12. БИБЛИОТЕЧНИ МОДУЛИ В ТУРБО ПАСКАЛ

12.1. Въведение

Както вече отбелязахме, функционално завършените алгоритмични части на програмите в Паскал е целесъобразно да се оформят като процедури или функции. Това е особено важно при често използвани в дадена област алгоритми. Такива са например числените математически методи (решаване на системи уравнения, намиране на минимум или максимум на функции, операции с матрици и др.), операциите с външната памет (търсене на определен файл на дисково устройство, смяна на подразбиращ се каталог и др.), средствата за организация на потребителски интерфейс и т.н. Програмирани като подпрограми, такива алгоритми могат внимателно да се тестват и след това с увереност да се използват многократно в различни приложни програми. Това е равносилно на разширяване на изразните възможности на езика.

Стандартът на езика Паскал допуска използването на готови подпрограми и това става единствено чрез включване на изходните им текстове на съответни места в изходните текстове на програмите. Това обаче не е удобно, защото е трудоемко, изисква много внимание, оставя възможност за внасяне на грешки в текста и изисква всеки път транслирането им. Ето защо в много от конкретните реализации на транслатори за езика Паскал се предвижда възможност за отделна транслация на процедурите и функциите, оформянето им в *библиотеки в транслиран вид* и последващото им свързване с програмите, които ги използват. В Турбо Паскал тази идея е осъществена чрез така наречените *библиотечни модули*.

Библиотечният модул (**Unit**) може да съдържа декларации на константи, типове данни и променливи, както и процедури и функции, достъпни за използване от различни програми. За да има дадена програма достъп до определен библиотечен модул, в декларативната й част трябва да присъства оператор **Uses** (използва), който има следния общ вид:

Uses *списък от имена на библиотечни модули;*

Имената в списъка се отделят със запетаи. Библиотечните модули се съхраняват във файлове с разширение **.TPU (Turbo Pascal Unit).**

12.2. Структура на библиотечния модул

Библиотечният модул се състои от *заглавна, интерфейсна, реализационна и инициализационна части*.

12.2.1. Заглавна част

Тя се състои от заглавен оператор.

Заглавният оператор има следния общ вид:

Unit *име на библиотечния модул;*

Името на библиотечния модул трябва да съвпада с името на файла, в който се съхранява. Поради това то трябва да отговаря както на изискванията към имената на обекти в Паскал, така и на изискванията към имената на файловете в MS DOS.

12.2.2. Интерфейсна част

Интерфейсната част започва с ключовата дума

Interface

и може да включва: оператор **Uses,** декларативна част и заглавия на подпрограми.

Тя осигурява достъпа на външните програми до обекти на библиотечния модул. Всички константи, типове и променливи, декларирани тук, са достъпни като нелокални както за външни програми, така и за подпрограмите от библиотечния модул. Всички процедури и функции от библиотечния модул, чиито заглавия присъстват тук, са достъпни за използване от външни програми.

12.2.3. Реализационна част

Реализационната част започва с ключовата дума

Implementation

съдържа пълните текстове на всички подпрограми на библиотечния модул и може да включва: оператор **Uses** и декларативна част. Всички константи, типове и променливи, декларирани тук, са достъпни като нелокални за всички подпрограми от библиотечния модул, но за външни програми те не са достъпни.

В реализационната част са записани цялостно (включително и заглавните им оператори) всички процедури и функции, декларирани в интерфейсната част. Тук могат да присъстват и процедури и функции, които не са декларирани в интерфейсната част. Те не са достъпни за външни програми.

За всяка подпрограма от библиотечния модул са достъпни всички негови подпрограми, независимо дали са разположени преди нея или след нея. Ако в дадена програма има вложена подпрограма, то тя е достъпна само за нея.

12.2.4. Инициализационна част

Инициализационната част представлява съставен оператор, който се изпълнява задължително при използване на библиотечния модул и то преди изпълнението на първия изпълним оператор от външната програма, ползваща този модул. Обикновено се използва за някои предварителни операции, необходими за нормалната работа на процедурите и функциите от библиотечния модул - най-често инциализацията (присвояването на стойност) на някои променливи. Ако такива действия не са необходими, инициализационната част може да бъде празна, т.е. състояща се само от **Begin** и **End**, или дори само от **End**, задължително с точка на края.

