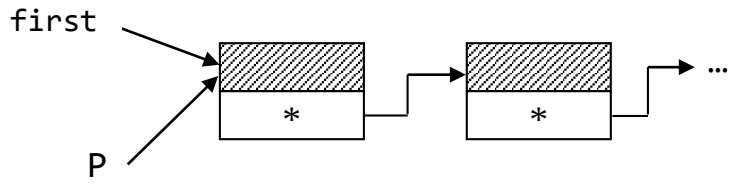
5. Распечатка информационной части элемента списка.

6. Распечатка всех элементов списка.

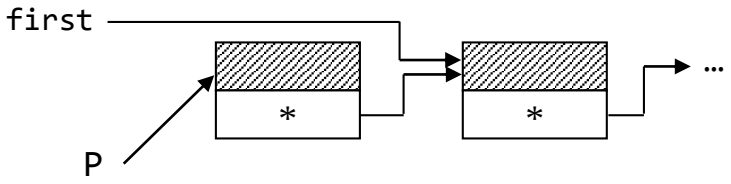
7. Нахождение и удаление конечного числа элементов списка.

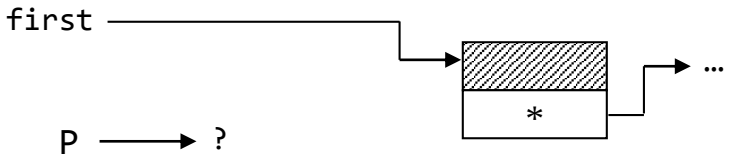
Рассмотрим, как в списке уничтожить первый элемент, элемент в середине списка и последний элемент.

Удаление первого звена в списке

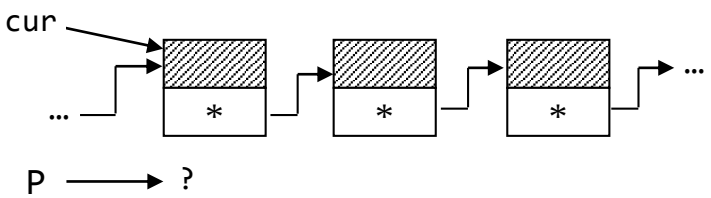
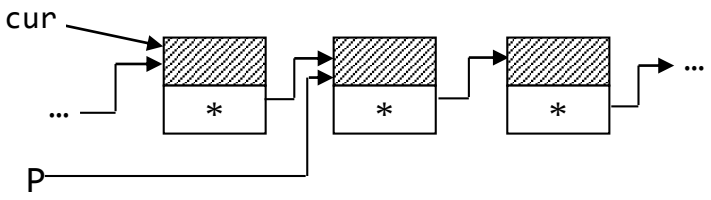
Действие	Схема
Удаление первого звена, на которое ссылается указатель `first`. Исходное состояние	
Ссылочная переменная `P` перенаправляется на `first`. `P := first`	

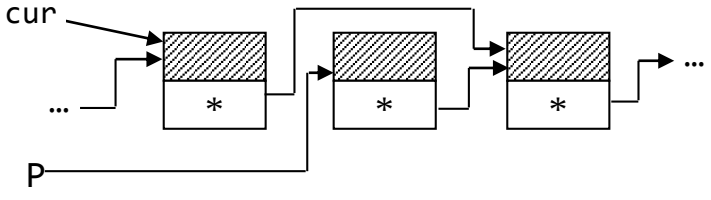
Действие	Схема

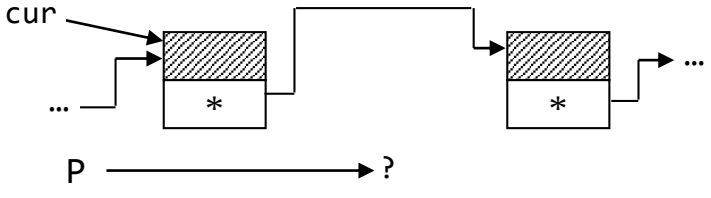
Указатель на начало списка `first` перенаправляется на второй элемент списка.
`first :=`
`first^.next`
	

Удаляется звено, которое раньше было первым. `Dispose(P)`	

Удаление звена в середине списка

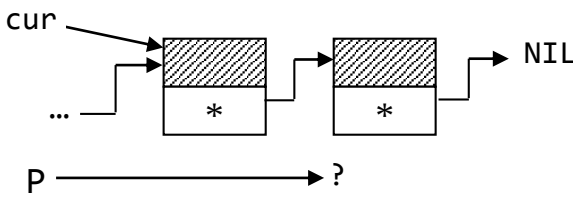
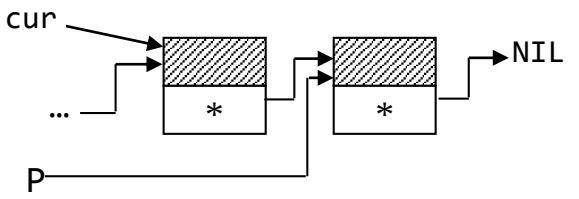
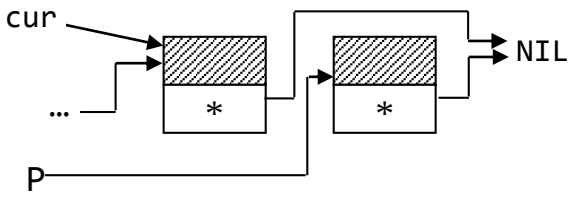
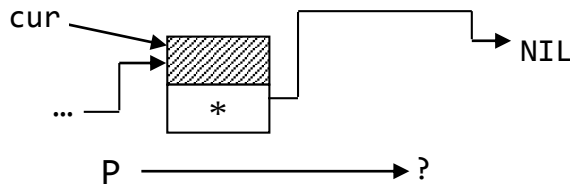
Действие	Схема
Удаление звена в середине списка, после звена, на которое ссылается указатель `cur`. Исходное состояние	
Ссылочная переменная `P` перенаправляется на `cur^next`. `P := cur^.next`	

Ссылочная часть звена, на которое ссылается указатель `cur`, перенаправляется на `cur^.next^.next`:
`cur^.next :=`
`cur^.next^.next`
	

Удаляется звено, которое размещено после звена, на которое ссылается указатель `cur`. `Dispose(P)`	

Замечание. Возможно удаление звена, на которое ссылается указатель `cur`, в середине списка. Разработайте такой алгоритм самостоятельно.

Удаление последнего звена из списка

Действие	Схема
Удаление последнего звена из списка. Указатель cur должен ссылаться на предпоследнее звено. Исходное состояние	
Ссылочная переменная P перенаправляется на последнее звено cur^.next. P := cur^.next	
Ссылочная часть звена, на которое ссылается указатель cur, перенаправляется на cur^.next^.next (фактически на nil) cur^.next := cur^.next^.next	
Удаляется звено, которое расположено после звена, на которое ссылается указатель cur. Dispose(P)	

8. Сортировка элементов списка.

Схема модуля:

```
unit  Spis_new;
interface
type      type_inf  = record …  end;
          Ptr_zveno = ^zveno;
          zveno     = record
                        inf  : type_inf;
                        next : Ptr_zveno
                      end;
procedure  Print_inf(P : zveno);
procedure  Init_inf (var P : zveno);
procedure  Sozd_spis(var  first  : Ptr_zveno);
procedure  Show_Spis(const first : Ptr_zveno);
           …
```

```
implementation
…   {В соответствии с предложенными схемами разработайте
процедуры самостоятельно}
end.
```

Задание. Для полинома $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ степени n, в котором отсутствует много одночленов, задаются пары чисел $\{i, a_i\}$, причем пара, у которой $a_i = 0$, отсутствует.

Требуется:

- ввести коэффициенты полинома;
- распечатать в виде полинома по спаданию степеней, например в таком виде: $5 * x\text{^}30 - x\text{^}10 + x + 1$;
- подсчитать значение полинома при различных значениях аргумента;
- подсчитать сумму коэффициентов полинома; подсчитать суммы коэффициентов полинома при четных и нечетных степенях.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 11
МОДУЛЬ DOS. МОДУЛЬ CRT

Содержание темы

- Модуль DOS.
- Модуль CRT. Основные положения.
- Управление звуком.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

МОДУЛЬ DOS

Процедуры и функции модуля DOS предназначены обеспечить удобный способ связи (интерфейс) с программами операционной системы MS DOS. Все они отсутствуют в стандарте языка.

Подключение модуля:

```
uses DOS;
```

Всего в модуле DOS можно насчитать шесть функциональных групп:

- опрос и установка параметров MS DOS;
- работа с датой и временем ПК;
- анализ ресурсов дисков;
- работа с каталогами и файлами;
- работа с прерываниями MS DOS;
- организация субпроцессора и резидентных программ.

Рассмотрим некоторые из них.

НЕКОТОРЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ МОДУЛЯ DOS

Название	Группа
`GetDate, SetDate, GetTime, SetTime`	Работа с часами и календарем
`DiskSize, DiskFree,`	Анализ ресурсов дисков
`GetFTime, SetFTime, FExpand, FSplit, FindFirst, FindNext, FSearch, GetFAttr`	Работа с каталогами и файлами
`Intr, MsDos`	Работа с прерываниями MS DOS
`Exec`	Организация субпроцессора и резидентных программ

МОДУЛЬ CRT

Аббревиатура CRT расшифровывается как «электронно–лучевая трубка». В модуле CRT описаны типы, константы, переменные и реализованы специальные процедуры и функции для работы с текстовой информацией на дисплее, позволяющие:

- управлять текстовыми режимами;
- организовывать окна вывода на экране;
- настраивать цвета символов;
- управлять движением курсора.

Кроме того, в модуль включены функции опроса клавиатуры и процедуры управления встроенным в ПК динамиком.

Подключение модуля:

```
uses CRT.
```

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Рассмотрим далее более подробно следующие виды работ:

- Работа с клавиатурой, звуком и курсором.
- Работа в текстовом видеорежиме.

Собственно язык Pascal очень прост и лаконичен, стандартные библиотечные подпрограммы же в своей значительной части образуют связь между языковыми средствами Turbo Pascal и функциями операционной системы, а также помогают организовать работу с устройствами компьютеров, совместимых с IBM PC.

УПРАВЛЕНИЕ ЗВУКОМ

В ПК имеется возможность генерировать при помощи встроенного динамика звуковые сигналы частотой от 37 до 32767 Гц. Воспроизводятся только чистые тона без искажений, сила звука не регулируется (при использовании оператора Writeln(^G) воспроизводится звуковой сигнал длительностью 0,25с и частотой 800 Гц).

Для управления частотой звука и его продолжительностью используются стандартные процедуры Sound, NoSound и Delay.

Процедура Sound (Hz) включает внутренний динамик, Hz: Word задает частоту сигнала, который генерируется динамиком, в герцах. Звучание можно отменить процедурой NoSound.

