наблюдаем отличие от требования языков C и C++, в которых идентификаторы регистро-зависимы. Зато в Паскале регистр имен не имеет значения.

2. Аргументы передаются вызываемой функции через стек. Если аргумент укладывается в 32-битное значение и не подлежит модификации вызываемой функцией, он обычно записывается в стек непосредственно. В остальных случаях программист должен разместить значение аргумента в памяти, а в стек записать 32-битный указатель на него. Таким образом, все параметры, передаваемые функции API, представляются 32-битными величинами.

3. Вызывающая программа загружает аргументы в стек последовательно, начиная с последнего, указанного в описании функции, и кончая первым. После загрузки всех аргументов программа вызывает функцию командой call.

4. За возвращение стека в исходное состояние после возврата из функции API отвечает сама вызываемая функция. Программисту заботиться о восстановлении указателя стека esp нет необходимости. Эта идея тоже пришла из реализаций Паскаля и прилично экономит на размере кода.

5. Вызываемая функция API гарантированно сохраняет регистры общего назначения ebp, esi, edi. Регистр eax, как правило, содержит возвращаемое значение. Состояние остальных регистров после возврата из функции API следует считать неопределенным (полный набор соглашений stdcall регламентирует также сохранение системных регистров ds и ss. Однако для flat-модели памяти, используемой в win32, эти регистры значения не имеют.)

В применении к нашему примеру это означает следующее:

- аргументы мы должны передать через стек;
- сама фунция вызывается командой call;
- не "парим" себе мозги, задаваясь вопросом, "если мы сделали PUSH, то когда же нам сделать POP?", ибо правильный ответ — никогда. Об этом позаботиться сама функция.

#13. И, напоследок, топик #13 ;)

Читая эту главу, вы написали свою первую программу для Windows на языке ассемблера. Если вы уже имеете хотя бы небольшой опыт программирования, то даже беглого прочтения этой главы будет достаточно, чтобы сказать: "да, теперь я знаю, как ЭТО делается на асме". Если же вы еще совсем молоды и зелены, то просто "тупо" выполните мои инструкции, а комментарии расцените как своего рода предварительные наброски тех тем, которые мы подробно будем рассматривать впоследствии.

Более того, это единственная глава, которая действительно необходима для обучения программированию под Windows на ассемблере. Как только программист узнает, как вызывать API, все остальное он станет способен делать самостоятельно, без учебников и подсказок. Потребуется только справочник Platform SDK, да знание двух языков: С и английского (оба — в объеме церковно-приходской школы).

И да пребудет с вами сила!

Portion by Svet(r)off

(Продолжение следует)

Программирование алгоритмов с использованием файлов

А.ШАКИРИН,
г.Минск, БАГУ, каф.ВТ.

Цель этой работы — освоить применение компонентов OpenDialog и SaveDialog и создать приложение, в котором используются файлы.

1. Пример создания приложения

Необходимо создать Windows-приложение для записи в файл и чтения из файла ведомости успеваемости 10 учащихся. Каждая запись файла должна содержать фамилию, инициалы, а также оценки по физике, математике и сочинению. Следует вывести список учащихся, отсортированный в алфавитном порядке, и записать эту информацию в текстовый файл.

1.1. Размещение компонентов на Форме

Один из возможных вариантов панели интерфейса создаваемого приложения показан на **рис.1**.

При работе с файлами чтение и запись информации удобно организовывать с помощью компонентов OpenDialog и SaveDialog.

Компоненты OpenDialog и SaveDialog находятся на странице Dialogs. Все компоненты этой страницы являются невизуальными, т.е. не видны в момент выполнения приложения. Поэтому их можно разместить в любом удобном месте Формы. Оба рассматриваемых компонента имеют идентичные свойства и отличаются только внешним видом.

