Направление подготовки 09.03.03 «Прикладная информатика»
Дисциплина: Разработка программных приложений
Тема: Разработка динамических библиотек

Автор: Беляков А.Ю.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

ВВОДНАЯ ЧАСТЬ

Одна из основных особенностей обучения в высшей школе заключается в том, что постоянный внешний контроль заменяется самоконтролем, активная роль в учении принадлежит уже не столько преподавателю, сколько студенту. Значительное учебное время отводится для самостоятельной работы студента. Основными формами самостоятельной работы студента являются изучение конспекта лекций, анализ рекомендованной литературы, исследование дополнительных вопросов изучаемой темы, выполнение практических проблемных заданий. В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, рефлексии и становится активным самостоятельным субъектом учебной деятельности.

ОСНОВНАЯ ЧАСТЬ

Динамически подключаемая библиотека или *DLL* (от англ. *Dynamic Link Library* – дословно «библиотека динамической компоновки») в операционной системе *Microsoft Windows* библиотека подпрограмм, допускающая своё использование различными программными приложениями одновременно. К *DLL* относятся также элементы управления *ActiveX* и драйверы.

Преимущества от использования динамически подключаемых библиотек.

Эффективное использование ресурсов оперативной памяти и снижение объема расходуемого дискового пространства, за счет использования одного экземпляра библиотечного модуля для различных приложений.

Повышение эффективности сопровождения и обновления программных продуктов за счёт их модульности. Устранение «багов» (ошибок кода, выявленных уже после окончания проектирования) и обновление или наращивание приложений возможно путем замены динамически подключаемых библиотек с одной версии на другую. Кроме того, динамические библиотеки могли использоваться разнотипными приложениями от одного или даже разных производителей – например, *Microsoft Office* и *Microsoft Visual Studio*.

Данный вопрос выходит за рамки аудиторных занятий и выносится на самостоятельное изучение. Вам необходимо:

– вспомнить особенности организации подпрограмм вида процедуры и функции,

– уточнить технологию модульного проектирования приложений,

– освоить порядок проектирования динамических библиотек.

ВНИМАНИЕ!

Если у вас есть затруднения в понимании модульной организации приложений – обновите свои знания, используя методической пособие к лабораторной работе №1 События мыши (http://delphi-pgsha.narod.ru/labrab/labrab1.pdf), первый вопрос которой посвящен обсуждению модульной организации приложений.

Если у вас есть затруднения в понимании особенностей декомпозиции цельного программного кода на подпрограммы – обновите свои знания, используя презентацию к лекции №4 «Организация подпрограмм» (http://delphi-pgsha.narod.ru/prez/prez_podprogr.swf), в которой рассматриваются следующие вопросы:

– вид и структура подпрограмм;

– параметры подпрограмм;

– глобальные и локальные переменные;

– доступ к подпрограммам;

– рекурсия и досрочный выход;

– опережающее описание процедур;

– перегрузка подпрограмм;

– процедурный тип данных.

Дополнительно можно восполнить базовые понятия о функциях и процедурах из презентации к лекции «Подпрограммы» для дисциплины «Информатика и программирование» – http://algopro.narod.ru/lek11.htm .

Пример организации процедуры и функции в Delphi 7.

Разработаем рекурсивные подпрограммы вычисления суммы чисел от 1 до N.

Напомню, что рекурсивная подпрограмма сама себя вызывает. Корректная рекурсия должна обладать двумя особенностями:

– иметь точку останова;

– менять значения аргументов с каждой итерацией.

Пользователь задает целое число N. Рекурсию запустим от N до 1, поэтому точкой останова будет проверка равенства текущего аргумента единице и на каждом шаге рекурсии аргумент будет уменьшаться на единицу.