Програма 12.1. Програма, използваща библиотечен модул с подпрограми за работа с тригонометрични функции.

Известно е, че езикът Паскал предлага много ограничен брой вградени функции за работа с тригонометрични функции. Всеки програмист може да създаде собствен библиотечен модул, съдържащ допълнителни такива процедури и функции.

По-долу е даден такъв библиотечен модул. В реализационната му част се съдържат осем подпрограми. В интерфейсната му част обаче присъстват заглавията само на шест от тях и само те са достъпни за външни програми. Заглавията на процедурата RadGMS и функцията GMSRad не присъстват в интерфейсната част и затова те са достъпни само за процедурите и функциите от библиотечния модул. Константите PiNa2 и PiPo2 са декларирани в интерфейсната

част и затова те са достъпни както за процедурите и функциите от библиотечния модул, така и за външни програми.

а) Библиотечен модул

```
Unit UnTrifun;
  INTERFACE
    Const PiNa2=Pi/2; PiPo2=Pi*2;
    Function SinGMS(G,M,S:integer):real;
    Function CosGMS(G,M,S:integer):real;
    Function Tan(R:real):real;
    Function TanGMS(G,M,S:integer):real;
    Function Atn2Pi(Dx,Dy:real):real;
    Procedure Atn360(Dx,Dy:real;Var G,M,S:integer);
  IMPLEMENTATION
{ Процедура за превръщане на ъгъл от радиани в градуси, минути и секунди }
  Procedure RadGMS(R:real;Var G,M,S:integer);
    Var
      Gr,Mn,Rr:real;
    Begin
      Rr:=abs(R);
      Gr:=Rr/Pi*180;  G:=trunc(Gr);
      Mn:=(Gr-G)*60; M:=trunc(Mn);
      S:=round((Mn-M)*60);
      If S=60
        then begin
          M:=M+1; S:=0
        end;
      If M=60
        then begin
          G:=G+1; M:=0
        end;
      If R< 0 then G:=-G
    End;
{ Функция за превръщане на ъгъл от градуси, минути и секунди в радиани }
  Function GMSRad(G,M,S:integer):real;
    Begin
      GMSRad:=((S/60+M)/60+G)*Pi/180
    end;
{ Функция за намиране на синуса ъгъл, зададен в градуси, минути и секунди }
  Function SinGMS(G,M,S:integer):real;
    Var
      R:real;
    Begin
      R:=GMSRad(G,M,S); SinGMS:=sin(R)
    End;
{ Функция за намиране на косинуса ъгъл, зададен в градуси, минути и секунди }
  Function CosGMS(G,M,S:integer):real;
    Var
      R:real;
    Begin
      CosGMS:=cos(GMSRad(G,M,S))
    End;
{ Функция за намиране на тангенса ъгъл, зададен в радиани }
```

```
  Function Tan(R:real):real;
    Begin
      Tan:=sin(R)/cos(R)
    End;
{6.Функция за намиране на тангенса ъгъл, зададен в градуси, минути и секунди}
  Function TanGMS(G,M,S:integer):real;
    Var
      R:real;
    Begin
      TanGMS:=Tan(GMSRad(G,M,S))
    End;
{ Функция за намиране на ъгъла в радиани (0..2π), който сключва с оста Ox
вектор с проекции Dx и Dy }
  Function Atn2Pi(Dx,Dy:real):real;
    Var
      A:real;
    Begin
      If Dx=0
      then if Dy=0
           then begin
                  Writeln('Некор. аргум. в Atn2Pi');
                  Halt
                end
           else if Dy>0
                     then A:=PiNa2
                     else a:=3*PiNa2
      else begin
             A:=arctan(Dy/Dx);
             If Dx>0
             then begin
                    if Dy<0 then A:=A+PiPo2
                    end
             else A:=A+Pi
           end;
      Atn2Pi:=A
    End;
{ Функция за намиране на ъгъла в градуси (0..360), който сключва с оста Ox
вектор с проекции Dx и Dy }
  Procedure Atn360(Dx,Dy:real;Var G,M,S:integer);
    Var
      R:real;
    Begin
      RadGMS(Atn2Pi(Dx,Dy),G,M,S)
    End;
BEGIN
  Writeln;
  Writeln('                ВНИМАНИЕ!'); Writeln;
  Writeln('  Модулът UNTRIFUN е създаден с учебна цел.');
  Writeln('  Той може да се използва при разработване на');
  Writeln('  програми за решаване на геометрични задачи.');
  Writeln;
  Writeln('        У С П Е Ш Н А  Р А Б О Т А !');Writeln;
END.
```

