Модульность

---

1. Понятие модульности

При составлении программ программисты стремятся к обобщению своего опыта, накапливая процедуры или просто фрагменты программ, которые бы допускали многократное использование. Весь вопрос в том в какой форме это следует делать. Можно, конечно, хранить тексты старых программ, вырезая из них куски с помощью редактора текста и вставляя их в новую программу по мере необходимости. Однако, эта проблема оказалась на практике столь важной, что для её решения были придуманы более совершенные средства, которые кратко принято обозначать словом модульность.

Модульность - это возможность представить программу в виде нескольких частей, называемых ***модулями***. Модульность можно рассматривать как следствие применения принципа абстракции для борьбы со сложностью программы (принцип "разделяй и властвуй"). В этой связи естественными модулями программы являются процедуры. Но в данном случае речь идёт о более крупных программных единицах, с одной стороны, и некоторой технологической поддержке их соединения в единую программу, с другой. В этой связи появляется понятие модуля как ***единицы компиляции***, поддерживаемое технологически системной ***программой компоновки*** модулей в единую исполняемую программу.

Совершенно ясно, что такие модули могут служить прекрасной формой накопления программистского опыта. Программист в своей программе должен лишь указать из каких модулей необходимо включить те или иные программные единицы и они будут включены компоновщиком.

К счастью, в среде Turbo Pascal компиляция многомодульных программ и их компоновка хорошо автоматизирована, что позволяет нам без задержки перейти к рассмотрению логической организации многомодульной программы.

2. Организация модуля в системе Turbo Pascal

Структуру модуля рассмотрим на примере организации модуля Stack, который реализует модель стека. ***Стек*** - это очень популярное в информатике безадресное устройство для хранения данных. Он работает по принципу "последним вошёл - первым вышел". Про элемент стека, который вошёл в него последним, говорят, что он находится в вершине стека. Модуль Stack должен быть размещён в одноимённом файле Stack.pas:

```
unit Stack;
interface

function Push(v: Integer): Boolean;
function Pop(var v: Integer): Boolean;
function Top(var v: Integer): Boolean;
```

```
implementation

const m = 5;
var St: array[1..m] of Integer;
    sp: Integer;

function Push(v: Integer): Boolean;
var Res: Boolean;
begin
Res:=sp<m;
if Res then begin Inc(sp); St[sp]:=v end;
Push:=Res;
end; {Push}

function Pop(var v: Integer): Boolean;
var Res: Boolean;
begin
Res:=sp>0;
if Res then begin v:=St[sp]; Dec(sp) end;
Pop:=Res;
end; {Pop}

function Top(var v: Integer): Boolean;
var Res: Boolean;
begin
Res:=sp>0;
if Res then v:=St[sp];
Top:=Res;
end; {Top}

begin sp:=0 end.
```

Модуль начинается с ключевого слова **unit** с указанием имени модуля (Stack) и состоит из трёх разделов (частей) - ***интерфейсного***, ***реализации*** и ***инициализации***. Интерфейсная часть модуля расположена в его начале после ключевого слова **interface**. Далее после ключевого слова **implementation** располагается часть реализации. Необязательная часть инициализации располагается в конце модуля за ключевым словом **begin**. Завершает модуль ключевое слово **end** с точкой.

Организация модуля подчинена принципу абстракции. Интерфейсный раздел модуля может включать описания констант, типов, переменных и заголовков процедур, которые должны быть видимы в модулях-клиентах. Раздел реализации должен содержать описание тех процедур, заголовки которых приведены в интерфейсной части модуля. В этой части модуля также могут быть расположены описания констант, типов, переменных и процедур (заголовки которых отсутствуют в интерфейсной части), но они не видимы для модуля-клиента. Их назначение состоит в поддержке реализации процедур объявленных в интерфейсе модуля. Операторы раздела инициализации модуля выполняются только один раз сразу же после загрузки модуля в основную память.

