Міністерство освіти і науки України
Державний вищий навчальний заклад
“Ужгородський національний університет”
Математичний факультет
Кафедра системного аналізу і теорії оптимізації

МОДУЛІ ТА БІБЛІОТЕКИ DLL У DELPHI

Методичні вказівки до лабораторних робіт для студентів II-го курсу математичного факультету спеціальності "прикладна математика"

Ужгород – 2011

Семйон І.В., Чупов С.В., Брила А.Ю., Апшай Н.І. Модулі та бібліотеки DLL у Delphi. Методичні вказівки до лабораторних робіт для студентів ІІ-го курсу математичного факультету спеціальності "прикладна математика". – Ужгород, 2011 . – 43с.

Розглядаються основи модульного програмування, структура модулів та методика їх використання. Всі теми ілюструються на конкретних прикладах та схемах. Ґрунтуючись на понятті модуля, розглядаються бібліотеки DLL, їх структура та методи підключення. Проведено ґрунтовний аналіз переваг і недоліків модулів та бібліотек, подано рекомендації щодо їх застосування. Запитання для самоконтролю та завдання для самостійного виконання дозволяють закріпити розглядуваний матеріал.

Рецензенти: канд. фіз.-мат. наук, доц.

канд. фіз.-мат. наук, доц.

Рекомендовано до друку Вченою радою математичного факультету ДВНЗ “Ужгородський національний університет” __ _____2011 року, протокол №__.

МОДУЛІ

Передумови виникнення модульного програмування

Як відомого, програма – це алгоритм, записаний за допомогою мови програмування, тобто послідовність команд, виконання яких дозволяє одержати розв'язок деякої задачі. При розв'язанні складних задач застосовують так званий *структурний підхід*, суть якого полягає у тому, що складна задача розбивається на простіші підзадачі, до розв'язання яких зводиться початкова задача. Цей підхід реалізовується за допомогою *технології процедурного програмування*, яка передбачає розбиття основної задачі на підзадачі та оформлення останніх у вигляді підпрограм (процедур та функцій). Звичайно, технологія процедурного програмування значно спрощує розв'язання основної задачі, оскільки дає можливість зосереджуватися на розв'язанні кожної окремої підзадачі, як автономної одиниці. Окрім цього, підпрограми дозволяють частково розв'язувати проблему повторного використання розроблених програмних одиниць. Так один раз розробивши деяку підпрограму, її можна з легкістю використати при розробці іншої програми просто скопіювавши її текст. Але ж для цього ми завжди повинні пам'ятати у яких програмах які розроблено підпрограми. А також може скластися ситуація, коли нам необхідно копіювати десятки, а то й сотні підпрограм. Звичайно, об'єм такої програми буде значним, що значно ускладнює розробку програми. Зрозуміло, що в цьому випадку зручним був би механізм, який би дозволяв використовувати раніше створені підпрограми не копіюючи кожен раз текст відповідної підпрограми та не компілювати цю підпрограму кожного разу. Тобто доцільніше було б розробляти програму по частинам, а далі з'єднувати ці частини в єдине ціле – кінцеву програму. Саме ці проблеми і покликані розв'язувати модулі. *Модуль* – це автономно компілюємо програмна одиниця, яка може включати в себе опис констант, типів даних, змінних та підпрограм, які можуть бути використані в інших модулях та програмах. Саме завдяки автономності

модулів та можливості їх багаторазового використання модулі набули широкого поширення при розробці програм.

Окрім проблем повторного використання коду, модулі часткового дозволяють розв'язувати проблеми організації даних. Використання модулів дозволяє "сховати" дані і процедури їх обробки всередині модуля, що дозволяє уникнути неправильного використання даних.

Структура модуля

Модуль має наступну структуру:

```
UNIT <Ім'я модуля>;
INTERFACE
  USES      <Список модулів>;
  CONST     <Опис констант>;
  TYPE      <Опис типів користувача>;   } Описова (інтерфейсна) частина
  VAR <Опис змінних>;
  <Заголовки процедур і функцій>
IMPLEMENTATION
  USES      <Список модулів>;
  CONST     <Опис констант>;
  TYPE      <Опис типів користувача>;   } Частина реалізації
  VAR       <Опис змінних>;
  <Опис процедур і функцій>
INITIALIZATION
  <Частина ініціалізації>
FINALIZATION
  <Частина деініціалізації>
END.
```

Отже, модуль складається із заголовку і чотирьох складових частин, кожна з яких може бути порожньою.

Заголовок модуля складається із зарезервованого слова UNIT та імені модуля

```
UNIT <Ім'я модуля>;
```

Ім'я модуля задається у формі ідентифікатора і повинно співпадати з іменем дискового файлу, в якому зберігається текст модуля. Так, наприклад, якщо маємо модуль, заголовок якого

```
UNIT Vector;
```

то текст відповідного модуля повинен зберігатися у однойменному дисковому файлі з розширенням "Pas" ("Vector.pas"). При компіляції створюється файл з розширенням "Dcu" (Global.Dcu). Саме це ім'я і використовується для підключення модуля до основної програми або до іншого модуля. Робиться це в розділі описання модулів USES

```
USES Vector;
```

Описова частина відкривається зарезервованим словом INTERFACE. Тут містяться описання всіх глобальних об'єктів модуля (типів, констант, змінних та підпрограм), які повинні бути доступними основній програмі та/або іншим модулям, які підключають даний модуль. При описанні глобальних підпрограм в інтерфейсній частині вказується тільки їх заголовок.

```
INTERFACE
    USES      <Список модулів>;
    CONST     <Опис констант>;
    TYPE      <Опис типів користувача>;
    VAR       <Опис змінних>;
    <Заголовки процедур і функцій>
```

Приклад.

```
INTERFACE
    USES      SysUtils;
    CONST     n=20;
    TYPE      TVector=array[1..n] of real;
    VAR       V:TVector;
    Procedure Vvod(n:integer; V:TVector);
    Function Sum(n:integer; V:TVector):Real;
```

Якщо тепер в основній частині програми в розділі описання модулів підключити цей модуль

```
USES Vector;
```

то у програмі стануть доступними всі описані константи, типи, змінні та процедури і функції так, ніби вони були описані в самій основній програмі.

Тобто наступні описи є еквівалентними

З використанням модуля	*Без використання модуля*

UNIT **Vector;**
INTERFACE
 	
CONST n=20;
TYPE TVector=array[1..n] of real;
VAR V:TVector;
Procedure Vvod(n:integer; V:TVector);
Function Sum(n:integer; V:TVector):Real;
 	
IMPLEMENTATION
……
INITIALIZATION
……..
FINALIZATION
……..
END.
	PROGRAM Project1;
	{$APPTYPE CONSOLE}
	
	CONST n=20;
	TYPE TVector=array[1..n] of real;
	VAR V:TVector;
	Procedure Vvod(n:integer; V:TVector);
	Begin
	……..
	End;
	Function Sum(n:integer; V:TVector):Real;
	Begin
	……..
	End;
	BEGIN
	……..
	END.

PROGRAM Project1; {$APPTYPE CONSOLE} **USES Vector;** BEGIN …….. END.	

Слід зазначити, що всі константи та змінні, описані в інтерфейсній частині модуля, як і глобальні константи та змінні основної програми, поміщаються компілятором у загальний сегмент даних. Порядок запису різних розділів описання та їх кількість можуть бути довільними.