Для установки компонентов OpenDialog и SaveDialog на Форму необходимо на странице Dialogs Палитры Компонентов щелкнуть мышью соответственно по пиктограмме  или  и разместить ее в любом свободном месте Формы. При выполнении приложения, в момент вызова компонента, появляется диалоговое окно, с помощью ко-

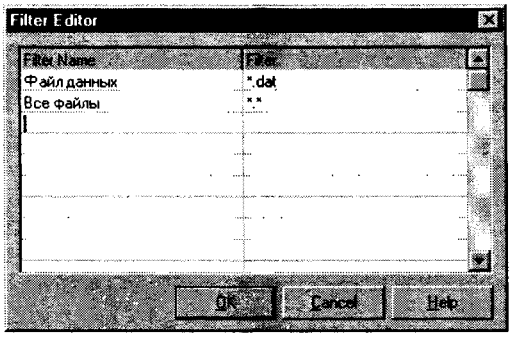
Рис. 1

торого пользователь выбирает имя файла и маршрут к нему. В случае успешного завершения диалога имя выбранного файла и маршрут поиска содержатся в свойстве FileName.

Пользователь имеет возможность настроить параметры окна диалога по своему усмотрению. В частности, изменить заголовок окна можно с помощью свойства Title. В свойстве DefaultExt можно указать расширение файла, если оно не задано пользователем. Свойство Filter используется для поиска (фильтрации) файлов, отображаемых в окне.

Установка фильтра производится следующим образом. Выделив соответствующий компонент, необходимо дважды щелкнуть по правой (белой) части свойства Filter Инспектора Объектов. В появившемся окне редактора фильтра (Filter Editor) необходимо в колонке Filter Name набрать текст, характеризующий соответствующий фильтр, а в колонке Filter — маску. Для компонента OpenDialog1 установим значения масок, как показано на **рис.2**.

Рис. 2



Маска *.dat означает, что будут видны файлы данных с любым именем и с расширением dat, а маска *.* — что будут видны все файлы (с любым именем и с любым расширением).

Для того чтобы файл автоматически записывался с расширением dat, в свойстве DefaultExt запишем требуемое расширение — dat. Аналогичным образом настроим компонент SaveDialog1 для текстового файла (расширение — txt).

1.2. Создание процедур обработки событий

Для удобства работы с несколькими различными процедурами обработки событий в свойстве Name каждого компонента BitBtn необходимо заменить программные имена кнопок: BitBtn1 — на BitBtnNew, BitBtn2 — на BitBtnOpen, BitBtn3 — на BitBtnSort, BitBtn4 — на BitBtnSave. Двойным нажатием клавиши мыши на кнопках BitBtn создаются соответствующие процедуры обработки событий. Пользуясь текстом модуля UnFile, внимательно наберите операторы этих процедур.