Процедура `Delay (Ms)` осуществляет задержку в выполнении программы на `Ms: Word` миллисекунд (хотя, как оказалось время задержки зависит от тактовой частоты процессора).

Пример 1. Программа, имитирующая сигнал звоночка. Логическую функцию `keypressed`, которая фактически делает паузу до нажатия какой-либо клавиши, рассмотрим ниже.

```
program bell;
uses crt;
var  i:Integer;
 begin
  repeat
   for i:=9 to 12 do
    begin
     sound(i*100);
     delay(20);
    end;
  until keypressed;
  nosound;
 end.
```

Пример 2. Программа, имитирующая звук сирены.

```
program sirena;
uses crt;
var
  i:Integer;
 begin
 repeat
  for i:=300 to 1800 do
   begin
    sound(i);    delay(400);
   end;
  nosound;
  for i:=1800 downto 300 do
   begin
    sound(i);    delay(400);
   end;
   delay(400);  nosound;
   Writeln('Нажми любую клавишу');
 until keypressed;
end.
```

Ввести в программу нехитрые мелодии можно, зная ноты и их частотные эквиваленты в герцах.

Пример 3.

```
program DemoGamma;
uses crt;
const
   M : array [1..7] of Integer=
             (262, 294, 330, 349, 392, 440, 494);
         {малая октава ноты До, Ре, Ми, Фа, Соль, Ля, Си}
   T : array [1 .. 7] of Integer=
                      (10, 11, 12, 13, 14, 15, 16);
                  {время звучания}
 var
      i, j : Byte;
 begin
 for j := 1 to 100 do
     begin
     for i := 1 to 7 do
         begin
          Sound(M[i]);   Delay(T[i]);
          NoSound
         end;
     end
 end.
```

Изменяя значения элементов массивов M и T, можно добиться хорошей имитации музыкальных произведений.

Для вычисления частоты сигнала имеется формула:

```
Round(440*exp(ln(2)*(akt-(10-nota)/12))),
```

где `akt` – номер одной из восьми октав, причем самая низкая тональная октава имеет номер “–3” ⇒ –3, –2, –1, ..., 3, 4;

`nota` – номер ноты в октаве:

“До” – 1, “До–диез” – 2, “Ре” – 3, ... “Си” – 13.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?

ТЕМА 12
МОДУЛЬ CRT. РАБОТА С КЛАВИАТУРОЙ

Содержание темы

- Работа с клавиатурой.
- Управление курсором.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

РАБОТА С КЛАВИАТУРОЙ

Главным средством ввода информации в ПК является клавиатура. Все клавиши клавиатуры можно разделить на шесть групп:

№	Назначение	Пример
1	Алфавитно–цифровые и знаковые	`A .. Z, a ..z, 0 ..9, +, –, *, /, Esc, Tab, Enter, BackSpace, …`
2	Функциональные	`F1 ..F12`
3	Служебные для управления перамещением курсора и редактирования	`↑, →, ↓, ←, End, Home, PageUp, PageDown, Del, …`
4	Служебно-управляющие	`Alt, Ctrl, Shift`
5	Служебные – для фиксации регистров	`Caps Lock, Scroll Lock, Num Lock, Ins`
6	Вспомогательные	`Prt_Sc, Pause/Break …`

Каждая клавиша имеет свой номер, называемый `Scan`–кодом (кодом считывания): Esc – 1; 1! – 2 и т. д. Поэтому `Shift` левый и `Shift` правый имеют разные `Scan`–коды.

В системной памяти компьютера находится буфер клавиатуры. Это циклическая очередь из 15 двухбайтовых символов.

Различают три уровня представления и обработки сигналов от клавиатуры: физический, логический, функциональный.

На *физическом* уровне анализируются сигналы, поступающие при нажатии и отпускании клавиш (длительное нажатие воспринимается как многократное).

Клавиатура имеет свой самостоятельный микропроцессор, который по Scan-коду анализирует, какая клавиша нажата (или отпущена), и передает информацию в процессор компьютера.

Клавиши разделяют на три типа:

1. Клавиши и комбинации клавиш, которые после нажатия пересылают в буфер клавиатуры ASCII-код.

Это алфавитно-цифровые клавиши, нажатые одновременно с Shift или без, а также нажатие комбинации клавиш: Ctrl и алфавитно-цифровых; Ctrl и некоторых специальных символов.

2. Клавиши и комбинации клавиш, нажатие которых посылает в буфер клавиатуры расширенный код. Это, например, функциональные клавиши: F1, …, F12; функциональные нажатые одновременно с Shift: Shift+F1, …, Shift+F12; функциональные нажатые одновременно с Ctrl: Ctrl +F1, … Ctrl+F12; функциональные нажатые одновременно с Alt: Alt+F1, …, Alt+F12; алфавитно-цифровые нажатые одновременно с Alt: Alt+ алфавитно-цифровые; управляющие ↓ → … и некоторые другие.

Расширенный код состоит из двух символов: первый символ есть #0, второй – ASCII-код.

3. Клавиши и комбинации клавиш, нажатие которых не посылает в буфер клавиатуры никаких кодов. Состояние их или фиксируется в специальном месте оперативной памяти: это клавиши Shift, Alt, Ctrl, или распознается комбинация клавиш как вызов подпрограммы: например, Ctrl+Alt+Del; Alt+PrintScreen, или игнорируются.

Логический уровень поддерживается базовой системой ввода-вывода. После обработки прерывания, поступившего от клавиатуры, или изменится информация о состоянии нажатия клавиш Shift, Alt, Ctrl, Caps Lock, или в буфер клавиатуры запишется простой или расширенный коды, и управление передается на следующий уровень – функциональный.

На *функциональном* уровне отдельным клавишам программным путем ставятся в соответствие собственные функции, реализуемые при нажатии этих клавиш.

Опрос клавиатуры

Основой алгоритмов управления клавиатуры является анализ в программе буфера клавиатуры. Считывать символы из буфера можно как процедурой Read, Readln, так и функцией модуля CRT

```
Readkey : Char.
```

Команда ch: = Readkey как бы вынимает последовательно введенные в буфер клавиатуры символы по одному за каждое обращение.

Особенности работы функции Readkey:

- полученные функцией символы не отображаются на дисплее;
- режим работы Readkey зависит от состояния буфера ввода:
  - если в буфере есть символы, то функция прочитает очередной символ из буфера (тот, который был введен первым) и удалит его из буфера;
  - если буфер пуст, то функция, а вместе с ней и программа, ожидает ввода символа.

Функция

```
KeyPressed : Boolean
```

возвращает значение true, если в буфере ввода с клавиатуры имеется хотя бы один символ, и false, если буфер пуст.

Следующая группа операторов очищает буфер клавиатуры:

```
while KeyPressed do ch := ReadKey;
```

Можно их оформить процедурой, например, с названием ClrKeyBuf.

Ожидание нажатия клавиши (клавиш), которая вырабатывает простой или расширенный код, осуществляется таким образом:

```
repeat … until KeyPressed.
```

При старте программы буфер обычно пуст.

При работе с клавиатурой возможны следующие ситуации. Если пользователь нажмет комбинацию клавиш:

- может возникнуть и программное прерывание, например (если «система» включила (on) анализ этих ситуаций):
  1) Ctrl + Alt + Del;
  2) Ctrl + Break;
  3) Alt + PrtScr;
  4) Ctrl + Alt;
- может образоваться расширенный код;
- комбинация может быть проигнорирована. Если сразу нажать Ctrl, Alt, Shift (последовательно), тогда приоритет имеет Alt, затем Ctrl, затем Shift;
- обычные (символьные) клавиши, даже если они нажаты одновременно с клавишами Ctrl или Shift, выдают символы с ASCII-кодом в диапазоне 1..127 (на русифицированных ПК диапазон до 255). При нажатии клавиш Ctrl + алфавитная эффект определяется ее латинским названием (даже если выбрана русская раскладка клавиатуры): Ctrl + A → 1; ... ; Ctrl + Z → 26. Это управляющие символы из таблицы кодов. Клавиши

Esc и «пробел» в любой комбинации дают одни и те же коды: 27 и 32 соответственно.

Введенные символы остаются в буфере клавиатуры и доступны для дальнейшей обработки.

Опрос расширенных кодов

Алфавитно-цифровые клавиши, нажатые одновременно с клавишей Alt, и функциональные клавиши посылают в буфер ввода с клавиатуры сразу два символа: первый – #0 (нулевой символ), и второй – с числовым кодом. Вторая часть расширенного кода может совпадать с кодировкой некоторого символа. Так, функциональная клавиша F1 дает расширенный код #0#59; если не учесть, что это расширенный код, тогда код #59 можно трактовать как символ «;».

Подобный механизм значительно повышает информационную отдачу клавиатуры.

Рассмотрим примеры обработки нажатия клавиш клавиатуры.

Пример 1. Программа вывода кодов нажатых клавиш.

```
program PR_key;
uses CRT;
var
    CH : Char;
begin
clrscr;
repeat
      CH := Readkey;
      if CH = #0 then
         begin                              {расширенный код}
         CH := Readkey;
         Write(' спецклавиша ');
         end
         else
             begin
              Write (' Char= ');     {алфавитно-цифровой}
              if CH >= #32 then Write(CH)
              else Write(' ^ ', CHR(ORD(CH) + 64));
             end;              {управляющий символ}
      Writeln (' ASCII= ', ORD(CH));
{т.к. код может совпадать с управляющим, поэтому печатаем номер}
until CH = #27;                              {выход по ESC}
end.
```

Если код клавиши расширенный, тогда надо к буферу клавиатуры обращаться два раза. Опишем функцию, которая будет возвращать символ и знак расширенного кода.

Пример 2.

```
program PR_KeyD;
uses CRT;
const
      ESC    = #27;        DEL  = #83;
      INSKEY = #82;        HOME = #71;
      F1     = #59;        F10  = #68;
      SF1    = #84;        SF10 = #93;
      CF1    = #94;        CF10 = #103;
      AF1    = #104;       AF10 = #113;
var
     ExtendKey : Boolean;
     CH        : Char;
function GetKey(var ExtendKey : Boolean) : Char;
 var
      ch : Char;
 begin
 ExtendKey := false;
 ch        := Readkey;
 if  ch = #0    then
     begin
     ExtendKey := true;
     ch        := Readkey;
     end;
 GetKey := ch
 end;
begin
ClrScr;
repeat
  ch := GetKey(ExtendKey) ;
  if  not  ExtendKey  then
      Writeln ('символ клавиши с кодом=', Byte(ch))
  else
    case ch of
      SF1..SF10 :
        Writeln('Нажата SHIFT+функциональная клавиша');
      CF1..CF10 :
        Writeln('Нажата CTRL +функциональная клавиша');
```

```
      F1 .. F10 :
        Writeln('Нажата функциональная клавиша');
      INSKEY    :
        Writeln('Нажата клавиша «вставка» ');
      DEL :
        Writeln('Нажата клавиша «del» ');
      HOME :
        Writeln('Нажата клавиша «home» ');
      else Writeln('Расширенный код #00+', Byte(ch));
   end;
until ch = ESC
end.
```

За каждой клавишей можно условно закрепить определенную ноту и таким образом и на компьютере имитировать музыкальное устройство.