Частина реалізації відкривається зарезервованим словом IMPLEMENTATION та містить описання реалізації підпрограм, описаних в інтерфейсній частині. У ній також можуть описуватися локальні для модуля об'єкти – допоміжні типи, константи, змінні, процедури і функції, які використовуються в інших процедурах і функціях цього ж модуля.

У частині реалізації при описанні підпрограм, заголовки яких описано в інтерфейсній частині, заголовки підпрограм можуть вказуватися без формальних параметрів. Але якщо в частині реалізації заголовок підпрограми вказується повністю, то він повинен у точності співпадати з раніше описаним в інтерфейсній частині.

Приклад. Описання реалізації підпрограм наведених раніше може бути наступним

```
UNIT Vector;
INTERFACE
  CONST     n=20;
  TYPE      TVector=array[1..n] of real;
  VAR       V:TVector;
  Procedure Vvod(n:integer; V:TVector);
  Function Sum(n:integer; V:TVector):Real;
IMPLEMENTATION
  Procedure Vvod;                              {Параметри не вказано}
   Begin
      for i:=1 to n do
        begin
           write('V[',i,']= ');
            readln(V[i]);
         end;
   End;
  Function Sum(n:integer; V:TVector):Real;     {Параметри повністю співпадають}
     var s:Real;
   Begin     s:=0;
        for i:=1 to n do
          s:=s+V[i];
     Result:=s;
   End;
END.
```

Як було зазначено, у частині реалізації можуть також описуватися локальні об'єкти (константи, змінні, підпрограми), які використовуються тільки в межах модуля та недоступні для програм, які будуть використовувати даний модуль. Наведемо приклад такого опису.

Приклад. Створити модуль, який містить функцію для знаходження суми максимального та мінімального елементів.

```
UNIT Vector;
INTERFACE
  CONST     n=20;
  TYPE      TVector=array[1..n] of real;
  VAR       V:TVector;
  Function SumMinMax(n:integer; V:TVector):Real;
IMPLEMENTATION
  Function GetMin(n:integer; V:TVector):Real;
     var min:Real;
   Begin     min:=V[1];
        for i:=2 to n do
         if min>A[i] then
           min:=A[i];
      Result:=min;
   End;
  Function GetMax(n:integer; V:TVector):Real;
     var max:Real;
   Begin     max:=V[1];
        for i:=2 to n do
         if max < A[i] then
           max:=A[i];
      Result:= max;
   End;
  Function SumMinMax (n:integer; V:TVector):Real;
   Begin
      Result:= GetMin(n,V)+ GetMax(n,V);
   End;
END.
```

Тут функції **GetMin** та **GetMax** можуть бути використані тільки у функції SumMinMax і недоступні для програм, які будуть використовувати даний модуль, оскільки їх заголовки не описано в інтерфейсній частині модуля.

Частина ініціалізації відкривається зарезервованим словом INITIALIZATION та містить фрагмент програми, який виконується до передачі управління основній програмі і, як правило, використовуються для підготовки модуля до роботи (початкової ініціалізації). У цій частині можуть ініціалізовуватися змінні, відкриватися потрібні файли, створюватися динамічні масиви і т.д. Якщо подібних дій виконувати не потрібно, то частина ініціалізації не вказується. У випадку, коли у програмі використовується декілька модулів, то на початку програми виконуються

частини ініціалізації всіх використаних модулів у порядку, вказаному під час опису.

Частина деініціалізації відкривається зарезервованим словом FINALIZATION та містить команди, які виконуються при завершенні основної програми. У цій частині можуть закриватися файли, звільнятися динамічно виділена пам’ять та ін. Як і частина ініціалізації, частина деініціалізації може бути відсутньою. Якщо у програмі використовується декілька модулів, то при закінченні програми виконуються частини деініціалізації модулів у порядку, протилежному до порядку їх опису у програмі.

Приклад. Дослідник щодня здійснює виміри температури та знаходить середнє арифметичне. Знайдене середнє арифметичне зберігається у файлі для подальших досліджень (якщо файлу не існує, то він створюється, якщо ж файл існує, то результат разом із датою вимірювань дописується у кінець файлу). Створити модуль, який містить процедури, для розв’язання поставленої задачі.

Текст модуля

```
UNIT Unit1;
INTERFACE
   uses SysUtils;
   Type TVector=array of Real;
   Var f:TextFile;
       n:integer;
       V:TVector;
   Procedure GetData(var n:integer; var V:TVector);
   Procedure SaveToFile(n:integer; var V:TVector);
IMPLEMENTATION
 Function GetAverage(n:integer; var V:TVector):Real;
   var s:Real;
       i:integer;
  Begin
   s:=0;
   for i:=0  to n-1 do
   begin
    s:=s+V[i];
   end;
   Result:=s/n;
  End;
```

```
 Procedure GetData;
   var i:integer;
  Begin
    writeln('Кількість вимірів = ');
    Readln(n);
    SetLength(V,n);
    for i:=0 to n-1 do
    begin
      write(i,'-вий вимір : ');
      readln(V[i]);
    end;
  End;
 Procedure SaveToFile;
  Begin
    writeln(f,DateToStr(Date),'  -  ',GetAverage(n,V));
  End;
INITIALIZATION    {Виконується на початку програми}
     AssignFile(f,'MyFile.txt');
     if FileExists('MyFile.txt') then
        Append(f)
     else
        Rewrite(f);
FINALIZATION      {Виконується на кінці програми}
     CloseFile(f);
END.
```

Текст проекта

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Unit1 in 'Unit1.pas';
BEGIN
 GetData(n,V);
 SaveToFile(n,V);
END.
```

Підключення модулів

Підключення модулів в основній програмі і їх можлива компіляція здійснюються у порядку їх описання у розділі описання модулів USES.

```
USES <Модуль 1>,<Модуль 2>,…< Модуль N>;
```

Якщо модуль знаходиться у поточному каталозі, то достатньо вказати тільки ім’я модуля.

Приклад.

```
USES SysUtils, Vector;
```

Якщо ж модуль знаходиться не у поточному каталозі, то для підключення модуля необхідно вказати повний шлях до нього.

```
USES <Модуль > in <Шлях до модуля>;
```

Приклад.

```
USES SysUtils,
     Vector in 'D:\MyFolder\ Vector.pas';
```

Проблема перевизначення

Як було зазначено, об’єкти, описані в інтерфейсній частині модуля, є доступними для програм, що їх використовують. Але ж може статися випадок, коли у програмі та модулях описано величини з тими ж самими іменами, але різних типів. У цьому випадку у програмі тип величини буде визначатися на основі наступних правил:

1) якщо величину описано у програмі, що використовує модуль, то величини буде саме того типу, який вказано у цій програмі;
2) якщо програма використовує декілька модулів, у яких описано однойменні величини, то величина буде того типу, який вказано у модулі, що вказаний описанім у розділі описання модулів.

Приклад. У проекті спочатку використовується **Unit1** а потім **Unit2.**

Модуль Unit 1

```
unit Unit1;
interface
var x:integer;
implementation
initialization
 x:=25;
end.
```

Модуль Unit 2

```
unit Unit2;
interface
 var x:String;
implementation
initialization
x:=' Unit 2 ';
end.
```

Проект

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Unit1, Unit2;
BEGIN
 writeln('x= ',x);
 readln;
END.
```

На екран буде виведено

```
x= Unit 2
```

оскільки останнім у розділі описання модулів Uses вказано Unit2, в якому x описано як величину типу String і в розділі ініціалізації їй надано початкове значення рядок символів **' Unit 2 '.**