1.3. Текст модуля UnFile

```
Unit UnFile;

Interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,StdCtrls, Buttons, Grids, Spin;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    BitBtnNew: TBitBtn;
    BitBtnOpen: TBitBtn;
    BitBtnSort: TBitBtn;
    BitBtnSave: TBitBtn;
    SaveDialog1: TSaveDialog;
    SpinEdit1: TSpinEdit;
    Label1: TLabel;
    OpenDialog1: TOpenDialog;
    procedure TForm1.FormCreate(Sender: TObject);
    procedure BitBtnSortClick(Sender: TObject);
    procedure BitBtnNewClick(Sender: TObject);
    procedure SpinEdit1Change(Sender: TObject);
    procedure BitBtnOpenClick(Sender: TObject);
    procedure BitBtnSaveClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

 Implementation

 {$R *.DFM}

type
 zap=record                    // объявление  записи
      fio          :string[20];
     mat,fiz,soch:integer;
     end;
var
 MZap:array[1..25] of zap;     // объявление  массива записей
 FileZap:file of zap;          // объявление файла записей
 FileText:TextFile;            // объявление текстового файла
 FileNameZap,FileNameText:string; // имена файла записей и
                                  //текстового файла
 n:integer;                    // текущее кол-во элементов
                               //массива записей
{ Обработчик  события создания Формы }
procedure TForm1.FormCreate(Sender: TObject);
begin
 with StringGrid1 do
 begin              // занесение информации в ячейки  StringGrid1
  Cells[0,0]:='Fпп';
  Cells[1,0]:='Фамилия,инициалы';
  Cells[2,0]:='Математика';
  Cells[3,0]:='Физика';
  Cells[4,0]:='Сочинение';
 end;
 BitBtnSort.Hide;     // спрятать кнопку "Sort"
 BitBtnSave.Hide;     // спрятать кнопку "Save"
end;

{ Обработчик нажатия кнопки Sort }
procedure TForm1.BitBtnSortClick(Sender: TObject);
var
 i,j :integer;
 vper:zap;
begin
  for i:=1 to n do
   with StringGrid1,MZap[i] do
    begin
     fio:=Cells[1,i];
     mat:=StrToInt(Cells[2,i]);
     fiz:=StrToInt(Cells[3,i]);
     soch:=StrToInt(Cells[4,i]);
    end;
{ сортировка методом "пузырька"}
 for i:=2 to n do
  for j:=n downto i do
```


```
   if MZap[j-1].fio>MZap[j].fio then
                      begin
                        vper:=MZap[j-1];
                        MZap[j-1]:=MZap[j];
                        MZap[j]:=vper;
                       end;
  for i:=1 to n do    // заполнение ячеек StringGrid1 полями
                      //массива записей
   with StringGrid1,MZap[i] do
    begin
      Cells[0,i]:=IntToStr(i);
      Cells[1,i]:=fio;
      Cells[2,i]:=IntToStr(mat);
      Cells[3,i]:=IntToStr(fiz);
      Cells[4,i]:=IntToStr(soch);
    end;
 end;

  { Создание нового файла записей}
procedure TForm1.BitBtnNewClick(Sender: TObject);
var
 i:integer;
begin  // вывод на экран окна с предупреждающим сообщением
  if MessageDlg('Содержимое существующего файла будет'+
  'уничтожено. Вы уверены?',
   mtConfirmation, mbYesNoCancel, 0)=mrYes then
 begin
 for i:=1 to n do
   with StringGrid1,MZap[i] do
    begin             // формирование полей массива записей
    fio:=Cells[1,i];
    mat:=StrToInt(Cells[2,i]);
    fiz:=StrToInt(Cells[3,i]);
    soch:=StrToInt(Cells[4,i]);
    end;
 with OpenDialog1 do
  begin
  Title:='Создание файла'; // заголовок окна диалога
  if Execute then          // выполнение стандартного диалога
                           //выбора имени файла
   begin
   FileNameZap:=FileName; // присваивание имени файла
   AssignFile(FileZap,FileNameZap); // назначить файлу FileZap
                                    //имя FileNameZap
   ReWrite(FileZap);                // открыть файл на запись
   for i:=1 to n do
    write(FileZap,MZap[i]); // запись в файл массива записей
   CloseFile(FileZap);              // закрытие файла записей
   end;
  end;
 end;
end;

{Обработчик  кнопки изменения размера}
procedure TForm1.SpinEdit1Change(Sender: TObject);
var
 i,m:integer;
begin
m:=StrToInt(SpinEdit1.Text); // присвоить новое значение размера
with StringGrid1 do
 begin
  RowCount:=m+1; // пересчитать количество строк
  if m>n then           // если строки добавлены то
   for i:=n+1 to m do // инициализировать новые ячейки
    begin
     Cells[0,i]:=IntToStr(i);
     Cells[1,i]:='';
     Cells[2,i]:='';
     Cells[3,i]:='';
     Cells[4,i]:='';
    end;
 end;
 n:=m;                // запомнить новое значение размера
end;

{ обработчик нажатия кнопки Open }
procedure TForm1.BitBtnOpenClick(Sender: TObject);
var
 i:integer;
begin
 with OpenDialog1 do
  begin
   Title:='Открытие файла'; // заголовок окна диалога
   if Execute then          // выполнение стандартного диалога
                            // выбора имени файла
   begin
   FileNameZap:=FileName;  // присваивание имени файла
   AssignFile(FileZap,FileNameZap); // назначить файлу FileZap
                                    // имя FileNameZap
   ReSet(FileZap);          // открыть файл на чтение
   n:=0;                    // инициализация счетчика кол-ва
                            // прочитанных из файла элементов
   while not EoF(FileZap) do
    begin
     n:=n+1; // увеличение счетчика
     read(FileZap,MZap[n]); // чтение из файла n-го элемента
                            // массива записей
    end;
   SpinEdit1.Text:=IntToStr(n); // занести кол-во прочитанных
                                //записей в SpinEdit
   StringGrid1.RowCount:=n+1;   // присвоить кол-во строк
   for i:=1 to n do
    with StringGrid1,MZap[i] do
     begin  // заполнение ячеек StringGrid1 полями массива записей
     Cells[0,i]:=IntToStr(i);
     Cells[1,i]:=fio;
     Cells[2,i]:=IntToStr(mat);
     Cells[3,i]:=IntToStr(fiz);
     Cells[4,i]:=IntToStr(soch);
     end;
   CloseFile(FileZap); // закрытие файла записей
   end;
  end;
BitBtnSort.Show;             // показать кнопку "Sort"
BitBtnSave.Show;             // показать кнопку "Save"
end;

{ Сохранение данных в текстовом файле}
procedure TForm1.BitBtnSaveClick(Sender: TObject);
var
 i:integer;
begin
 with SaveDialog1 do
 if Execute then    // выполнение стандартного диалога выбора
                    //имени файла
  begin
  FileNameText:=FileName;   // присваивание имени файла
  AssignFile(FileText,FileNameText); // назначить файлу
                                     // FileText имя FileNameText
  ReWrite(FileText);        // открыть текстовый файл на запись
  for i:=1 to n do
   with MZap[i] do          // запись в текстовый файл
   writeln(FileText,i:3,fio:20,mat:5,fiz:5,soch:5);
  CloseFile(FileText);      // закрытие текстового файла
                            // по окончании записи
  end;
 BitBtnSort.Hide;            // спрятать кнопку "Sort"
 end;
end.
```