*б) Примерна програма, използваща библиотечния модул Un*Trifun.

```
Uses UnTrifun;
Var
  I, G, M, S :integer;
  A, Dx, Dy :real;
Begin
  Writeln('Синусът от 25 градуса, 15 минути и 30 секунди е ',
                                                SinGMS(25,15,30):0:5);
  Writeln('Косинусът от 25 градуса, 15 минути и 30 секунди е ',
                                                CosGMS(25,15,30):0:5);
  Writeln('Тангенсът от 0.5 rad е ',Tan(0.5):0:5);
  Writeln('Тангенсът от 25 градуса, 15 минути и 30 секунди е ',
                                                TanGMS(25,15,30):0:5);
{ Цикъл за намиране ъгъла на вектор при 8 негови положения }
  For i:=0 to 8 do
  begin
    Dx:=100*cos(I*PiNa2/2);
    Dy:=100*sin(I*PiNa2/2);
    A:=Atn2Pi(Dx,Dy);
    Write('Dx=',Dx:8:3,'  Dy=',Dy:8:3,'  A=',A:8:5);
    Atn360(Dx,Dy,G,M,S);
    Writeln('  G=',G:3,'  M=',M:2,'  S=',S:2)
  end;
  readln
End.
```

12.3. Стандартни библиотечни модули в Турбо Паскал

Средата Турбо Паскал се разпространява с няколко стандартни библиотечни модула, предоставящи различни допълнителни възможности на програмиста. Някои от тях са:

Модул **System** - Включва някои математически функции, средства за работа с оперативната памет и с периферните устройства. Този модул се използва винаги и не е необходимо да се указва в оператор **Uses**.

Модул **CRT** - Включва допълнителни средства за работа с дисплея в текстов режим, с клавиатурата и с високоговорителя на компютъра (генератора на звук).

Модул **Graph** - Включва средства за компютърна графика, т.е. за работа с дисплея в графичен режим.

Модул **Printer** - Включва средства за работа с печатащо устройство.

Ще опишем накратко два от тези модули, като пълна информация за възможностите, които те предоставят, може да се получи от фирмените ръководства по Турбо Паскал или от помощната функция (Help) на програмната среда.

12.4. Модул CRT

12.4.1. Работа с екрана

Модулът включва константи, определящи цвета на изобразяване на символите и на фона, върху който те се изобразяват. Допустимите цветове на символите са 16 и са разделени на две групи - тъмна с кодове от 0 до 7 и светла с кодове от 8 до 15. За фон могат да се използват само цветове от тъмната група. Имената на константите и съответните им кодове са:

0 - Black (Черен)
1 - Blue (Син)
2 - Green (Зелен)
3 - Cyan (Лазурно син)
4 - Red (Червен)
5 - Magenta (Цикламов)
6 - Brown (Кафяв)
7 - LightGray (Светло сив)
8 - DarkGray (Тъмно сив)
9 - LightBlue (Светло син)
10 - LightGreen (Светло зелен)
11 - LightCyan (Светло лазурно син)
12 - LightRed (Светло червен)
13 - LightMagenta (Светло цикламов)
14 - Yellow (Жълт)
15 - White (Бял).

Към цвета на символите може да се прибави (аритметично) и константата Blink (мигащ) с код 128. Например записът Red+Blink е еквивалентен на 4+128 или направо на 132 и означава изобразяване на червени мигащи символи на екрана.

1. Задаването на цвета на символите и цвета на фона става чрез следните две процедури:

```
Procedure TextColor(CS:intetger);
Procedure TextBackground(CF:integer);
```

Тук CS и CF са цели константи или променливи, задаващи атрибутите на изписване на символите (респективно цвят на символите и цвят на фона) при последващите оператори за изход на екрана (Write или WriteLn). Промяната на атрибутите с тези две процедури не влияе на намиращия се вече на екрана текст.