Приклад. У проекті спочатку використовується **Unit2** а потім **Unit1.**

Модуль Unit 1

```
unit Unit1;
interface
var x:integer;
implementation
initialization
 x:=25;
end.
```

Модуль Unit 2

```
unit Unit2;
interface
 var x:String;
implementation
initialization
x:=' Unit 2 ';
end.
```

Проект

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Unit2, Unit1;
BEGIN
 writeln('x=',x);
 readln;
END.
```

На екран буде виведено

```
x= 25
```

оскільки останнім у розділі описання модулів Uses вказано Unit1, в якому x описано як величину типу Integer і в розділі ініціалізації їй надано початкове значення 25.

Приклад. У проекті описано власну змінну типу Boolean.

Модуль Unit 1

```
unit Unit1;
interface
var x:integer;
implementation
initialization
 x:=25;
end.
```

Модуль Unit 2

```
unit Unit2;
interface
 var x:String;
implementation
initialization
x:=' Unit 2 ';
end.
```

Проект

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Unit1, Unit2;
 Var x:boolean;
BEGIN
 x:=True;
 writeln('x=',x);
 readln;
END.
```

На екран буде виведено

```
x= True
```

оскільки у програмі описано власну змінну типу Boolean, якій надано початкове значення True.

Але, тим не менше, у програмі є можливість звертання до перевизначених величин, описаних у модулях. Для цього необхідно вказати перед іменем величини назву модуля, у якому ця величина описана.

```
<Ім'я модуля> . <Ім'я змінної>
```

Приклад. У проекті описано власну змінну типу Boolean.

Модуль Unit 1

```
unit Unit1;
interface
var x:integer;
implementation
initialization
 x:=25;
end.
```

Модуль Unit 2

```
unit Unit2;
interface
 var x:String;
implementation
initialization
x:=' Unit 2 ';
end.
```

Проект

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,
  Unit1, Unit2;
 Var x:boolean;
BEGIN
 x:=True;
writeln('Project x=',x);
writeln('Unit1 x=',Unit1.x);
writeln('Unit2 x=',Unit2.x);
 readln;
END.
```

На екран буде виведено

```
Project x=True
Unit1 x=25
Unit2 x= Unit 2
```

При цьому

x – змінна типу Boolean, описана у проекті Project1,

Unit1.x – змінна типу Integer, описана у модулі **Unit1**,

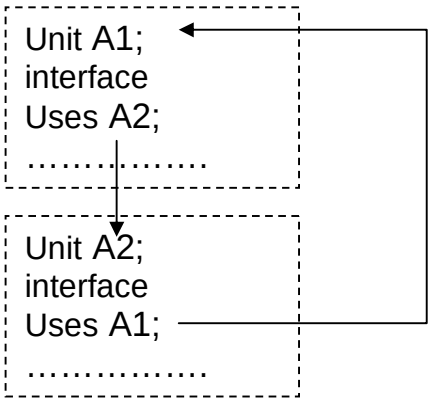
Unit2.x – змінна типу String, описана у модулі **Unit2.**

Проблема зациклення

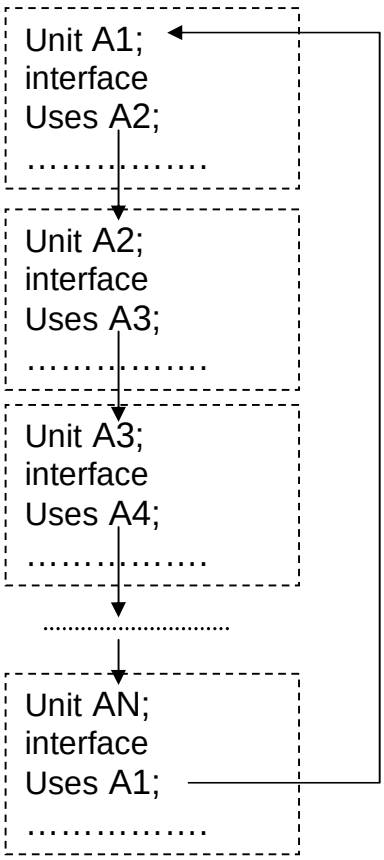
При переході до наступного модуля система попередньо відшукує всі модулі, на які він посилається. Посилання модулів один на одного можуть утворювати деревовидну структуру довільної складності, однак **забороняється явне або опосередковане звертання модуля самого до себе.**

Приклад.

Явне зациклення



Опосередковане зациклення



Таке обмеження можна обійти, перенісши розділ описання модулів USES в частину реалізації IMPLEMENTATION. Наприклад, наступне недопустиме описання

```
Unit A;
INTERFACE
      Uses A2;
……..
IMPLEMENTATION
……..
END.
```

```
Unit A2;
INTERFACE
      Uses A1;
……..
IMPLEMENTATION
……..
END.
```

можна виправити наступним чином

