Basrah University
College of Education for pure Sciences
Computer Science Department

Computer Graphics

Nada Ali Noori

(Third Stage) - (2022-2023)

Chapter One

Introduction to Computer Drawing and its Uses

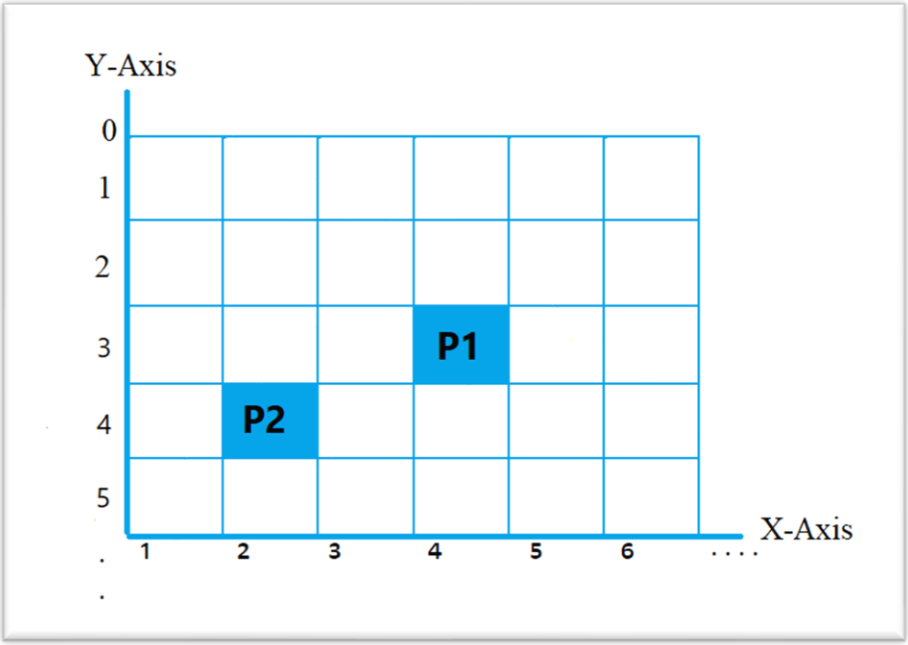
1.1 Introduction to Computer Graphics

The computer is an information processing machine. It is a tool for storing, manipulating, and correlating data. There are many ways to communicate the processed information to the user. Computer graphics is one of the most effective and commonly used ways to communicate the processed information to the user. It displays the information in the form of graphics objects such as pictures, charts, graphs, and diagrams instead of simple text**. Computer Graphics refers to designing or generating images using computers.**

Thus, we can say that computer graphics makes it possible to express data in pictorial form (شكل تصويري). The picture or graphics object may be an engineering drawing (الرسم الهندسي), business graphs (الرسوم البيانية للأعمال), architectural structures (الهياكل المعمارية), a single frame from an animated movie or a machine part illustrated for a service manual (توضيح جزء الماكينة في دليل الخدمة).

In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called **pixels**. **The pixel is the smallest addressable screen element. It is the smallest piece of the display screen that we can control.** The control is achieved by setting the intensity (شدة) and color of the pixel which composed the screen.

Each pixel on the graphics display does not represent a mathematical point. Rather, it represents a region that theoretically can contain an infinite number of points. For example, if we want to display point P_1 whose coordinates are (4,3) and point P_2 whose coordinates are (2,4) then P_1 and P_2 are represented as shown in the figure below:



The special procedures determine which pixel will provide the best approximation to the desired picture or graphics object. The process of determining the appropriate pixels for representing a picture or graphics object is known as **rasterization (تنقيط)**, and the process of representing a continuous picture or graphics object as a collection of discrete pixels is called **scan conversion**.

The computer graphics allows rotation(تدوير), translation (تحويل), scaling (تقييس), and performing various projections on the picture before displaying it. It also allows adding effects such as hidden surface removal, shading, or transparency to the picture before the final representation. It provides the user the control to modify the contents, structure, and appearance of pictures or graphics objects using input devices such as a keyboard, mouse, or touch-sensitive panel on the screen. There is a close relationship between the input devices and display devices. Therefore, graphics devices include both input devices and display devices.

1.2 Advantages of Computer Graphics

1. High-quality graphics displays of personal computers provide one of the most natural means of communicating with a computer.
2. It has the ability to show moving pictures, and thus it is possible to produce animations with computer graphics.
3. Computer graphics can also control the animation by adjusting the speed, the portion of the total scene in view, the geometric relationship of the objects in the scene to one another, the amount of detail shown and so on.

4. Computer graphics also provides a facility called update dynamics. With updated dynamics, it is possible to change the shape, color, or other properties of the objects being viewed.
5. With the recent development of digital signal processing (DSP) and audio synthesis chip, interactive graphics can now provide audio feedback along with graphical feedback to make the simulated environment even more realistic.

1.3 Applications of Computer Graphics

1. **User interfaces (واجهات المستخدم):** It is now a well-established fact that graphical interfaces provide an attractive and easy interaction between users and computers. The built-in graphics provided with user interfaces use visual control items such as buttons, menus, icons, scroll bars, etc., which allows the user to interact with the computer only by mouse-click. Typing is necessary only to input text to be stored and manipulated.
2. **Plotting of graphics and chart (رسم المخططات والرسوم البيانية)**: In industry, business, government, and educational organizations, computer graphics is most commonly used to create 2D and 3D graphs of mathematical, physical and economic functions in form

of histograms, bars, and pie-charts. These graphs and charts are very useful for decision-making.

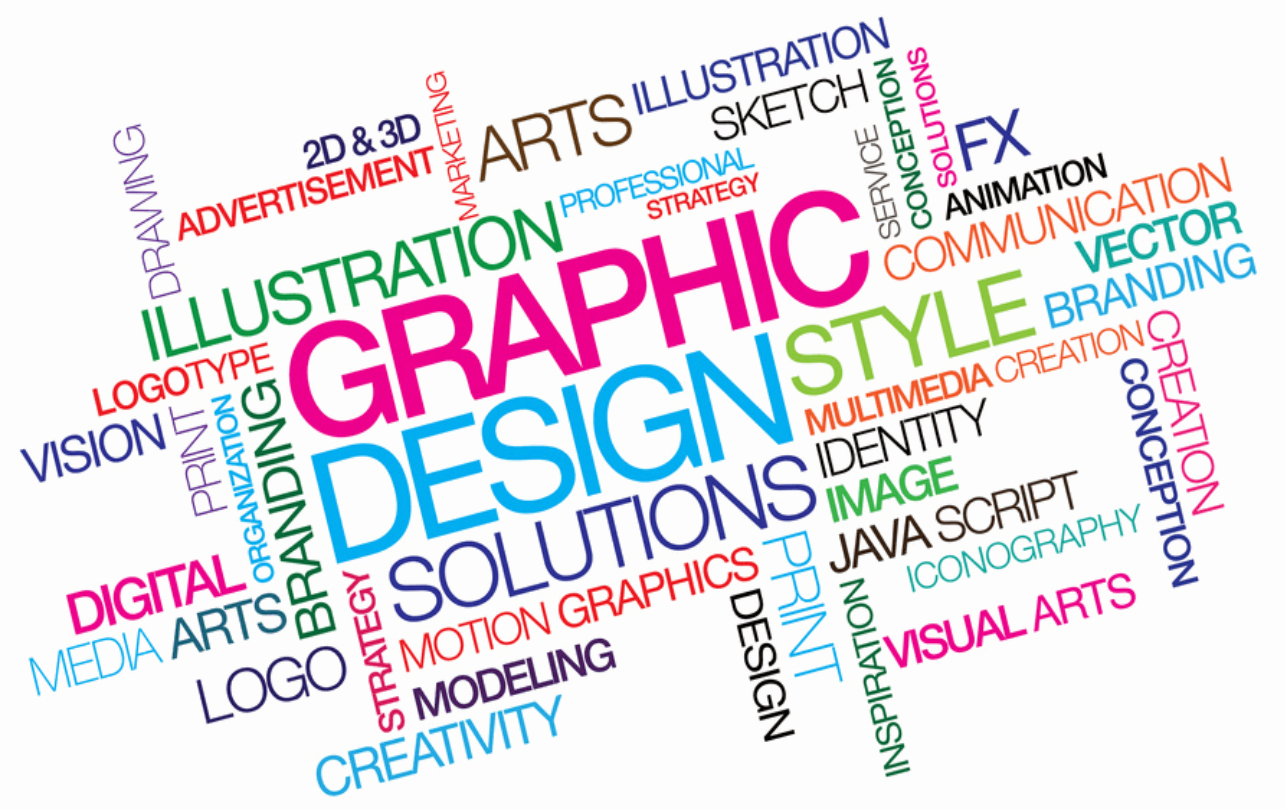
3. **Computer-aided drafting and design (الصياغة بمساعدة الحاسوب والتصميم)**: computer-aided drafting uses graphics to design components and systems electrical, mechanical, electromechanical, and electronic devices such as automobile bodies, structures of buildings, airplanes, ships, very large-scale integrated chips, optical systems, and computer networks.
4. **Simulation and Animation (المحاكاة والرسوم المتحركة)**: use of graphics in simulation makes mathematic models and mechanical systems more realistic and easier to study. The interactive graphics supported by animation software proved their use in the production of animated movies and cartoon films.
5. **Art and Commerce (الفن والاعلانات)**: There is a lot of development in the tools provided by computer graphics. This allows users to create artistic pictures which express the message and attract attention. Such pictures are very useful in advertising.
6. **Process Control (عمليات التحكم):** With the use of computers now it is possible to control various processes in the industry from a remote-control room. In such cases, process systems and processing parameters are shown on the computer with graphic

symbols and identifications. This makes it easy for an operator to monitor and control various processing parameters at a time.

7. **Cartography (رسم الخرائط):** Computer graphics is also used to represent geographic maps, weather maps, oceanographic charts, population density maps, and so on.
8. Education and training (التعليم والتدريب): computer graphics can be used to generate models of physical aids. Models of physical systems, physiological systems, population trends, or equipment, such as color-coded diagrams can help trainees to understand the operation of the system.
9. Image processing: in computer graphics, a computer is used to create pictures. Image processing, on the other hand, applies techniques to modify or interpret existing pictures such as photographs and scanned images. Image processing and computer graphics are typically combined in many applications such as to model and studying physical functions, designing artificial limbs, and planning and practicing surgery. Image processing techniques are most commonly used for picture enhancements to analyze satellite photos, X-ray photography, and so on.

Chapter Tow

Graphics Libraries



2.1 Graphics Libraries

C++ graphics using **graphics.h** functions can be used to draw different shapes, display text in different fonts, change colors, and many more. Using functions of graphics.h in C++ compiler can make graphics programs, animations, projects, and games. You can draw circles, lines, rectangles, bars, and many other geometrical figures. You can change their colors using the available functions and fill them. Following is a list of functions of graphics.h header file. Every function is discussed with the arguments it needs, its description, possible errors while using that function and a sample C++ graphics program with its output.

Syntax: **#include<graphics.h>**

2.2 Initializing C++ Graphics Mode:

1.InitGraph ()

The computer display system must be initialized into graphics mode before calling the graphics function.

The "**initgraph**" function is used to initialize the display into graphics mode. This function is a part of the "**graphics.h**" header file. So this file is included in the program before executing "initgraph" function. The syntax of initgraph" function is:

Void initgraph(int &Graphriver, int &Graphmode,Char "pathtoDriver");

Graph Driver OR (gd):

Represents the graphics driver installed on the computer. It may be an integer variable or an integer constant identifier, e.g. CGA, EGA, SVGA, etc.

The graphics driver can also be automatically detected by using the keyword “DETECT”. Letting the compiler detect the graphic driver is known as auto-detect.

If the driver is to be automatically detected, the variable driver is declared as of integer type and **DETECT** value is assigned to it as shown below.

```
int driver, mode;
driver = DETECT;
```

This statement must be placed before “initgraph” function. When the above statement is executed. the computer automatically detects the graphic driver and the graphics mode.

Graph Mode OR (gm):

Represents output resolution on the computer screen. The normal mode for VGA is VGAHI. It gives the highest resolution If the driver is auto-detected, then its use is optional. The computer automatically detects the driver as well as the mode.

& :

represents the addresses of constant numerical identifiers of driver and mode. If constants (e.g., VGA, VGAHI) are used, then "&" operator is not used as shown below:

initgraph (VGA, VGAHI, "path");

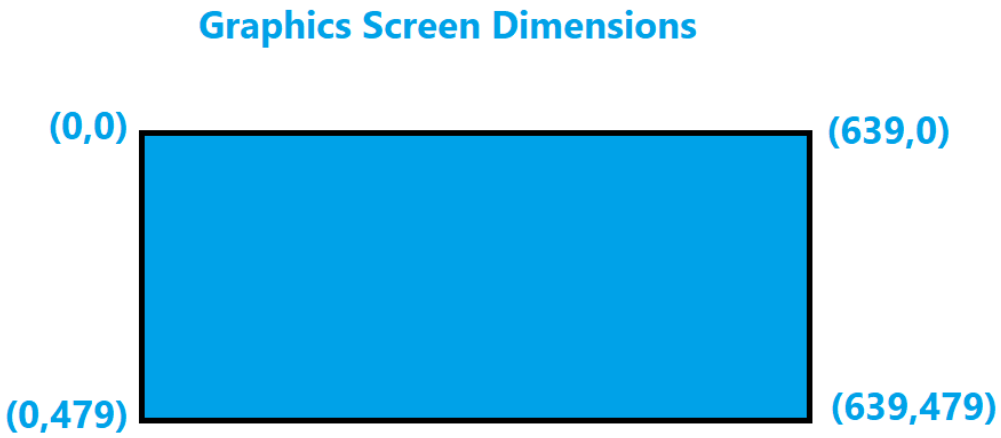
Path to driver:

Represents the path of graphic drivers. It is the directory where the BGI files are located. Suppose the BGI files are stored in "C:\TC\BGI", then the complete path is written as:

initgraph (&gd, &gm, "C:\TC\\BGI");

The use of double backslash "\" is to be noted. One backslash is used as an escape character and the other for the directory path. If the BGI files are in the current directory, then the path is written as:

Initgraph(&gd,&gm,"");



2.Grapherrormsg ():

This function used to return the error message.

the syntax: **Char *grapherrormsg(int errorcode);**

for example, if we want to print the error message write the following:

```
int gd, gm, errorcode;
initgraph(&gd, &gm, "C:\\TC\\BGI");
errorcode = graphresult();
if(errorcode != grOk)
 {
   printf("Graphics error: %s\n", grapherrormsg(errorcode));
   exit (1);

 }
```

3.Graphresult ():

This function is used to return the error code at any A drawing process when an error occurs. When no error occurs in the drawing operations, the function returns the value (**grok).** For example

```
int errorcode = graphresult();
if (errorcode != grOk)
{
 printf(grapherrormsg(errorcode);
 getch();
 return 0;
 }
```

4. Closegraph():

The **close graph** function is used to restore the screen to text mode. When graphics mode is initialized, memory is allocated to the graphics system. When "closegraph" function is executed, it de-allocates all memory allocated to the graphics system. This function has usually used at the end of the program.

Its syntax is **Void closegraph();**

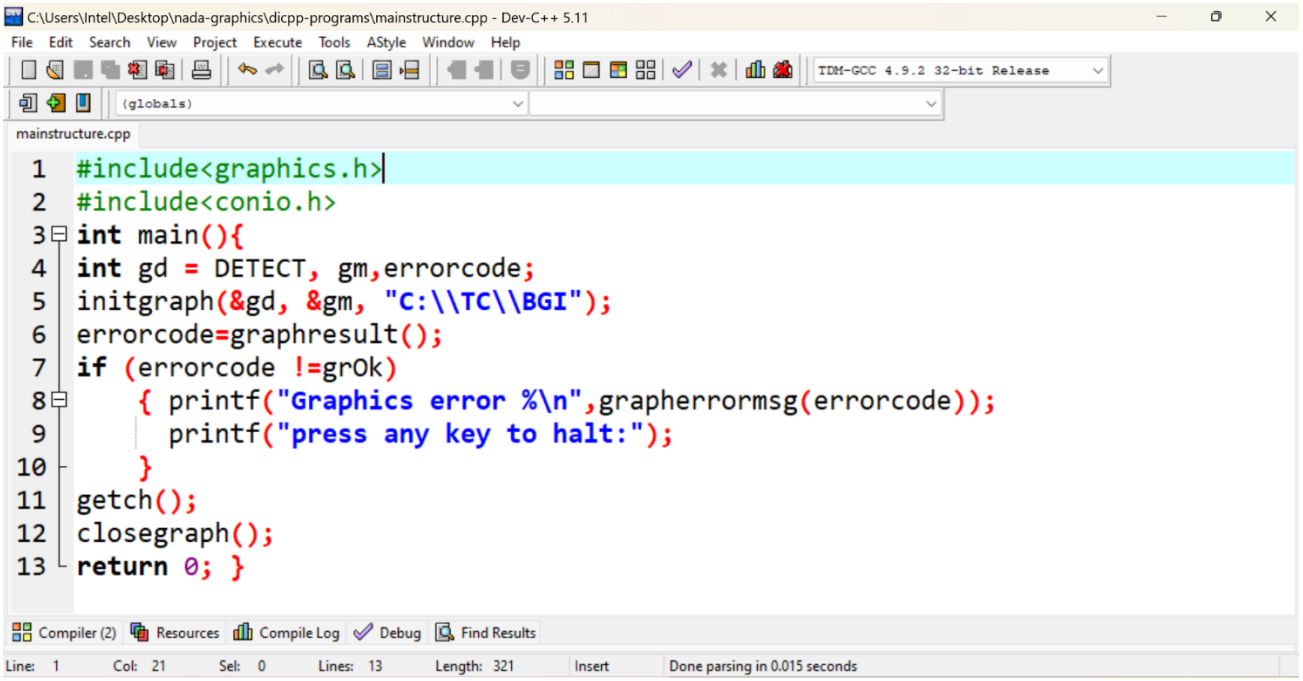
For example: if we want to convert the computer system from (**text mode**) to (**graphics mode**) and write the sentence **(Press any Key to Close the Graphics Mode...**), then close the graphics mode and return the system to the text mode.

```
initgraph(&gd, &gm, "");
outtext("Press any key to close the Graphics Mode...");
getch();
closegraph();
```

5.cleardevice ():

The "cleardevice" function is used to clear the screen in graphics mode. It is similar to "clrscr" function that is used to clear the screen text mode. Its syntax is: **cleardevice();**

The general structure of all graphic programs in C++:



HOMEWORK

What is the output of this code?

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main ()
{
int gd=DETECT,gm;
initgraph(&gd,&gm, "");
outtext("Enter any key to come outside from the graphics mode");
getch();
closegraph();
printf("Text Mode ");
}
```

Chapter three

Graphics Functions

3.1 Graphics Functions in C++:

This chapter explains in detail the main graphics function to draw shapes like points, circles, rectangles, lines ...etc. Also, we are going to learn how to set colors, select fonts, writing text with different sizes, fonts, and colors. We will also discuss important functions such as scaling, motion, translation.... etc.

1.Putpixel ()

The header file graphics.h contains putpixel() function which plots a pixel(point)at location(coordinates) (x,y) of the specified color.

Syntax :

```
void putpixel(int x, int y, int color);
```

where, (x, y) is the location at which the pixel(point) is to be put, and color specifies the color of the pixel.

Example: A RED color pixel at (50, 40) can be drawn by using **putpixel (50, 40, RED);**

putpixel() function can be used to draw circles, lines, and ellipses using various algorithms.

2. Getpixel ()

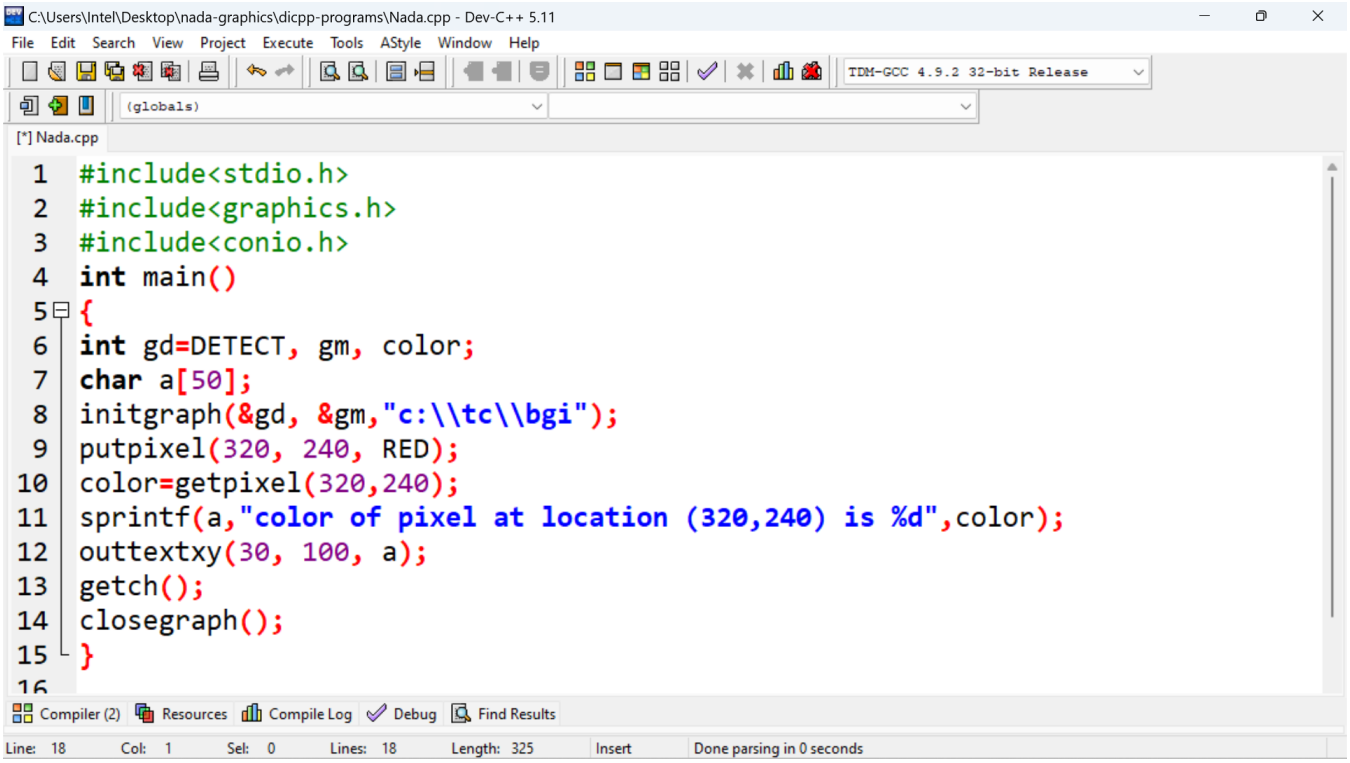
returns the color of the pixel present at the location (x, y).

Syntax: `int getpixel(int x, int y);`

Example: what is the color of the point (0,0)?