Таким образом можно считать, что те программные объекты, описание которых находятся в интерфейсной части модуля, включены в контекст модуля-клиента и он может ими свободно пользоваться. В случае конфликта имен, имя из модуля можно уточнить с помощью префикса из имени модуля, который присоединяется через точку, например, Stack.Push(5).

Интерфейсная часть модуля *Stack* включает заголовки трёх функций, вызовы которых играют роль операций со стеком. Все три функции возвращают коды завершения, позволяющие судить об успешности выполнения операций. Операция *Push* помещает

параметр *v* в вершину стека, если, конечно, он не переполнен. Операция *Pop* выталкивает из стека элемент, который находится в его вершине, если, конечно он не пуст. Параметр *v* передаёт значение вытолкнутого элемента. И, наконец, операция *Top* позволяет просто определить значение элемента в вершине стека, если он не пуст.

В часть реализации модуля *Stack* необходимо включить всё то, что позволяет реализовать операции из его интерфейсной части. Мы совершенно сознательно должны постараться скрыть в этой части все детали реализации, знать о которых пользователю модуля нет никакой необходимости. В частности, в данном случае стек моделируется массивом *St* из *m* элементов. Переменная *sp* играет роль указателя вершины стека. В начале работы стека эта переменная должна принять нулевое значение (см. часть инициализации модуля).

Теперь рассмотрим пример модуля-клиента модуля Stack. Его роль играет главная программа TstStack.pas:

```
program TstStack;
uses Stack;

const m = 6;
      Dat: array[1..6] of Integer =
           (2,3,5,7,11,13);
var i,Res: Integer;

begin
i:=1;
while Push(Dat[i]) do Inc(i);
if Top(Res) then WriteLn('Top = ', Res);
while Pop(Res) do Write(Res:3);
WriteLn;
end.
```

Выход программы TstStack.pas:

```
 Top = 11
 11  7  5  3  2
```

Программа достаточно очевидна. Предложение **uses** в её начале содержит список используемых модулей. В качестве модулей-клиентов могут выступать не только программы, но и другие модули. В этом случае предложения **uses** в них включается сразу после слов **interface** и/или **implementation** в зависимости от того, к какой части это подключение требуется.

В системе Turbo Pascal откомпилированные модули имеют расширение tpu (Turbo Pascal Unit).

3. Стандартный модуль Crt

В систему Turbo Pascal входит большое количество стандартных модулей. Большое значение имеет модуль поддержки выполнения Pascal-программ System. Он по умолчанию подключается к каждой программе. Из других модулей мы рассмотрим в общих чертах только модули Crt и Graph. Оба предназначены для создания удобного интерфейса пользователя.

Модуль Crt предназначен для работы с клавиатурой и монитором в алфавитно-цифровом режиме. Из всех его богатых возможностей мы рассмотрим только три. Вот так они выглядят в интерфейсной части этого модуля.

```
unit Crt;
...
procedure ClrScr;              {Очистка экрана}
function KeyPressed: Boolean; {"Нажата какая-то клавиша"}
function ReadKey: Char;        {Возвращает литеру нажатой клавиши}
...
```

Смысл функций достаточно очевиден. Некоторых пояснений требует лишь функция ReadKey. Дело в том, что для специальных клавиш она возвращает литеру с кодом 0. В этом случае для получения кода такой литеры функцию ReadKey надо вызвать ещё раз. Следующая программа позволяет определить, какие коды соответствуют тем или иным клавишам.

```
program Key;
uses Crt;
var ch,che: Char;
begin
ClrScr;
WriteLn('Press "Esc" for exit.');
repeat
   Write('Press any key...');
   while not KeyPressed do; {Цикл ожидания нажатия на клавишу}
   ch:=ReadKey;
   Write(Ord(ch):4);
   if Ord(ch)=0 then
      begin
      che:=ReadKey;
      Write(Ord(che):4);
      end;
   WriteLn;
   if Ord(ch)=27 then Exit;
until False;
end.
```