1.4. Работа с приложением

Запустите созданное приложение. Занесите в соответствующие поля панели интерфейса информацию об ус-

певаемости учащихся. Кнопкой "New" сохраните данные в файле. Завершите выполнение приложения.

Вновь запустите приложение и кнопкой "Open" откройте только что созданный файл. Убедитесь, что информация не содержит ошибок. При необходимости, обнаруженные ошибки можно исправить, а также дополнить ведомость новой информацией. Для сортировки ведомости в алфавитном порядке воспользуйтесь кнопкой "Sort" и сохраните отсортированную информацию кнопкой "New".

Еще раз завершите и вновь запустите приложение.

Кнопкой "Open" откройте файл и убедитесь, что в нем теперь содержится ведомость, отсортированная в алфавитном порядке. Кнопкой "Save" сохраните информацию в текстовом файле. Для просмотра содержимого текстового файла воспользуйтесь, например, Microsoft Word.

Используя все управляющие компоненты панели интерфейса, убедитесь в правильном функционировании приложения во всех предусмотренных режимах работы.

2. Индивидуальные задания

Во всех заданиях необходимо предусмотреть возможность сохранения вводимых данных в файле и чтения из ранее созданного файла. Результаты следует выводить в панель интерфейса и в текстовый файл.

1. Ведомость абитуриентов, сдавших вступительные экзамены в университет, содержит Ф.И.О. абитуриента и оценки каждого. Определить средний балл по университету и вывести список абитуриентов, средний балл которых выше среднего балла по университету. Первыми в списке должны идти студенты, сдавшие все экзамены на 5.

2. В радиоателье хранятся квитанции о сданной в ремонт радиоаппаратуре. Каждая квитанция содержит следующую информацию: наименование группы изделий (телевизор, радиоприемник и т.п.), марка изделия, дата приемки в ремонт, состояние готовности заказа (выполнен, не выполнен). Вывести информацию о состоянии заказов на текущие сутки по группам изделий.