```
char array[50];
int color = getpixel(0, 0);
sprintf(array,"color of pixel at (0,0) = %d",color);
outtext(array);
```

Exercise: Write a C++ program to draw a RED pixel at location (320, 240) then print the color number of the same point.

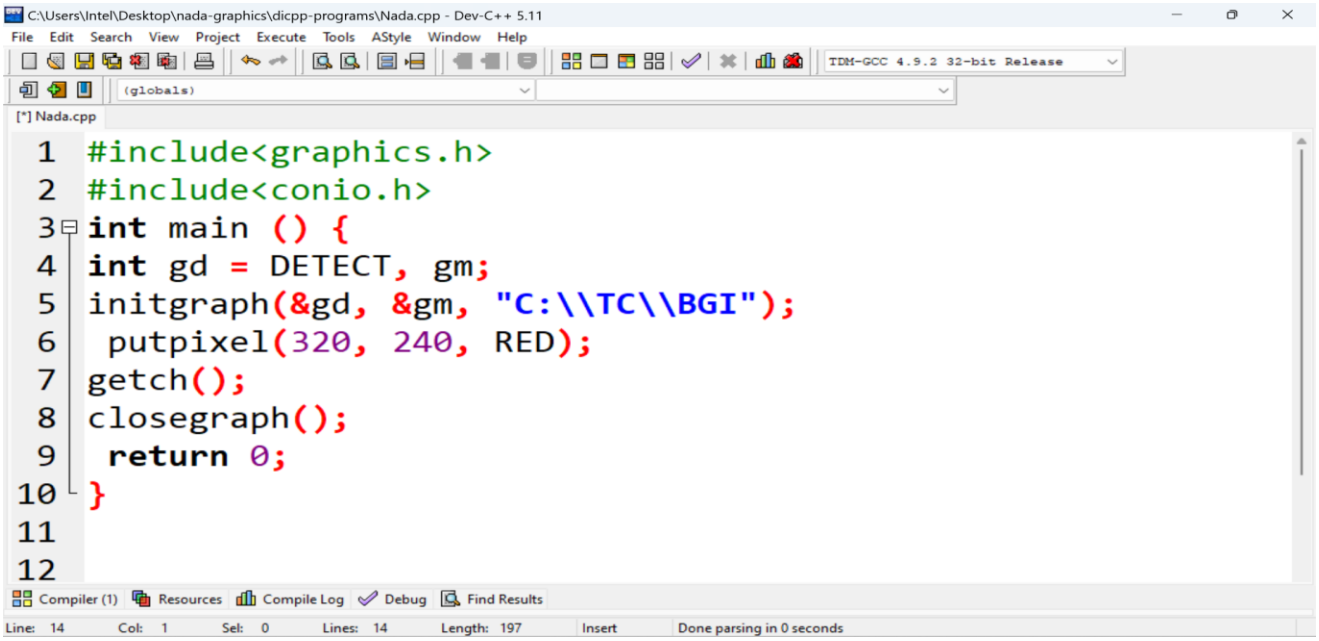


THE OUTPUT:

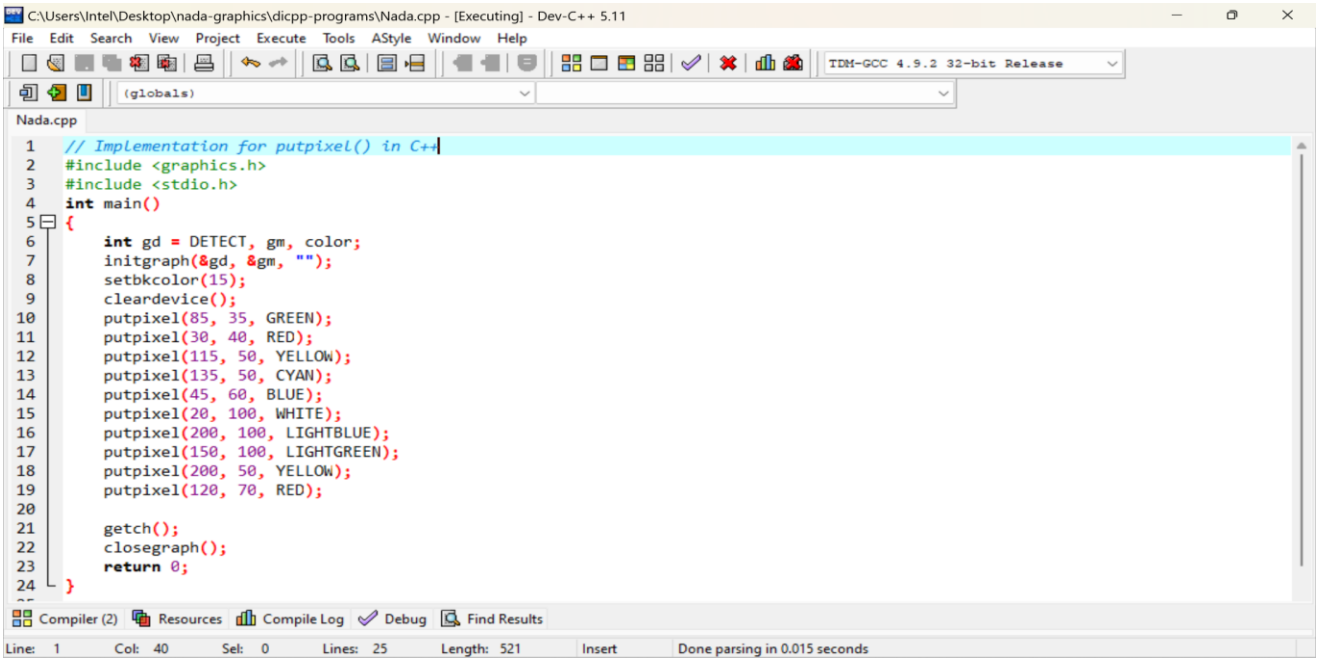
color of pixel at location (320,240) is 4

Exercise:

Write a C++ program to draw RED pixel at location (320, 240) using putpixel() function.

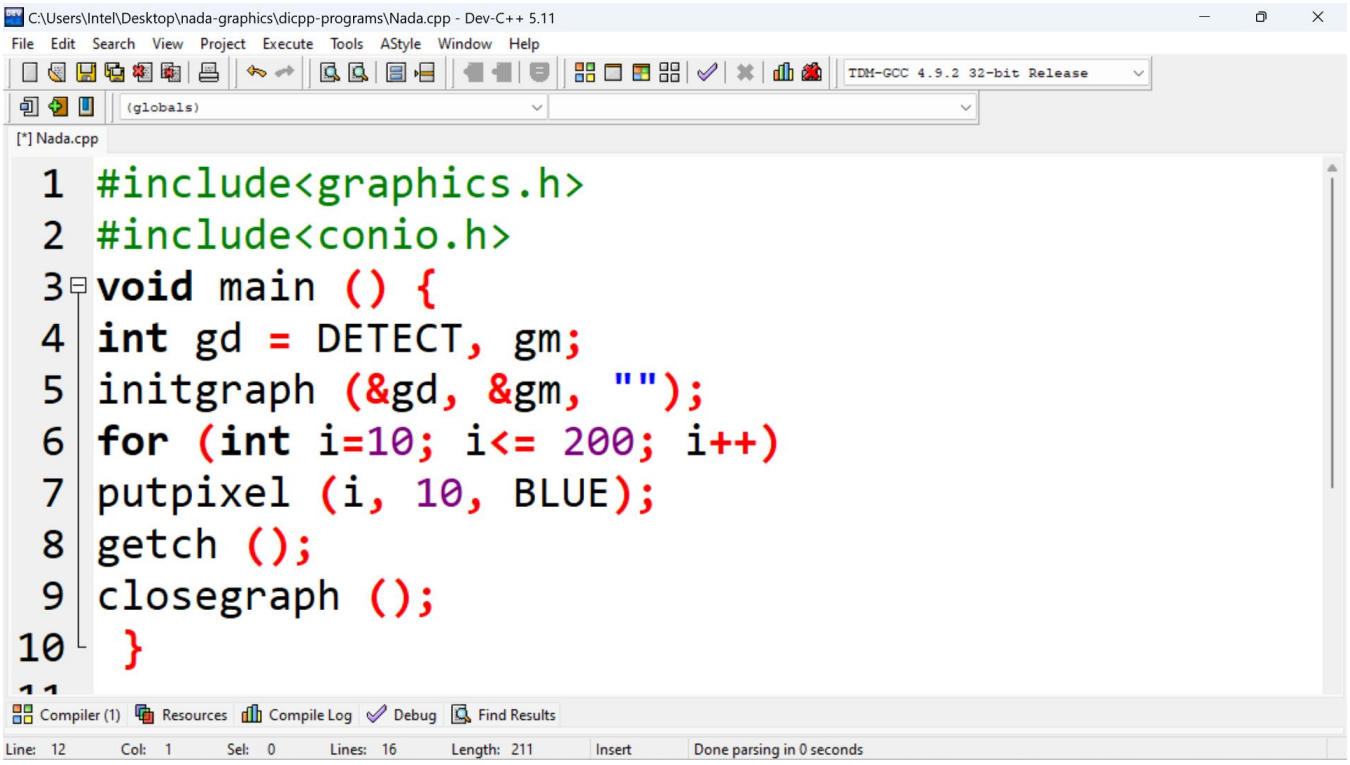


HOMEWORK **What is the output of this code?**



Exercise:

Write a C++ program to draw a horizontal line with BLUE color using putpixel() function, starting from point**(10, 10)** to point**(200,10)**.



The C++ language provides us with important standard functions:

- **getmaxx():** this function returns the maximum X coordinate for the current graphics mode and driver. Syntax : **int getmaxx();**
- **getmaxy():** this function returns the maximum Y coordinate for the current graphics mode and driver. Syntax : **int getmaxy();**

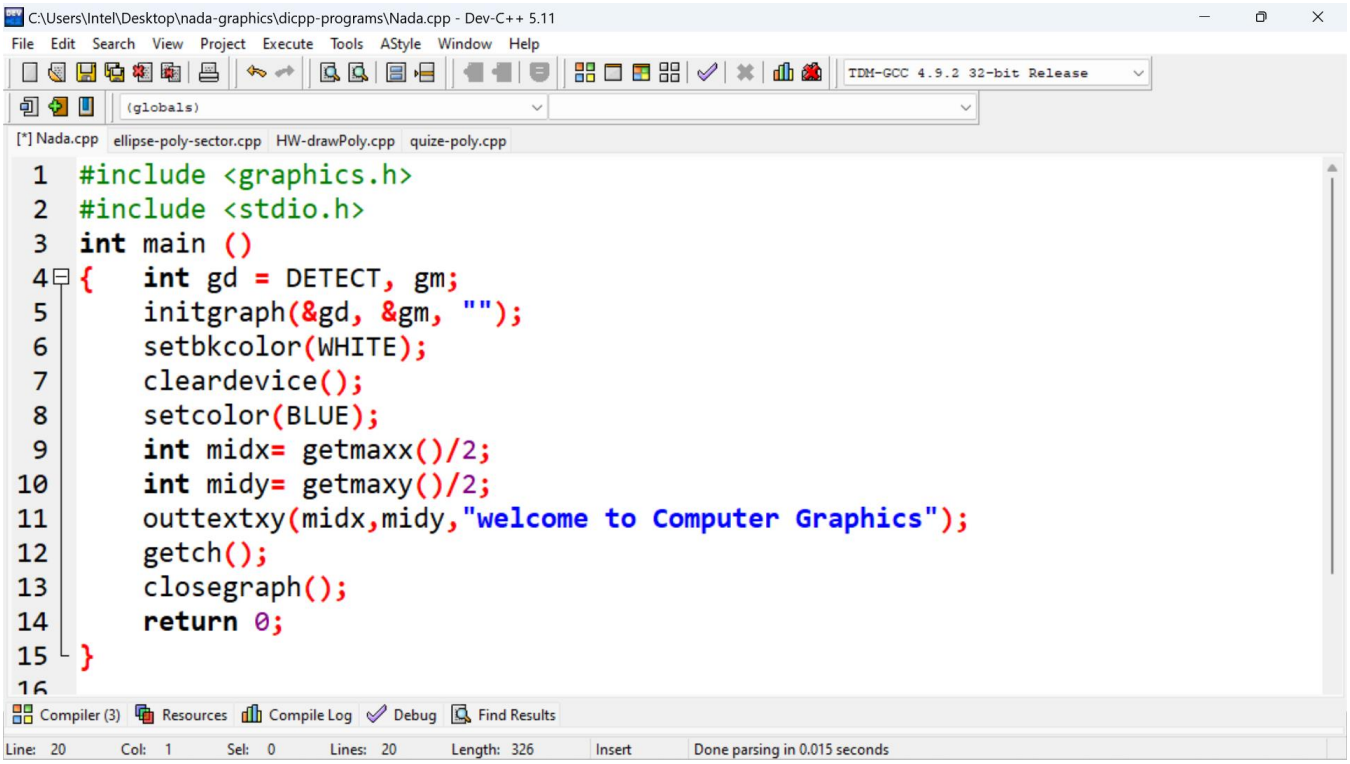
example:

```
// C++ Implementation for getmaxx() and getmaxy()
#include <graphics.h>
#include <stdio.h>
int main ()
{     int gd = DETECT, gm;
      char array [100];
      initgraph(&gd, &gm, "");
      int maxx= getmaxx();
      int maxy= getmaxy();
      sprintf(array, "Maximum X coordinate and Maximum
       Y coordinate for current Graphics Mode and Driver
        = %d,%d", maxx,maxy);
      outtext(array);
      getch();
      closegraph();
      return 0;
}
```

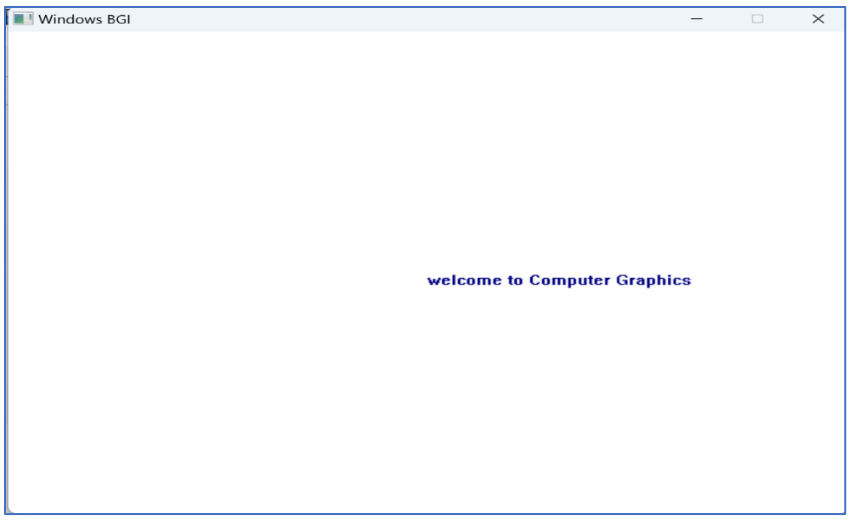
THE OUTPUT:

Maximum X coordinate and Maximum coordinate for current Graphics Mode and Driver = 639, 479

Exercise: write a C++ program to display "**Welcome to Computer Graphics**" in the middle of the screen.



THE OUTPUT:



- **getx()** : returns the X coordinate of the current position.

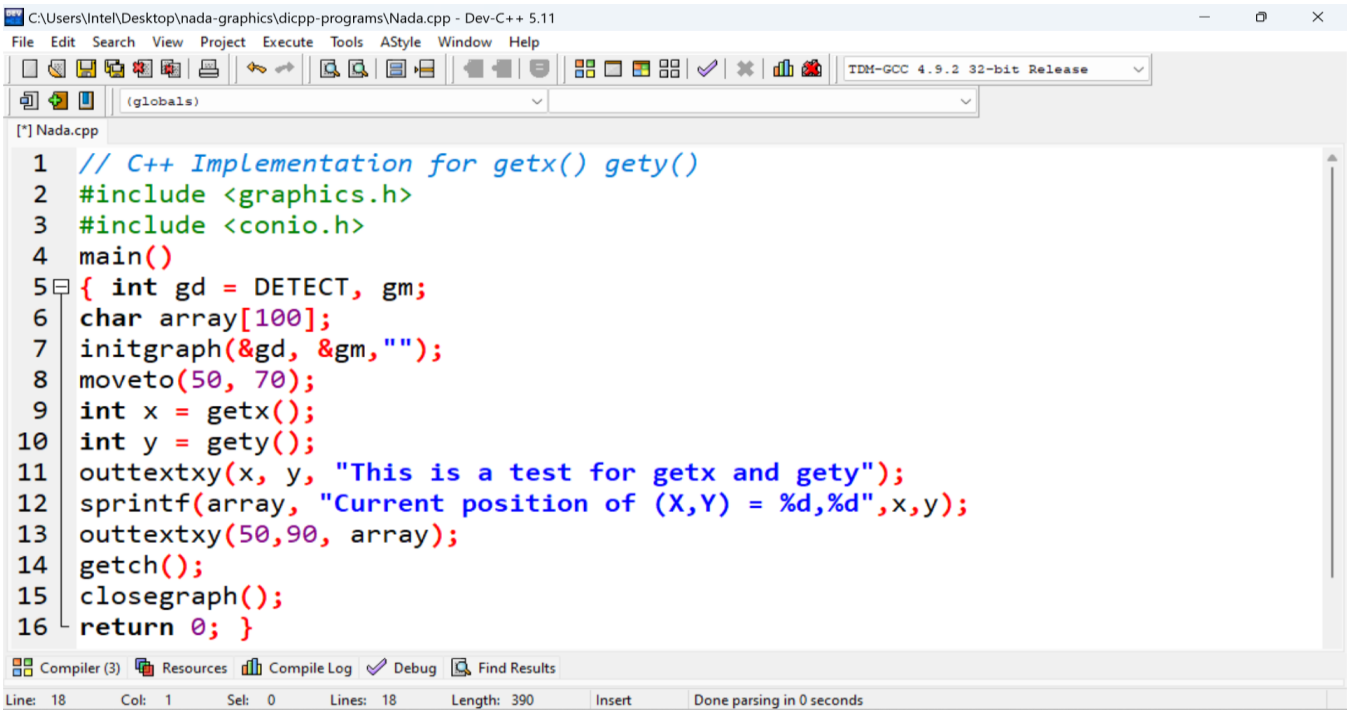
  Syntax : **int getx();**

- **gety():** returns the Y coordinate of the current position.

  Syntax : **int gety();**

Initially, the X coordinate and Y coordinate of the current(default) position are **(0,0).** On moving the coordinates(cursor/pointer) using **moveto()** function or **moverel()** function, the X, and Y coordinates change to a new position.

Exercise: write a C++ program to display the sentence "**This is a test for getx and gety**" at position (50,70).



THE OUTPUT: **This is a test for getx and gety**

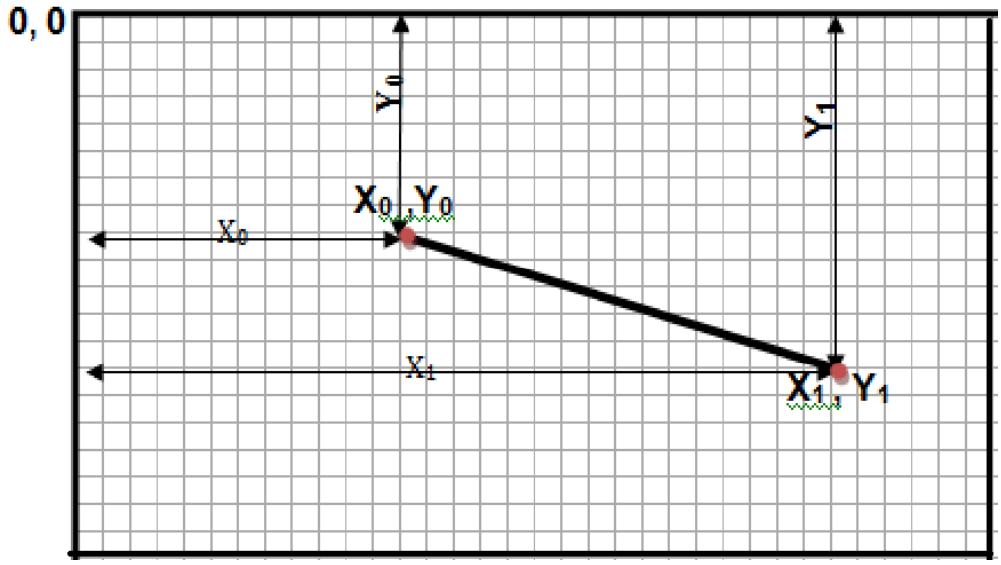
Current position of (X, Y) =50,70

Line functions in C++

3-line () function:

line function is used to draw a line from a point (x0, y0) to point (x1, y1) i.e. (x0, y0) and (x1, y1) are end points of the line

syntax: **void line (int x0, int y0, int x1, int y1);**



Example: if we want to draw a line from point (10,10) to point (120,120), write the code:

Line(10,10,120,120);

Exercise: write a C++ program to draw a triangle using the line() function where the heads of the triangle are A(100,200), B(300,200), and C(200,100).

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

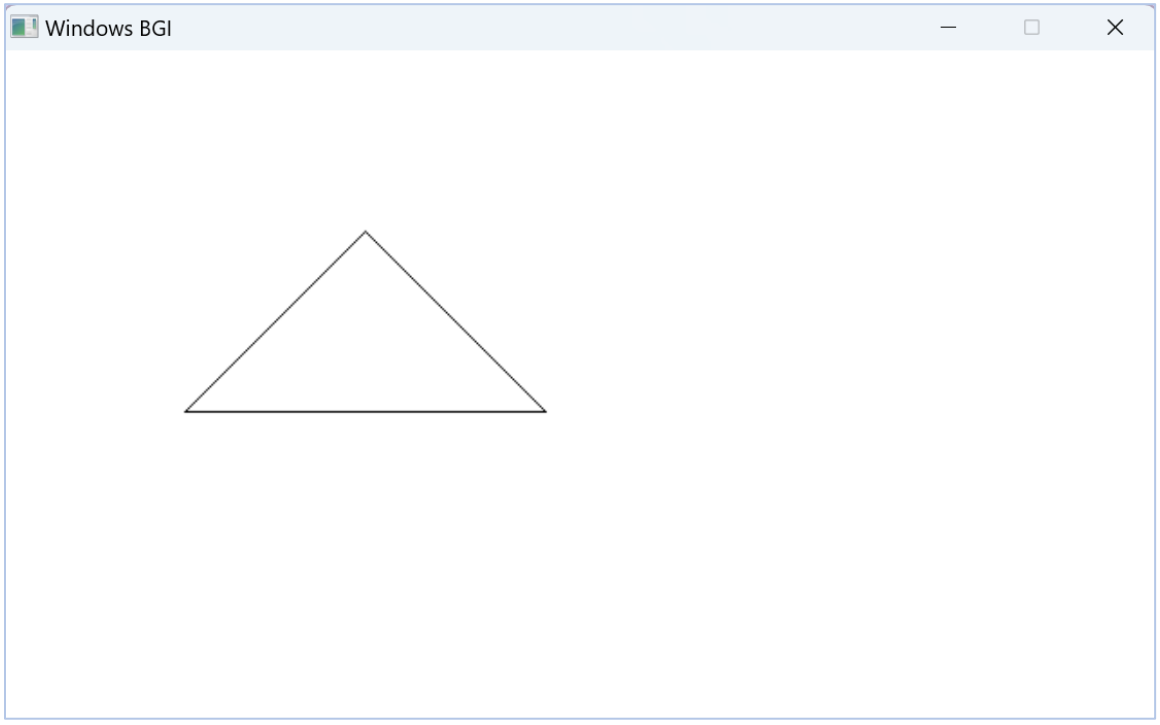
[*] Nada.cpp