Пример 3. Имитация музыкального инструмента.

Решение. За цифровыми клавишами 0, 1, … , 8, коды которых #48-#55, закрепим ноты 262, 294, 330, 349, 392, 440, 494, 523. Нажатие любой цифровой клавиши позволит воспроизвести соответствующий звук.

```
program DemoInstrument;
uses CRT;
const
      M : array [1 .. 8] of Integer =
                (262, 294, 330, 349, 392, 440, 494, 523);
var    I  : Integer;
       Ch : Char;
begin
while true do
 begin
  Ch:= Readkey;
  case Ch of
    #48       : Halt;                  {клавиша 0}
    #49.. #55 : I := Ord(ch) – 48;
   else
     Write('клавише звук не назначен', ^G, 'Повторите!');
  end;
  Sound(M[i]);
  Delay(100); {здесь можно repeat until KeyPressed}
  NoSound
 end
end.
```

УПРАВЛЕНИЕ КУРСОРОМ

Работа процедур `Write` и `Writeln` вызывает перемещение курсора по экрану дисплея.

Процедура `GOTOXY(x, y)` перемещает курсор в позицию `x` (столбец) и `y` (строка) относительно текущего окна ($1 \le x \le 80$, $1 \le y \le 25$).

Функции `WhereX`, `WhereY` – дают соответственно значение `x` и `y` – координат курсора относительно текущего окна.

Функции `WhereX`, `WhereY` – дают соответственно значение `x` и `y` – координат курсора относительно текущего окна.

```
Write('Курсор находится в столбце', WhereX);
Write('Курсор находится в строке ', WhereY);
```

Пример. Звуковое сопровождение украсило и следующую программу. Что выводит следующая программа?

```
uses crt;
type
    Stroka = String[160];
var
    vhod   : Stroka;
procedure Gostring(x, y : Byte; inst : Stroka);
var
       st1 : stroka;
       i   : Byte;
procedure Zwon;
 begin  Sound(1000); Delay(50); Nosound;  end;
Begin
 St1:='';
 Clrscr;
 St1:=st1+inst;
 Writeln(st1);
 Writeln;
 for i:=1 to Length(st1) do
  begin
   Delete(st1,1,1);
   Gotoxy(x, y);
   Write(st1);
   Zwon;
   Delay(500);
   DelLine; {удаляет с экрана строку, в которой
               находится курсор}
```

```
  Gotoxy(20,10);
  Writeln;
  Writeln(st1);
end;
begin
 Gostring(10, 20,'abcdefghijklmnopqrstuvwxyz');
end.
```

Задание. В зависимости от нажатия клавиш ↓ → ↑ ← заставить двигаться символ *, и при помощи этого движения написать свое имя.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 13
ВИДЕОДОСТУП. МОДУЛЬ CRT

Содержание темы

- Видеодоступ.
- Установка текстового режима.
- Вывод на экран.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ВИДЕОДОСТУП

Основным устройством для отображения информации, которая вводится и выводится, является *дисплей*. Поддержка его работы осуществляется при помощи соответствующего *адаптера* – специальной платы для подключения его к компьютеру. Изображение на экране формируется в двух основных режимах – текстовом и графическом. В первом – на экран выводятся символы, во втором – изображение выводится как объединение цветовых точек – *пикселов*.

Каждый пиксел, или точка, при цветном режиме может светиться разными цветами, пиксел таких размеров, что промежутки между соседними пикселами почти отсутствуют. Если группа смежных пикселов светится, то они воспринимаются как цельный участок.

Адаптер связывает микропроцессор с дисплеем устройством, которое называется контроллером электронно-лучевой трубки, имеющим в своем составе: *программируемые порты ввода-вывода, знакогенератор, оперативную электронную память.*

В первых моделях ПК применяли адаптеры `MDA` – монохромный, `CGA` – Color Graphics Adapter. Затем `MCGA` – multyCGA, `EGA` – Enhanced Graphics Adapter, `VGA` – Video Graphics Adapter, сейчас существуют `SVGA`-адаптеры и т. д.

Образ одного экрана хранится в электронной оперативной памяти в закодированном виде. *Страница* – образ экрана в памяти ПК (полная копия экрана), которую можно отобразить мгновенно в любое время.

Существует жесткая взаимосвязь между размером электронной памяти (видеобуфером), разрешением экрана, количеством видеостраниц и «диапазоном» цветов пиксела.

В текстовом режиме под один символ дается поле – матрица определенных размеров. Размеры матрицы зависят от разрешимости экрана (обычно 8×8, 8×14, 8×16, 9×14, 9×16).

Если разрешение экрана 640×200, то выбрано 25 строк и 80 столбцов при матрице 8×8 (640×200 ==25×8×80×8).

Разрешение экрана для VGA-адаптера

Адаптер	Разрешение	Кол-во цветов	Число видеостраниц	Объем видеобуфера
SVGA, VGA	640×200 640×350 640×480	16 16 $>2^8$	4 2 1	256k

В VGA-адаптере видеобуфер – 256 кб и более. Начало видеобуфера зависит от адаптера.

Замечание. Если у вас имеется многостраничный видеодоступ, тогда с каждой страницей связан свой курсор и местоположение его хранится в специальном месте оперативной памяти.

Текстовый режим работы дисплея поддерживает модуль `CRT`, графический – `Graph`.

Установка текстового режима

Текстовые режимы служат для отображения символов из таблицы кодов ПК (ASCII-кодов) и характеризуются количеством символов в строке и строк на экране.

Процедура

```
TextMode (M),
```

где `M` – типа `Word`, переключает текстовые режимы вывода информации на дисплей.

Специально для этой процедуры в модуле `CRT` определены следующие константы (можно пользоваться как названиями, так и числовыми эквивалентами):

Значение M		Разрешение	Замечание
0	BW40	40×25	Черно-белый при цветном адаптере
1	CO40 = C40	40×25	Цветной
2	BW80	80×25	Черно-белый при цветном адаптере
3	CO80 = C80	80×25	Цветной
256	Font8x8	80/40×43 80/40×50	Цветной, адаптор EGA Цветной, адаптор VGA

Все остальные числовые значения до 65 535, которые не совпадают ни с одной из указанных констант, включают процедурой TextMode режим C80.

Константа Font8×8 используется в адаптерах EGA и VGA, является дополнительной для первых четырех. Например: TextMode (CO80+Font8×8) изменит режим разрешимости экрана 80 × 25 на режим 80 × 43 (EGA) или 80 × 50 (VGA), поскольку константа Font8×8 обеспечивает построение символов не из матриц 8 × 14 и 8 × 16 , а из матриц 8 × 8. Когда основной режим не задан, то по умолчанию берется режим BW80.

Процедура TextMode очищает экран текущим цветом, устанавливает курсор в левую верхнюю позицию с координатами (1, 1), выполняет некоторые другие действия по наилучшему отображению информации на экран.

Значение установленного режима запоминается в переменной LastMode типа Word, описанной в CRT.

Вывод на экран

Для установки цветов символов, которые выводятся, используется процедура TextColor(Color), где 0≤Color≤15. Для установки цвета фона – процедура TextBackGround(Color), где 0≤Color≤7.

Эти процедуры связаны с описанной в модуле CRT переменной TextAttr типа Byte, где фиксируется установка цветов символа и фона, а изменение цветов на экране происходит при выполнении каких-либо операторов работы с экраном, например Write, clrscr и др.

Если цвет символа совпадает с цветом фона, тогда символ становится невидимым. Так можно «спрятать» на определенное время текст.

Для управления яркостью используются стандартные процедуры LowVideo (наименьшая), NormVideo (нормальная, восстанавливает первоначальный режим яркости), HighVideo (повышенная).

Очистка экрана и управление строками на экране

ClrScr полностью очищает экран или окно, и курсор помещается в левый верхний угол (цвет определяется текущим цветом фона).

ClrEol удаляет все символы в строке от позиции с курсором до конца строки, при этом можно изменить цвет фона в строке. Курсор остается на месте.

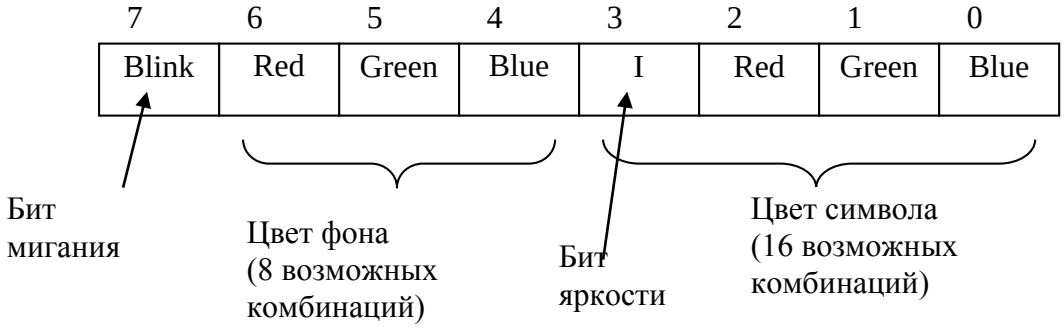
DelLine удаляет с экрана строку, в которой находится курсор. Все строки, расположенные ниже, перемещаются на одну строку вверх. В нижнюю часть активного окна добавляется новая строка.

`InsLine` в рамках текущего окна вставляет пустую строку с позиции расположения курсора.

Установка атрибутов цвета символа и фона

Если дисплей работает в текстовом режиме, тогда каждой позиции символа на экране отводится два байта памяти. Первый байт содержит номер кода `ASCII` символа, а второй – атрибуты символа. Цветные адаптеры позволяют выводить в цвете как сам символ, так и фон. Монохромный адаптер ограничен только черным и белым цветами, но он позволяет выводить подчеркнутые символы. Преобразование двух байт, в которых размещены код символа и цвет символа, в пиксельное представление выполняется специальным аппаратным устройством – генератором символов – знакогенератором.

Рассмотрим, как формируется байт атрибута `TextAttr`. По умолчанию устанавливается палитра – набор цветов – на основе цветов: `Red`, `Green`, `Blue`.