2. Както е известно извеждането на символите на екрана става от позиция, указана от указател, наречен курсор. Позицията на курсора на екрана може да се определи с две координати - номер на колоната и номер на реда, на който той се намира. Модул CRT дава възможност извеждането на текст да става не на целия екран, а на правоъгълна част от него, наречена прозорец, като останалия текст извън прозореца остава непроменен. Задаването на прозорец става с процедурата:

```
Procedure Window(X1,Y1,X2,Y2:integer);
```

Параметрите на процедурата са координатите (колона, ред) на горния ляв и долния десен ъгъл на прозореца.

След задаване на прозорец последващите оператори Write и WriteLn извеждат само в него, като координатите на курсора се отчитат относно горния ляв ъгъл на прозореца. Възстановяване на извеждането върху целия екран може да стане с оператор:

```
Window(1,1,80,25);
```

3. Ако е нужно извеждането на текст да започне на точно определено място от прозореца, курсорът се позиционира на това място предварително. За целта служи процедурата:

```
Procedure GoToXY(X,Y:integer);
```

Параметрите са координатите, на които се позиционира курсорът.

4. Изчистването (т.е. запълването с фонов цвят) на активния прозорец се постига чрез процедурата:

Procedure ClrScr;

12.4.2. Работа с клавиатурата

Модул CRT предлага две полезни функции за работа с клавиатурата.

1. Функцията

Function KeyPressed:boolean;

връща стойност true, ако има натиснат, но непрочетен клавиш и стойност false в обратния случай. Често се използва в цикъл

Repeat until KeyPressed;

за да реализира задържане преминаването на следващ оператор до натискане на произволен клавиш.

2. Функцията

Function ReadKey:char;

чете символ от клавиатурата без да го извежда на екрана (без ехо) за разлика от оператора Read.

Тези две функции са особено полезни например при реализация на менютехника за потребителски интерфейс. Ако натиснатият клавиш е непечатаем, специален (например клавишите със стрелки, клавишите F1, F2 и т.н.), натискането му е равностойно на подаването на два символа - първият с код 0, а вторият с код, определящ натиснатия клавиш. Следователно анализът на специалните клавиши може да се осъществи с четене без ехо (чрез ReadKey), проверка дали има още непрочетени символи (чрез KeyPressed) и дали прочетеният символ е с код нула, второ четене без ехо, ако това е така и избор на действие в зависимост от прочетения символ (най-често чрез оператора **Case**).

12.4.3. Работа с високоговорителя

Управлението на високоговорителя става чрез две процедури от модула CRT.

1. Процедурата

Procedure Sound(F:integer);

предизвиква включването на звук с честота F херца.

2. Издаването на звук продължава независимо от останалата работа, извършвана от компютъра, до изпълнението на процедурата

Procedure NoSound;

която го прекратява.

3. За получаване на звук с точно определена продължителност е уместно да се използва процедурата

Procedure Delay(T:integer);

която спира работата на микропроцесора на компютъра за T милисекунди.

Така изпълнението на последователността от оператори

```
Sound(440);
Delay(1000);
NoSound;
```

ще предизвика изпълнение на тона ла от първа октава, който е с честота 440 Hz, в течение на една секунда.

12.5. Модул Graph

Турбо Паскал предоставя допълнителна възможност за работа в графичен режим посредством модула **Graph**. Той предоставя около 80 процедури и функции за работа на персоналния компютър в графичен режим. В този раздел ще разгледаме малка част от тях, като ще използваме възможността да изясним основните принципи за получаване на графично изображение.