```
1  #include <graphics.h>
2  #include <conio.h>
3  int main() {
4  int gd = DETECT, gm;
5  initgraph(&gd, &gm, "C:\\TC\\BGI");
6  line(100, 200, 300, 200);
7  line(300, 200, 200, 100);
8  line(200, 100, 100, 200);
9  getch(); closegraph();
10 return 0; }
```

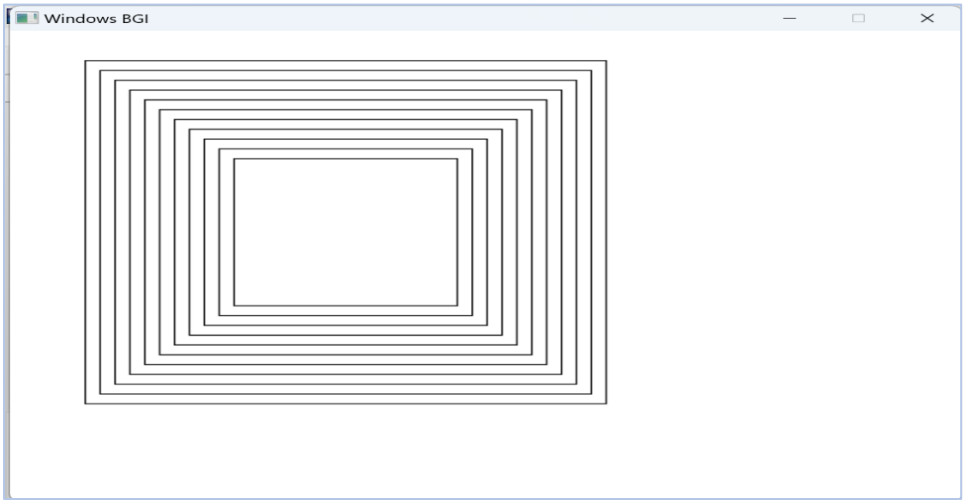
Compiler (1) Resources Compile Log Debug Find Results

Line: 13 Col: 1 Sel: 0 Lines: 16 Length: 255 Insert Done parsing in 0 seconds

THE OUTPUT:



Exercise: write a C++ program to draw the following figure using Line() function.



C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

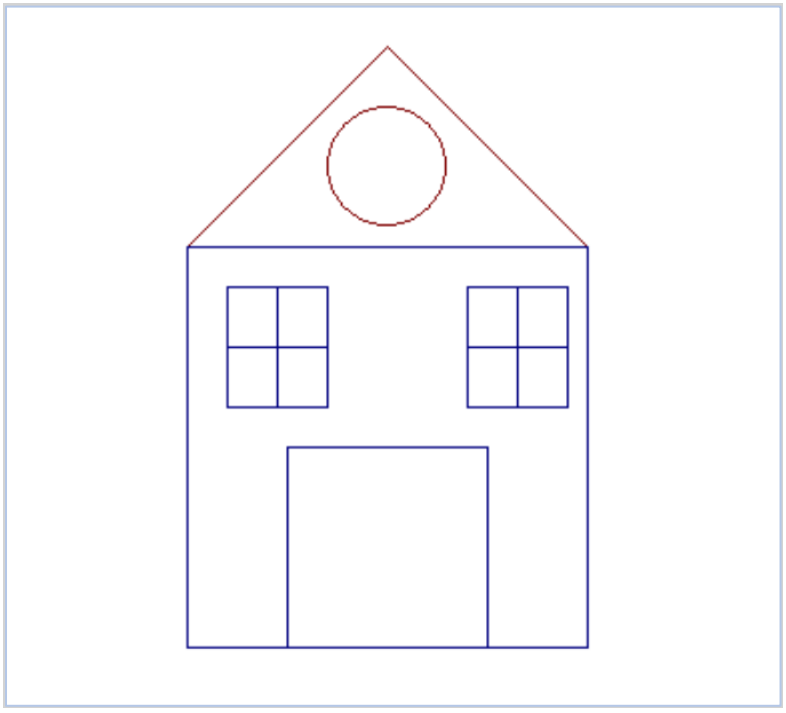
[*] Nada.cpp

```
1  #include <graphics.h>
2  #include <conio.h>
3  void sequare(int x1,int y1,int x2,int y2);
4  int main() {
5  int gd = DETECT, gm;
6  initgraph(&gd, &gm, "C:\\TC\\BGI");
7  int x1=150,x2=300,y1=130,y2=280,count=0,i=10;
8  setbkcolor(WHITE);
9  cleardevice();
10 setcolor(BLACK);
11 while(count<=10)
12 {count++;
13 sequare(x1,y1,x2,y2);
14 x1=x1-i;y1=y1-i;
15 x2=x2+i;y2=y2+i;
16 }getch(); closegraph();
17 return 0; }
18 void sequare(int x1,int y1,int x2,int y2)
19 {
20     line(x1,y1,x2,y1);
21     line(x2,y1,x2,y2);
22     line(x2,y2,x1,y2);
23     line(x1,y2,x1,y1);
24 }
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 29 Col: 1 Sel: 0 Lines: 29 Length: 536 Insert Done parsing in 0 seconds

HOMEWORK: write a C++ program to draw the following figure using line() function.



4. lineto() function

lineto() function draws a line from current position to the point(x,y).

Note: Use getx() and gety() to get the current position.

Syntax: **Void lineto(int x, int y);**

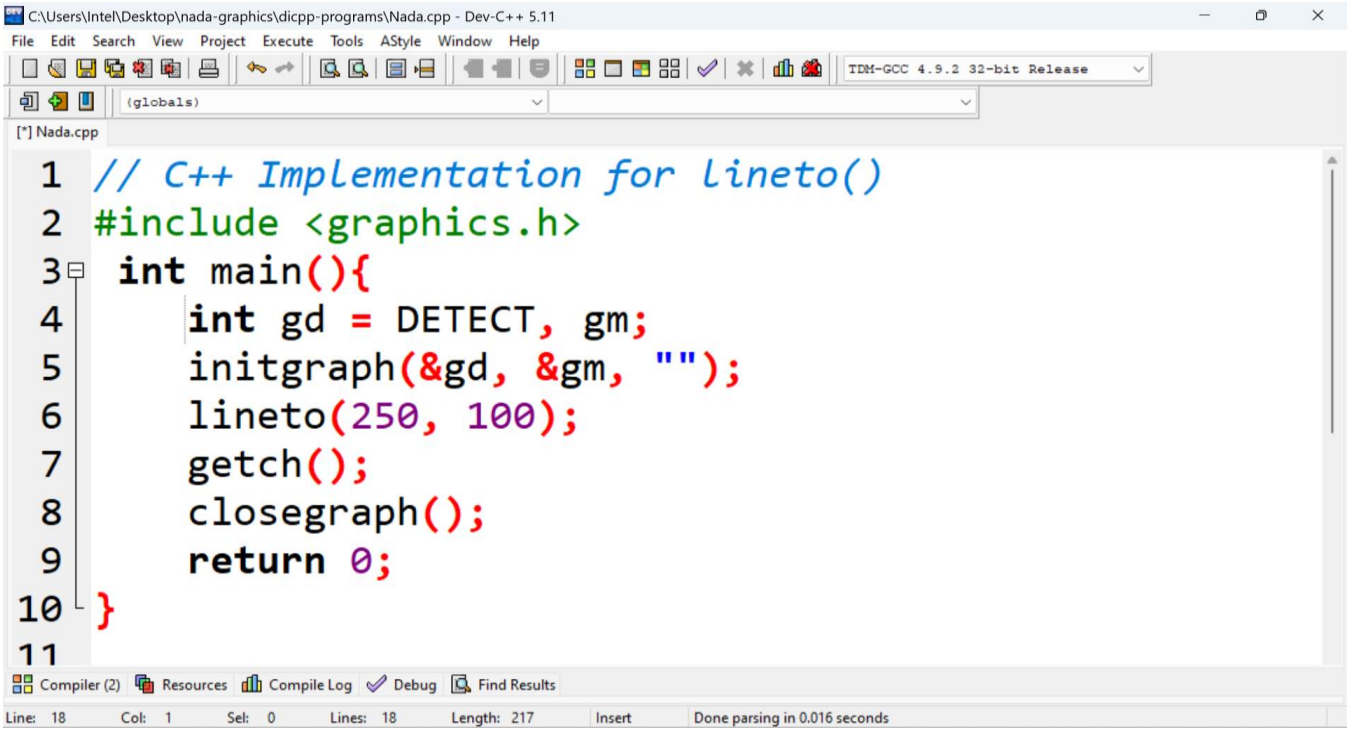
where, (x, y) are the coordinates up to which the line will be drawn from the previous point.

Example: If we want to draw a line from (100,100) to (320,240) using

Moveto() and lineto() as follow:

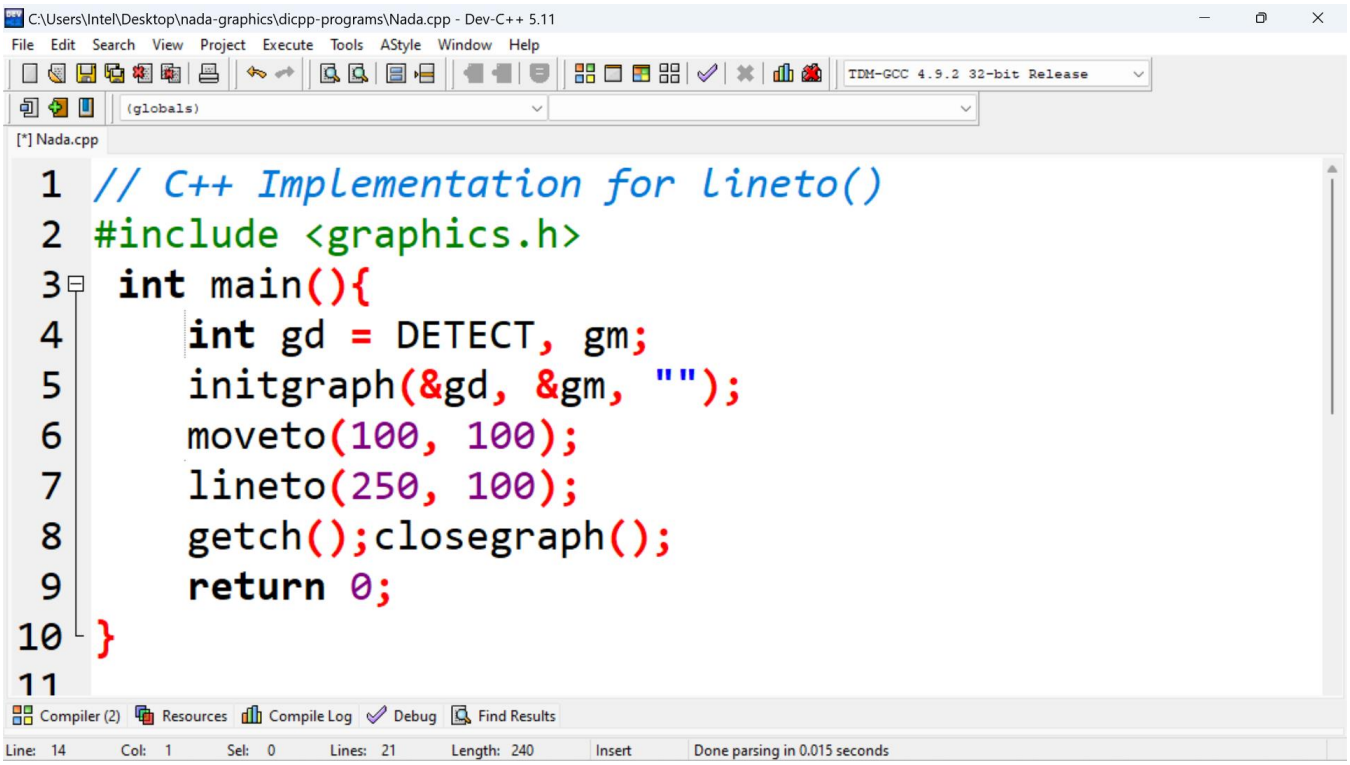
Moveto(100,100); lineto(320,240);

Example: Draw a line from the initial position (0,0) to point (250,100).

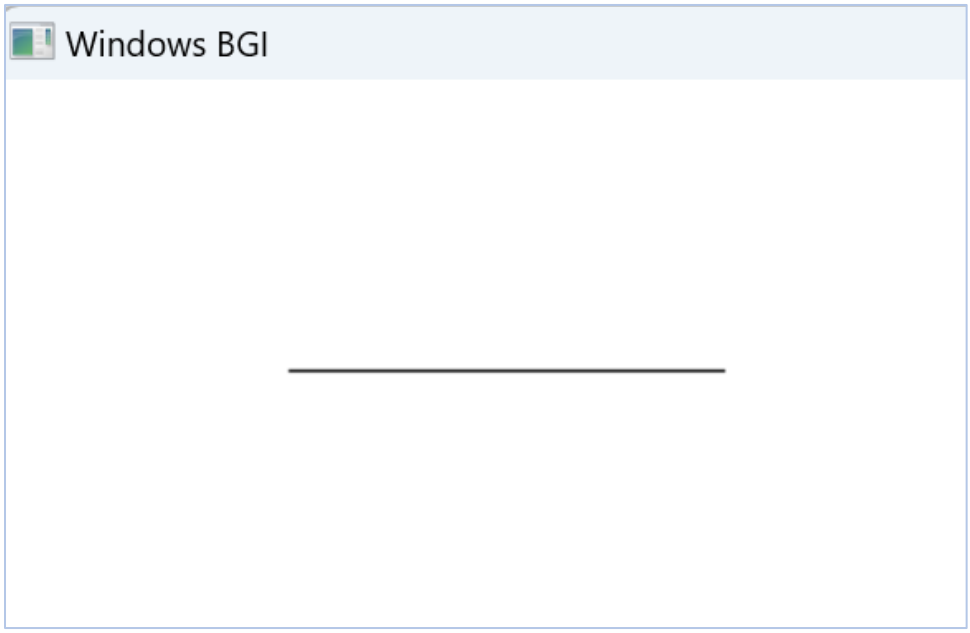


THE OUTPUT:

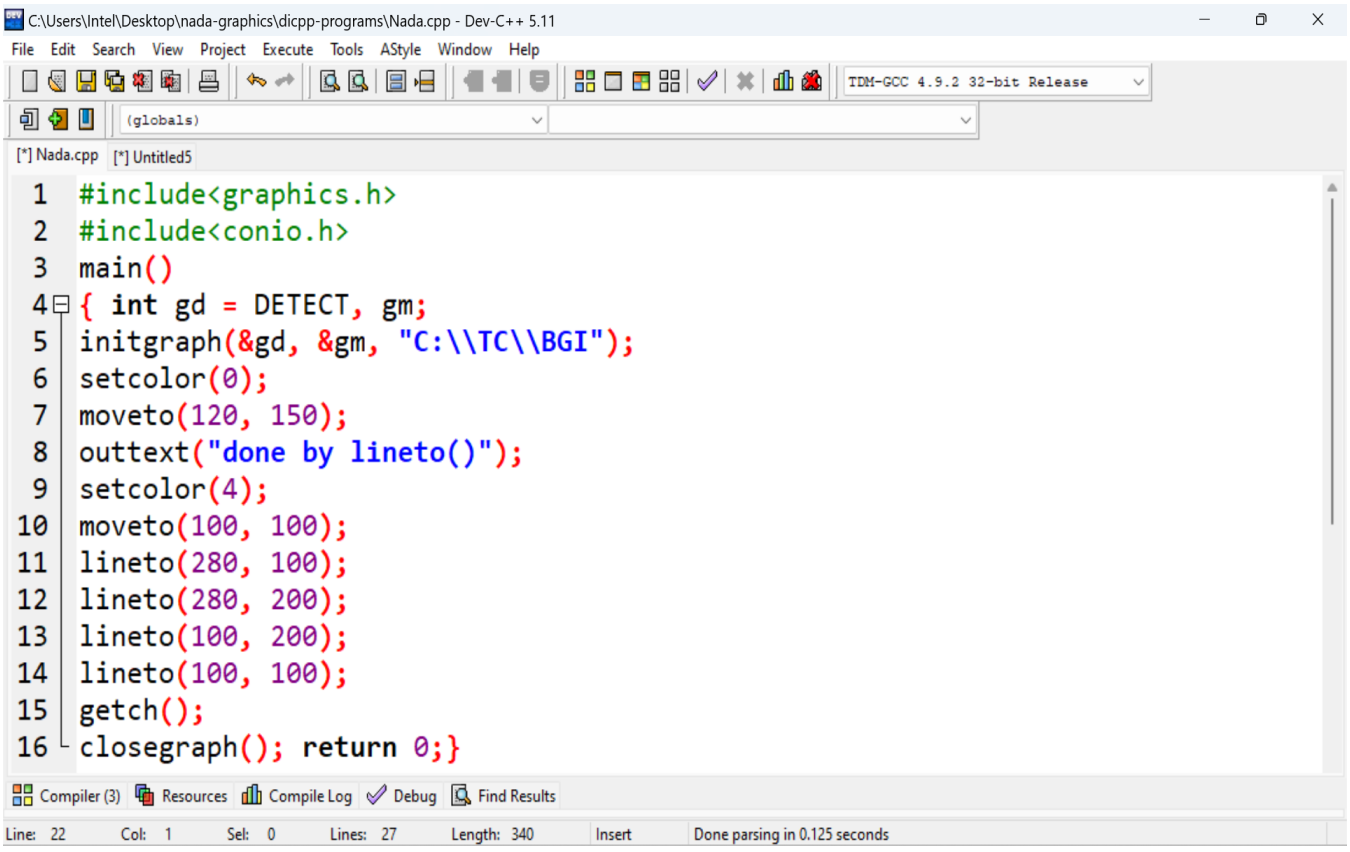
Example: Draw a line from position (100,100) to point (250,100).