Далее приведены примеры функции и процедуры. Исходное значение находится в первом текстовом поле, а результат, после вычисления, помещается во второе.

```
function sum_f(n: Byte): Word;
begin
  if n=1
    then result:=1
    else result:=sum_f(n-1)+n;
end;

procedure sum_p(n: Byte; var r: Word);
begin
  if n=1
    then r:=1
    else begin sum_p(n-1,r); r:=r+n; end;
end;

procedure TForm1.btn1Click(Sender: TObject);
```

```
begin
  edt2.Text:=IntToStr(sum_f(StrToInt(edt1.Text)));
end;

procedure TForm1.btn2Click(Sender: TObject);
var r: Word;
begin
  sum_p(StrToInt(edt1.Text),r);
  edt2.Text:=IntToStr(r);
end;
```

Используются сокращенные названия компонентов.
Создайте приложение и апробируйте работу рекурсивных подпрограмм.

Пример размещения процедуры и функции в модуле.

Теперь перенесем рекурсивные подпрограммы в другой модуль. Создайте модуль Unit2 (меню File / New / Unit) и сохраните его в той же папке, что и само приложение с первым модулем. Сохранение приложения со всеми модулями является обязательным. Ниже приведено исходное содержание второго модуля:

```
unit Unit2;

interface

implementation

end.
```

В интерфейсном разделе следует декларировать подключаемые модули, пользовательские типы данных, константы и переменные, заголовки процедур и функций. В разделе реализаций описываются сами подпрограммы.
Перенесите разработанные ранее подпрограммы во второй модуль:

```
unit Unit2;

interface
  function sum_f(n: Byte): Word;
  procedure sum_p(n: Byte; var r: Word);

implementation

function sum_f(n: Byte): Word;
begin
  if n=1
    then result:=1
    else result:=sum_f(n-1)+n;
end;

procedure sum_p(n: Byte; var r: Word);
begin
  if n=1
    then r:=1
    else begin sum_p(n-1,r); r:=r+n; end;
end;

end.
```

Первый модуль сократите, но не забудьте произвести подключение второго модуля:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    edt1: TEdit;
    edt2: TEdit;
    btn1: TButton;
    btn2: TButton;
    procedure btn1Click(Sender: TObject);
    procedure btn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

uses  Unit2;   // это обязательно !!!

{$R *.dfm}

procedure TForm1.btn1Click(Sender: TObject);
begin
  edt2.Text:=IntToStr(sum_f(StrToInt(edt1.Text)));
end;

procedure TForm1.btn2Click(Sender: TObject);
var r: Word;
begin
  sum_p(StrToInt(edt1.Text),r);
  edt2.Text:=IntToStr(r);
end;

end.
```

Итак, значимые подпрограммы мы вынесли в отдельный модуль, который уже можно отдавать, совершенствовать, заменять независимо от основной программы. Однако его всё ещё нужно компилировать совместно с программным кодом основной программы.

Пример размещения процедуры и функции в динамической библиотеке.

Сохраните все изменения и закройте приложение. Через меню создайте новую заготовку под библиотеку: меню File / New / Other / DLL Wizard. В созданной пока пустой библиотеке вы увидите комментарий, который гласит буквально следующее:

Если динамическая библиотека в процессе работы использует переменные или функции, осуществляющие динамическое выделение памяти под собственные нужды (длинные строки, динамические массивы, функции New и GetMem), а также, если такие переменные передаются в параметрах и возвращаются в результатах, то в таких библиотеках обязательно должен использоваться модуль ShareMem. При этом в секции uses модуль должен располагаться на первом месте. Управление этими операциями осуществляет специальная библиотека BORLANDMM.DLL, которая должна распространяться вместе с разрабатываемыми динамическими библиотеками, использующими модуль ShareMem.

В нашей библиотеке мы пока не используем указанные особенности, поэтому список подключаемых модулей оставим без изменений, комментарий уберем, а наши рекурсивные подпрограммы вставим:

```
library P_DLL;

uses
  // ShareMem  // этот модуль нужно ставить тут, но сейчас он не нужен
  SysUtils,
  Classes;