```
Unit A;
INTERFACE
……..
IMPLEMENTATION
  Uses B;
……..
END.
```

```
Unit B;
INTERFACE
……..
IMPLEMENTATION
  Uses A;
……..
END.
```

Зауваження. При створенні модулів у частині INTERFACE здійснюється підключення тільки тих модулів, які містять необхідні типи і константи для створення нових типів та опису інших елементів описової частини, а всі інші модулі підключаються у частині реалізації IMPLEMENTATION.

Переваги і недоліки модульного програмування

Розглянемо основні переваги та недоліки модульного програмування. Звичайно, найбільшою перевагою у використанні модулів є те, що їх використання дає можливість застосувати структурний підхід до розв'язання складних задач за методикою "зверху-вниз". Цей підхід дозволяє складну задачу розбити на більш простіші, розв'язання яких може бути оформлено у вигляді підпрограм, що можуть бути об'єднані за деякою ознакою у модулі.

Другою позитивною рисою модульного програмування є те, що модулі дозволяють певною мірою розв'язати проблему повторного використання вже розробленого програмного коду. Так один раз розроблений модуль з легкістю можна використати у будь-якій програмі.

Третя перевага модульного програмування полягає у тому, що модулі, як і класи, певною мірою дозволяють “інкапсулювати” дані та методи їх обробки в середні одного модуля, а це дозволяє уникнути проблеми неправильного використання даних і “сховати” реалізацію методів від користувача.

Ще однією характерною рисою модулів є те, що при компіляції код модуля приєднується до коду основної програми. З одного боку це позитивна риса, оскільки такий підхід полегшує перенос програми на інший комп’ютер. Але з іншого – це є значним недоліком модульного програмування. Уявімо ситуацію, коли три різні програми використовують один і той же модуль. У цьому випадку, якщо всі три програми запущено на виконання, то в оперативній пам’яті один і той же модуль буде присутній у трьох екземплярах. Слід зазначити також, що за рахунок збільшення об’єму програми збільшується і час завантаження програми у оперативну пам’ять. При цьому завантажуються всі процедури і функції, частина з яких під час роботи можливо використовуватися не буде. Звичайно, це призводить до неефективного використання оперативної пам’яті та інших ресурсів.

Не дивлячись на значні недоліки модульного програмування, модулі широко використовуються у сучасному програмуванні.

Запитання для самоконтролю

1. *Що таке модуль?*
2. *Що таке модульне програмування?*
3. *Коли доцільно використовувати модулі?*
4. *Яка структура модуля?*
5. *Які особливості іменування модуля та файлу, у якому він зберігається?*
6. *Яке розширення мають відкомпільовані модулі?*
7. *Що може бути описано у описовій частині модуля?*
8. *Де описується реалізація підпрограм модуля?*
9. *Що таке частина ініціалізації і коли вона використовується?*
10. *Що таке частина деініціалізації і коли вона використовується?*
11. *Як підключити модуль до програми?*
12. *Роз'ясніть, коли відбувається зациклення посилань при використанні модулів і як його уникнути?*
13. *Які переваги і недоліки модульного програмування?*

Завдання для самостійного виконання

1. Створити модуль, який містить такі функції для роботи з натуральними числами:
   - знаходження всіх дільників вказаного натурального числа;
   - знаходження найбільшого спільного дільника двох чисел;
   - знаходження найменшого спільного кратного двох чисел.
2. Створити модуль, який містить такі функції для роботи з натуральними числами:
   - визначення того, чи є дане число простим;
   - виведення всіх простих чисел, які менші за дане.
3. Створити модуль, який містить такі підпрограми для роботи з натуральними числами:
   - визначення того, чи може дане число бути представлено у вигляді суми двох простих чисел;
   - визначення того, чи може дане число бути представлено у вигляді добутку двох простих чисел.
4. Створити модуль, який містить такі підпрограми для роботи з натуральними числами:
   - знаходження суми цифр;
   - знаходження добутку цифр;
   - підрахунку кількості входження деякої цифри;
   - визначення порядку числа.
5. Створити модуль, який містить такі підпрограми для роботи з натуральними числами:
   - інвертування числа;
   - формування числа, у якому цифри йдуть у зростаючому порядку;
   - формування числа, у якому цифри йдуть у спадному порядку;
   - формування числа, у якому відсутні цифри 0.

6. Створити модуль, який містить такі підпрограми для роботи з десятковими дробами (ціла і дробова частини зберігаються окремо як цілі числа):
   - введення та виведення;
   - виконання арифметичних операцій.
7. Створити модуль, який містить такі підпрограми для роботи з натуральними числами як грошовими сумами (число $n \le 9999$):
   - запису числа прописом;
   - визначення того, як дану грошову суму можна утворити з купюр номіналом 50, 20, 5 і 1 гривні при мінімальній кількості купюр.
8. Створити модуль, який містить функції для знаходження факторіала числа, та визначення за вказаним числовим значенням того, факторіалом якого числа є це значення.
9. Створити модуль, який містить функцію для знаходження n-го числа Фібоначчі, та функцію, яка за вказаним числовим значенням Фібоначчі визначатиме його порядковий номер.
10. Створити модуль, який містить такі підпрограми для роботи з раціональними дробами:
    - введення та виведення;
    - виконання арифметичних операцій.
11. Створити модуль, який містить такі підпрограми для роботи з різними мірами довжини:
    - переведення міліметрів у сантиметри, метри, кілометри, милі, дюйми і навпаки.
    - порівняння двох довжин, заданих у сантиметрах і дюймах.
12. Створити модуль, який містить такі підпрограми для роботи з різними мірами ваги:
    - переведення грам у кілограми, центнери, тони фунти і навпаки;
    - порівняння двох ваг, заданих у кілограмах і фунтах.

13. Створити модуль, який містить процедури для роботи із комплек сними числами.
14. Створити модуль, який містить та підпрограми для роботи з арифметичними прогресіями:
    - знаходження *n*-го члена арифметичної прогресії;
    - знаходження суми перших *n* членів арифметичної прогресії.
15. Створити модуль, який містить та підпрограми для роб оти з геометричними прогресіями:
    - знаходження *n*-го члена геометричної прогресії;
    - знаходження суми перших *n* членів геометричної прогресії.
16. Створити модуль, який містить процедури переведення числа з десяткової системи числення у довільну *q*-кову систему числен на і навпаки.
17. Створити модуль, який містить процедури переведення числа записаного у десятковій системі численна у римську систему числення і навпаки.
18. Стоврити модуль, який містить процедури переведення числа з двійкової у вісімкову, шістнадцяткову і навп аки.
19. Створити модуль, який містить процедури додавання, множення та ділення многочленів.
20. Створити модуль, який містить підпрограми піднесення двочлена до вказаної степені та виведення трикутника Паскаля.
21. Створити модуль, який містить функцію для обчисленн я значення многочлена за схемою Горнера.
22. Створити модуль, який містить функції визначення паралельності чи перпендикулярності двох прямих на площині та знаходження точки їх перетину.
23. Створити модуль, який містить функції визначення паралельності чи перпендикулярності двох прямих у просторі та знаходження точки їх перетину.

24. Створити модуль, який містить функції для визначення того, чи належить точка вказаній прямій та чи лежать дві точки по один бік від прямої.
25. Створити модуль, який містить функції для обчислення площі трикутника різними способами (за трьома сторонами, за двома сторонами і кутом між ними, за стороною і висотою, опущеною на нею).
26. Створити модуль, який містить такі функції для роботи з трикутником, заданим довжинами сторін:
    - здійснити перевірку, чи можна із заданих трьох відрізків утворити трикутник;
    - визначення типу трикутника;
    - визначення кутів трикутник.
27. Створити модуль, який містить такі функції для роботи з прямокутним трикутником:
    - знаходження площі різними способами;
    - знаходження катетів різними способами (коли відома гіпотенуза і кути, і коли відома гіпотенуза і однин з катетів)
    - знаходження радіусів вписаного і описаного кіл.
28. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин на площині:
    - визначення довжини сторін;
    - визначення площі трикутника;
    - визначення периметра трикутника.
29. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин на площині:
    - виведення вершин трикутника проти годинникової с трілки;
    - знаходження трикутника (його вершин), який розташований по відношенню до даного симетричний відносно осей координат та початку координат.

30. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин н а площині:
    - паралельний перенос;
    - поворот відносно початку координат.
31. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин на площині:
    - визначення того, чи перетинаються два трикутники, задані координатами своїх вершин;
    - чи знаходиться однин трикутник у середині іншого.
32. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин на площині:
    - визначення типу трикутника;
    - визначення градусної міри кутів.
33. Створити модуль, який містить наступні функції для роботи з трикутником, заданим координатами своїх вершин на площині:
    - знаходження довжин висот трикутника;
    - знаходження довжин медіан;
    - знаходження довжин середніх ліній.
34. Створити модуль, який містить функції для роботи з прямокутниками:
    - знаходження площі;
    - знаходження периметру;
    - знаходження радіуса вписаного і описаного кіл:
    - знаходження довжин діагоналей.
35. Створити модуль, який містить функції для роботи з паралелограмами:
    - знаходження площі різними способами;
    - знаходження периметру;
    - знаходження довжин діагоналей.
36. Створити модуль, який містить функції для роботи з трапеціями:
    - знаходження площі;
    - знаходження периметру;

- знаходження середньої лінії.

37. Створити модуль, який містить наступні функції для роботи з опуклим чотирикутником, заданим координатами своїх вершин на площині:
    - визначення типу чотирикутника;
    - знаходження периметру чотирикутника;
    - знаходження площі чотирикутника;
    - визначення того, чи лежить задана точка в межах чотирикутника;
    - чи перетинаються два задані чотирикутники.
38. Створити модуль, який містить процедури та функції для виконання операцій з векторами (встановлення паралельності та перпендикулярності двох векторів, визначення кута між векторами) .
39. Створити модуль, який містить процедури та функції для виконання операцій з векторами (введення, виведення вектора стовпця та транспонованого вектора).
40. Створити модуль, який містить процедури та функції для виконання операцій з векторами (знаходження скалярного добутку, множення вектора на число, знаходження скалярного добутку) .
41. Створити модуль, який містить процедури та функції для виконання операцій з векторами (встановлення паралельності та перпендикулярності двох векторів, визначення кута між векторами) .
42. Створити модуль, який містить процедури та функції для виконання операцій з векторами (поворот на вказаний кут, нормування вектора, визначення проекції одного вектора на інший) .
43. Створити модуль, який містить підпрограми сортування одновимірного масиву різними способами.
44. Створити модуль, який містить підпрограми для роботи з упорядкованим за зростанням одновимірним масивом (додавання/вилучення елемента не повинно порушувати упорядкованості масиву).

45. Створити модуль, який містить підпрограми для знаходження максимального і мінімального елементів одновимірного масиву.
46. Створити модуль, який містить підпрограми для знаходження суми, добутку всіх елементів одновимірного масиву, а також тільки додатних та тільки від'ємних.
47. Створити модуль, який містить підпрограми для знаходження середнього арифметичного, середнього арифметичного, квадратичного та середнього гармонійного елементів одновимірного масиву.
48. Створити модуль, який містить процедури введення та виведення матриць.
49. Створити модуль, який містить функції для підрахунку суми та добутку елементів матриці а також тільки тих, які знаходяться вище/нижче головної діагоналі.
50. Створити модуль, який містить функції для знаходження максимального, мінімального елементів матриці, а також тих, які знаходяться вище/нижче головної діагоналі.
51. Створити модуль, який містить підпрограми множення матриці на матрицю та множення матриці на вектор.
52. Створити модуль, який містить процедури для виконання операцій додавання матриць та множення матриці на число.
53. Створити модуль, який містить функції для обчислення детермінанта матриць порядку 2 і 3.
54. Створити модуль, який містить функцію для транспонування квадратної матриці порядку n та булеву функцію для визначення чи містить дана матриця нулі.
55. Створити модуль, який містить функції для наближеного обчислення інтегралу різними методами (лівих/правих/середніх прямокутників, трапецій, сімпсона).

56. Створити модуль, який містить підпрограми для знаходження кореня рівняння $f(x)=0$ на проміжку $[a;b]$ (метод поділу відрізка пополам, метод Ньютона).
57. Створити модуль, який містить підпрограми для знаходження розв'язків системи n лінійних рівнянь з n невідомими.
58. Створити модуль, який містить підпрограми для роботи з динамічною структурою стек (додавання нового елемента, вилучення, виведення елементів на екран).
59. Створити модуль, який містить підпрограми для роботи з динамічною структурою черга (додавання нового елемента, вилучення, виведення елементів на екран).
60. Створити модуль, який містить підпрограми для роботи з динамічною структурою бінарне дерево пошуку(додавання нового елемента, вилучення, виведення елементів на екран).
61. Створити модуль, який містить такі підпрограми для роботи з текстовими файлами:
    - визначення кількості рядків:
    - визначення кількості рядків, довжина яких не перевищує k символів;
    - визначення суми всіх чисел, які присутні у текстовому файлі;
62. Створити модуль, який містить такі підпрограми для роботи з файлом дійсних чисел:
    - знаходження максимального/мінімального елементів:
    - знаходження найбільшого серед від'ємних та найменшого серед додатних.
63. Створити модуль, який містить такі підпрограми для роботи з нетипізованим файлом:
    - запис/зчитування певної кількості цілих чисел;
    - запис/зчитування певної кількості дійсних чисел;
    - створення копії довільного файлу.

БІБЛІОТЕКИ DLL

Поняття бібліотеки DLL

При розгляді модульного програмування було відмічено основні його недоліки:

- модуль може бути присутнім в оперативній пам’яті в декількох екземплярах;
- оскільки код модуля приєднується до коду програми, то збільшується час її завантаження за рахунок завантаження підпрограм, які можливо використовуватися не будуть.

Саме ці недоліки призвели до появи бібліотек з динамічним підключенням, або ж бібліотек DLL (Dynamic-link Library). Бібліотеки DLL мають багато спільного з модулями. Вони, як і модулі, є окремо скомпільованими програмними одиницями, що можуть містити описання підпрограм, та можуть бути використані в основній програмі. Окрім цього бібліотеки можуть містити дані або ресурси, які спільно можуть використовувати декілька програм. І в той же час у бібліотеках відсутні наведені раніше недоліки. Отже, основними перевагами таких бібліотек є:

- як і модулі, бібліотеки дозволяють спростити процес розробки програм, за рахунок використання структурного підходу;
- які і модулі, бібліотеки дозволяють розв’язувати проблему повторного використання вже розробленого програмного коду;
- бібліотека може містити дані та інші ресурси, що можуть бути використані в інших програмах;
- код бібліотеки не приєднується до програми у процесі компіляції;
- бібліотека DLL в оперативній пам’яті завжди знаходиться тільки в одному екземплярі і її спільно використовують інші програми;
- бібліотеки DLL можуть завантажуватися у оперативну пам’ять не одразу з основною програмою, а за необхідністю у процесі роботи програми;

– оскільки бібліотеки DLL не об'єднуються з програмою у процесі компіляції, то простою їх заміною легко здійснити оновлення програми та здійснювати виправлення помилок.
– бібліотеки дозволяють розширювати функціональні можливості програми шляхом підключення до неї нових бібліотек (плагінів).
– бібліотеки розроблені на одній мові програмування можуть бути використані при створенні програм на інших мовах програмування.

Структура бібліотеки DLL

Наведемо загальну структуру бібліотеки DLL.