THE OUTPUT:



Exercise: Write a C++ program to draw a red rectangle using lineto() function, its coordinates from the top left angle are (100,100) and the coordinates from the bottom right angle are (280,200).



THE OUTPUT:

5.linerel () function

linerel() function draws a line from the current position (x_pos1, y_pos1) to a point that is at a relative distance (x, y) from the Current Position, then advances the Current Position by (x, y). Note: Use getx() and gety() to find the current position.

Syntax: **void linerel(int x, int y);**

The end position is calculated as:

x_pos2 = x_pos1 + x;

y_pos2 = y_pos1 + y;

Example: Draw a line from point (250,250) to point (350,150)

```
Moveto(250,250);
Linerel(100,-100);
```

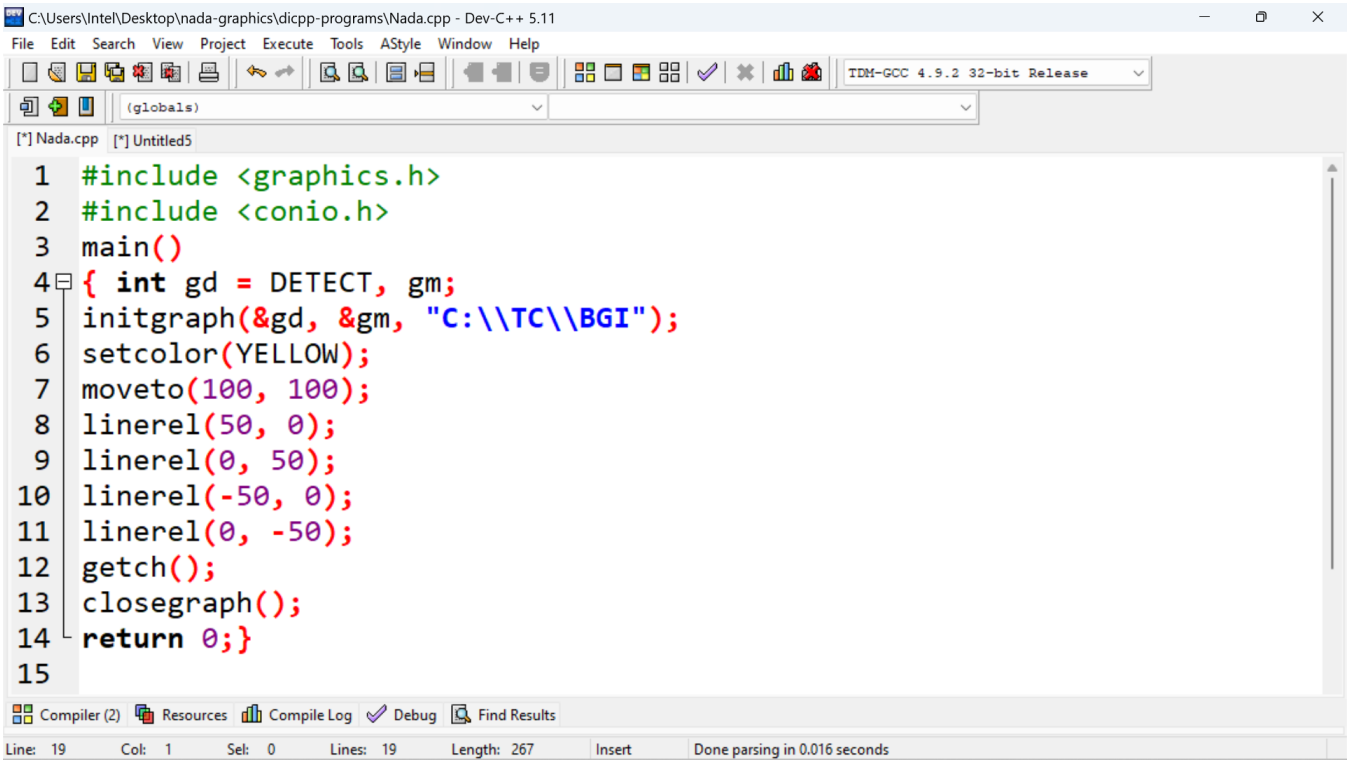
Exercise: what is the line coordinates drawn by the following code

```
Moveto(20,30);
Moverel(100,30);
Linerel(320,240);
```

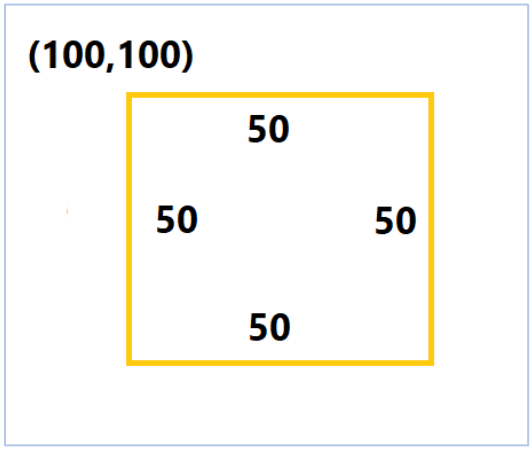
Solution:

```
Line(120,60,440,300);
```

Exercise: Write a C++ program to draw a yellow square with a radius of (50), its coordinates start from (100,100). Using lineto() function.



THE OUTPUT:



6.Moveto () function:

The "moveto" function moves the current cursor position to a specified location on the screen (x,y) where the output will be printed. It is similar to "gotoxy" function used in text mode.

Syntax: **void moveto (int x, int y);**

Where:

x: Represents the new x-coordinate of the new position.

y: Represents the new y-coordinate of the new position.

Example: how to use moveto() function and print Programming digest into C++ graphics mode:

```
moveto(400,200);
outtext("Programming digest");
```

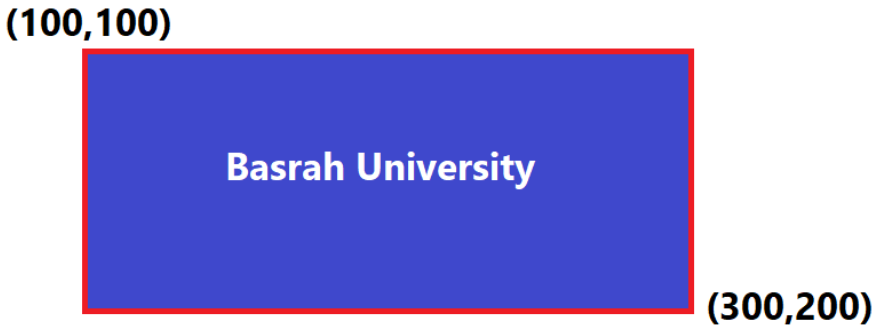
Example: if we want to write "**Computer Graphics**" on the position (100,100).

```
moveto(100,100);
outtext("Computer Graphics");
```

Exercise: write a C++ program to print "Basrah University" in white color inside a red rectangle filled with blue paint, its top left angle has the coordinates (100,100) and the bottom right angle has the coordinates (300,200).

```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
setbkcolor(15);
cleardevice();
setcolor(RED);
setfillstyle(SOLID_FILL, BLUE);
rectangle(100, 100, 300, 200);
floodfill(110, 110, 4);
moveto(110, 140);
setcolor(15);
outtext("Basrah university");
getch();
closegraph(); return 0; }
```

THE OUTPUT:



7.Moverel()Function: moves a point from the current position(x_pos1, y_pos1) to a point that is at a **relative distance**(x, y) from the Current Position and then advances the Current Position by (x,y).

Note: getx and **gety** can be used to find the current position.

Syntax: **moverel(int x, int y);**

The end position is calculated as:

x_pos2 = x_pos1 + x;

y_pos2 = y_pos1 + y;

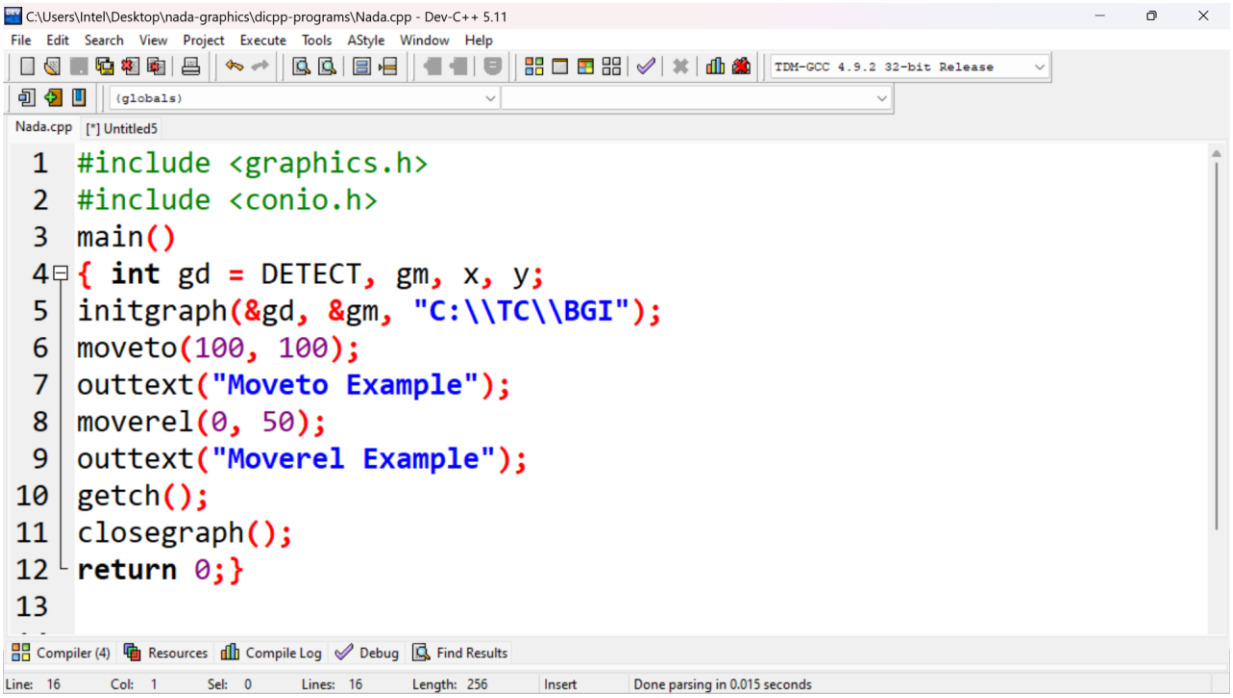
Example: if we want to move the cursor from the point p1 (100,100)

To new coordinates p2 (150,50), using moverel()

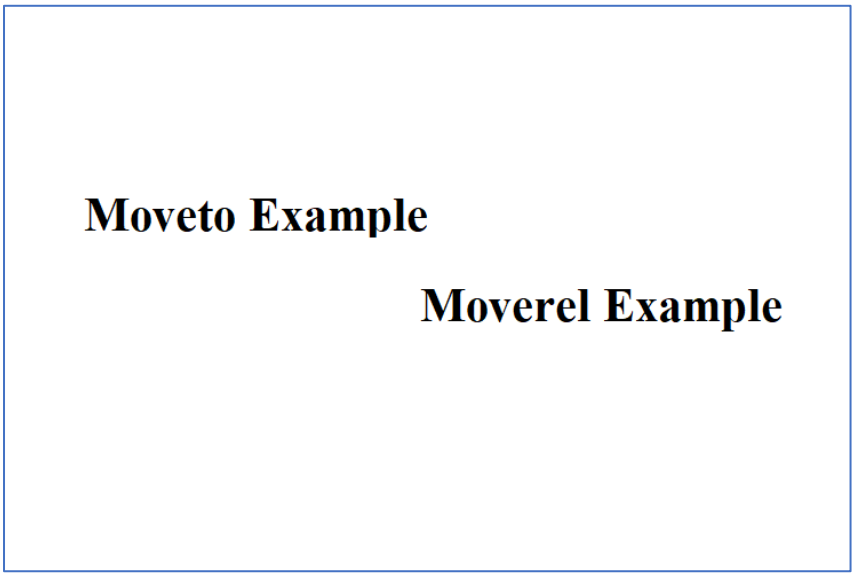
Moveto(100,100);

Moverel(50,-50);

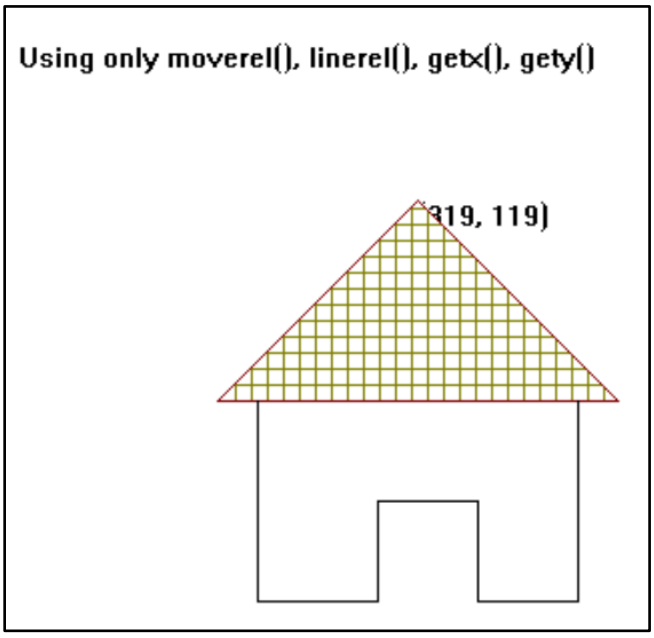
Exercise: **write a C++ program to print ” Moveto Example” at the point(100,100).then move the cursor to the new position(100,150) using the function moverel() then write the sentence ”Moverel Example”.**



THE OUTPUT:



Exercise: write a c++ program to display the following figure Using only moverel(), linerel(), getx(), gety().



```
#include <graphics.h>
#include<conio.h>
Int main(){
 int gd = DETECT, gm ,maxx, maxy, x, y;
initgraph(&gd, &gm, "");
char str[100];
setbkcolor(15);cleardevice();setcolor(0);
maxx = getmaxx(); maxy = getmaxy();
outtextxy(120, 40, "Using only moverel(), linerel(), getx(), gety()");
setcolor(0);moverel(maxx/2, maxy/4);x = getx(); y = gety();
sprintf(str, "(%d, %d)", x, y);outtextxy(x, y, str);
setcolor(RED);linerel(100, 100);linerel(-200, 0);
linerel(100, -100);setfillstyle(HATCH_FILL, BROWN);
floodfill(maxx/2+5, maxy/4+10, RED);setcolor(0);
moverel(-80, 100);linerel(0, 100);
linerel(60, 0);linerel(0, -50);linerel(50, 0);
linerel(0, 50);linerel(50, 0);linerel(0, -100);
getch();
closegraph();}
```

Colors Functions دوال التلوين

8.Setcolor() function

Setcolor function is used to set the current drawing color to the new color.

Syntax: **void setcolor (int COLOR);**

In Graphics, each color is assigned a number. The total number of colors available is 16 (0-15). The number of available colors depends on the current graphics mode and driver. For example, setcolor(RED) or setcolor(4) changes the current drawing color to RED. Remember that the default drawing color is WHITE. The Colors table is given below.

COLORS (VALUES)

BLACK (0)	**BLUE (1)**
GREEN (2)	**CYAN (3)**
RED (4)	**MAGENTA (5)**
BROWN (6)	**LIGHTGRA (7)**
DARKGRAY (8)	**LIGHTBLUE (9)**
LIGHTGREEN (10)	**LIGHTCYAN (11)**
LIGHTRED (12)	**LIGHTMAGENTA (13)**
YELLOW (14)	**WHITE (15)**

EXAMPLE: Write a C++ program to draw a triangle, each side with a different color. Using line() function.