{$R *.res}

function sum_f(n: Byte): Word;
begin
  if n=1
    then result:=1
    else result:=sum_f(n-1)+n;
end;

procedure sum_p(n: Byte; var r: Word);
begin
  if n=1
    then r:=1
    else begin sum_p(n-1,r); r:=r+n; end;
end;

exports
  sum_f, sum_p;

begin
end.
```

Обратите внимание на новый раздел **exports**, он нужен для того чтобы подпрограммы библиотеки были доступны во внешних приложениях. Сохраните текущий проект в ту же папку, что и основное приложение под интуитивно понятным именем (например, **P_DLL**). Откомпилируйте динамическую библиотеку нажав CTRL+F9 и проверьте, что в текущей папке появился новый файл **P_DLL.dll** уже готовый к использованию сторонними программами. Проверим это, для чего закройте редактируемую библиотеку и откройте наш основной проект. На текущий момент в нем два обработчика события (две процедуры), готовые использовать рекурсивные подпрограммы. Но пока адресация установлена на использование второго модуля. Удалите ссылку на второй модуль, но добавьте описание рекурсивных подпрограмм со ссылкой на нашу динамическую библиотеку:

```
unit Unit1;

interface

uses
```

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

type
  TForm1 = class(TForm)
    edt1: TEdit;
    edt2: TEdit;
    btn1: TButton;
    btn2: TButton;
    procedure btn1Click(Sender: TObject);
    procedure btn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

//uses  Unit2;  // это уже нужно убрать !!!

{$R *.dfm}

function sum_f(n: byte): Word;
  external 'P_DLL.dll' name 'sum_f';
procedure sum_p(n: byte; var r: Word);
  external 'P_DLL.dll' name 'sum_p';

procedure TForm1.btn1Click(Sender: TObject);
begin
  edt2.Text:=IntToStr(sum_f(StrToInt(edt1.Text)));
end;

procedure TForm1.btn2Click(Sender: TObject);
var r: Word;
begin
  sum_p(StrToInt(edt1.Text),r);
  edt2.Text:=IntToStr(r);
end;

end.
```

При наличии откомпилированной библиотеки наше приложение можно уже запустить и испытать работу рекурсивных подпрограмм.

Обратите внимание на параметр **name** в декларации внешней библиотеки. Он задает имя подпрограммы из библиотеки, однако в вашей программе вы можете использовать другое имя, главное чтобы количество аргументов и их типы данных совпали.

После внесения изменений раздел реализации может выглядеть так:

```
implementation

{$R *.dfm}

function sum_f(n: byte): Word;
  external 'P_DLL.dll' name 'sum_f';
```

```
procedure s_p(n: byte; var r: Word);
  external 'P_DLL.dll' name 'sum_p';

procedure TForm1.btn1Click(Sender: TObject);
begin
  edt2.Text:=IntToStr(sum_f(StrToInt(edt1.Text)));
end;

procedure TForm1.btn2Click(Sender: TObject);
var r: Word;
begin
  s_p(StrToInt(edt1.Text),r);
  edt2.Text:=IntToStr(r);
end;

end.
```

Внимание! Подумайте над вопросом: для чего может потребоваться смена имени подпрограммы.

Задания для самостоятельного исполнения.

Внимание:
– удобнее всего обмениваться строковыми данными с динамической библиотекой через переменную типа **`TStringList`**, даже если там всего одна строка (*иначе можно столкнуться со сложностями передачи строковых данных, так как строковый тип String можно использовать для передачи в библиотеку, но обратно можно возвращать только через тип PChar; на странице http://delphi-pgsha.narod.ru/faq.htm можете найти пример по передаче строки в динамическую библиотеку, обработке строки и возвращении её обратно из библиотеки в основную программу; код примера можете увидеть ниже в Приложении с поясняющими комментариями*);
– для выполнения заданий вам может потребоваться подключение модуля ShareMem;
– при выполнении заданий можно не делать каждый раз новую библиотеку, достаточно просто дополнять нашу.

1. Разработайте динамическую библиотеку, содержащую рекурсивную функцию подсчета произведения нечетных чисел от 1 до N.

2. Разработайте динамическую библиотеку вычисления корней квадратного уравнения (в библиотеку передаются коэффициенты a, b, c; обратно возвращаются x1, x2, строка с текстом о количестве корней – через нулевой элемент переменной типа **`TStringList`**).