`Blink=`2^7`=128`

Основная палитра имеет следующие комбинации цветов:

Код цвета	Номер цвета в палитре	Название цвета
Основные цвета		
0000	0	Черный (black)
0001	1	Синий (blue)
0010	2	Зеленый (green)
0100	4	Красный (red)
Смесь основных цветов		
0011	3	Циан (бирюзовый) (cyan)
0101	5	Фиолетовый (magenta)
0110	6	Коричневый (brown)
0111	7	Светло-серый (неярко-белый) (LightGray)

Код цвета	Номер цвета в палитре	Название цвета
Дополнено интенсивностью		
1000	8	Серый (DarkGray)
1001	9	Светло-синий (LightBlue)
1010	10	Светло-зеленый (LightGreen)
1011	11	Светло-циан (LightCyan)
1100	12	Светло-красный (LightRed)
1101	13	С
1110	14	Светло-коричневый (желтый) (yellow)
1111	15	Ярко-белый (white)

Выводы:

Номер бита	Значение бита	Толкование
0	1	Синий включен в основной цвет
1	1	Зеленый включен в основной цвет
2	1	Красный включен в основной цвет
3	1	Символ выводится с высокой интенсивностью
4	1	Синий включен в основной цвет фона
5	1	Зеленый включен в основной цвет фона
6	1	Красный включен в основной цвет фона
7	1	Символы мигают

Описанные комбинации битов образуют 16 регистров 0-й палитры.

Бывают и другие палитры. Они переключаются при помощи прерываний.

Процедуры установки цвета символа `TextColor` и цвета фона `TextBackGround` связаны с переменной `TextAttr`, но можно сразу менять значение переменной:

`TextAttr:=$1F;` 00011111 – ярко-белый на синем фоне;

`TextAttr:=$71;` 01110001 – синим на белом фоне;

`LowVideo` ⇔ `TextAttr:=TextAttr and $F7` (11110111) (установка в ноль бита яркости и контроль его до отмены).

`HighVideo` ⇔ `TextAttr:=TextAttr or $08` (00001000) (установка в 1 бит яркости).

Вместо процедур:
```
TextColor (Yellow+Blink);
TextBackGround (Red);
```
можно написать: `TextAttr:=Yellow+Blink+Red shl 4;`

Задача 1. Рассмотрим программу вывода 40 окон, координаты которых и цвет фона выбираются случайно.

```
program DemoRandomWindow;
uses Crt;
```

```
const  randWx = 80;  randWy = 25;
var    x, y, i : Byte;
begin
Randomize;
ClrScr;
for i:=1  to  40  do
    begin
    x := succ (Random(randWx));
    y := succ (Random(randWy));
    Window(x, y, x+Random(20), y+Random(8));
    TextBackGround (Random(8));
    ClrScr;
    Write(' Window', ^G, i);
    delay(300);
    end;
Window (1, 1, 80, 25);
ClrScr;
GotoXY (33, 25);
Write ('Автор – я!');
Window (1, 1, 80, 24);
ClrScr;
GotoXY (33, 15);
Write ('Okey!');
delay(300);
ClrScr;                    {останется – "Автор – я!"}
end.
```

Текстовые окна

Модуль CRT поддерживает возможность в любой момент работы программы использовать для вывода определенную часть экрана, так называемое *окно*. Размеры окна задает программист процедурой

```
Window (X1, Y1, X2, Y2),
```

где параметры X1, X2, Y1, Y2 типа Byte. (X1, Y1) – координаты верхнего левого, (X2, Y2) – координаты нижнего правого угла окна.

Если следующие условия не выполняются: $1 \le X1 \le X2 \le X_{max}$ и $1 \le Y1 \le Y2 \le Y_{max}$, то окно создано не будет. После выполнения процедуры Window окно становится текущим. Это значит, что все операции с экраном относятся к той его части, которая определена координатами открытого окна. При этом перемещение курсора происходит только в пределах текущего окна, и позиция с координатами (1, 1) – это левый верхний угол окна.

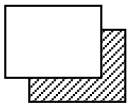
После активизации процедуры Window модуль CRT формирует две специальные переменные WindMin и WindMax типа Word, в которых фиксируются размеры текущего окна (отсчет при этом ведется от (0, 0)). Можно получить значения текущего окна

X1: = Lo (WindMin)+1, Y1: = Hi (WindMin),

X2: = Lo (WindMax)+1, Y2: = Hi (WindMax).

Здесь Lo – младший байт младшего слова своего целочисленного аргумента; Hi – старший байт младшего слова своего аргумента. Эти координаты фиксируются в специальных местах памяти. Местоположение курсора, его вид также фиксируются в оперативной памяти. Эти переменные WindMin и WindMax нам нужны, если окно открыто через случайные координаты.

Задание. В середине экрана сделать окно с тенью следующего вида и окантовать его.



Алгоритм. Создаем темное окно (открываем и закрашиваем). Открываем поверху первого второе окно и закрашиваем светлым цветом; стандартная процедура ClrScr после открытия окна стирает в видеопамяти всё, что попало «под окно». Окантовываем последнее окно символами псевдографики. Открываем новое третье окно так, чтобы окантовка попала в предыдущее окно.

Задача. Написать программу, которая создает несколько не наложенных окон и затем совершает переход из одного окна в другое.

Решение. Для сохранения координат окон, которые создаются, опишем специальные типы и данные.

```
uses Crt;
type
      WinRecord = record
                  xl, yl, xr, yr : Byte
                  end;
const     maxWin = 3;
type      TMW    = array[1..maxWin]  of  WinRecord;
var       i      : Integer;
const
      MW : TMW = ((xl : 10; yl : 5; xr : 15; yr : 10),
                  (xl : 20; yl : 5; xr : 25; yr : 10),
                  (xl : 30; yl : 5; xr : 35; yr : 10));
```

```
begin
ClrScr;
for I := 1  to  maxWin  do
    begin
    with MW[i] do Window(xl, yl, xr, yr);
         ClrScr;
         delay(500);
         end;
    Readln;      {окна созданы по неслучайным координатам.
                           Обращение делаем аналогично.}
    for I := 1  to  maxWin  do
        begin
        with MW[i] do
             Window (xl, yl, xr, yr);
             delay(5000);
             Writeln ('Old');
             end;
end.
```

Отметим, что на экране находится несколько окон, но в каждый отдельный момент времени активным является только одно, с которым связан курсор. Все процедуры ввода-вывода выполняются относительно активного окна.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 14
РЕСУРСЫ МОДУЛЯ GRAPH

Содержание темы

- Графическое программирование.
- Ресурсы модуля GRAPH.
- Базовые процедуры и функции:
  - управление видеостраницами,
  - перемещение курсора,
  - вывод точки и определение параметров пиксела,
  - вывод отрезка.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Стандартное состояние ПК после его включения соответствует работе экрана в текстовом режиме. Любая программа, использующая графические средства компьютера, должна определенным образом инициировать графический режим работы дисплейного адаптера.

Более 80 подпрограмм, содержащихся в модуле `Graph` в файле `GRAPH.TPU`, являются мощными средствами для работы с графической информацией. Подключение модуля стандартное:

```
uses Graph.
```

Настройка графических процедур на работу с конкретным адаптером происходит путем подключения соответствующего графического драйвера. *Драйвер* – это специальная программа, которая осуществляет управление теми или иными техническими средствами ПК. Загрузочные модули драйверов хранятся в специальном файле с расширением `BGI` (`Borland Graphics Interface`). Обычно такие модули записаны в поддиректории `BGI` директории, содержащей `Borland Pascal`.

Для `EGA` и `VGA` адаптеров используется драйвер `EGAVGA.BGI`. В библиотеке модуля `Graph` нет драйверов для новейших адаптеров, и поэтому приходится использовать драйвер `EGAVGA.BGI` и довольствоваться его относительно скромными возможностями.

В графическом режиме экран дисплея условно делится прямоугольной сеткой, каждый элемент которой имеет свои координаты (x, y). Максимальная величина x и y зависит от типа дисплея, от драйвера и от режима его работы. Графическому режиму, как и текстовому, свойственно понятие текущего указателя-курсора, который в графическом режиме невидим.

Образ любого изображения на экране хранится в видеопамяти и восстанавливается примерно через каждые 1/25 с (в текстовом режиме – 1/60с). Там каждому пикселу отводится свой участок (бит, 2 бита, 4 бита, 8 битов и т. д.), где хранится информация о его состоянии и цвете.

Драйвер имеет несколько режимов работы. При инициализации графического режима процедурой `InitGraph` надо указать драйвер, который используется, желательный режим и путь к драйверу.

Если используется драйвер `VGA` (`GraphDriver` = 9), тогда значение режима можно выбирать из следующего множества, приведенного в таблице:

Название режима	Значение константы режима	Разрешение	Количество цветов	Количество видеостраниц
VGA	VGAL0=0 VGAMed=1 VGAHi=2	640×200 640×350 640×480	16 16 16	4 2 1

Драйвер можно задавать зарезервированной константой `detect` (= 0), это означает, что при инициализации графического режима будет происходить определение доступного драйвера и установки лучшей разрешимости экрана.

РЕСУРСЫ МОДУЛЯ GRAPH

Весь набор стандартных графических подпрограмм можно разделить на такие группы:

- процедуры для подготовки графической системы и переход в текстовый режим;
- опрос текущих режимов;
- процедуры для установки параметров изображения (цвет, стиль заполнения, толщина линии и т. д.);
- процедуры для получения изображения на экране;
- процедуры и функции для получения параметров изображения.

При работе с процедурами и функциями модуля Graph могут возникнуть ошибки. Код ошибки возвращается функцией GraphResult и соответствует списку констант: grOk (= 0) {ошибок нет}, grNoInitGraph = –1 графика не инициализирована и т. д. (всего 15 констант).

Процедура GraphErrorMsg (GraphResult) дает текст ошибки. Запомните, что дважды использовать GraphResult для одной и той же операции нельзя – будет плохой результат (флаг ошибок будет сброшен в нуль).

Инициализация и выход из графического режима

Каждый драйвер находится в отдельном файле на диске и содержит выполняемый код и данные. Процедура

```
InitGraph(Graphdriver, Graphmode, Pathdriver)
```

анализирует графическое аппаратное обеспечение, загружает в динамическую память и инициализирует соответствующий графический драйвер, переводит систему в графический режим.

Процедура CloseGraph освобождает память от драйвера и восстанавливает предыдущий видеорежим. Процедура RestoreCrtMode выполняет переход в текстовый режим, а SetGraphMode – в графический. SetGraphMode – устанавливает опять по умолчанию все параметры (палитру, текущий указатель, цвет символов, фон и т. д.).

Очистка экрана проще выполняется процедурой ClearDevice, а сложнее – GraphDefaults. Последняя процедура неявно вызывается при инициализации графического режима.