```
LIBRARY  <Ім'я бібліотеки>;
  USES       <Список модулів>;
  CONST      <Опис констант>;
  TYPE       <Опис типів користувача>;      Описова частина
  VAR <Опис змінних>;
  <Опис процедур і функцій>
  {$R <Ім'я файлу ресурсів>.RES}            Підключення файлу ресурсів
EXPORTS
      <Список процедур і функцій для експорту >   Експортна частина
BEGIN
      <Частина ініціалізації >
END.
```

Заголовок бібліотеки складається із зарезервованого слова LIBRARY та імені бібліотеки

```
LIBRARY  <Ім'я бібліотеки>;
```

Ім'я бібліотеки задається у формі ідентифікатора і повинно співпадати з іменем дискового файлу проекту. Так, наприклад, якщо маємо бібліотеку, заголовок якої

```
LIBRARY MyLib;
```

то текст відповідної бібліотеки повинен зберігатися у однойменному дисковому файлі проекту з розширенням "Dpr" (MyLib.Dpr). При компіляції

створюється файл з розширенням “Dll” (MyLib.Dll). Саме це ім’я і використовується для підключення бібліотеки до основної програми.

Описова частина бібліотеки йде одразу після її заголовку. Тут містяться описання всіх глобальних об’єктів бібліотеки (типів, констант, змінних та підпрограм). На відміну від модулів, описані типи, константи і змінні не є доступними для програми, що підключає даний модуль. Вони розглядаються лише як глобальні власні об’єкти бібліотеки. І хоча бібліотека в оперативній пам’яті завжди існує тільки в одному екземплярі, екзем плярів змінних створюється стільки, скільки є програм, що використовують дану бібліотеку (для кожної програми окремі змінні). Звертатися до цих змінних програма можна за допомогою підпрограм цієї ж бібліотеки.

Опис процедур і функцій здійснюється аналогічно до їх опису у програмі. Єдиною особливістю опису є те, що після заголовку підпрограми необхідно вказати директиву, за допомогою якої визначається спосіб передачі параметрів підпрограмі. Оскільки бібліотеки можуть бути використані у різних мовах програмування, то, як правило, при описі вказують директиву StdCall, яка вказує на стандартний спосіб передачі параметрів (через стек). За замовчуванням вважається, що вжито директиву Register (передача здійснюється з використанням регістрів). Хоча такий спосіб передачі параметрів є швидшим, але він є несумісним при використанні інших мов програмування. У зв’язку з цим, рекомендується завжди вказувати директиву StdCall.

Приклад.