При масово разпространените персонални компютри, когато е необходимо да се изобрази графично изображение, то се формира на база на точки, разположени на повърхността на графичното устройство – най-често дисплей. Тези най-малки елементи от графичното изобрежение се наричат *пиксели* и са разположени на точно определени позиции в така наречана *растерна мрежа*. Качеството на изображението зависи от гъстотата на точките при образуването на растерната мрежа. Броят на пискелите на единица разстояние се нарича *разделителна способност* и е важна характеристика за всички растерни графични устройства. Разделителната способност най-често се измерва с брой точки на инч (*dots per inch* или *dpi*). Средствата за компютърната графика в Турбо Паскал се определят от архитектурата на графичните адаптери, които се използват в Intel базираните персонални компютри. Всеки модел може да поддържа няколко различни графични режима, отличаващи се с различен брой пиксели в растерната мрежа и брой на цветовете, които могат да бъдат изобразени. Ще направим кратък преглед на графичните възможности, които може да има компютър от този клас, като ще разглеждаме само броя на пикселите по двете оси и броя на цветовете, които могат да се изобразят. При това трябва да се знае, че конструкторите са се погрижили по-новите графични адаптери с по-големи възможности да реализират всички функции на по-старите им предшественици.

Тъй като при създаването на програми е желателно да се предвиди възможността те да работят на произволен компютър в библиотечния модул Graph е предвидена възможността за поддържане на всички разпространени графични адаптери. В приложената таблица са описани най-често използваните от тях и най важните им характеристики:

Графичен драйвер	Режим*	Брой пиксели	Цветове
CGA	CGAC0	320 x 200	Палитра 0 - светлозелено, светлочервено, жълто
	CGAC1		Палитра 1 - светло лазурно, светлопурпурно, бяло
	CGAC2		Палитра 2 - зелено, червено, кафяво
	CGAC3		Палитра 3 - лазурно, пурпурно, сиво
	CGAHi	640 x 200	Едноцветно
EGA	EGALo	640 x 200	16 от 64
	EGAHi	640 x 350	16 от 64
VGA	VGALo	640 x 200	16 от 320хил.
	VGAMed	640 x 350	16 от 320хил.
	EGAHi	640 x 480	16 от 320хил.
SVGA**		640 x 480	Зависи от режима, обема на видеопаметта на видеоконтролера и от драйвера. Може да достигне до 16,7 милиона цвята (обикновено палитра 256 от 16,7 милиона цвята)
		800 x 600	
		1024 x 768	
		1280 x 1024	
		1600 x 1280	

* Наименованието на режимите е според декларираните константи в GRAPH.TPU.

** SVGA драйверите не се поддържат стандартно от Турбо Паскал. Съществуващите драйвери не са стандартизирани и наименованията на режимите зависят от конкретната реализация.

Пикселите се задават с целочислени координати. Горният ляв ъгъл на областта за изобразяване е с координати (0,0), а Y координатната ос е насочена надолу.

Превключването на дисплея от текстов в графичен режим става чрез извикване на процедурата:

Procedure InitGraph(**Var** Drive,Mode:integer; Path:**string**);

С променливите Drive и Mode се указва типа на дисплея (по-точно на управляващия го адаптер) и на режима на работа (определящ най-вече разрешаващата способност и максималния брой цветове). За техни стойности обикновено се използват константи, определени в модула Graph с имена аналогични на тези посочени в таблицата. Path е текстова константа или променлива и представлява път към каталога, в който се намират файловете за управление на графичния дисплей. Те са различни в зависимост от типа устройство и е задължително да са налични, за да можем да работим в избрания графичен режим. Разширението на тези файлове е **BGI** – например за работа с CGA е необходим файл CGA.BGI, за EGA и VGA – EGAVGA.BGI и т.н.

Пример:

```
GrD := EGA; GrMode := EGALo;
InitGraph(GrD, GrMode, '');
```

В посочения пример се предполага, че компютърът разполага с графичен адаптер, поддържащ EGA, и се активира режим EGALo (640x200). Ако адаптерът не разполага с такива графични възможности се извежда съобщение за грешка и се прекратява работата на програмата. По-добър подход на работа е този, при който програмата сама установява типа на графичния адаптер и го активира в най-добрия възможен режим. За да се реализира това в графичния модул е предвидено при стойност на първия параметър 0 (или при използване на дефинираната константа със стойност 0 – **Detect)** процедурата InitGraph да извърши такова разпознаване. Стойността, съответстваща на избрания режим, се връща в променливата Mode. След активирането на графичен режим по този начин е необходимо да знаем какви са възможностите на автоматично избрания режим. За тази цел са предвидени следните полезните функции, които дават максималната възможна координата по X и по Y за дадения режим, максималния номер на цвят и максималния възможен номер на режим за графичния адаптер:

```
Function GetMaxX:integer;
Function GetMaxY:integer;
Function GetMaxColor:integer;
Function GetMaxMode:integer;
```

Използването на този подход ще илюстрираме след разглеждане на процедурите за изчертаване на основните графични примитиви в приложения по-долу пример.