Пример.

```
program Test;
uses Graph;
var
      GraphDriver : Integer;                    {драйвер}
      GraphMode   : Integer;                    { режим }
      ErrorCode   : Integer;
begin
DirectVideo  :=  false;
GraphDriver  :=  detect;
    {detect   –   для   автоматического   определения   типа
 драйвера аппаратных средств. Если стоит detect, то GraphMode
установится автоматически, иначе надо задавать GraphMode }
```

```
InitGraph(GraphDriver, GraphMode, '');
    {если драйвер в текущем каталоге, то путь задается
     пустой строкой, иначе ставим путь поиска библиотеки}

ErrorCode := GraphResult;
       {после опроса этой функции она сбрасывает
        свои результаты в 0. Так как повторный вызов
        функции GraphResult дал бы не тот результат, то мы
          код ошибки сохраняем в переменной ErrorCode }
if ErrorCode <> grOk then            {  Ошибка!}
  begin
    Writeln('Ошибка графики:',GraphErrorMsg(ErrorCode));
    Writeln ('Программа завершена');
    Halt(1);
   end;
      {*******Работа в графическом режиме*******}
Outtext('  Графический режим. Нажмите <Enter>');
Readln;
       {Задержка экрана, пока не нажмем <ввод> }

RestoreCrtMode;

       {********Работа в текстовом режиме******}
Writeln('Текстовый режим. Нажмите <Enter>');
Readln;
SetGraphMode (GraphMode);
Outtext('Снова графический режим. Нажмите <Enter>');
Readln;
Closegraph;
end.
```

Через соответствующие подпрограммы до или после инициализации графического режима можно получить сведения об установленном режиме:

`GetDriverName` – функция возвращает имя графического драйвера;

`GetGraphMode` – функция возвращает номер графического режима;

`GetMaxMode` – функция возвращает максимальное значение номера режима для текущего загруженного драйвера;

`DetectGraph` – процедура возвращает значение номера текущего драйвера и режима и служит для тестирования графического адаптера;

`GetModeName` – функция возвращает имя заданного графического текущего режима (строка типа '640 × 200 VGA').

Эти процедуры и функции используются для организации диалогового управления графическими режимами.

БАЗОВЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

Рассмотрим далее базовые процедуры и функции в следующей классификации.

- Управление графическими режимами и их анализ.
- Рисование графических примитивов, фигур и перемещение «текущего указателя».
- Управление цветом и заливкой.
- Битовые операции и сохранение графических изображений в памяти.
- Управление страницами.
- Графические окна.
- Управление выводом текста.

Управление видеостраницами

Память адаптеров (видеобуфер) `EGA` и `VGA` имеет более одной видеостраницы. Обычно страницы нумеруются начиная с 0. Для работы с видеостраницами предназначены две процедуры: `SetActivePage` и `SetVisualPage`. Их часто используют при создании анимационных программ.

Процедура

```
SetVisualPage (Page)
```

устанавливает в качестве текущей, т. е. такой, которая отображается на экране, видеостраницу с номером `Page`, а процедура

```
SetActivePage (Page)
```

делает страницу с номером `Page` активной. Это означает, что все графические операции будут происходить на активной странице.

Перемещение курсора

Текущий указатель в графическом режиме невидим, но он присутствует как курсор. Его координаты фиксируется относительно текущей страницы в определенном месте оперативной памяти.

Процедура

```
MoveTo(x, y)
```

перемещает текущий указатель в точку с координатами (x, y). Например, `MoveTo(200, 100).`

Процедура

```
MoveRel(dx, dy)
```

перемещает текущий указатель на `dx` точек по горизонтали и на `dy` – по вертикали. Например, `MoveRel` (5, 10).

Функции `GetX` и `GetY` – возвращают соответственно значения `x`- и `y`-координат текущего указателя. Чтобы координаты не выходили за пределы экрана (в таком случае система не дает ошибки, а отсекает выход), требуется их контролировать функциями

`GetMaxX : Integer` и `GetMaxY : Integer,`

которые возвращают максимально допустимые значения для координат `x` и `y` соответственно.

При помощи этих функций можно следующим образом контролировать принадлежность координат точки экрана:

```
if not((x>GetMaxX) or (y>GetMaxY)) then MoveTo(x, y).
```

Для вычисления координат центра получим

```
Xcenter := GetMaxX div 2;
Ycenter := GetMaxY div 2;
```

Вывод точки и определение параметров пиксела

Изображение точки можно получить процедурой

```
PutPixel (x, y, Color),
```

где `x, y : Integer` – заданные координаты, `Color : Word` – цвет (цвет задается по номеру или по названию из таблицы цветов).

Например, изображение точки в центре экрана зеленым цветом получим вызовом процедуры

`PutPixel` (`Xcenter`, `Ycenter`, 2).

Функция

```
GetPixel(x, y) : Word
```

дает номер цвета пиксела с координатами (`x`, `y`).

Вывод отрезка

Процедура

```
Line(x1, y1, x2, y2)
```

рисует отрезок прямой, с началом в точке (x1, y1) и концом в точке (x2, y2) текущим цветом.

Цвет можно предварительно задать процедурой

```
SetColor(Color),
```

где Color задается номером или именованной константой.

Из текущей точки с приращением dx, dy линию чертит процедура

```
LineRel(dX, dY).
```

Из текущей точки в точку с координатами (x, y) линию чертит процедура

```
LineTo(X, Y).
```

Turbo Pascal позволяет чертить линии различного стиля: тонкие, широкие, штриховые, пунктирные и т. д., а также задавать собственный режим вывода. Стиль линии устанавливается процедурой

```
SetLineStyle(LineStyle, Pattern, Thickness);
```

Здесь LineStyle : Word – тип линии, Pattern : Word – образец, Thickness : Word – толщина.

Тип линии LineStyle может принимать значения, которые приведены ниже в таблице:

Константа типа линии	Тождественное значение	Описание линии
Solidln	0	Непрерывная
Dottedln	1	Линия из точек
Centerln	2	Штрих-пунктир
Dashedln	3	Штриховая линия
UserBitln	4	Тип программиста

Если определяется тип линии из стандартных типов, тогда Pattern=0; если задается пользовательский, т. е. номером 4, тогда Pattern≠0. В этом случае вторым параметром указывается примитив.

Например, примитив $5555 (=0101010101010101$_2$) на месте Pattern показывает, что линия будет перемежаться пробелами и точками по принципу: там, где стоит 0, будет выводиться пиксел цветом фона, а где стоит 1 – цветом линии.

Thickness – толщина линии принимает следующие значения:

NormWidth (= 1) нормальная толщина в 1 пиксел,

ThickWidth (= 3) жирная линия в 3 пиксела.

Если какие-либо параметры, кроме последнего, имеют недопустимые значения, тогда процесс игнорируется, а graphresult принимает значение 11, если же Thickness задан некорректно, тогда толщина берется как NormWidth.

В ряде случаев при использовании Line, LineRel, LineTo и некоторых других процедур требуется устанавливать режим вывода линии на экран (копирование или использование логических операций). Для установки режима вывода кусочно-линейных примитивов предназначена процедура

```
SetWriteMode(Mode).
```

Значение параметра Mode определяется стандартными константами:

```
const
        CopyPut = 0;
        XORPut  = 1;
        ORPut   = 2;
        ANDPut  = 3;
        NOTPut  = 4.
```

Если Mode=0, тогда пикселы, расположенные на отрезке прямой линии, переопределяют пикселы на экране, и, таким образом, линия на экране имеет заданный текущий цвет.

Если Mode=1, то пикселы, образующие линию, имеют код цвета, которые образуется операцией xor кода текущего цвета и кода цвета пикселов на экране, через которые линия проходит.

Как частный случай такого поведения можно стереть выведенную линию с экрана, исполнив вывод линии еще раз, так как

```
A xor B xor B = A.
```

Для определения текущих характеристик линий служит процедура

```
GetLineSettings(LST),
```

где LST : LineSettingsType, а тип LineSettingsType такой:

```
type
     LineSettingsType = record
                         linestyle : Integer;
                         UPattern  : Word;
                         thickness : Integer;
                        end;
```

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 15
ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ОБРАЗОВ

Содержание темы

- Управление параметрами образов:
  - цвета для разных адаптеров,
  - установка цвета,
  - установка палитры,
  - установка стиля заполнения.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

УПРАВЛЕНИЕ ПАРАМЕТРАМИ ОБРАЗОВ

Цвета для разных адаптеров

В CGA-адаптерах используется так называемая RGBI-система (Red, Green, Blue, Intensiv) работы с цветом. На базе трех основных цветов путем смешивания и установки различной яркости свечения формируются четыре палитры. Определенный цвет палитры получается путем смешивания трех основных цветов – синего, зеленого и красного. Если для задания цвета служат четыре бита, то три из них используются для указания цветового составляющей: синего (0001), зеленого (0010) и красного (0100). Четвертый бит управляет яркостью свечения: 1000 – высокая интенсивность свечения, 0000 – низкая. По этому принципу формируется 16 цветов, которые поддерживает для адаптера CGA драйвер CGA.BGI. Такую таблицу цветов мы рассматривали в текстовом режиме.

Новые модели видеоадаптеров – EGA / VGA – имеют ряд конструктивных особенностей, позволяющих значительно расширить возможности работы с цветом. Для задания цвета пиксела здесь используются уже шесть битов, а не четыре, как в CGA. Соответственно расширяется гамма цветов. Принцип их формирования примерно то же, но в качестве основы используется система RrGgBb, где RGB – красный, зеленый и синий цвета нормальной яркости, а rgb – те же цвета, но яркость их в 2 раза меньше.

Для EGA/VGA-адаптеров драйвер EGAVGA.BGI устанавливает 64 цвета. Именованные константы для них начинаются словосочетанием EGA.

Основная палитра имеет следующие комбинации цветов:

Константа	Код цвета	Значение	Название цвета	Компоненты цвета
EGABlack			Черный	
EGABlue			Синий	B
EGAGreen			Зелёный	….G.
EGACyan			Голубой	….GB
EGARed			Красный	…R..
EGAMagenta			Фиолетовый	…R.B
EGABrown			Коричневый	g
EGALightGray			Светло-серый	…RGB
EGADarkGray			Тёмно-серый	rgb…
EGALightBlue			Светло-синий	rgb..B
EGALightGreen			Светло-зелёный	r
EGALightCyan			Светло-голубой	rgb.GB
EGALightRed			Светло-красный	r
EGALighMagentat			Светло-фиолетовый	rgbR.B
EGAYellow			Жёлтый	rgbRG.
EGAWhite			Белый	rgbRGB

Установка цвета

Для различных типов адаптеров количество цветов, которые одновременно отображаются на экране в графическом режиме, может быть разной. Но для всех `BGI`-драйверов она ограничена диапазоном целочисленных значений от 0 до 15. Для того чтобы узнать максимальный номер цвета, который воспринимается данным адаптером в текущем графическом режиме, нужно использовать функцию `GetMaxColor: WORD`. По умолчанию для изображения используется цвет с максимальным номером (т. е. белый), а для фона – с минимальным (черный). Если в процедуре `SetColor(Color)` в качестве `Color` указан недопустимый номер цвета, текущий цвет не меняется. Процедура

```
SetBkColor(Color)
```

устанавливает новый цвет фона, который определяется параметром `Color`.

Получить значения текущих установок цвета можно с помощью двух специальных функций

`GetColor: Word` и `GetBkColor: Word`.

Установка палитры

Палитрой называется максимальный набор цветов, которые поддерживаются драйвером. Она включает 16 цветов (от 0 до 15), которые называются «программными» цветами или цветовыми атрибутами и используются как в текстовом, так и в графическом режимах.