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

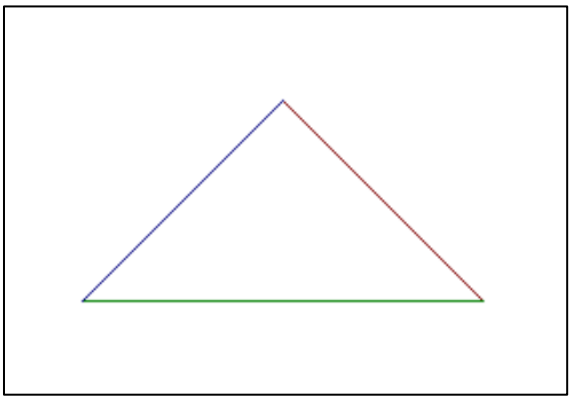
Nada.cpp [*] Untitled5

```
1  #include <graphics.h>
2  #include <conio.h>
3  main()
4  { int gd = DETECT, gm, x, y;
5  initgraph(&gd, &gm, "C:\\TC\\BGI");
6  setbkcolor(15);
7  cleardevice();
8  setcolor(4);line(300,100,400,200);
9  setcolor(2);line(400,200,200,200);
10 setcolor(1);line(200,200,300,100);
11 getch();
12 closegraph();
13 return 0;}
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 16 Col: 1 Sel: 0 Lines: 17 Length: 304 Insert Done parsing in 0 seconds

THE OUTPUT:



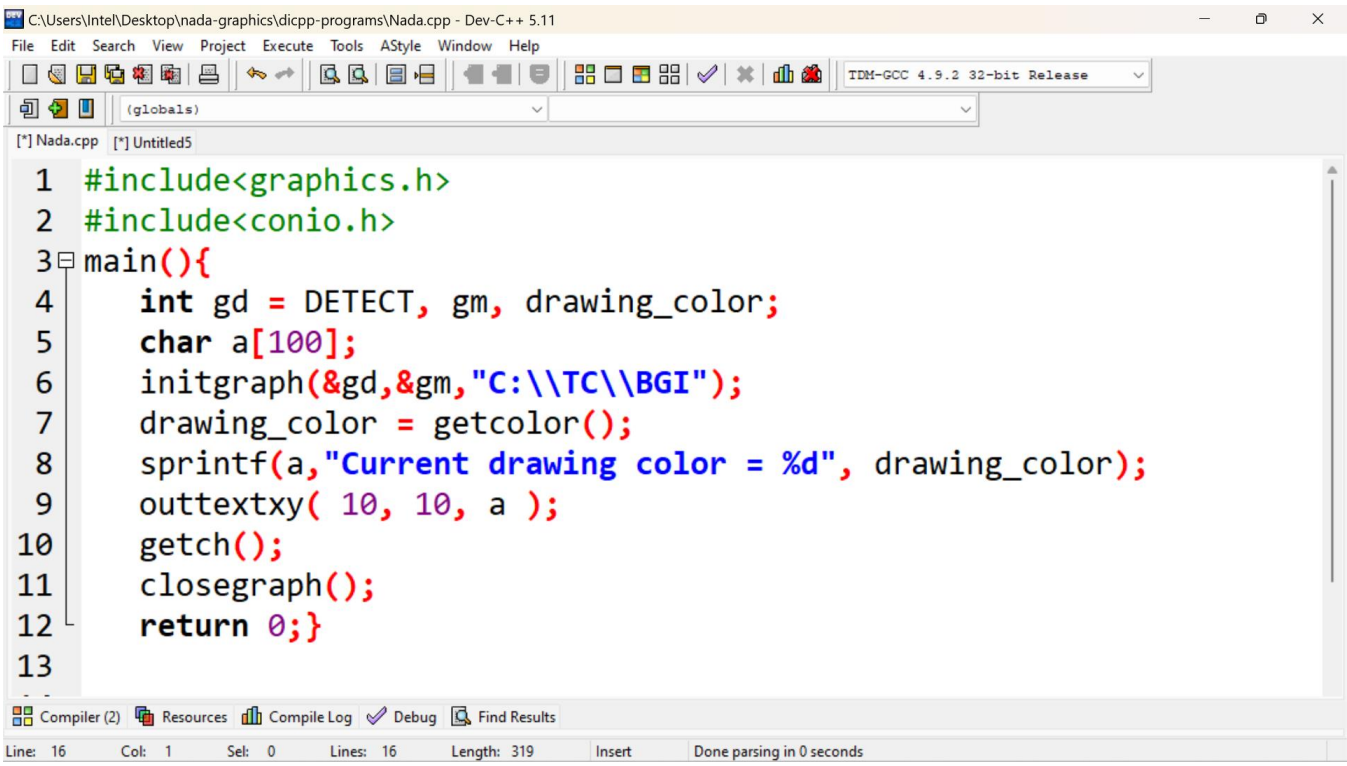
9.Getcolor() function

Getcolor function returns the current drawing color.

Syntax: `int getcolor();`

Example: a = getcolor(); // a is an integer variable
if the current drawing color is WHITE then a will be 15.

Example: implement getcolor() function



THE OUTPUT:

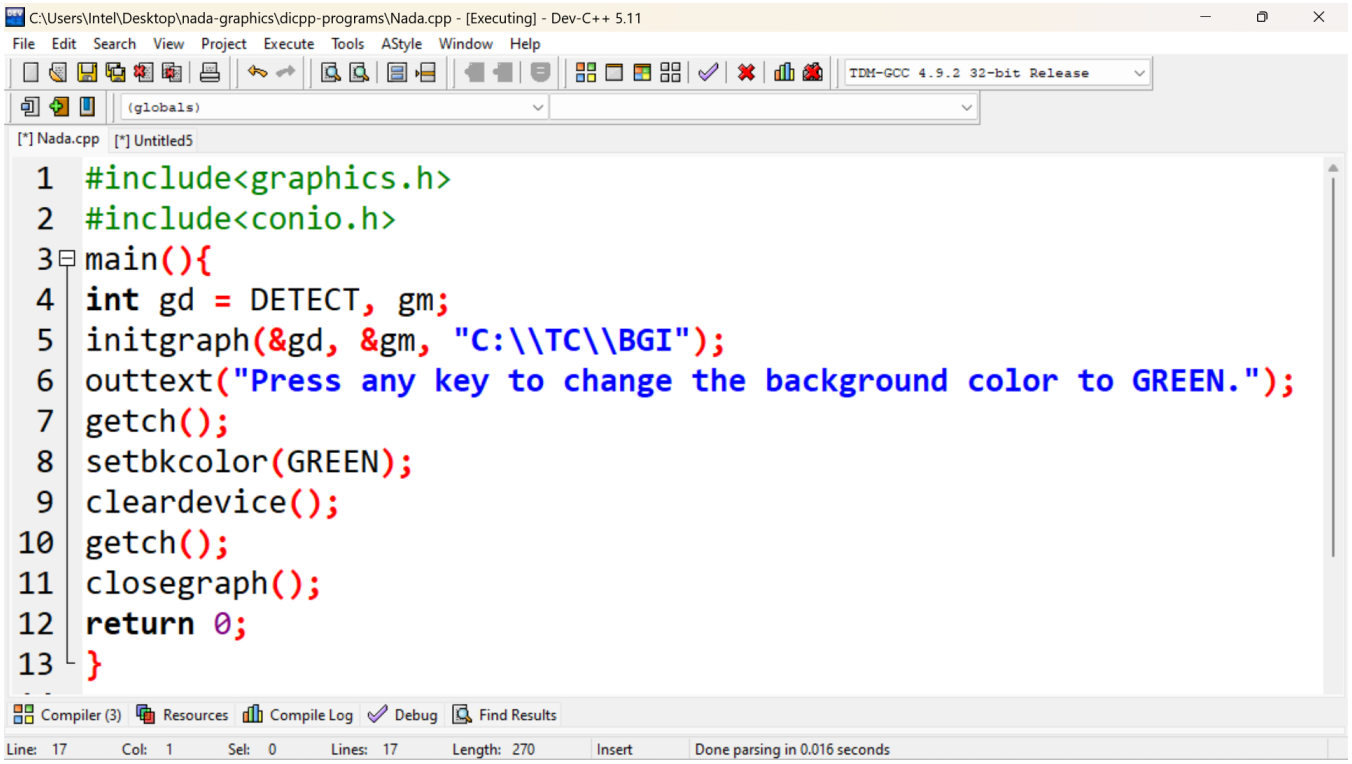
Current drawing color = 15

10.Setbkcolor() function

setbkcolor function changes the current background color e.g. setbkcolor(YELLOW) changes the current background color to YELLOW. Remember that the default drawing color is WHITE and the background color is BLACK.

Syntax: **void setbkcolor(int color);**

Example: in this program when we Press any key the background color change to GREEN.



11.Getbkcolor() function

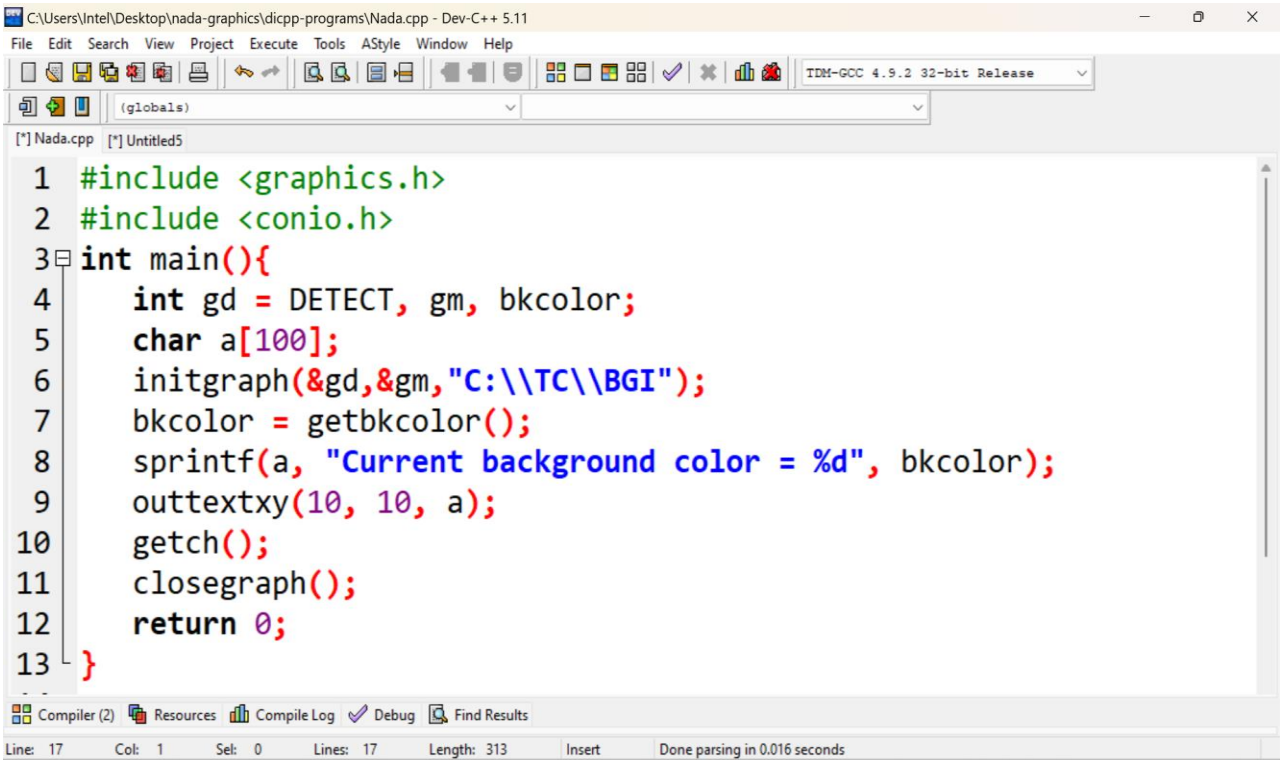
Function getbkcolor returns the current background color.

Syntax: **int getbkcolor();**

e.g. color = getbkcolor(); // color is an int variable

If the current background color is GREEN, the color will be 2.

Example: If we want to print the number of the current background color along with a message stating that at the point (10,10) as follows:

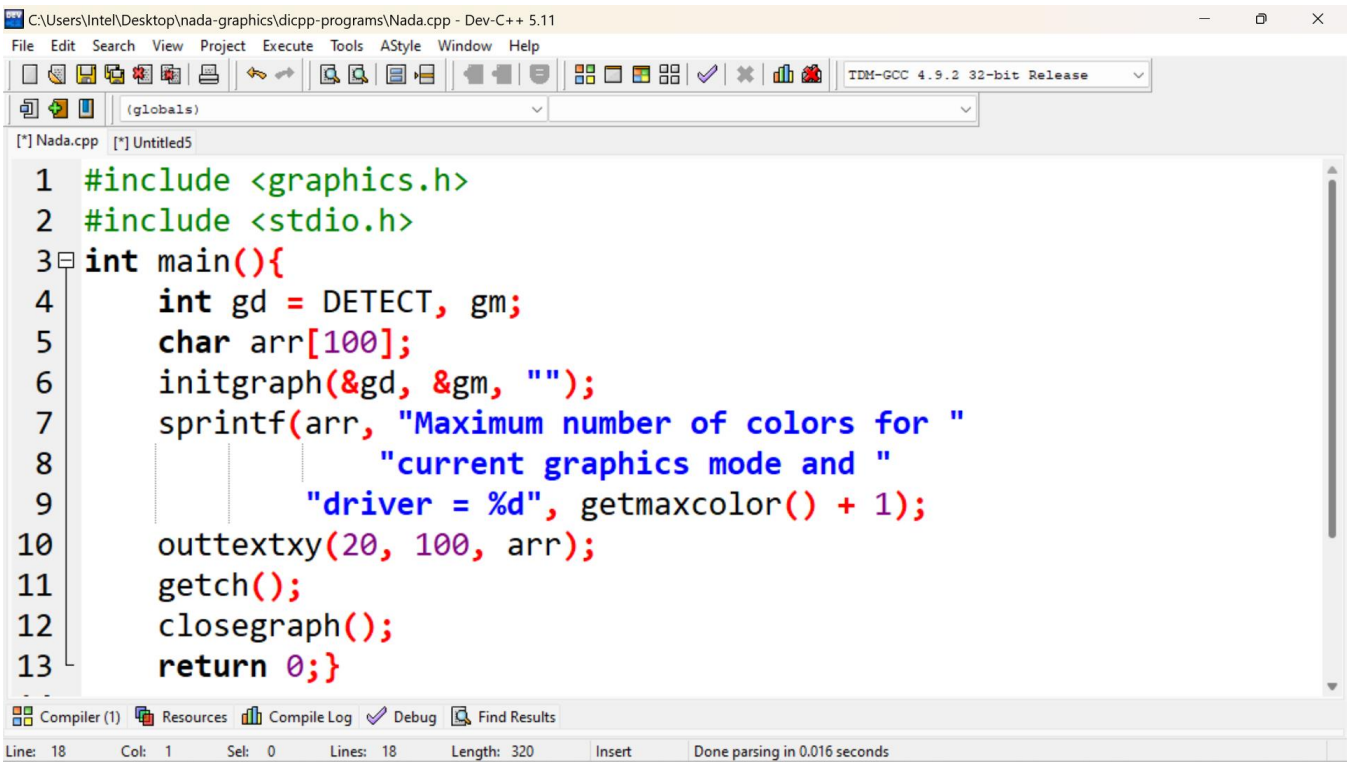


THE OUTPUT: Current background color =0

12.Getmaxcolor() function

this function returns the maximum color value for the current graphics mode and driver. As color numbering starts from zero, the total number of colors available for the current graphics mode and driver are **(getmaxcolor() + 1)** .

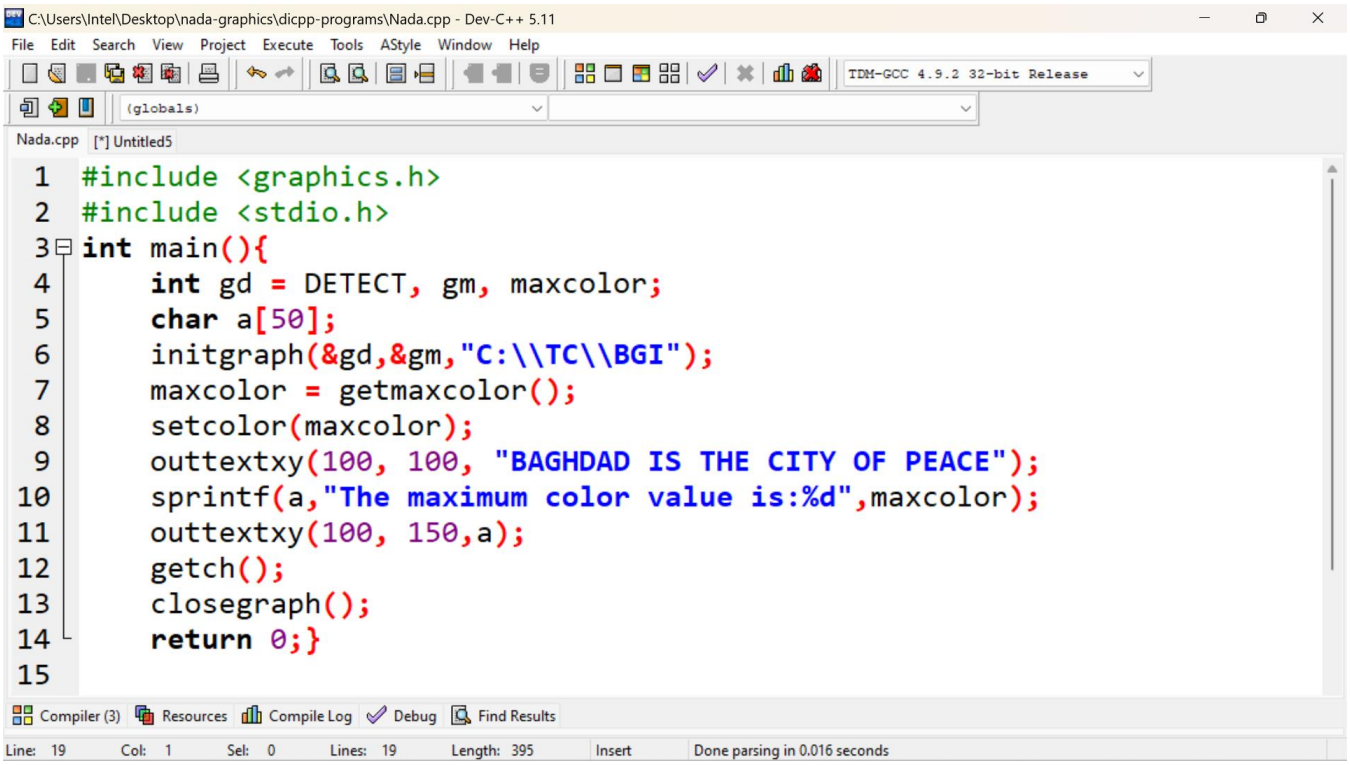
Syntax: **int getmaxcolor();**



THE OUTPUT:

Maximum number of colors for current graphics mode and driver =16

Exercise: Write a program in C++ language to print the text "**BAGHDAD IS THE CITY OF PEACE**" in a color obtained from the function **getmaxcolor()** on the drawing screen at the location **(100,100).** Then print the color number with an explanatory message at location **(100,150).**



THE OUTPUT:

BAGHDAD IS THE CITY OF PEACE

The maximum color value is:15

13.Setfillstyle() function

This function sets the current fill pattern and fills the color. it is set before the drawing function and used with floodfill() function

Syntax: **void setfillstyle(int pattern, int color);**

Where pattern represents the form of the current fill pattern, and color represents the color of the fill pattern. Below is the table showing INT VALUES corresponding to Patterns:

PATTERN	INT VALUES
EMPTY_FILL	0
SOLID_FILL	1
LINE_FILL	2
LTSLASH_FILL	3
SLASH_FILL	4
BKSLASH_FILL	5
LTBKSLASH_FILL	6
HATCH_FILL	7
XHATCH_FILL	8
INTERLEAVE_FILL	9
WIDE_DOT_FILL	10
CLOSE_DOT_FILL	11
USER_FILL	12

Exercise: write a c++ program to draw a circle filled with:

pattern = HATCH_FILL, Color = RED

circle : x = 250, y = 250, radius = 100

floodfill : x = 250, y = 250, border color=4.

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

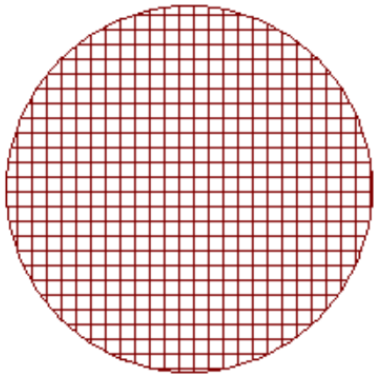
[*] Nada.cpp [*] Untitled5

```
1  #include <graphics.h>
2  #include <stdio.h>
3  int main(){
4      int gd = DETECT, gm, maxcolor;
5      initgraph(&gd,&gm,"C:\\TC\\BGI");
6      setbkcolor(15);
7      cleardevice();
8      setcolor(4);
9      setfillstyle(HATCH_FILL, RED );
10     circle(250,250,100);
11     floodfill(250,250,RED);
12     getch();
13     closegraph();
14     return 0;}
15
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 19 Col: 1 Sel: 0 Lines: 19 Length: 330 Insert Done parsing in 0 seconds

THE OUTPUT:



14.floodfill()Function

floodfill function is used to fill an enclosed area.

Syntax: **void floodfill(int x, int y, int border);**

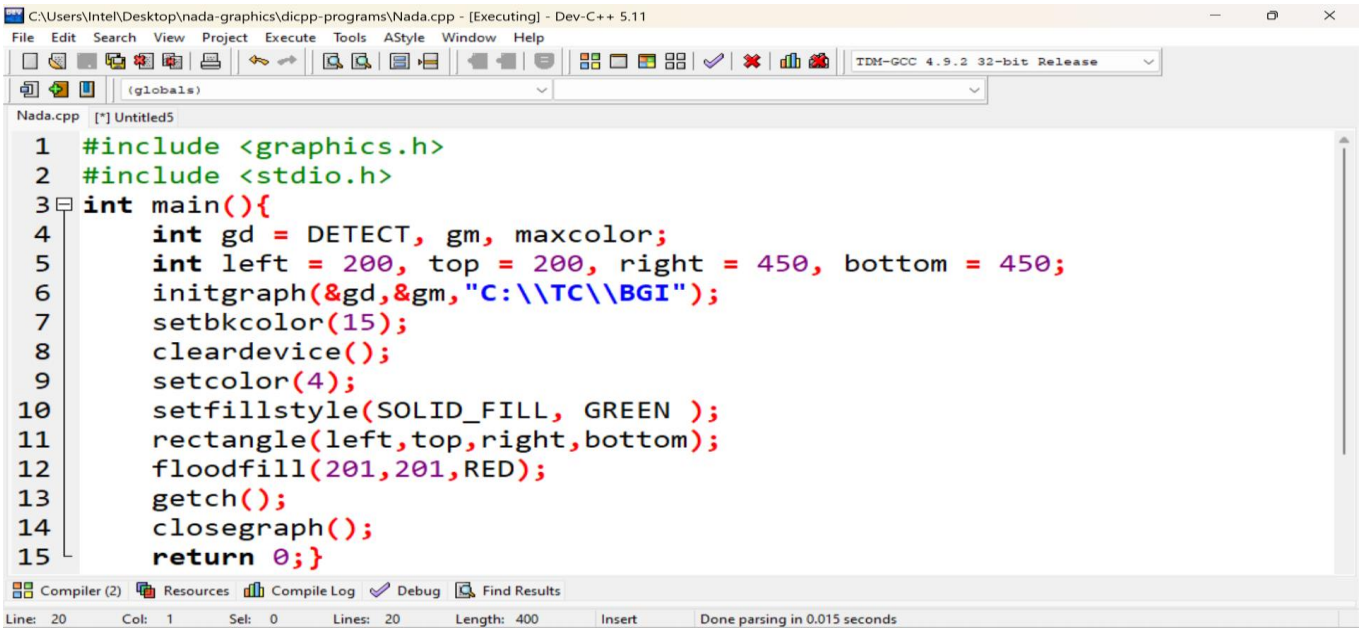
The current fill pattern and fill color are used to fill the area. (x, y) is any point on the screen if (x,y) lies inside the area then inside will be filled otherwise outside will be filled, border specifies the color of the boundary of the area. To change the fill pattern and fill color use setfillstyle().