```
LIBRARY  MyLib;
  USES   SysUtils,  Classes;
  function Sum(x,y:Real):Real;  StdCall;
  Begin
    Result:=x+y;
  End;
……..
```

Слід також зазначити, що у випадку, коли один із параметрів підпрограми є величиною типу String або функція повертає значення цього типу, то обов'язково першим у списку використовуваних модулів повинен бути модуль ShareMem. Крім цього додаткового використовується бібліотека BORLNDMM.DLL . Про це йдеться у повідомленні-коментарі, що включено у шаблон бібліотеки при її створенні (File ► New ► Other ► Dll Wizard).

```
library Project1;
{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  Project-View Source) USES clause if your DLL exports any procedures or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL --even those that
  are nested in records and classes. ShareMem is the interface unit to
  the BORLNDMM.DLL shared memory manager, which must be deployed along
  with your DLL. To avoid using BORLNDMM.DLL, pass str ing information
  using PChar or ShortString parameters. }
uses SysUtils, Classes;
{$R *.res}
begin
end.
```

Експортна частина бібліотеки розпочинається службовим словом EXPORTS, після якого необхідно вказати список підпрограм, які експортуються і можуть бути використані в основній програмі. При цьому для підпрограм можна визначати інші (зовнішні) імена та індекси (цілі числа). Хоча індекси прискорюють пошук відповідної підпрограми під час виклику, але їх використовують рідко (іноді їх використання може приз вести до збоїв). Як правило, підпрограми експортуються за власними або іншими зовнішніми іменами.

```
EXPORTS
<Ім'я функції> index <індекс> name  <Зовнішнє ім'я функції (для експорту) >,
        ……..
```

Приклад.

```
LIBRARY  MyOperations;
  USES   SysUtils,  Classes;
  function Sum(x,y:Real):Real;  StdCall;
  Begin
    Result:=x+y;
  End;
  function Sub(x,y:Real):Real;  StdCall;
  Begin
    Result:=x-y;
  End;
  function Mul(x,y:Real):Real;  StdCall;
  Begin
    Result:=x*y;
  End;
  function Diviz(x,y:Real):Real;  StdCall;
  Begin
     if y=0 then   Result:=0
      else Result:=x/y;
  End;
  Procedure Cub(var x:real);   StdCall;
  Begin
     x:=x*x*x;
   End;
EXPORTS
  Sum name 'Suma',                  //Вказано зовнішнє ім'я 'Suma'
  Sub,                              //Зовн. ім'я співпадає з внутрішнім 'Sub'
  Mul index 1 name 'Mult',          //Вказано індекс 1 та зовн. ім'я 'Mult'
  Diviz index 2,                    //Вказано індекс 2
  Cub;
END.
```

Якщо в експортній частині для підпрограми вказано і індекс і інше зовнішнє ім'я, то викликати цю підпрограму можна або за індексом, або за зовнішнім

іменем. Хочеться відмітити, що експортна частина бібліотеки чимось нагадує описову частину INTERFACE у модулях, з тієї відмінністю, що у експортній частині описують тільки підпрограми, які можуть б ути викликані з основної програми.

Підключення бібліотек

Бібліотека може бути використана у програмі шляхом неявного або явного завантаження. У випадку неявного завантаження іноді кажуть, що бібліотеку підключено статично, у випадку ж явного завантаження – динамічно. Оскільки при підключенні бібліотеки не вказується шлях до неї, то бібліотека повинна знаходитись або у тому ж каталозі, що і програма, або у якомусь із стандартних, у яких операційна система автоматично знайде її (System,System32, коренева папка Windows). При цьому якщо бібліотека не була завантажена, то її буде завантажено, в іншому випадку буде збільшено лічильник бібліотеки, у якому міститься кількість програм, що її на даний момент використовують.

Неявне завантаження

Неявне завантаження тому називається неявним, бо воно відбувається одразу із завантаженням основної програми, якщо звичайно до цього моменту бібліотеку не було вже завантажено. При неявному завантаженні підпрограми з бібліотеки необхідно виконати наступні кроки:

- в описовій частині модуля, у якому використовуватиметься підпрограма з бібліотеки, необхідно описати заголовок підпрограми, параметри якої в точності відповідають підпрограмі, яку будемо експортувати з бібліотеки (ім’я підпрограми у модулі може бути довільним);
- у частині реалізації вказати на те, що описана підпрограма є зовнішньою і вона експортується з бібліотеки за індексом чи за зовнішнім іменем. Необхідно мати на увазі, що при експорті

підпрограми за іменем, зовнішнє ім’я підпрограми необхідно вказувати з точністю до регістра.

```
UNIT <Ім’я модуля>;
INTERFACE
   ……….....
  function <ім’я підпрограми у модулі>(<параметри>):<тип>;  StdCall;
IMPLEMENTATION
      //Експорт за іменем
function <ім’я підпрограми у модулі>; external '<ім’я бібліотеки>.dll' name 'зовнішнє ім’я підпрограми';
      //Експорт за індексом
function <ім’я підпрограми у модулі>; external '<ім’я бібліотеки>.dll' index 'індекс підпрограми';
  ……..
END.
```

Після такого опису описані підпрограми можуть бути використані у межах цього модуля як звичайні власні підпрограми.

Приклад. Використаємо функції Sum та Mul, які описані у бібліотеці MyOperations. Функцію Sum у модулі назвемо ADD, а функцію Mul назвемо Multiplication. Відмітимо, що параметри відповідних функцій повністю співпадають.

```
UNIT unit1;
INTERFACE
   ……….....
  function ADD(x,y:Real):Real;  StdCall;
  function Multiplication(x,y:Real):Real;  StdCall;
IMPLEMENTATION
  function ADD;             external 'MyOperations.dll' name 'Suma';
  function Multiplication; external 'MyOperations.dll' index 1;
  ……..
procedure TForm1.Button1Click(Sender: TObject);
 var a,b,s,m:real;
begin
 a:=strtofloat(Edit1.Text);
 b:=strtofloat(Edit2.Text);
 s:= ADD (a,b);                  //Виклик бібліотечної функції Sum
 m:= Multiplication (a,b);       // Виклик бібліотечної функції Mul
 ……..
end;
END.
```

Явне завантаження

Явне завантаження відрізняється від неявного тим, що завантаження бібліотеки відбувається не одразу з програмою, а у разі потреби (явно), з використанням відповідних операторів. Для явного завантаження бібліотеки необхідно виконати наступні кроки:

1) в розділі описання типів користувача описати відповідні типи підпрограм;
2) описати відповідні змінні типу підпрограм;
3) описати змінну вказівник (типу THandle) для збереження адреси бібліотеки, яку буде завантажено;
4) завантажити бібліотеку за допомогою функції LoadLibrary (якщо завантаження не відбулося то функція повертає значення 0);
5) знайти у завантаженій бібліотеці адресу необхідної підпрограми за допомогою функції GetProcAddress (якщо необхідну функцію не знайдено, то функція повертає nil);
6) використовувати підпрограму, як звичайну внутрішню підпрограму;
7) після використання підпрограм бібліотеки звільнити бібліотеку за допомогою процедури FreeLibrary. При відключенні бібліотеки лічильник бібліотеки зменшується на одиницю. На віть якщо він дорівнює 0 бібліотека може залишатися в оперативній пам'яті. Але якщо операційна система потребує цю пам'ять, то бібліотека в цьому випадку остаточно вивантажується.