Каждому программному цвету соответствует «аппаратный» цвет из так называемой полной палитры.

В модуле предусмотрен ряд процедур, которые охватывают практически все возможные операции с цветами палитры.

Процедуры и функции для работы с палитрой:

Название	Предназначение
`GetDefaultPalette(Palette)`	Получение информации о текущей палитре
`GetPalette(Palette)`	Размещает в переменную Palette информацию о текущей палитре
`SetPaletteSize:Integer`	Возвращает число цветов в текущей палитре
`SetPalette(ColorNum,Color)`	Используется для установки одного цвета палитры
`SetAllPalette(Palette)`	Используется для установки всех цветов палитры

Установка стиля заполнения

Для заполнения внутренних или внешних областей графических фигур в модуле `Graph` встроена группа наперед определенных (стандартных) комбинаций символов-заполнителей, которые можно назвать *маской*. Маска может окрашиваться в допустимые цвета. Комбинацию цвет-маска называют *стилем заполнения*.

Стандартные стили:

Константа	Значение	Маска (заполнение)
`EmptyFill`	`0`	Цвет фона
`SolidFill`	`1`	Текущий цвет
`LineFill`	`2`	Символы – горизонтальные линии
`LtSlashFill`	`3`	Символы наклонные линии// нормальной толщины
`SlashFill`	`4`	Символы // удвоенной толщины
`BkSlashFill`	`5`	Символы \\ удвоенной толщины
`LtBkSlashFill`	`6`	Символы \\ нормальной толщины
`HatchFill`	`7`	Вертикально-горизонтальная штриховка

Константа	Значение	Маска (заполнение)
XhatchFill	8	Штриховка крест-накрест по диагоналям редкими тонкими линиями
InterLeaveFill	9	Штриховка крест-накрест по диагоналям частыми тонкими линиями
WideDotFill	10	Заполнение «частыми» точками
CloseDotFill	11	Заполнение «редкими» точками
UserFill	12	Заполнение по определенной пользователем маске заполнения

Процедура

```
SetFillStyle(Pattern, Color)
```

устанавливает стиль Pattern и цвет Color. Значение маски Pattern приведено в предыдущей таблице и может быть задано константой или числом, Color берется из установленной палитры. Например:

```
SetFillStyle(SlashFill, Yellow);
Bar(10, 10, 50, 150);
      {столбец заполнен слешами // желтого цвета}
```

Только когда Pattern = UserFill (Pattern = 12), становится активным шаблон (маска), заданный в SetFillPattern.

Если не подходят предопределенные маски и нужно установить свою, используется процедура

```
SetFillPattern(PattMatrix, Color)
```

PattMatrix : FillPatternType – новая маска, Color – ее цвет. Маска занимает матрицу 8×8 (64 пиксела). Для ее задания используется 8 байт (64 бит), причем каждый бит «зажигает» или «гасит» соответствующий пиксел в матрице 8 × 8. Для этого применяют предопределенный 8-байтовый массив типа

```
type FillPatternType = array[1..8] of Byte;
```

Пример.

```
const
     MyPattern : FillPatternType=
                 ($10, $38, $7C, $FE, $7C, $38, $10, $00);
```

А это в двоичной системе счисления имеет вид ромбика:

```
00010000
00111000
01111100
11111110
01111100
00111000
00010000
00000000
```

Необходимую информацию о нестандартных стилях можно получить при помощи процедуры

`GetFillPattern(inf),`

которая выгружает в массив `Var inf : FillPatternType` все содержимое маски.

Процедура

`GetFillSetting(inf),`

где `inf` типа `FillSettingType`, а

```
type
      FillSettingType = record
                          Pattern : Word;
                          Color : Word
                        end
```

дает информацию о текущем стиле.

Процедура

`FloodFill (x, y, border)`

служит для заполнения внутренней или внешней области фигур (эллипсов, окружностей, многоугольников) при помощи стиля, который был установлен процедурами `SetFillStyle` и `SetFillPattern`.

Если `x, y` – координаты внутри фигуры, контур которой имеет цвет `border`, тогда закрашивается внутренняя часть фигуры, а наоборот – выполняется внешнее заполнение текущим образцом зарисовки.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?

ТЕМА 16
ГРАФИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ФИГУР

Содержание темы

- Построение графических фигур.
- Работа с текстом.
- Экран и окно.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

ПОСТРОЕНИЕ ГРАФИЧЕСКИХ ФИГУР

Библиотека `Graph` содержит ряд процедур, формирующих разные фигуры на основе заданных параметров. Цвет, стиль и толщина линии для вычерчивания берутся по умолчанию или устанавливаются соответственно процедурами `SetColor`, `SetFillPattern`, `SetFillStyle`.

Прямоугольники

Изображение контура одномерного прямоугольника дает процедура

```
Rectangle(x1, y1, x2, y2),
```

где `x1, y1` – координаты левого верхнего угла, `x2, y2` – правого нижнего. Область внутри прямоугольника не закрашивается и совпадает по цвету с фоном.

Закрашенный прямоугольник установленным наперед определенным заполнителем и цветом дает процедура

```
Bar(x1, y1, x2, y2).
```

Трехмерный закрашенный прямоугольник (параллелепипед) изображает процедура

```
Bar3D(x1, y1, x2, y2, Delph, Top).
```

Параметр `Delph` – это количество пикселов, которое задает глубину трехмерного контура (чаще всего `Delph := (x2-x1) div 4`). Параметр `Top:Boolean` определяет, строить над прямоугольником вершину (`Top=True`), или нет (`Top=False`).

Многоугольники

Прямоугольники, не ориентированные относительно сторон экрана, можно рисовать разными способами, например, при помощи `Line` или `LineTo`. Но следующая процедура позволяет строить любые многоугольники линией текущего цвета, стиля и толщины:

```
DrawPoly (NumPoints, PolyPoints)
```

Параметр `NumPoints: Word` задает количество целочисленных координат, которые попарно записываются в нетипизированном параметре переменной `PolyPoints`.

Замечание. Для вычерчивания замкнутого `n`-угольника нужно передать (`n` + 1) координату ((`n` + 1)-я совпадает с первой).

При выполнении этой процедуры мы будем использовать тип `PointType`, который введен в модуле `Graph` следующим образом:

```
type
      PointType = record
                   x, y : Integer;
                  end;
```

Например,

```
const
      T : array[1..5] of PointType = ((X :  10; Y :  50),
                  (X : 100; Y :  60), (X : 100; Y : 100),
                  (X :  10; Y : 110), (X :  10; Y :  50));
           …
begin
          DrawPoly (5, T); … end.
```

Будут соединяться первая со второй, вторая с третьей точками и т. д. Чтобы замкнуть фигуру задали массив из пяти элементов и последний элемент равен первому.

Пример. Программа чертит в центре экрана треугольник красными линиями.

```
program DemoDrawPoly;
uses CRT, Graph;
var
          DV, MV                : Integer;
          Pp                    : array [1..4] of PointType;
          xm, ym, ym4, xm4      : Word;
begin
DV      :=  detect;
```

```
InitGraph(DV, MV, '');
xm      := GetMaxX;
ym      := GetMaxY;
xm4     := xm div 4;
ym4     := ym div 4;              {координаты вершин}
pp[1].x := xm4;
pp[1].y := ym4;
pp[2].x := xm-xm4;
pp[2].y := ym4;
pp[3].x := xm div 2;
pp[3].y := ym-ym4;
pp[4]   := pp[1];
SetColor(LightRed);               {цвет для вычерчивания}
DrawPoly(4, pp);
Readln;
CloseGraph
end.
```

При рисовании этой фигуры опросили соответствующими подпрограммами максимальную разрешимость экрана и потом получили образ.

Задание. В центре экрана изобразить окружность случайного радиуса $r \in [10, 20]$ и описать около ее правильный треугольник.

Предыдущий треугольник можно зарисовать, т. е. изменить фон внутри треугольника. Для этого вместо `DrawPoly` используют другую процедуру, но с такими же параметрами:

```
FillPoly (NumPoints, PolyPoints)
```

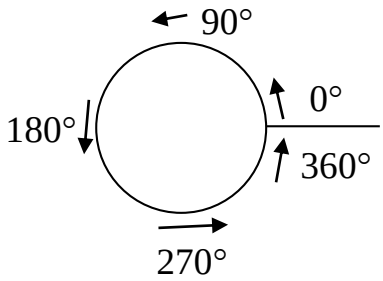
Задача. Нарисовать четырехконечную звезду.

```
program DemoFillPoly;
uses
      CRT, Graph;
const
          Star : array[1..18] of Integer =        {9 пар}
               (75, 0, 100, 50, 150, 75, 100, 100, 75,
                150, 50, 100, 0, 75, 50, 50, 75, 0);
var
      DV, MV     : Integer;
begin
DV           := detect;
InitGraph    (DV, MV, '');
SetFillStyle (1, Green);
```

```
FillPoly (9, Star);          {количество пересечений =8+1}
Readln;
CloseGraph;
end.
```

Построение дуг и окружностей

В таких случаях используется полярная система координат.



Процедура

```
Circle(x, y, R)
```

чертит окружность текущим цветом, где (x, y) – центр окружности, R – радиус (все типа Word). Например:

```
SetColor (LightGreen);
Circle (450, 100, 50);
```

Процедура

```
Arc(x, y, St, End, R)
```

чертит дугу окружности, где St, End – начальный и конечный углы (в градусах) дуги окружности с центром в точке (x, y) и радиуса R (если St = 0, End = 360 – получим полную окружность).

Информацию о координатах последнего обращения к Arc дает следующая процедура:

```
GetArcCoords(ArcCoords).
```

Здесь `var ArcCoords : ArcCoordsType,`

```
ArcCoordsType = record
                  x, y           : Integer;
                  xStart, yStart : Integer;
                  xEnd, yEnd     : Integer
                end;
```

где (x, y) – координаты центра, (xStart, yStart) – начальная позиция, (xEnd, yEnd) – последняя позиция последней команды Arc.

Эта информация, например, бывает нужной для построения сложных фигур.

Для построения эллиптических дуг предназначена процедура

```
Ellipse (x, y, St, End, xR, yR),
```

где `x, y` – координаты центра эллипса, `xR, yR` – длины горизонтальной и вертикальной осей, `St, End` – начальный и конечный углы (все типа `Word`). При `St=0, End=360` получается полный эллипс. Фон внутри эллипса не изменяется. Например,

```
SetColor (EgaLightCyan);
Ellipse (100, 100, 0, 360, 30, 50);
```

Следующая процедура строит и заливает эллипс:

```
FillEllipse (x, y, xR, yR).
```

Заполнитель устанавливается процедурами `SetFillStyle` или `SetFillPattern`.

В программах деловой графики часто требуется разделить окружность или эллипс на секторы и залить их. Это делается процедурами:

```
PieSlice (x, y, St, End, R)     {сектор в окружности}
Sector (x, y, St, End, xR, yR)     {сектор в эллипсе}
```

Секторы рисуются текущим цветом, а при заливке используются тип и цвет, заданные через `SetFillStyle` и `SetFillPattern`.