Рассмотренные функции модуля Crt позволяют достаточно просто организовать меню пользователя. Пример.

```
program Menu;
uses Crt;
var ch:  Char;

procedure Wait;
begin
Write('Pres any key...');
while not KeyPressed do;
ReadKey;
if Ord(ch)=0 then ReadKey;
end; {Wait}

procedure Proc1;
begin
ClrScr;
WriteLn('Proc1 executed!');
Wait;
end; {Proc1}
```

```
procedure Proc2;
begin
ClrScr;
WriteLn('Proc2 executed!');
Wait;
end; {Proc2}

begin
repeat
   ClrScr;
   WriteLn('    M E N U');
   WriteLn('--------------------  ');
   WriteLn('Esc: Exit.');
   WriteLn('1:   First selection.');
   WriteLn('2:   Second selection.');
   WriteLn('----------------------');
   Write('Your choice: ');

   while not KeyPressed do;
   ch:=ReadKey; if Ord(ch)=0 then ReadKey;

   case ch of
     '1': Proc1;
     '2': Proc2;
     #27: Exit;
   end; {case}

until False;
end.
```

4. Стандартный модуль Graph

Модуль Graph предназначен для работы с монитором в графическом режиме. В этом режиме весь экран монитора представляется в виде прямоугольной матрицы графических элементов - ***пикселов***.

В модуле Graph положение на экране отдельного пиксела задаётся его целочисленными координатами в системе координат (СК) экрана. Начало этой СК находится в левом верхнем углу экрана. Ось 0X проходит горизонтально от начала СК вправо, а ось 0Y - вертикально от начала СК вниз.

Надо отметить, что существует большое разнообразие графических систем и режимов их работы. В примере, который приводится ниже, предполагается использовать так называемый режим VGA, который поддерживается драйвером EGAVGA.BGI из каталога Bgi системы Turbo Pascal. Этот режим характеризуется размерами матрицы пикселов 640 x 480. Все приводимые далее данные по графической системе будут относиться к режиму VGA.

Кроме положения на экране каждый пиксел характиризуется цветом. Цвет пиксела кодируется целым числом из диапазона 0..15 (например, 1 - голубой, 2 - зелёный, 4 - красный). Кроме того, существует понятие "цвет фона", который кодируется целым числом из диапазона 0..7.

Интерфейс модуля Graph достаточно велик и сложен. Для первого знакомства с графическим вводом-выводом нам будет достаточно небольшой его части.

Введём понятие графического курсора (ГК), который невидимо указывает на определённый пиксел графической матрицы. Некоторые функции устанавливают положение ГК явным образом. Ряд функций, например, для проведения линий, устанавливают его в конечную точку неявно.

```
unit Craph;
...
procedure ClearDevice;        {Заполняет экран цветом фона}
procedure CloseGraph;         {Закрывает графический режим}
function GetX:Integer;        {Возвр. координату X ГК}
function GetY:Integer;        {Возвр. координату Y ГК}
function GetMaxX:Integer;     {Возвр. макс. значение координаты X}
function GetMaxY:Integer;     {Возвр. макс. значение координаты X}
procedure SetColor(Color:Word);       {Уст. цвет линий и литер текста}
procedure SetBkColor(Color:Word);     {Уст. цвет фона}
procedure MoveTo(X,Y:Integer)         {Уст. ГК}
procedure MoveRel(dX,dY:Integer);     {Уст. ГК относительно}
procedure Line(X1,Y1,X2,Y2:Integer);  {Проводит линию}
procedure LineTo(X,Y:Integer);        {Проводит линию от ГК}
procedure LineRel(dX,dY:Integer);     {Проводит линию от ГК относительно}
procedure Circle(X,Y:Integer; Radius:Word);  {Рисует окружность}
procedure Rectangle(X1,Y1,X2,Y2:Integer);    {Рисует прямоугольник}
procedure PutPixel(X,Y:Integer; Color:Word);    {Выводит пиксел ук. цвета}
procedure OutTextXY(X,Y:Integer;TextString:string); {Выводит текст}
...
```