```
UNIT <Ім'я модуля>;

INTERFACE
        ……..
Type
  <ім'я типу підпрограми> = function (<параметри>):<тип>; StdCall;      {1}
        ……..
IMPLEMENTATION
        ……..
procedure <ім'я процедури>(<параметри>);
 var   <вказівник на бібліотеку>: THandle;                                {2}
```

```
        <змінна підпрограма >:<ім'я типу підпрограми>;                    {3}
        ……..
begin
        ……..
   <вказівник на бібліотеку>:= LoadLibrary('<ім'я бібліотеки>');          {4}
@<змінна підпрограма>:= GetProcAddress(<вказівник на бібліотеку>,
                                        <зовнішнє ім'я функції>);         {5}
        ……..
        {Використання бібліотечної функції}                              {6}
        ……..

        FreeLibrary(<вказівник на бібліотеку>);                           {7}
        ……..
end;
END.
```

Приклад. Проілюструємо явне завантаження функції на прикладі функції Sum, описаної у бібліотеці MyOperations.

```
unit Unit1;

INTERFACE
……..
TYPE
……..
  Tfunc = function (x,y:Real):Real;  StdCall;        {1 опис типу підпрограми}
……..
IMPLEMENTATION
procedure TForm1.Button1Click(Sender: TObject);
 var x,y,s:real;
     DllHandle:THandle;                              {2 опис вказівника на бібліотеку}
     f:Tfunc;                                        {3 опис змінної підпрограми}
begin
 x:=strtofloat(Edit1.Text);
 y:=strtofloat(Edit2.Text);
 DllHandle:=LoadLibrary('MyOperations.dll');         {4 завантаження бібліотеки}
 if DllHandle=0 then
    ShowMessage('Бібліотеку не знайдено!')
 else
  begin
   @f:=GetProcAddress(DllHandle,'Suma');             {5 пошук адреси функції 'Suma'}
   if @f=nil then
     ShowMessage('Функцію не знайдено!')
   else
    begin
     s:=f(x,y);                                      {6 використання функції}
     Edit3.Text:=floattostr(s);
```

```
   end;
  end;
  FreeLibrary(DllHandle);                    {7 звільнення бібліотеки }
end;
END.
```

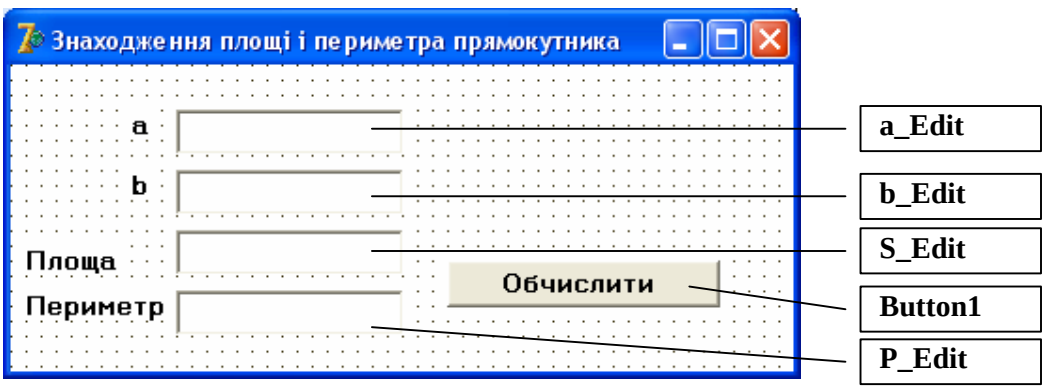
Розглянемо ще один приклад використання бібліотеки Dll.

Приклад. Створити бібліотеку, яка містить функції для знаходження площі та периметра прямокутника.

При розв'язанні даного завдання створимо проект Project1

```
program Project1;
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
{$R *.res}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

та форму Form1



Створимо також бібліотеку SquareLib.Dll та модуль Unit1, у якому буде підключатися бібліотека. Проілюструємо два способи підключення цієї бібліотеки (відповідно наведемо два можливі варіанти модуля Unit1).

Бібліотека SquareLib.Dll

```
library SquareLib;
uses
  SysUtils,
  Classes;
function Square(a,b:Real):Real ;
                              stdcall;
begin
 Result:=a*b;
end;
function
Perimeter(a,b:Real):Real;
                              stdcall;
begin
 Result:=2*a+2*b;
end;
{$R *.res}
exports Square,
          Perimeter;
END.
```

Модуль Unit1
(неявне завантаження)

```
unit Unit1;
interface
uses  Windows, Messages, SysUtils,
Variants, Classes, Graphics, Controls,
Forms,  Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    a_Edit: TEdit;    Label1: TLabel;
    b_Edit: TEdit;    Label2: TLabel;
    S_Edit: TEdit;    Label3: TLabel;
    P_Edit: TEdit;    Label4: TLabel;
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  end;
 function fS(a,b:Real):Real; stdcall;
 function fP(a,b:Real):Real; stdcall;
var  Form1: TForm1;  a,b,S,P:real;
implementation
 function fS; external 'SquareLib.dll'
                              name 'Square';
 function fP; external 'SquareLib.dll'
                              name
'Perimeter';
procedure TForm1.Button1Click(Sender:
TObject);
begin      a:=strtofloat(a_Edit.Text);
           b:=strtofloat(b_Edit.Text);
 S:=fS(a,b);
 S_Edit.Text:=floattostr(S);
 P:=fP(a,b);
 P_Edit.Text:=floattostr(P);
end;
end.
```

Модуль Unit1
(явне завантаження)

```
unit Unit1;
interface
uses   Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms,  Dialogs, StdCtrls;
type  TForm1 = class(TForm)
        a_Edit: TEdit;    Label1: TLabel;  b_Edit: TEdit;
        Label2: TLabel;  S_Edit: TEdit;    Label3: TLabel;
        P_Edit: TEdit;    Label4: TLabel;   Button1: TButton;
        procedure Button1Click(Sender: TObject);
      end;
 TFunc=function (a,b:Real):Real; stdcall;
var  Form1: TForm1;  a,b,S,P:real;

implementation
procedure TForm1.Button1Click(Sender: TObject);
 var   DllHandle:THandle;
       fS,fP:TFunc;
begin
 a:=strtofloat(a_Edit.Text);
 b:=strtofloat(b_Edit.Text);
 DllHandle:=LoadLibrary('SquareLib.dll');

@fS:=GetProcAddress(DllHandle,'Square');
@fP:=GetProcAddress(DllHandle,'Perimeter');
    P:=fP(a,b);
    P_Edit.Text:=floattostr(P);
    S:=fS(a,b);
    S_Edit.Text:=floattostr(S);
  FreeLibrary(DllHandle);
end;
end.
```

Частина ініціалізації

Частина ініціалізації відкривається зарезервованим словом BEGIN та містить фрагмент програми, який виконується при підключенні бібліотеки до програми. Як правило, частина ініцілізації використовуються для підготовки бібліотеки до роботи (початкової ініціалізації). У цій частині можуть ініціалізовуватися змінні, відкриватися потрібні файли, створюватися динамічні масиви і т.д. Якщо подібних дій виконувати не потрібно, то частина ініціалізації не вказується.