Завършването на работа в графичен режим (и преминаване в текстов) става с процедурата:

Procedure CloseGraph;

При извеждане на графично изображение често се налага то да бъде разположено само на определена част от екрана и ако в програмата се извежда графичен елемент, част от който е извън тази област, да не се допуска той да променя изображението извън нея. За да не е необходимо програмистът постоянно да следи това да не се случва, в графичния модул е предвидена възможност за дефиниране на така наречената *област за изобразяване* (ViewPort). След като тя е определена премахването на графичните елементи (или техни части), излизащи извън този прозорец, се извършва автоматично от самата графична библиотека. Дефинирането на област за изобразяване става чрез процедурата:

Procedure SetViewPort (X1,Y1,X2,Y2:integer;Clip:boolean);

Първите четири параметъра са като при процедурата Window от модула CRT, а параметъра Clip може да приема стойност **ClipOn** или **ClipOff,**, определящи какво да се случва с графичните елементи, излизащи извън прозореца. Ако стойността на последния параметър е true или **ClipOn** всичко, което излиза извън областта, просто не се визуализира. В противен случай частите от геометричните примитиви, излизащи извън областта за изобразяване се изчертават, ако са в рамките на екрана. След първоначално активиране на графичен режим областта за изобразяване е разположена върху целия екран. Важно е да се помни, че началото на координатната система е в горния ляв ъгъл на областта за изобразяване. Изключение правят само координатите, указвани в SetViewPort, които се задават винаги спрямо екранната координатна система.

Изчистването на областта за изобразяване става с процедурата:

Procedure ClearViewPort.

Визуализирането на точка от екрана T(X,Y) се извършва с процедурата:

Procedure PutPixel(X,Y,Color:integer);

X и Y са координатите на точката, Color – номер на цвета на точката. Цвят 0 означава цвят на фона на изображението

В графичен режим неявно се поддържа текуща позиция на изчертаването (подобно на позицията на курсора в текстов режим, но позицията не се изобразява на екрана). Първоначално (или след ClearViewPort) текущата позиция е с координати (0,0) - горния ляв ъгъл на прозореца. Те могат да бъдат променени с процедурите:

Procedure MoveTo(X,Y:integer);
Procedure MoveRel(DX,DY:integer);

В първата от тях X и Y са новите координати на текущата позиция, а във втората DX и DY са отместванията по хоризонтала и вертикала от старата до новата текуща позиция. Координатите на текущата позиция могат да бъдат получени чрез функциите:

Function GetX:integer;
Function GetY:integer

Изчертаването на отсечка може да се извърши с една от следните три процедури:

Procedure LineTo(X,Y:integer);
Procedure LineRel(DX,DY:integer);
Procedure Line(X1,Y1,X2,Y2:integer);

При първите две се изчертава отсечка от текущата позиция до позиция с координати X, Y или с отместване DX, DY. Текуща става позицията на крайната

точка. При третата се указват координатите на началната точка T1(X1,Y1) и крайната точка T2(X2,Y2) на отсечката. Процедурата Line не променя положението на текущата позиция за изчертаване

Както се вижда по-горе в процедурите за изобразяване на отсечка няма параметър определящ цвета, с който тя се изобразява. Задаването на цвета, с който ще се изчертават обектите посредством всички процедури, с изключение на PutPixel, става с процедурата:

Procedure SetColor(C:integer);

а на фона с процедурата:

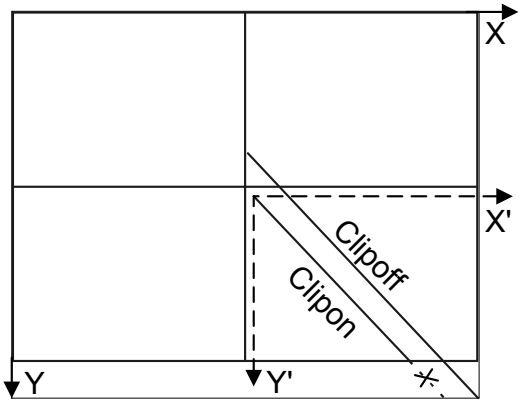
Procedure SetBkColor(C:integer);

За разлика от текстовия режим, втората от тези процедури незабавно променя фоновия цвят на целия екран.