Рассмотрим взаимосвязь некоторой заданной СК с СК экрана. В заданной СК с помощью четырёх параметров *xmin*, *xmax*, *ymin* и *ymax* определим прямоугольную область. Рассмотрим задачу отображения этой прямоугольной области на экран монитора. Исходя из предположения, что угловые точки отображаемой области должны отобразиться в угловые пикселы графической матрицы экрана, получим следующие соотношения:

$$mx = GetMaxX/(xmax - xmin),\ my = GetMaxY/(ymax - ymin),$$
$$xs = Round((-xmin+x)*mx),\ ys = Round((ymax-y)*my),$$

где *x*, *y* - координаты некоторой точки в отображаемой СК, *xs*, *ys* - координаты отображения точки в СК экрана, *mx* и *my* - масштабы преобразования по соответствующим осям.

Эти соотношения использованы в следующем примере программы с применением модуля Graph. Программа Curve.pas строит график функции *y = Sin(x)* на отрезке *(-2*Pi, 2*Pi)*.

```
program Curve;
uses Graph;
const
   xmin = -6.5; xmax = 6.5;      { Размеры заданной           }
   ymin = -1.5; ymax = 1.5;      {                   области  }
   a = -2*Pi;    b = 2*Pi;       { Отрезок построения графика }
   n = 50;                       { Кол. интервалов графика    }
   h = (b - a)/n;                { Длина одного интервала     }
var
   mx,my: Real;         { Масштабы                          }
   x,y:   Real;         { Текущая точка кривой в зад. СК    }
   xs,ys: Integer;      { Текущая точка графика а СК экрана }
   xc,yc: Integer;      { Начало заданной СК в СК экрана    }
   i:     Integer;      { Номер точки графика               }

procedure GrInit;      {Инициирует графическую систему     }
var   GraphDriver, GraphMode, ErrorCode: integer;
```

```
begin
GraphDriver:=Detect;
InitGraph(GraphDriver,GraphMode,'');
ErrorCode:=GraphResult;
if ErrorCode<>grOk then
   begin
   WriteLn('Ошибка графики: ',GraphErrorMsg(ErrorCode));
   WriteLn('Программа остановлена.');
   ReadLn;
   Halt(1);
   end
end; { GrInit }

procedure ToScr(x,y:Real; var xs,ys:Integer);
   { Отображает точку из заданной СК в СК экрана }
begin
xs:=Round((-xmin+x)*mx); ys:=Round((ymax-y)*my);
end; { ToScr }

begin
GrInit;  { Инициируем графическую систему }

   { Определяем масштабы по осям 0X и 0Y}
mx := GetMaxX/(xmax - xmin); my := GetMaxY/(ymax - ymin);

   { Определяем координаты начала заданной СК в СК экрана }
ToScr(0,0,xc,yc);

SetColor(1);                    { Устанавливаем цвет линий }
SetBkColor(7);                  { Устанавливаем цвет фона  }
Line(0,yc,GetMaxX,yc);          { Проводим ось 0X          }
Line(xc,0,xc,GetMaxY);          { Проводим ось 0Y          }
OutTextXY(20,20,'Y = Sin(X)');  { Выводим заголовок графика}

                { Рисуем график }
SetColor(12);
x:=a; y:=Sin(x);
ToScr(x,y,xs,ys);
MoveTo(xs,ys);
for i:=1 to n do
   begin
   x:=x+h; y:=Sin(x);
   ToScr(x,y,xs,ys);
   LineTo(xs,ys);
   end;

ReadLn;       { Для выхода из гр. режима надо нажать Enter }
CloseGraph;   { Возвращаемся в алфавитно-цифровой режим    }
end.
```