3. У администратора железнодорожных касс хранится информация о свободных местах в поездах дальнего следования на ближайшую неделю в следующем виде: дата выезда, пункт назначения, время отправления, число свободных мест. Оргкомитет международной конференции обращается к администратору с просьбой зарезервировать *m* мест до города *N* на *k*-й день недели с временем отправления поезда не позднее *t* часов вечера. Вывести время отправления или сообщение о невозможности выполнить заказ в полном объеме.

4. Разработать программу формирования ведомости об успеваемости студентов. Каждая запись этой ведомости должна содержать: номер группы, Ф.И.О. студента, оценки за последнюю сессию. Вывести списки студентов по группам. В каждой группе Ф.И.О. студентов должны быть расположены в порядке убывания среднего балла.

Литература

1. В.В.Феофанов. Delphi 3. Учебный курс. — М.: Нолидж, 2000.

2. Э.Возневич. Delphi. Освой самостоятельно. — М.: Восточная книжная компания, 1996.

3. Дж.Матчо, Д.Р.Фолкнер. Delphi. — М.: БИНОМ, 1995.

4. М.Канту. Delphi 2 для Windows 95/NT. — М.: ООО "Малип", 1997.

А.БЕЛОУСОВ,
E-mail: ark@mos.ru
WWW: http://hi-tech.nsys.by

Контрольные суммы: сумма Флетчера

Циклические избыточные коды (CRC) — популярный метод построения контрольных сумм, служащих для обнаружения и исправления ошибок при передаче и хранении данных. Широкое распространение получили алгоритмы вычисления CRC16 и CRC32. Однако для получения контрольных сумм можно использовать и другие алгоритмы. Например, очень простой, короткий и быстрый алгоритм — сумму Флетчера.

Обоснование

Сумма Флетчера — это остаток от деления на 255 потока данных, интерпретируемого как длинное число. Остаток от деления на 2^n получить просто (достаточно взять n младших битов в двоичной системе счисления), остаток же от деления на 2^n-1 получить сложнее. Ниже дано обоснование этого процесса:

```
G % D  = (xn*B^n + ... + x1*B + x0) % D
       = (xn*(...)*D + xn + ... + x1*D + x1 + x0) % D
       = ((...)*D % D + (xn + ... + x1 + x0) % D) %.D
       = (xn + ... + x1 + x0) % D
```

Здесь D — это делитель, а G — поток данных, интерпретируемый как полином или как число в позиционной системе счисления с постоянным основанием B=D+1. Для суммы Флетчера $D=2^8-1=255$ и $B=2^8=256$. Знак "%" — это операция получения остатка от деления. Использованы следующие соотношения:

1. $(D+1)^n = D^n + ... + D + 1 = (...)*D + 1$
2. $(a + b) \% d = (a \% d + b \% d) \% d$

Отсюда видно, что остаток от деления полинома на D равен остатку от деления на D суммы всех коэффициентов (в данном случае байтов потока) этого полинома. Остаток от деления на D суммы вычисляется аналогично — сумма разбивается на байты, которые складываются до тех пор, пока результат не станет меньше B. Итог, равный D, означает нулевой остаток. Отсюда также видно, что порядок коэффициентов не важен, и суммировать байты потока можно в любом порядке.

Как факт — вычисление суммы Флетчера аналогично проверке делимости числа на 9, для чего суммируются все десятичные цифры числа и промежуточных сумм до тех пор, пока итог не станет меньше 10. Соответственно, число делится на 9 тогда и только тогда, когда итог равен 0 или 9.

Как оптимизацию отметим, что сумму можно сокращать до окончания суммирования всех байтов. Например, байты суммы можно сложить сразу, как только сумма станет больше B:

```
(xn + ... + x1 + x0) % D
       = (S + xk + ...) % D
       = (S % D + (xk + ...) % D) % D
       = ((sm*B^n + ... + s0) % D + (xk + ...) % D) % D
              ...
       = ((sm + ... + s0) % D + (xk + ...) % D) % D
       = (sm + ... + s0 + xk + ...) % D
```