Exercise: write a c++ program to draw a rectangle filled with:

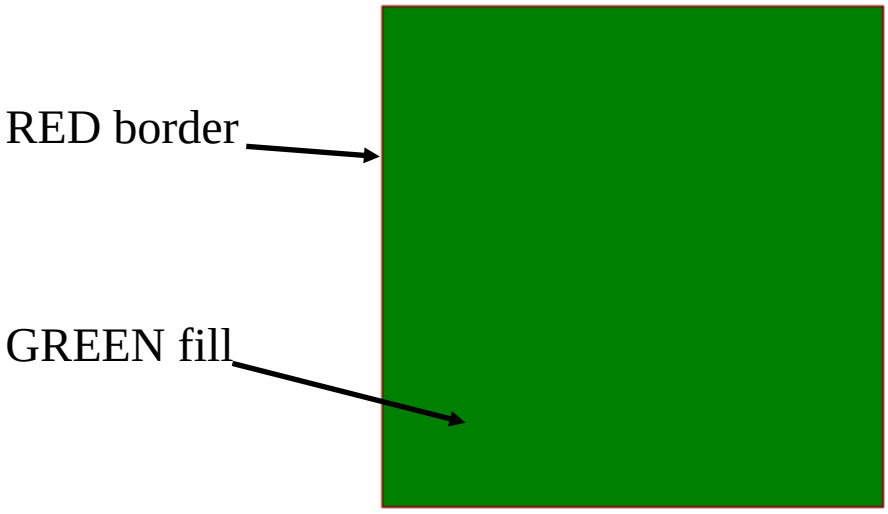
pattern = SOLID_FILL, Color = GREEN.

rectangle: left = 200, top = 200, right = 450, bottom = 450.

floodfill : x = 201, y = 201, border_color = RED.



THE OUTPUT:



HOMEWORK: Write a C++ program to draw the following figure using functions: line(), setfillstyle(), floodfill().

15.setlinestyle() function

sets the style for all lines drawn by line, lineto, rectangle, drawpoly, and so on.

Syntax:

```
Void setlinestyle(int linestyle,unsigned pattern,int thickness);
```

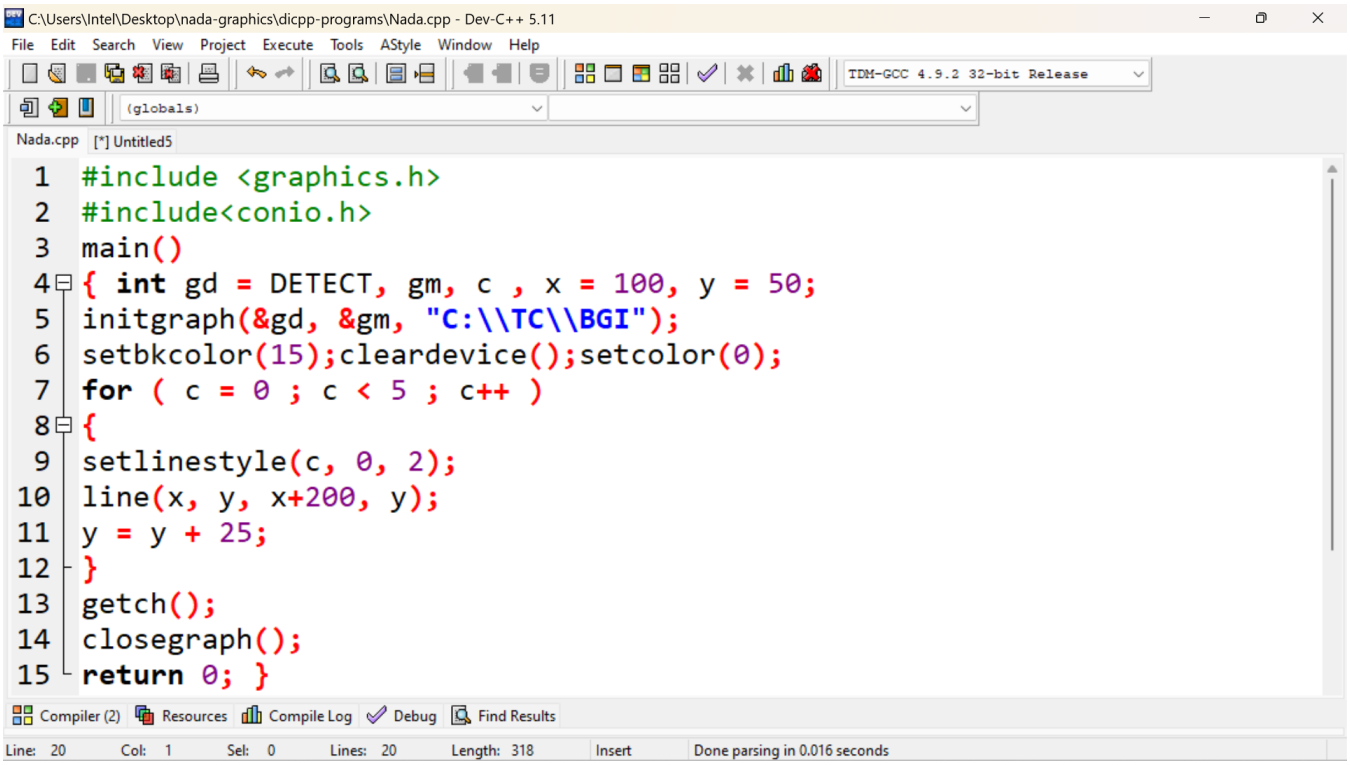
Where: linestyle specifies in which of several styles subsequent lines will be drawn (such as solid, dotted, centered, dashed).

Name	Value	Description
SOLID_LINE	0	solid line
DOTTED_LINE	1	dotted line
CENTER_LINE	2	center line
DASHED_LINE	3	dashed line
USERBIT_LINE	4	user defined line style

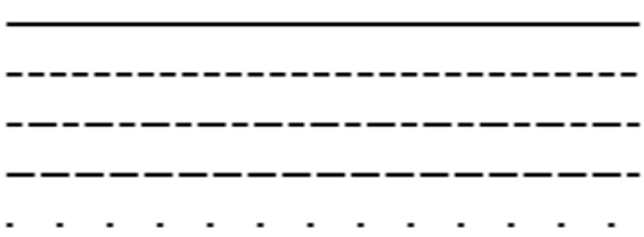
An unsigned pattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. A value for an 'unsigned pattern' must always be

supplied. It is simply ignored if 'linestyle' is not USERBIT_LINE(4). **thickness** specifies whether the width of subsequent lines drawn will be normal or thick.

Exercise: draw multiple lines with different line styles.



THE OUTPUT:



Text functions دوال النصوص

16.Outtext() function

Outtext function display text at the current position on the screen in graphics mode. If we want to display the text at the new position, use the function moveto() or moverel() before outtext function.

Syntax: **void outtext(char *string);**

Note: Do not use printf, gotoxy, and other textmode functions in graphics mode. Do not use escape sequences such as '\n','\t' etc in outtext and outtextxy functions as they are meant to be used in textmode.

Example:

```
Setcolor(BLUE);
outtext("Hello , Have a good day !");
```

Example:

Display the following text at location (10,10),

Text=” Display this text at position (10,10) use outtext() function”

Solution:

```
Moveto(10,10);
Outtext(“Display this text at position(10,10) use outtext() function”);
```

17.Outtextxy() Function

Outtextxy function displays the text or string at a specified point (x, y) on the screen.

Syntax: **void outtextxy(int x, int y, char *string);**

where, x, and y are coordinates of the point and, the third argument contains

the string to be displayed.

Examples: display "Have a good day !" at locations (200,150).

Solution 1: outtextxy(200,150,"Hello, Have a good day !");

Solution 2:

Moveto(200,150);

Outtext("Hello, Have a good day !");

HOMEWORK: which one of the following codes is not correct, and what is the correct output?

```
//*****Code 1 *****************
for(int i=0;i<10;i++){cleardevice();
 outtextxy(100,100,i); //First try
 outtextxy(100,100,"%d",i);
//Second try
 delay(1000);}
```

```
//*****Code 2 *****************
char str[3];
for(i=0;i<10;i++){    cleardevice();
    sprintf(str,"%d",i);
    outtextxy(100,100,str);
    delay(1000);}
```

18.Settextstyle() Function

This function is used to change how text appears. Using it we can modify the size of text, change the direction of text and change the font of the text.

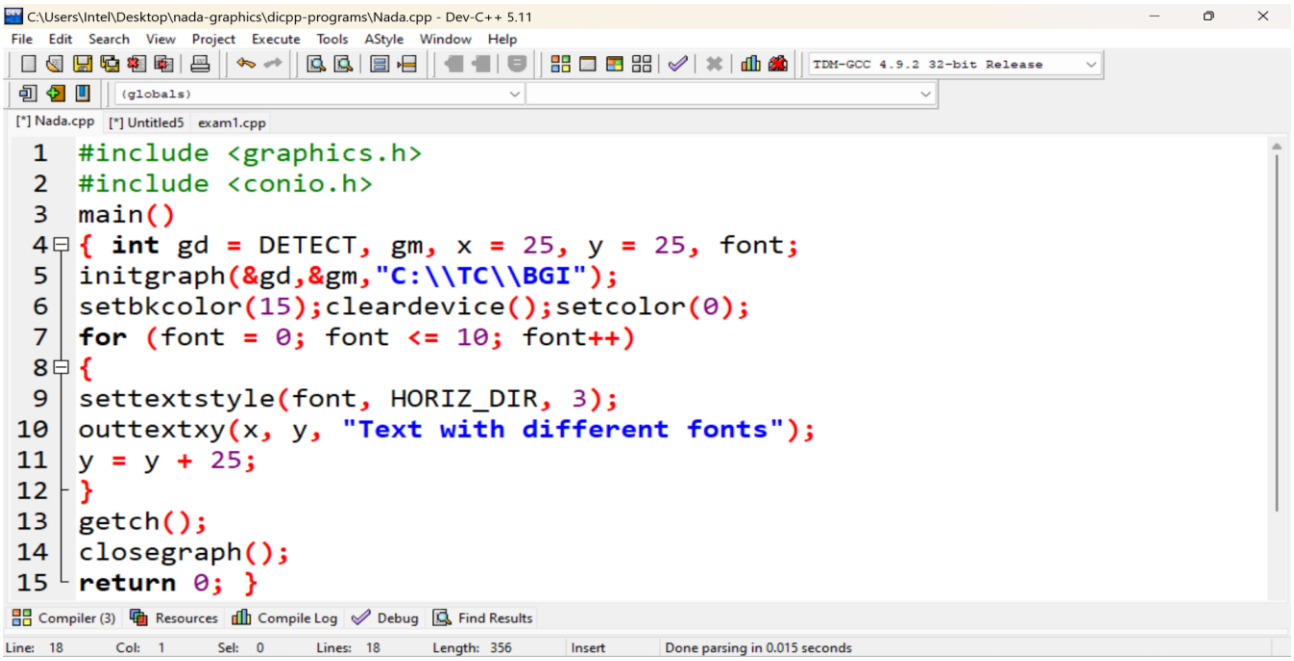
Syntax: **void settextstyle(int font, int direction, int font_size);**

where **font** argument specifies the font of the text, **Direction** can be HORIZ_DIR (Left to right) or VERT_DIR (Bottom to top).

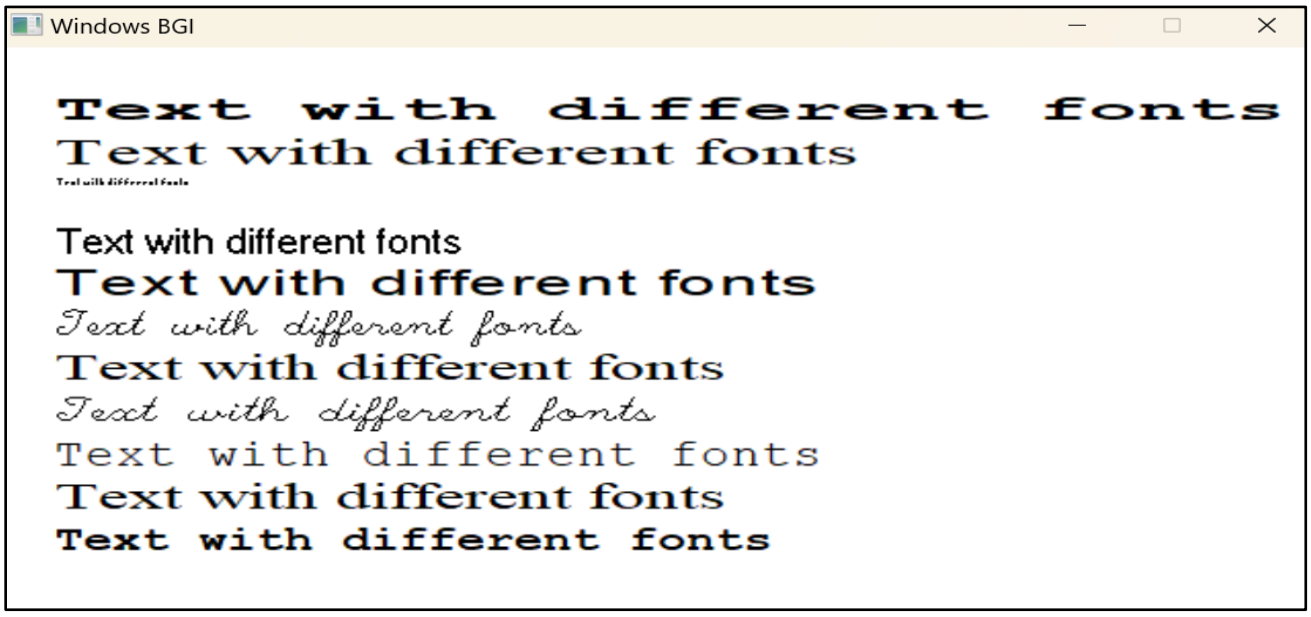
Font value takes one of the following

Fonts	INT Values
DEFAULT_FONT	0
TRIPLEX_FONT	1
SMALL_FONT	2
SANS_SERIF_FONT	3
GOTHIC_FONT	4
SCRIPT_FONT	5
SIMPLEX_FONT	6
TRIPLEX_SCR_FONT	7
COMPLEX_FONT	8
EUROPEAN_FONT	9
BOLD_FONT	10

Exercise: Write a C++ program to display "Text with different fonts", with horizontal direction, and different fonts and sizes at position(x,y).



THE OUTPUT:



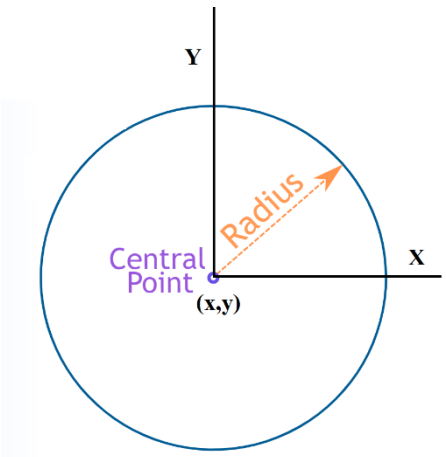
Draw 2D Shapes

Graphics.h library is used to include and facilitate graphical operations in program. C++ graphics using graphics.h functions can be used to draw different shapes, display text in different fonts, change colors and many more. Using functions of graphics.h you can make graphics programs, animations, projects and games. You can draw circles, arc, rectangles, bars and many other geometrical figures. You can change their colors using the available functions and fill them.

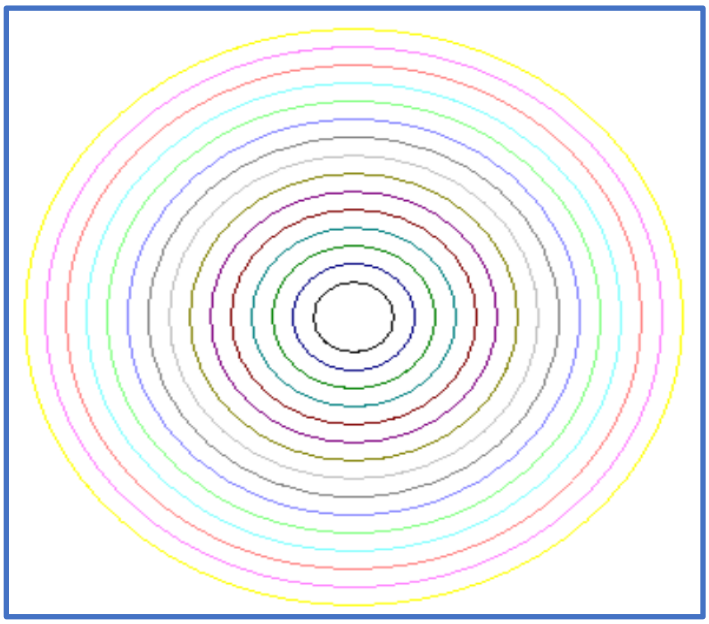
19.Circle() Function

Circle function in c++ draws a circle with center at (x, y) and given radius.**Syntax:** **circle(int x,int y,int radius);**

where,**(x, y)** is center of the circle, and '**radius**' is the Radius of the circle.



Examples: Write a c++ program to draw the following figure.



Solution:

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

[*] Nada.cpp [*] Untitled5

```
1  #include <graphics.h>
2  #include <conio.h>
3  main()
4  { int gd = DETECT, gm, x, y ,radius;
5  initgraph(&gd,&gm,"C:\\TC\\BGI");
6  setbkcolor(15);cleardevice();setcolor(0);
7  x = 250; y = 250; radius = 20;
8  for(int i=0;i<=15;i++)
9  {setcolor(i);
10 circle(x,y,radius);
11 radius+=10;
12 }
13 getch();
14 closegraph();
15 return 0; }
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 18 Col: 1 Sel: 0 Lines: 18 Length: 319 Insert Done parsing in 0 seconds

Exercises: What is the output of this code:

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

[*] Nada.cpp [*] Untitled5

```
5
6  #include <graphics.h>
7  int main()
8  {
9  int gdriver = DETECT,gmode;
10 initgraph(&gdriver,&gmode,"");
11 x = 250; y = 250; radius = 100;
12 setfillstyle(1,4);
13 circle(x,y,radius);
14 circle(x,y,radius-20);
15 floodfill(340,250,4);
16 outtextxy(x-35,y,"circle center");
17 getch();
18 closegraph();
19 return 0;}
20
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 23 Col: 1 Sel: 0 Lines: 25 Length: 365 Insert Done parsing in 0.016 seconds

Solution:

20.Rectangle() Function

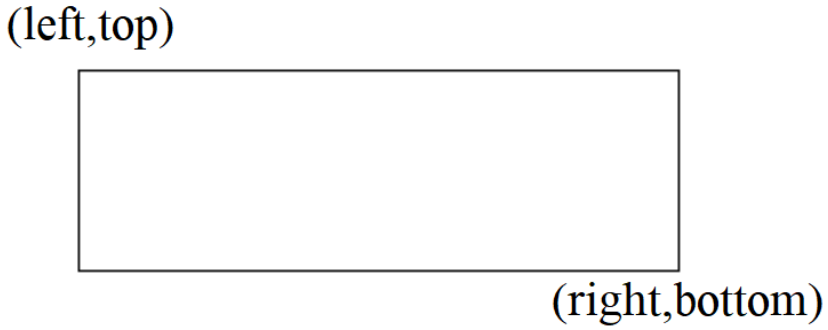
This function is used to draw a rectangle. Coordinates of the left top and right bottom corners are required to draw the rectangle. **left** specifies the X-coordinate of the top left corner, the **top** specifies the Y-coordinate of the top left corner, the **right** specifies the X-coordinate of the right bottom corner, **bottom** specifies the Y-coordinate of the right bottom corner.

Syntax: **rectangle(int left, int top, int right, int bottom);**

Examples:

int left = 150, top = 250, right = 450, bottom = 350;

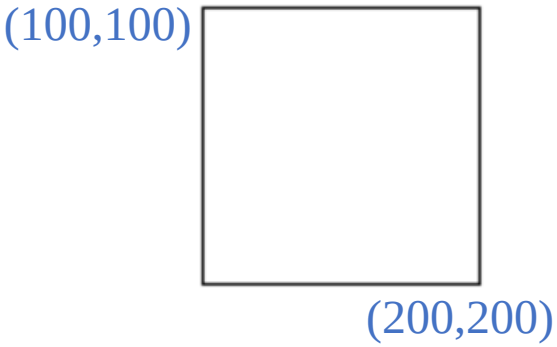
rectangle(left,top,right,bottom);



Exercise: use the rectangle() function to draw a square.
solution:

int left = 100, top = 100, right = 200, bottom = 200;

rectangle(left,top,right,bottom);



21.Bar() Function

The bar () function is used to draw a 2-dimensional, rectangular filled-in bar without a border, if we need to draw a border, we need to use the rectangle() function with the same coordinates as the bar() function. The default fill color is white, if we need to fill it with different color we use the function setfillstyle().