Пример: Програма, която изчертава рамка около екрана и демонстрира работата на процедурата SetViewPort.

```
Program Grafika;
  Uses Graph;
  Var
    Grd,GrM,MaxX,MaxY,MaxColorNum,Cx,Cy:integer;
  Begin
    GrD := Detect;
    InitGraph (GrD,GrM,'C:\Turbo\BGI');
    MaxX := GetMaxX;
    MaxY := GetMaxY;
    MaxColorNum := GetMaxColor;
    MoveTo(0,0);
{ Изчертаване на рамка на екрана }
    LineTo(MaxX,0); LineTo(MaxX, MaxY);
    LineTo(0, MaxY); LineTo(0, 0);
    Cx := MaxX div 2; Cy := MaxY div 2;
    SetColor (Red); Line(0, cy, MaxX,cy); Line(cx, 0, cx, MaxY);
    SetViewPort(cx+10,cy+10,MaxX,MaxY,ClipOn);
    Line (0,0,1000,1000);
    SetViewPort(cx+10,cy+10,MaxX,MaxY,ClipOff);
{ Забележете, че размера на областта за изчертаване е (Cx-10)*(Cy-10) }
    Line (0,-50,Cx,Cy+100);
    readln;  { Задържа изпълнението на програмата в графичната страница }
    CloseGraph;
End.
```

В следствие на изпълнение на програмата ще се изчертае следната графика:



Ето още няколко изчертаващи процедури:

Procedure Circle(XC,YC,R:integer);

изчертава окръжност с център XC,YC и радиус R.

Procedure Arc(XC,YC,StartA,EndA,R:integer);

изчертава дъга от окръжност с център XC,YC, стартов ъгъл StarA, краен ъгъл EndA и радиус R. Ъглите се задават в градуси

Procedure Ellipse (XC,YC,StartA,EndA,A,B:integer);

изчертава дъга от елипса. Първите четири параметъра са като при процедурата Arc, а A и B са съответно дължината на хоризонталната и вертикалната полуос. Цяла елипса може да се изчертае чрез задаване стойности на StartA и EndA съответно 0 и 360 градуса.

Възможно е да се зададе стила на линията, която се изчертава като се използва процедурата:

Procedure SetLineStyle(LineStyle,Pattern,Thickness:word);

Параметърът LineStyle задава стила на линията:

0 - непрекъсната линия;
1 - линия от точки;
2 - осева линия;
3 - пунктирана линия;
4 - потребителски стил линия.

Thickness определя дебелината на линията и може да приема стойности

1 или 3 (може да се използват константите NormWidth или ThickWidth).

Вторият параметър Pattern има значение само ако се дефинира потребителски тип линия (LineStyle = 4). В този случай видът на линията се формира чрез повтаряне на еднотипни части с дължина 16 пиксела. Всеки бит от двоичното представяне на стойността на Pattern съответства на това дали съответната точка да се визуализира или не. Например ако искаме да имаме прекъсната линия със съотношение 3:1 прекъсната към непрекъсната част двоичното представяне на Pattern ще изглежда така: 0000000000001111(2). Това съответства на десетична стойност 15.

Извеждане на текст в графичен режим се извършва с една от процедурите:

Procedure OutText(Text:**string**);
Procedure OutTextXY(X,Y:integer; Text:**string**);

Първата процедура извежда текста, който е в променливата Text от текущата позиция на курсора, а втората - от точка T(X,Y).

Процедурата:

Procedure Rectangle(X1,Y1,X2,Y2:integer);

изчертава правоъгълник с координата T1(X1,Y1) на горния ляв ъгъл и T2(X2,Y2) на долния десен ъгъл.