Пример. В следующей программе на экране имитируется вид часового циферблата, однако ход часов выбран без связи с системными часами.

```
uses Graph, Crt;
var
    d, r, r1, r2, rr, c, x1, y1, x2, y2, x01, y01, k
                      : Integer;
          Xasp, Yasp : Word;
               XYasp : Real;
begin
d := detect;
InitGraph (d, r, '');
c := GraphResult;
if c <> Grok then Writeln (GraphErrorMsg(c))
   else
   begin
   X1 := GetMaxX div 2;
   Y1 := GetMaxY div 2;
   GetAspectRatio(Xasp, Yasp);     {подсчет радиусов}
```

```
   XYasp := Yasp / Xasp;
   r := round(3 * GetMaxY / 8/XYasp);
   r1 := round(0.9 * r);           {часовые деления }
   r2 := round(0.95 * r);          {минутные деления}
   Circle(x1, y1, r);              {циферблат       }
   Circle(x1, y1, round(1.02 * r));
   for k := 0 to 59 do
      begin
       if k mod 5=0 then rr := r1 {часовые деления }
          else rr := r2;          {минутные деления}
       X01 := X1+Round(rr*sin(2*pi*k/60));
       X2  := X1+Round(r*sin(2*pi*k/60));
       Y01 := Y1-Round(rr*cos(2*pi*k/60)*XYasp);
       Y2  := Y1-Round(r*XYasp*cos(2*pi*k/60));
       Line(X01, Y01, X2, Y2);     {деления}
      end;
   SetWRITEMode(XORPut);
   SetLineStyle(Solidln, 0, ThickWidth);
   r := 0;         {счетчик минут в каждом часе}
   repeat
     for k := 0 to 59 do
     if not KeyPressed then
      begin             {координаты часовой стрелки}
       X2  :=X1+Round(0.85*r1*sin(2*pi*r/60/12));
       Y2  :=Y1-Round(0.85*r1*XYasp*cos(2*pi*r/60/12));
       X01 :=X1+Round(r2*sin(2*pi*k/60));
       Y01 :=Y1-Round(r2*XYasp*cos(2*pi*k/60));
                         {координаты минутной стрелки}
       Line(X1, Y1, X2,  Y2);
       Line(X1, Y1, X01, Y01);
       Delay(1000);
       Line(X1, Y1, X2, Y2);
       Line(X1, Y1, X01, Y01);   {стерли вторым выводом}
       inc(r);
       if r=12*60 then r := 0;
      end;
   until KeyPressed;
   if Readkey=#0 then k := Ord(Readkey);
                              {очистили буфер клавиатуры}
   CloseGraph;
   end
end.
```

РАБОТА С ТЕКСТОМ

Задание шрифтов

В комплект поставки Turbo Pascal включается набор штриховых шрифтов. Файлы штриховых шрифтов имеют расширение `*.CHR`. В штриховых шрифтах при построении символа используется не матричный (как в стандартных шрифтах для текстового режима), а векторный способ. Это дает широкие возможности манипуляции размерами шрифтов без ухудшения качества их изображения. Для доступности штриховых шрифтов нужно, чтобы при инициализации графического режима были доступны файлы с расширением `*.CHR`, которые их содержат. По умолчанию подключается матричный (битовый) шрифт.

Имена встроенных шрифтов приведены в таблице.

Наперед определенные константы в модуле Graph	Значение	Пояснения
`DefaultFont`	`0`	8×8 стандартный битовый шрифт
`TriplexFont`	`1`	Штриховые шрифты
`SmalFont`	`2`	Малый шрифт
`SansSerifFont`	`3`	Шрифт без засечек
`GothicFont`	`4`	Готический шрифт

Все эти шрифты содержат только первую часть `ASCII`-таблицы (0 ÷ 127), и только `DefaultFont` имеет символы кириллицы (если системно подключена вторая часть таблицы с кириллицей).

Установить нужный шрифт можно процедурой

```
SetTextStyle(Font, Direction, CharSize).
```

Здесь `Font : Word` – номер выбранного шрифта (из таблицы), `CharSize: Word` – размер выводимых символов (1≤`CharSize`≤ 10). Если шрифт матричный (`Font` = `DefaultFont`), тогда при `CharSize` = 4 каждый символ, закодированный матрицей 8 × 8, будет выводиться на основе матрицы 32 × 32 пиксела. `Direction : Word` – направление:

`Horizdir`	0	Слева направо
`Vertdir`	1	Снизу вверх (на 90° повернутый)

Векторные шрифты в основе построения символа реализуют такую идею: в некоторой системе координат описывается последовательность прохождения контура, который образует символ, относительно предыдущей точки контура. Значит, и модификация шрифта (увеличение, расширение и т. д.) происходит простым умножением этих координат на соответствующее число.

Нужный размер шрифта можно установить процедурой

```
SetUserCharSize (multX, divX, multY, divY),
```

где отношения `n = multX / divX` и `m = multY / divY` установят ширину и высоту нового шрифта соответственно. Это означает, что, например, при матричном шрифте (8 × 8) каждый пиксел отображается матрицей `n×m`.

Функции

```
TextHeight (Textstring) : Word  и
TextWidth  (Textstring) : Word
```

дают сведения о высоте и толщине строки `Textstring` в пикселях. Эта информация нужна, чтобы рассчитать размеры текста для вывода на экран.

Текст выводится относительно текущей позиции (`x, y`). Текст можно выводить, совмещая координаты вывода с центром строки, или правым, или левым краем строки, или относительно толщины строки, строку поднять вверх или опустить вниз.

Вариант ориентировки задается процедурой

```
SetTextJustify (Horizontal, Vertical),
```

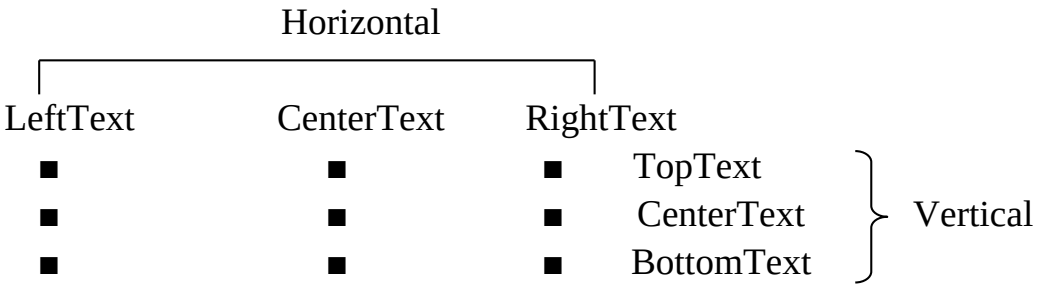
где `Horizontal : Word` содержит значения:

```
0 = LeftText;        1 = CenterText; 2 = RightText;
```

а `Vertical : Word` содержит значения:

```
0 = BottomText; 1 = CenterText; 2 = TopText;
```

На схеме знак ■ показывает координаты точки (`x, y`)



Вывод текста

Процедура

```
Outtext (TextString)
```

выводит текст `Textstring : String`, начиная с позиции `(x,y)` текущего указателя, а процедура

```
OuttextXY (x, y, Textstring)
```

выводит текст относительно заданных координат (x,y) установленным стилем, цветом и ориентировкой. Если текст не помещается в окно, то он отсекается (в случае шрифта defaultfont теряется вовсе).

Отметим, что текстовые процедуры GotoXY, Write/Writeln и установка цвета текста (TextBackGround, TextColor) в графическом режиме работают, только если переменная Crt.DirectVideo=false (или модуль Crt не подключен). Ввод Read/Readln действует всегда, но при этом текст ввода стирает часть экрана.

Процедура GettextSettings дает сведения о шрифте, направлении вывода, размерам символа и ориентации текущего текста.

Для вывода числовых значений предварительно их нужно перевести в строку символов процедурой Str.

Пример.

```
Max := 34.56;
Str(Max : 6 : 2, Smax);
OuttextXY (400, 40, 'max='+Smax);
```

Задача. В центре экрана вывести 8 строк разными шрифтами и обвести их рамками.

```
program DemotextFrame;
uses Graph;
var
     Driver, mode  : Integer;
     St, st1       : String;
     Height, width,
     cx, cy, x1, x2,
     y1, y2, i     : Integer;
     tinf          : TextSettingsType;
begin
     St      :=  'AaBbCc…';
     Driver  :=  detect;
     InitGraph(Driver, Mode, '');
     Cx      :=  GetMaxX div 2;
     Cy      :=  GetMaxY div 2;
     SettextJustify(CenterText, CenterText);
     for i := 1 to 8 do
       begin
          Settextstyle (i, 0, 1);
          clearviewport;
          Gettextsettings(tinf);
```

```
            str(tinf.font : 2, st1);
            st1    := 'Font : '+st1+'  '+st;
            Height := (Textheight(st1)+4) div 2;
            Width  := (Textwidth (st1)+4) div 2;
            x1     := cx-width;
            x2     := cx+width;
            y1     := cy-height;
            y2     := cy+height;
            SetColor(15);
            Rectangle(x1, y1, x2, y2);
            SetColor(11);
            OuttextXY (CX, CY, st1);
            Readln;
          end;
       Closegraph;
end.
```

ЭКРАН И ОКНО

Окно – это прямоугольная область экрана, которая выполняет все функции полного экрана.

После установки окна вся остальная площадь экрана как бы не существует, и весь ввод-вывод осуществляется только через окно. В каждый отдельный момент может быть активным только одно окно. Если окон несколько, за переключение ввода-вывода в нужное окно отвечает программист.

Процедура

```
SetViewPort(x1, y1, x2, y2, Clip)
```

создает окно, где `x1, y1` – координаты левого верхнего угла, `x2, y2` – правого нижнего. Параметр `Clip: Boolean` определяет, будет ли изображение отсекаться при выходе за пределы окна (тогда `Clip := true`), или нет (тогда `Clip := false`). Значит, создав окно, можно получить локальную систему координат и пользоваться отрицательными координатами.

Процедура

```
SetBrColor(Color)
```

задает цвет фона.

Процедура

```
ClearViewPort
```

очищает текущее окно.

Окно существует независимо от текущей страницы и не привязано к странице. По умолчанию сразу окно занимает весь экран, его размеры задаются процедурой инициализации `InitGraph`.

После очистки окна текущий указатель устанавливается в левый верхний пункт окна с координатами (0, 0) – это внутренние координаты окна.

Координатную систему полного экрана можно вернуть, установив новое окно `SetViewPort(0, 0, GetMaxX, GetMaxY, True)`, или выполнить процедуру `GraphDefauls`.

Атрибуты текущего окна можно получить при помощи процедуры

`GetViewSetting(Vp)`.