Аналіз подій для бібліотеки

Раніше зазначалося, що у бібліотеці є частина ініціалізації, яка виконується під час підключення бібліотеки. Окрім цього для бібліотеки визначено глобальну змінну вказівник DLLProc, яка містить вказівник на процедуру, що викликається у випадку, якщо для бібліотеки настає одна із подій:

- DLL_PROCESS_ATTACH – подія підключення бібліотеки;
- DLL_PROCESS_DETACH – подія відключення бібліотеки;
- DLL_THREAD_ATTACH – подія створення нового потоку;
- DLL_THREAD_DETACH – подія закриття нового потоку.

За замовчуванням вказівник DLLProc містить значення nil, але в разі необхідності цій змінній можна надати адресу певної процедури користувача, у якій буде здійснюватися аналіз вказаних подій. Як правило це роблять у частині ініціалізації бібліотеки. У загальному, така процедура користувача може мати структуру процедури DllEvents.

```
LIBRARY  <Ім’я бібліотеки>;
USES WINDOWS, … ;
……..
procedure DllEvents (event:DWord);
Begin
```

```
        case event of
          DLL_PROCESS_ATTACH : <Дії при підключенні бібліотеки>;
          DLL_PROCESS_DETACH :<Дії при відключенні бібліотеки>;
          DLL_THREAD_ATTACH  :<Дії при створенні нового потоку>;
          DLL_THREAD_DETACH  :<Дії при закритті нового потоку>;
        end;
End;
BEGIN
        DLLPROC:=@DLLEvents;          //Передаємо адресу DLLEvents
        DLLEVENTS(DLL_PROCESS_ATTACH); // Виклик DLLEvents
                                       // (подія підключення)
END.
```

Тут у розділі підключення модулів підключено модуль WINDOWS, який необхідний у зв'язку з використанням типу DWord для збереження параметра події.

Розглянемо використання бібліотеки на прикладі, розв'язаному раніше з використанням модулів.

Приклад. Дослідник щодня здійснює виміри температури та знаходить середнє арифметичне. Знайдене середнє арифметичне зберігається у файлі для подальших досліджень (якщо файлу не існує, то він створюється, якщо ж файл існує, то результат разом із датою вимірювань дописується у кінець файлу). Створити бібліотеку, яка містить процедури, для розв'язання поставленої задачі.

Текст бібліотеки

```
LIBRARY DoslidLib;
 uses
  SysUtils,
  Classes,Windows;
 Type TVector=array of Real;
 Var f:TextFile;
     n:integer;
     V:TVector;
 Function GetAverage:Real; stdcall;
```

```
   var s:Real;
       i:integer;
  Begin
   s:=0;
   for i:=0  to n-1 do
   begin
    s:=s+V[i];
   end;
   Result:=s/n;
  End;
 Procedure GetData;    stdcall;
   var i:integer;
  Begin
    writeln('Кількість вимірів = ');
    Readln(n);
    SetLength(V,n);
    for i:=0 to n-1 do
    begin
      write(i,'-вий вимір : ');
      readln(V[i]);
    end;
  End;
 Procedure SaveToFile; stdcall;
  Begin
    writeln(f,DateToStr(Date),'   -  ',GetAverage);
  End;
procedure DllEvents (event:DWord);
Begin
 case event of
   DLL_PROCESS_ATTACH : begin
       {Виконується при підключенні бібліотеки}
                         AssignFile(f,'MyFile.txt');
                         if FileExists('MyFile.txt') then
                          Append(f)
                         else
                          Rewrite(f);
                        end;

   DLL_PROCESS_DETACH :begin
       {Виконується при відключенні бібліотеки}
                        CloseFile(f);
                       end;
 end;
End;

EXPORTS GetData,
        SaveToFile;
BEGIN
 DLLPROC:=@DLLEvents;
```

```
 DLLEVENTS(DLL_PROCESS_ATTACH);
END.
```

Текст проекту

```
program Project1;
{$APPTYPE CONSOLE}
uses
  SysUtils,windows;
Type
  TProc=Procedure; stdcall;
Var
  DllHandle:THandle;
  GetData,SaveToFile:TProc;
begin
 DllHandle:=LoadLibrary('DoslidLib.dll');
 if DllHandle=0 then
     writeln('Бібліотеку не знайдено!')
 else
  begin
   @GetData:=GetProcAddress(DllHandle,'GetData');
   @SaveToFile:=GetProcAddress(DllHandle,'SaveToFile');
   if (@GetData=nil)or(@SaveToFile=nil) then
     writeln('Функції не знайдено!')
   else
    begin
     GetData;
     SaveToFile;
    end;
  end;
  FreeLibrary(DllHandle);
end.
```

Недоліки бібліотек

Не дивлячись на велику кількість позитивних якостей бібліотек Dll, очікуваних переваг від їх застосування на практиці отримати не вдалось. Це пов’язано з тим, що нерідко траплялися ситуації, коли декілька програм одночасно намагаються використовувати не повністю сумісні різні версії деякої бібліотеки. В результаті виникали збої у роботі цих прог рам. Тому у пізніх версіях операційної системи Microsoft Windows почали застосовувати підхід, при якому дозволяється одночасно паралельно використовувати різні версії бібліотек. А отже втрачається перевага у використанні бібліотек.

Запитання для самоконтролю

1. *Що таке бібліотека Dll?*
2. *Чим відрізняється бібліотека від модуля?*
3. *Що спільного між бібліотекою і модулем?*
4. *Яка структура бібліотеки?*
5. *Що можна експортувати з бібліотеки ?*
6. *Які способи експорту підпрограм ви знаєте?*
7. *Які особливості використання глобальних змінних у бібліотеках?*
8. *Які є способи завантаження бібліотеки?*
9. *Як реалізувати неявне завантаження (статичне підключення) бібліотеки?*
10. *Як реалізувати явне завантаження (динамічне підключення) бібліотеки?*
11. *Чи важливим є регістр у записі зовнішніх імен підпрограм при їх експорті?*
12. *Коли використовується частина ініціалізації?*
13. *Що таке* `DLLPROC` *?*
14. *Які переваги і недоліки бібліотеки?*

Завдання для самостійної роботи

В якості завдань для самостійного виконання можна взяти індивідуальні завдання, які наведені при розгляді модулів, і виконати їх з використанням бібліотек.

ЗМІСТ

Відповідальний за випуск: завідувач кафедрою системного аналізу і теорії оптимізації к. ф.-м. н., доц. Кузка О.І.

Автори: к. т. н., доц. Семйон І.В.,
к. ф.-м. н., доц. Чупов С.В.,
к. ф.-м. н., Брила А.Ю.,
к. пед. н., Апшай Н.І.

Рецензенти:

ВИКОРИСТАННЯ МАСИВІВ У DELPHI

Методичні вказівки до лабораторних робіт для студентів І І-го курсу математичного факультету спеціальності "прикладна математика "