3. Разработайте динамическую библиотеку определения самой длинной строки в текстовом файле. В библиотеку передается полное имя файла, обратно возвращается искомая строка (через нулевой элемент переменной типа **`TStringList`**).

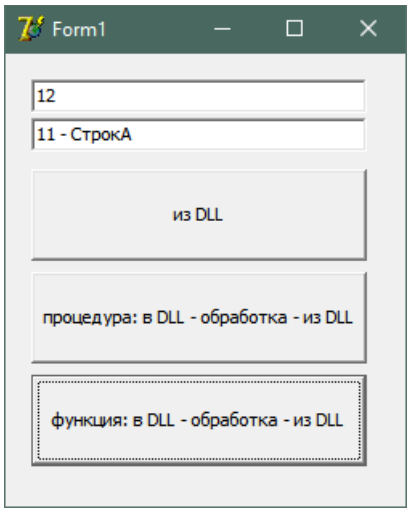
4. Пусть структура текстового файла такова: в каждой строке находятся Фамилия и Средний балл абитуриента, разделенные пробелом (или другим символом, по желанию). Разработайте динамическую библиотеку определения списка Фамилий абитуриентов имеющих средний балл выше указанного пользователем. В библиотеку передается полное имя файла и средний балл, а обратно возвращается список фамилий (список строк **`TStringList`**). Результат можно вывести в поле **`Memo`**.

ПОДВЕДЕНИЕ ИТОГОВ

Все разработанные приложения сохраните в отдельных папках, ко всем апробированным программам добавьте свои комментарии в тексте кода, а результаты проделанной работы на электронном носителе предоставьте на проверку преподавателю.

Оцениваться будет объем и полнота проделанной работы, сложность разработанных приложений, корректность кода и настроек свойств и событий компонентов.

Главная форма вид приложения:



Код динамической библиотеки:

```
library P_DLL;

uses
  SysUtils,
  Classes;

{$R *.res}
const str='СтрокА';

procedure P_Str(var s: string; var pp: PChar);
var e: string; n: Integer;
begin
  n:=StrToInt(s)+1;
  e:=IntToStr(n)+' - '+str;
  pp:=PChar(e);
end;

function F_Str(s: string): PChar;
var e: string; n: Integer;
begin
  n:=StrToInt(s)-1;
  e:=IntToStr(n)+' - '+str;
  result:=PChar(e);
end;

procedure Get_Str(var p: PChar);
begin
  StrCopy(p,str);
end;

function Get_Len(var s: string): integer;
begin
  Result:=Length(str);
end;

exports
  Get_Len, Get_Str, P_Str, F_Str;

begin
end.
```

Код приложения:

```
unit Unit1;

interface

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

type
  TForm1 = class(TForm)
    edt1: TEdit;
    edt2: TEdit;
    btn6: TButton;
    btn1: TButton;
    btn2: TButton;
    procedure btn6Click(Sender: TObject);
    procedure btn1Click(Sender: TObject);
    procedure btn2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure Get_Str(var p: PChar);
  external 'P_DLL.dll';
function Get_Len:integer;
  external 'P_DLL.dll';
procedure P_Str(var a: String; var z: PChar);
  external 'P_DLL.dll';
function F_Str(s: string): PChar;
  external 'P_DLL.dll';

procedure TForm1.btn6Click(Sender: TObject);
var p: PChar; // ссылка на символ в строке
  // PChar нужен для работы с ASCIIZ-строками
  // это строки длиной больше 255, в конце символ с кодом 0
begin
  GetMem(p,(Get_Len+1)*SizeOf(Char)); // выделяем память
  // +1 нужен на нуль-символ конца строки
  // SizeOf(Char) нужен чтобы работало и для двухбайтной кодировки
  Get_Str(p);
  edt2.Text:=p;
  FreeMem(p);  // освобождаем память
end;

procedure TForm1.btn1Click(Sender: TObject);
var p: PChar; a: string;
begin
  a:=edt1.Text;
  P_Str(a,p);
  edt2.Text:=p;
end;

procedure TForm1.btn2Click(Sender: TObject);
begin
  edt2.Text:=F_Str(edt1.Text);
end;

end.
```