Процедурата:

Procedure Bar(X1,Y1,X2,Y2:integer);

изчертава запълнен правоъгълник със зададения предварително стил.

Procedure SetFillStyle(Pattern,Color:integer);

задава стила на запълване. Pattern е число от 0 до 12, което показва номера на модела на запълване, дефиниран в модула, Color задава цвета на запълване. Съществува възможност и за създаване на модел от програмиста.

Процедурата:

Procedure GetImage(X1,Y1,X2,Y2:integer; **Var** Buffer);

запазва изображението, ограничено от правоъгълник с координати T1(X1,Y1) на горния ляв ъгъл и T2(X2,Y2) на долния десен ъгъл, в променливата Buffer побитово.

Процедурата:

Procedure PutImage(X1,Y1:integer; **Var** Buffer,Mode:integer);

въвежда запазеното изображение в променливата Buffer в точка с координати T(X1,Y1).

Програма 12.2. Програма за последователно чертане на квадрат в квадрат, като върховете на всеки следващ квадрат са среди на страните на неговия предшественик.

```
Uses Graph;
Type
  Masiv=array[1..5] of integer;
Var
  i,A,Amin:integer;
  Driver,Mode:integer;
  X,Y:Masiv;
Procedure Kvadrat(Var X,Y:masiv);
  Begin
    MoveTo(X[1],Y[1]);
    For i:=2 to 5 do LineTo(X[i],Y[i]);
    For i:=1 to 4 do
      begin
        X[i]:=round((X[i]+X[i+1])/2);
        Y[i]:=round((Y[i]+Y[i+1])/2)
      end;
    X[5]:=X[1];Y[5]:=Y[1]
  End;
Begin
  Write('Въведете страната на най-големия квадрат - до 450:');Readln(A);
  Write('Въведете страната на най-малкия квадрат - над 20:');Readln(Amin);
  X[1]:=10;   Y[1]:=10;
  X[2]:=10+A; Y[2]:=10;
  X[3]:=10+A; Y[3]:=10+A;
  X[4]:=10;   Y[4]:=10+A;
  X[5]:=X[1];Y[5]:=Y[1];
  Driver:=0;
  InitGraph (Driver, Mode, 'c:\oldd\dos-app\tp\bgi');
  Repeat
    Kvadrat(X,Y);Readln;
    A:=round(A*sqrt(2)/2)
  until A<Amin;
  Readln;
  CloseGraph
End.
```

Програма 12.3. Програма с елементи на анимация.

```
Uses graph, crt;
Var
  Grd,Grm,m,n,p,q,r:integer;
Procedure Oko(X,Y:integer;CvOko,CvZen:word);
  Begin
    Setfillstyle(1,CvOko);
    Fillellipse(X,Y,30,10);
    Setfillstyle(1,CvZen);
    Fillellipse(X,Y,12,10);
    Circle(X,Y,30);
  End;
Begin
  Writeln('За стартиране на графиката натиснете клавиша "Enter" ');
  Writeln('За спиране на графиката натиснете кой да е клавиш');
  Readln;
  Grd := Detect;
  InitGraph (grd, grm, 'c:\oldd\dos-app\tp\bgi');
  m := GetMaxX; n := GetMaxY;
  p:=trunc(m/2);q:=trunc(n/2);r:=trunc(n/2.5);
  Setbkcolor(lightred);
  Circle (p,q,200);                    {Овал на главата}
  Oko(trunc(m/2.7),r,white,blue);  {Дясно око}
  Line(p,q-20,p,q+50);                 {Нос}
  Arc(p,q,230,310,130);                {Уста}
  Repeat
    Oko(trunc(2*m/3.2),r,white,blue);        {Ляво око отворено}
    Delay(2000);
    Setcolor(0);
    Oko(trunc(2*m/3.2),r,red,red);  {Ляво око отворено - заличено}
    Setcolor(15);
    Line(trunc(2*m/3.2)-30,r,trunc(2*m/3.2)+30,r);{Ляво око затворено}
    Delay(250);
    Setcolor(0);
    Line(trunc(2*m/3.2)-30,r,trunc(2*m/3.2)+30,r);{Ляво око затворено - заличено}
    Setcolor(15);
  until keypressed;
  CloseGraph
End.
```