Здесь параметр `Vp` описан так: `var Vp : ViewPortType`, а `ViewPortType` – следующий тип:

```
type
          ViewPortType = record
                           x1, y1, x2, y2 : Integer;
                           Clip               : Boolean;
                           end;
```

Процедура `ClearViewPort` заливает текущим фоном весь экран и поэтому не выделяет границ окна. «Заливки» графического окна цветом, который отличается от общего фона экрана, делают так:

- процедурой `SetFillStyle` устанавливается шаблон и цвет заполнения;
- процедурой `Bar` – выводится залитый прямоугольник;
- открывается окно с теми же параметрами, что и в `Bar`.

Пример.

```
SetFillStyle (1, 4);      {1 – заполнение сплошным цветом,
                           4 – номер цвета Red (красный)}
Bar(100, 50, 500, 200);
SetViewPort(100, 50, 500, 200);
```

Следующая программа под случайное музыкальное сопровождение рисует разноцветные окна.

```
program Demo_SetViewPort;
uses
      Crt, Graph;
var
         I      : Integer;
         DV, DM : Integer;
```

```
begin
        DV := detect;
        InitGraph (DV, DM, '');
        SetViewPort (10, 10, 630, 320, True);
        i := 1;
        repeat
                i := i+1;
                Sound (Random(180)+40+i);
                Delay (Random(170));
                SetFillStyle (Random(4), Random(16));
                Bar (10, 10, 630, 320);
                NoSound;
                Delay(100)
        until KeyPressed;
        Readln;
        CloseGraph;
end.
```

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

ТЕМА 17
ПРОГРАММИРОВАНИЕ АНИМАЦИИ

Содержание темы

- Манипулирование фрагментами образов.
- Анимация.

Целеполагание студента

- Какие цели Вы перед собой ставите в рамках данной темы?
- Что интересует?
- Как данный материал повлияет на Ваше умение программировать?

МАНИПУЛИРОВАНИЕ ФРАГМЕНТАМИ ОБРАЗОВ

Использование оперативной памяти для хранения графических образов, чтобы в дальнейшем быстро восстановить этот образ на экране, полезно, например, для программирования образов, которые движутся.

Процедура

```
GetImage (x1, y1, x2, y2, BitMap)
```

сохраняет в оперативной памяти в переменной `BitMap` образ прямоугольной области графического экрана:

- `(x1, y1)` – координаты верхнего левого угла прямоугольной области,
- `(x2, y2)` – координаты нижнего правого угла.

Размер `BitMap` должен быть на 4 байта больше, чем необходимо для сохранения образа заданного объема (в этих байтах запоминаются ширина и высота прямоугольной области экрана, которая сохраняется). Переменная `BitMap` обычно располагается в `Heap`, где ей предварительно процедурой `GetMem` отводится память.

Функция

```
ImageSize(x1, y1, x2, y2) : Word
```

возвращает количество байт, необходимых для сохранения прямоугольной области графического образа экрана с координатами (`x1`, `y1`), (`x2`, `y2`). Если нужно памяти более 64 кб, тогда `ImageSize` равно 0, а `GraphResult` = –11, и сохранять образ нужно будет по частям. Величина образа зависит от драйвера и режима, а значит, от количества битов в видеопамяти для одного пиксела.

Процедура

```
PutImage(x, y, BitMap, Mode)
```

выводит на экран сохраненный в BitMap образ прямоугольной области графического образа экрана. Место вывода задается координатами (x, y) – верхний левый угол (а ширина и высота находятся в BitMap). Переменная Mode задает режим вывода на экран. Ее значение соответствует одному из вариантов следующего списка констант:

```
CopyPut   =0;       {MOV}
NormalPut =0;       {MOV}
XORPut    =1;       {xor}
ORPut     =2;       {or}
AndPut    =3;       {and}
NotPut    =4;       {not}
```

Каждая из этих констант соответствует записанной в комментарии бинарной операции между байтами видеопамяти и байтами сохраненного образа. Режим XORPut обеспечивает вывод с видимостью на любом фоне. Повторный вывод с операцией xor даст предварительное значение экрана, поскольку A xor B xor B = A.

Задача. Создать простой вариант движения закрашенного круга, который «падает» с левого верхнего угла в правый нижний.

Решение.

```
program PR_Graph;
uses Crt, Graph;
const
      R  = 20;
      SX : Word=20;
      SY : Word=20;
var
      GD, GM : Integer;
      P      : Pointer;
      Size   : Word;
begin
       GD := Detect;
       InitGraph (GD, GM, '');
       if GraphResult < > grOk then Halt(1);
       Circle (SX, SY, R);
       FloodFill(SX, SY, GetColor);
                                        {зарисовали круг}
       Size := ImageSize (SX-R, SY-R, SX+R, SY+R);
```

```
                        {сколько памяти надо запросить}
      GetMem (P, Size);
            {выделили память для хранения образа в Heap}
      GetImage (SX-R, SY-R, SX+R, SY+R, P^);
                              {записали в буфер в Heap}
      repeat
          Delay(300);
          PutImage (SX-R, SY-R, P^, xorPut);
                    {стираем изображение на старом месте}
          Inc (Sx, 6);
          Inc (Sy, 3);
          PutImage (SX-R, SY-R, P^, XorPut);
                  {пересылаем изображение на новое место}
       until (SX > GetMaxX-2*R) or (SY > GetMaxY-2*R);
       Readln;
       CloseGraph;
       FreeMem (P, Size);
end.
```

АНИМАЦИЯ

Первый вариант. Если драйвер обеспечивает работу со страницами графических образов, тогда можно чередовать страницы на экране и получать анимационные фрагменты.

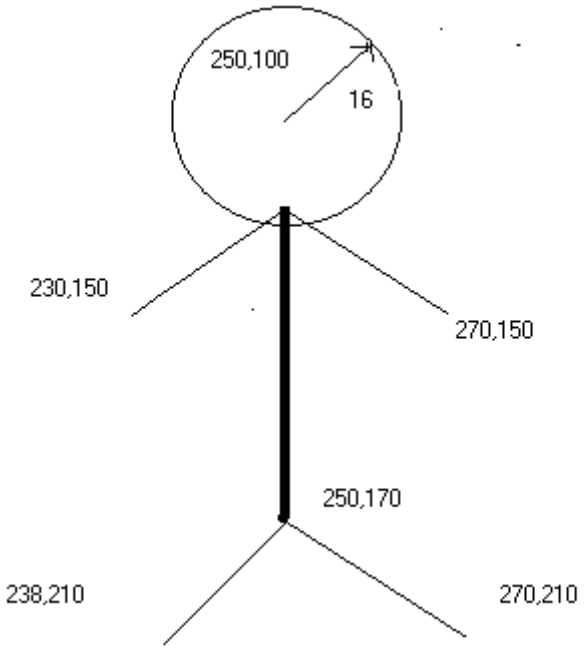
Второй вариант. Выводим рисунок, потом выводим его же цветом, совпадающим с цветом фона, – изображение исчезает, затем таким же методом выводим его в другом месте. Можно при выводе рисунка использовать операцию `xor`. Такой вариант подходит для небольших изображений на одноцветном фоне.

Третий вариант. Выводим изображение, очищаем экран (`ClearViewPort`). Выводим новый образ, очищаем экран и т. д. Такой метод не дает качественных рисунков.

Четвертый вариант. Выводим рисунок, процедурами `PutImage` и `GetImage` какую-то его часть храним в динамической памяти и нужное количество раз восстанавливаем на новых местах.

Простая анимация

Рассмотрим простую анимацию на примере движения конечностей какого-то мультяшного человечка. В системе экранных координат, рассчитаем координаты образа и получим, например, следующую схему для анимации.



Соответственно схеме напишем программу, уточнив некоторые детали.

```
program Demo_Simple_Anim;
uses
         Crt, Graph;
const
       Ver : array [1 .. 3] of Integer=(150, 112, 76);
       Hor : array [1 .. 3] of Integer=(230, 200, 191);
var       DriverVar,  ModeVar,  i : Integer;
begin
          DriverVar  :=  detect;
          InitGraph (DriverVar, ModeVar, '');
          while not Keypressed do
           begin
               SetColor(15);
               Circle(250, 100, 16);          {     голова}
               Line(250, 117, 250, 170);      {   туловище}
               Line(250, 170, 238, 210);      {левая  нога}
               Line(250, 170, 270, 210);      {правая нога}
               for i := 1 to 3 do
                   begin
                 SetColor(15);
                 Line(250, 120, Hor[i], Ver[i]);
                 Line(250, 120, 250+(250-Hor[i]), Ver[i]);
                 Delay(1000);
                 SetColor(0);
```

```
                Line(250, 120, Hor[i], Ver[i]);
                Line(250, 120, 250+(250-Hor[i]), Ver[i]);
                 end;
          end;
CloseGraph;
end.
```

Задание. Человечек должен подкидывать ногами мяч, который движется по дуге.

Рефлексия

- Каковы были ваши цели перед занятием и насколько их удалось реализовать?
- Перечислите трудности, с которыми вы столкнулись при изучении темы?
- Каков главный результат для вас лично при изучении темы?
- Каким образом вы преодолевали трудности? За счет чего?
- Чему вы научились лучше всего?
- Что вам удалось больше всего при изучении темы и почему?
- Что не получилось и почему?
- Опишите динамику Ваших чувств и настроений при изучении темы (можно в виде графика).

СПИСОК ЛИТЕРАТУРЫ

Расолька, Г. А. Pascal: тэорыя і практыка праграміравання : вучэб.-метад. дапам. / Г. А. Расолько, Ю. А. Кремень. – Мінск : БДУ, 2008.

Расолька, Г. А. Метады праграміравання. Алгарытмы апрацоўкі данных / Г. А. Расолько, Ю. А. Кремень. . – Мінск : БДУ, 2008.

Расолько, Г. А. Задания вычислительной практики по курсу «Методы программирования и информатика» : практикум : в 2 ч. / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2020. – Ч. I. – 66 с.
https://elib.bsu.by/handle/123456789/248827

Расолько, Г. А. Задания вычислительной практики по курсу «Методы программирования и информатика» : практикум : в 2 ч. / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2020. – Ч. II. – 85 с.
https://elib.bsu.by/handle/123456789/248828

Расолько, Г. А. Сборник задач по курсу «Методы программирования и информатика» : практикум : в 2 ч. / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2020. – Ч. I. – 97 с. https://elib.bsu.by/handle/123456789/248829

Расолько, Г. А. Теория и практика программирования на Pascal / Г. А. Расолько, Ю. А. Кремень. – Минск : Выш. шк., 2015.

Расолько, Г. А. Электронный учебно-методический комплекс по учебной дисциплине «Методы программирования и информатика» для специальностей: 1-31 03 01 «Математика (по направлениям)» «1-31 03 01-02 Математика (научно-педагогическая деятельность)» / Г. А. Расолько, Е. В. Кремень, Ю. А. Кремень. – Минск : БГУ, 2014. – 90 с. http://elib.bsu.by/handle/123456789/97905