Syntax : **void bar(int left, int top, int right, int bottom);**

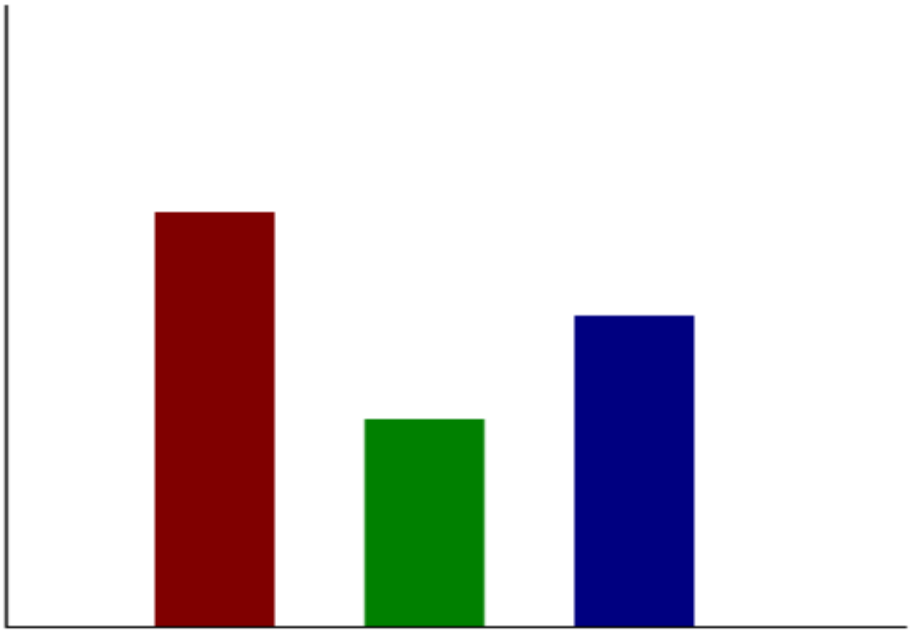
where,**left** specifies the X-coordinate of the top left corner,

top specifies the Y-coordinate of the top left corner,

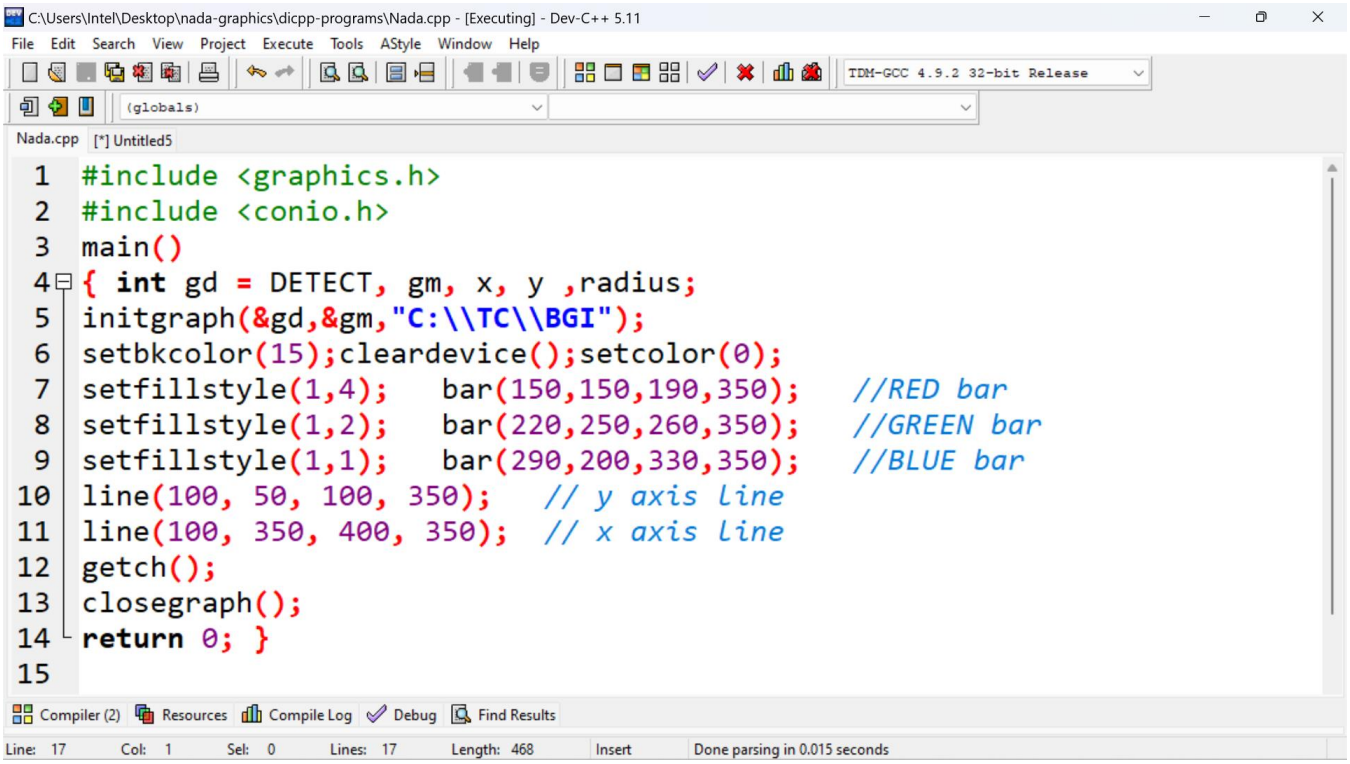
the **right** specifies the X-coordinate of the right bottom corner,

bottom specifies the Y-coordinate of the right bottom corner.

Examples: Write a c++ program to draw the following figure.



Solution:



22.Bar3d() Function

Bar3d function is used to draw a 2-dimensional, rectangular filled in bar . **Syntax:**

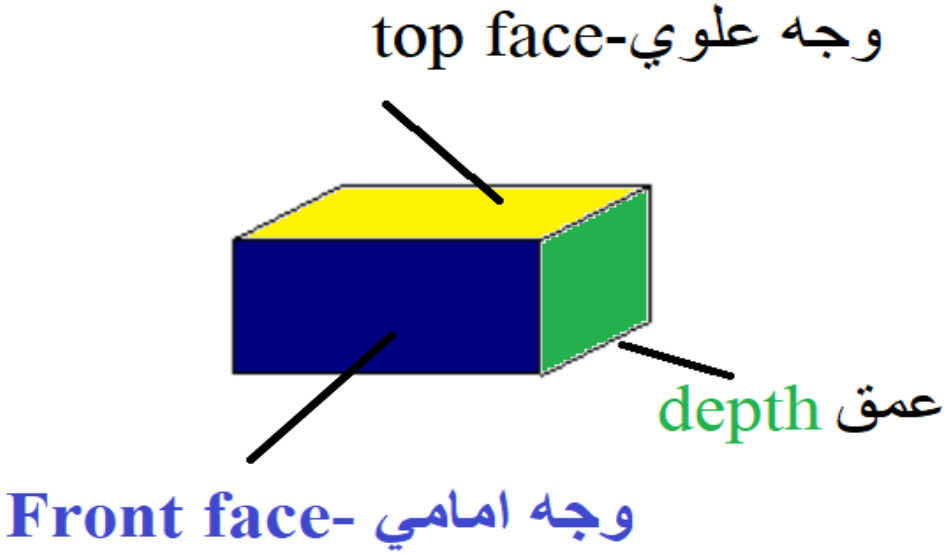
void bar3d(int left, int top, int right, int bottom, int depth, int topflag);

Coordinates of left top and right bottom corner of bar are required to draw the bar. **left** specifies the X-coordinate of top left corner, **top** specifies the Y-coordinate of top left corner, **right** specifies the X-coordinate of right bottom corner, **bottom** specifies the Y-coordinate of right bottom corner, **depth** specifies the depth of bar in pixels, **topflag** determines whether a 3 dimensional top is put on the bar or not

(if it's non-zero then it's put otherwise not). Current fill pattern and fill color is used to fill the bar. To change fill pattern and fill color use setfillstyle() function.

Example:

```
// Frontface - floodfill with BLUE color

setfillstyle(SOLID_FILL, BLUE); bar3d(100, 100, 200, 150, 35, 1);

//Top face - floodfill with YELLOW color

setfillstyle(SOLID_FILL, YELLOW); floodfill(150, 90, WHITE);

//Side face - floodfill with GREEN color

setfillstyle(SOLID_FILL, GREEN); floodfill( 210, 125, WHITE);
```



Exercise: What is the output of this code.

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

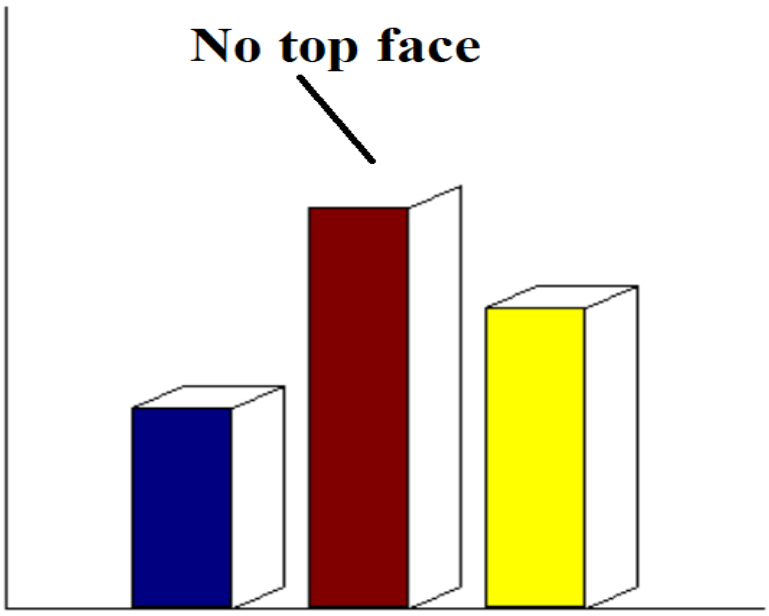
[*] Nada.cpp [*] Untitled5

```
1  #include <graphics.h>
2  #include <conio.h>
3  main()
4  { int gd = DETECT, gm, x, y ,radius;
5  initgraph(&gd,&gm,"C:\\TC\\BGI");
6   setbkcolor(15);cleardevice();setcolor(0);
7  int left, top, right, bottom,depth, topflag;
8  setfillstyle(1,1);
9  bar3d(left=150,top=250,right=190,bottom=350,depth=20,topflag=1);
10 setfillstyle(1,4);
11 bar3d(left=220,top=150,right=260,bottom=350,depth=20,topflag = 0);
12 setfillstyle(1,14);
13 bar3d(left=290,top=200,right=330,bottom=350,depth=20,topflag = 1);
14 // y axis line
15 line(100, 50, 100, 350);
16 // x axis line
17 line(100, 350, 400, 350);
18 getch();
19 closegraph();return 0; }
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 22 Col: 1 Sel: 0 Lines: 22 Length: 604 Insert Done parsing in 0.016 seconds

The output:



Example: draw multiple bar3d() with different style.

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11

File Edit Search View Project Execute Tools AStyle Window Help

TDM-GCC 4.9.2 32-bit Release

(globals)

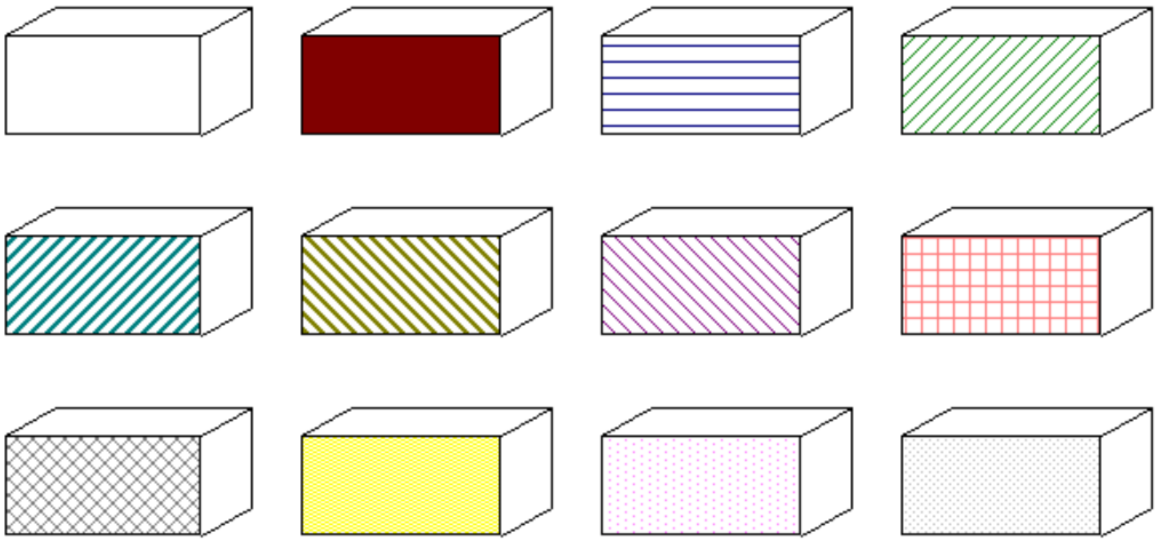
[*] Nada.cpp [*] Untitled5

```
1  #include <graphics.h>
2  #include <conio.h>
3  int main()
4  { int gd = DETECT, gm;
5  initgraph(&gd,&gm,"C:\\TC\\BGI");
6  setbkcolor(15);cleardevice();setcolor(0);
7  setfillstyle(EMPTY_FILL,YELLOW);bar3d(2,150,100,200,25,1);
8  setfillstyle(SOLID_FILL,RED);bar3d(150,150,250,200,25,1);
9  setfillstyle(LINE_FILL,BLUE);bar3d(300,150,400,200,25,1);
10 setfillstyle(LTSLASH_FILL,GREEN);bar3d(450,150,550,200,25,1);
11 setfillstyle(SLASH_FILL,CYAN); bar3d(2,250,100,300,25,1);
12 setfillstyle(BKSLASH_FILL,BROWN);bar3d(150,250,250,300,25,1);
13 setfillstyle(LTBKSLASH_FILL,MAGENTA);bar3d(300,250,400,300,25,1);
14 setfillstyle(HATCH_FILL,LIGHTRED);bar3d(450,250,550,300,25,1);
15 setfillstyle(XHATCH_FILL,DARKGRAY);bar3d(2,350,100,400,25,1);
16 setfillstyle(INTERLEAVE_FILL,YELLOW);bar3d(150,350,250,400,25,1);
17 setfillstyle(WIDE_DOT_FILL,LIGHTMAGENTA);bar3d(300,350,400,400,25,1);
18 setfillstyle(CLOSE_DOT_FILL,LIGHTGRAY);bar3d(450,350,550,400,25,1);
19 getch();closegraph();return 0; }
```

Compiler (2) Resources Compile Log Debug Find Results

Line: 22 Col: 1 Sel: 0 Lines: 22 Length: 972 Insert Done parsing in 0 seconds

The output:



23.Arc() Function

The arc() function draws an arc with center at (x, y) and given radius. start_angle is the starting point of angle and end_angle is the ending point of the angle. The value of the angle can vary from (0 to 360) degree.

Syntax:

```
void arc(int x, int y, int start_angle,int end_angle, int radius);
```

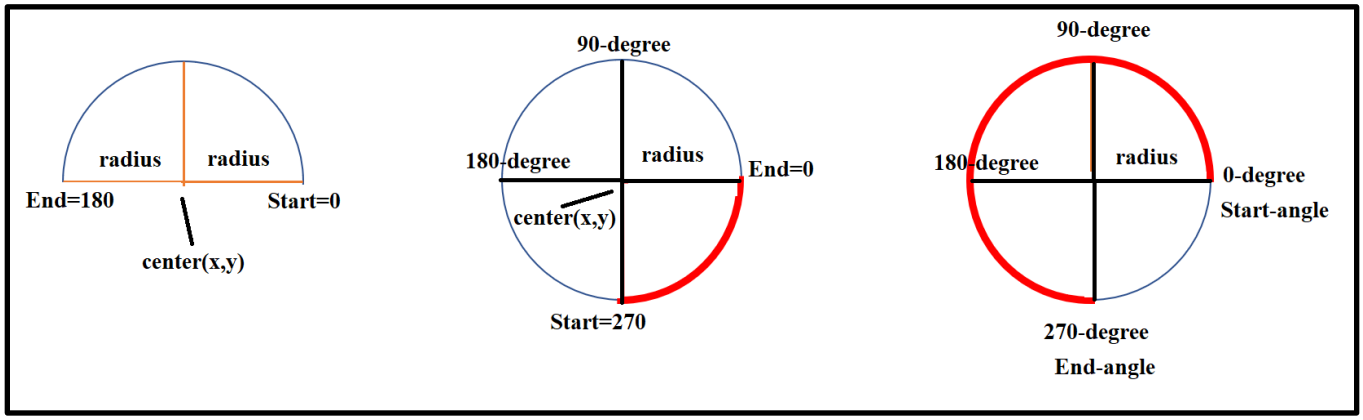
where:

(x, y): is the center of the arc.

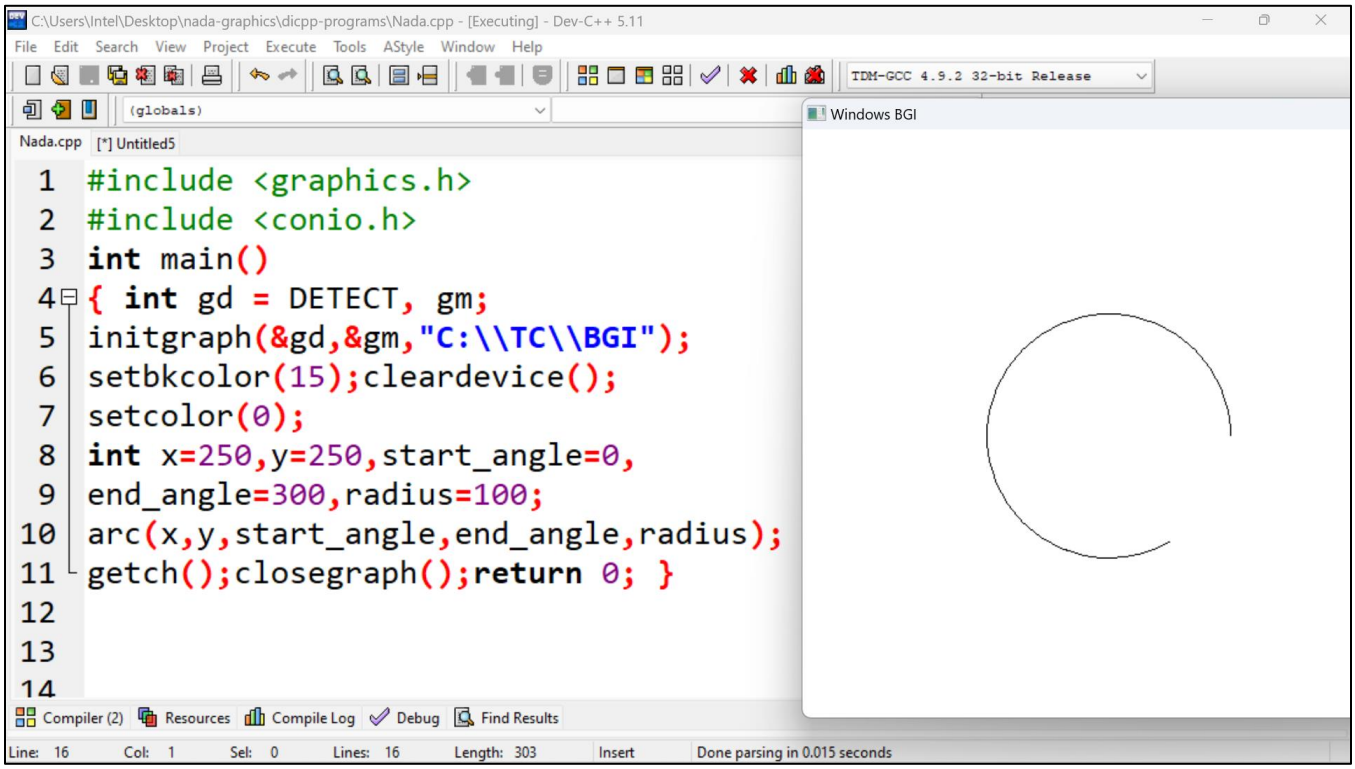
start_angle: is the starting angle and

end_angle: is the ending angle.

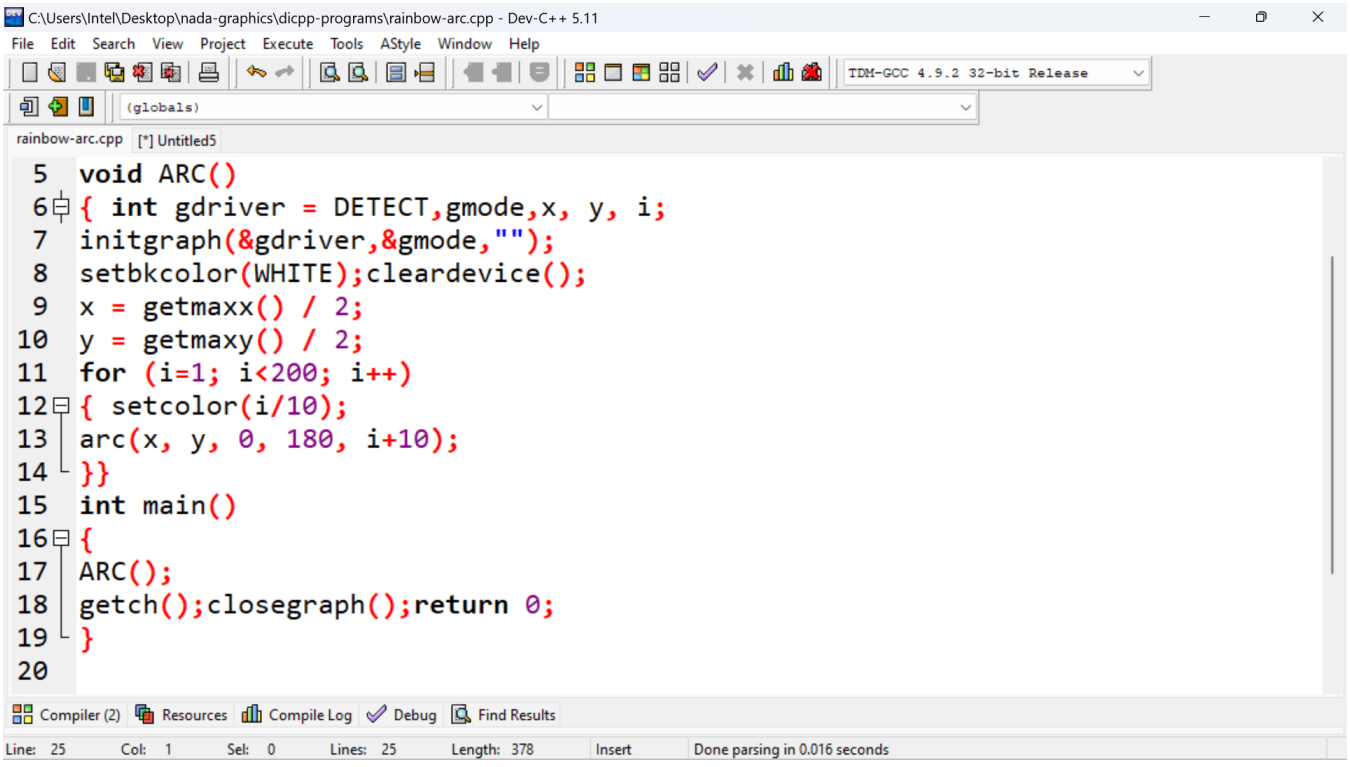
Radius: is the Radius of the arc.



Example:



HOMEWORK: what is the output of the following code.



24.Ellipse() Function

Ellipse is used to draw an ellipse on graphics screen.

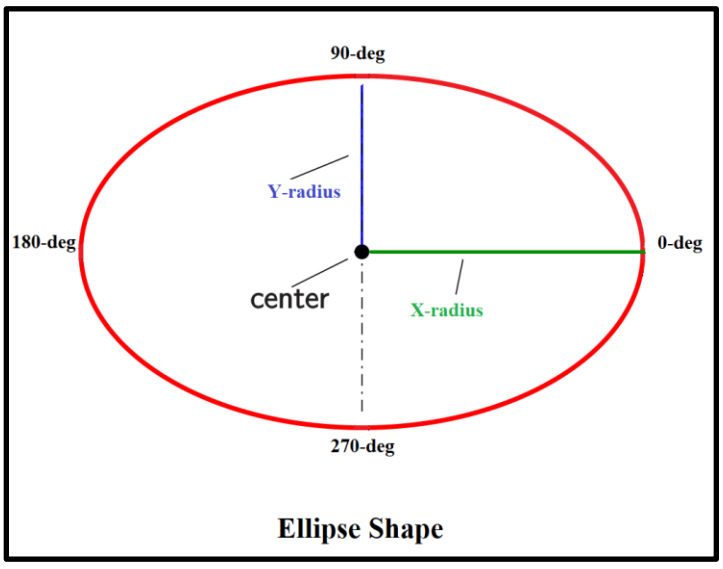
Suntax:

void ellipse(int x, int y, int start_angle, int end_angle, int x_radius, int y_radius)

In this function x, y is the center location of the ellipse.

x_radius and y_radius decide the radius of form x and y.

start_angle is the starting point of angle and end_angle is the ending point of angle. The value of angle can vary from 0 to 360 degree.

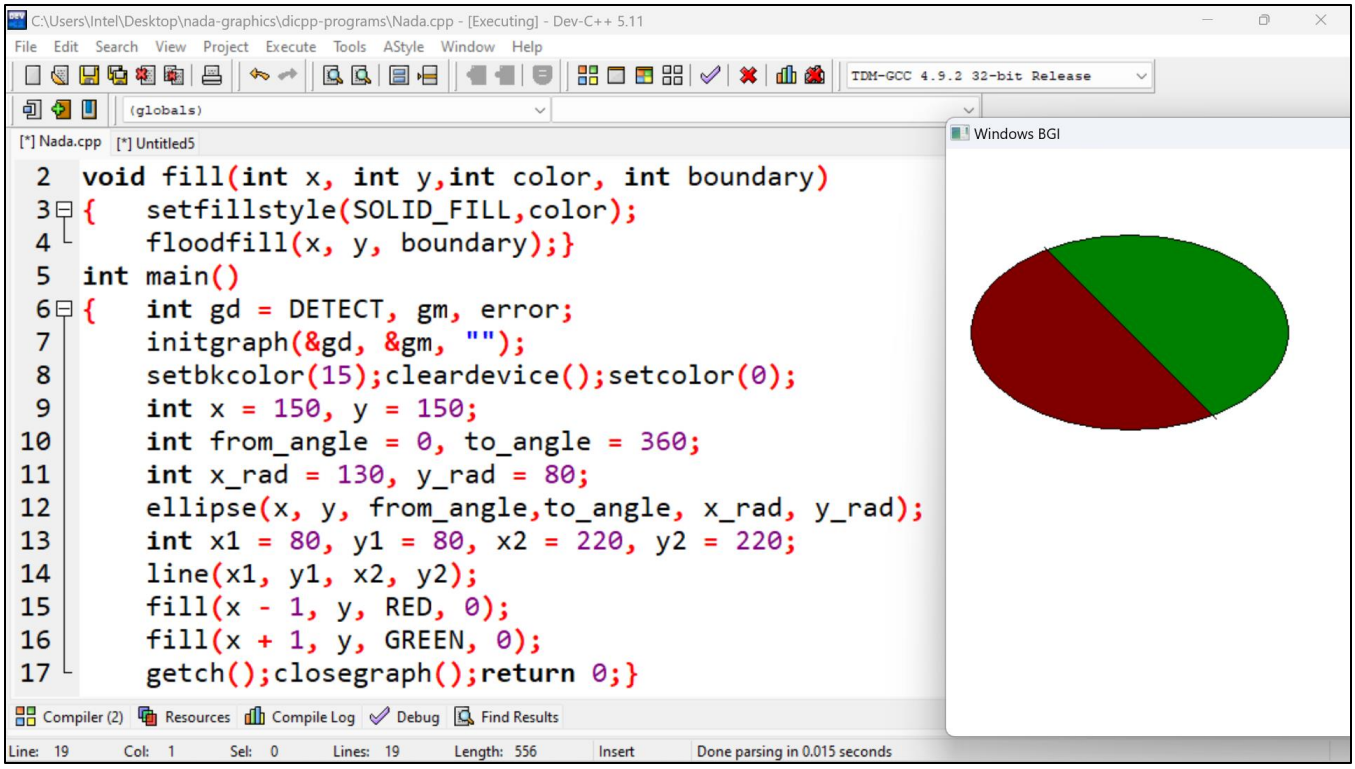


Ellipse Shape

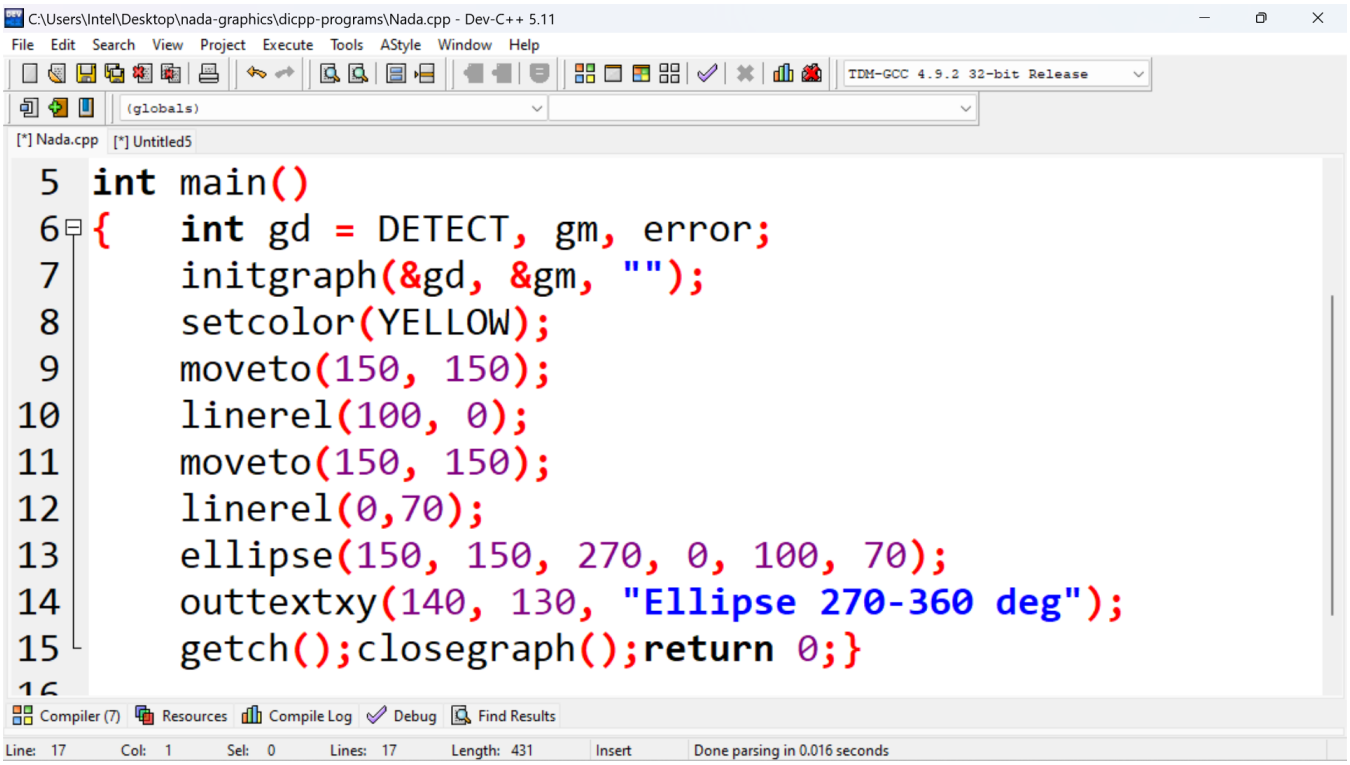
Exercise:

Draw an ellipse divided by straight line into two colored parts in C++:

Sulution:



Homework what is the output of this code?



25.Fillellipse() Function

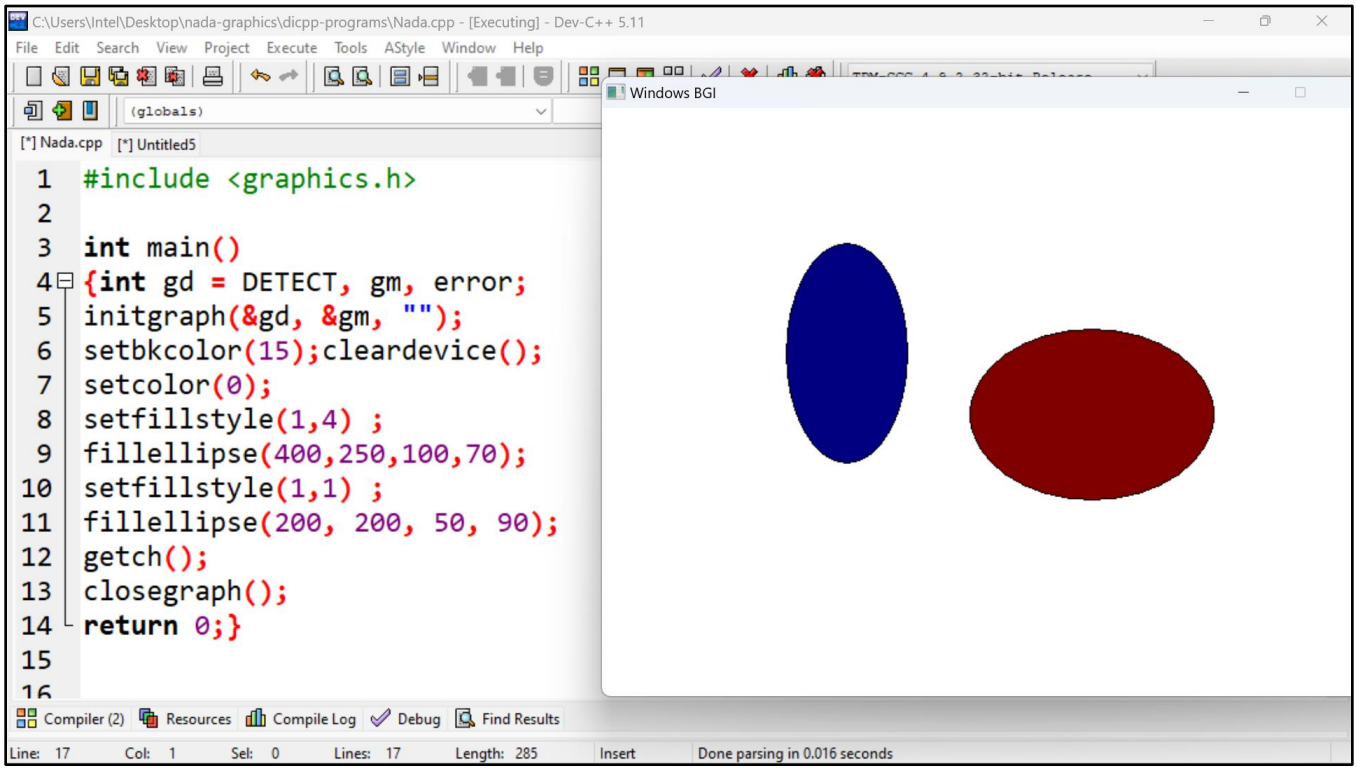
fillellipse() function draws and fills an ellipse with center at (x, y) and (x_radius, y_radius) as x and y radius of ellipse.

Syntax :

void fillellipse(int x, int y, int x_radius,int y_radius);

where,(x, y) is center of the ellipse.(x_radius, y_radius) are x and y radius of ellipse.

Example:



26.Pieslice() Function

pieslice() draws and fills a pie slice with center at (x, y) and given radius r. The slice travels from start-angle to end-angle which are starting and ending angles for the pie slice. The angles for pie-slice are given in degrees and are measured counterclockwise.

Syntax:

void pieslice(int x, int y, int start-angle,int end-angle, int r);

where,(x, y) is center of the circle.

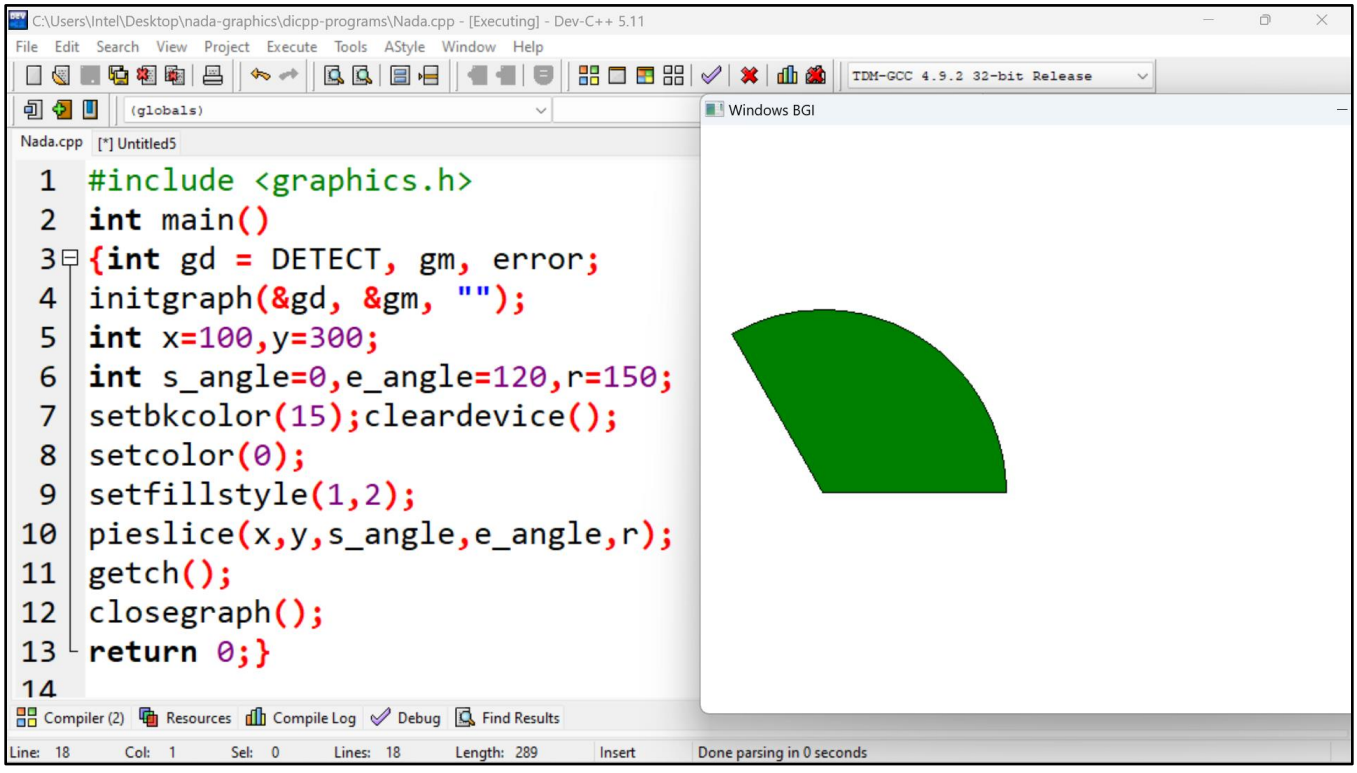
r is the radius of the circle.

Start-angle and end-angle are the starting and ending angles respectively. If

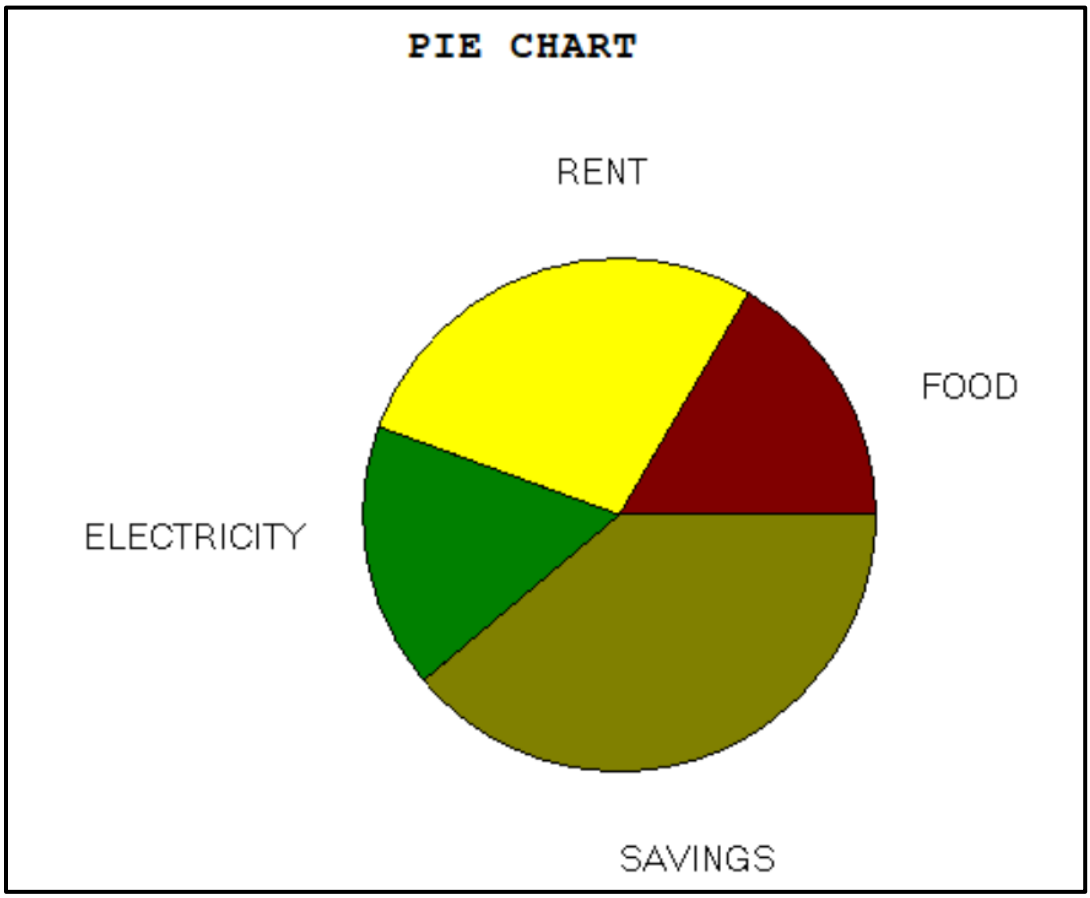
Examples:

Input: x = 300, y = 300, start_angle = 0, end_angle = 120, r = 150.

Output:



Exercise: Write a c++ program to draw the following figure using Pieslice() function.



Solution:

```
#include<graphics.h>
#include<conio.h>
int main() {
  int gd = DETECT, gm, x, y;
  initgraph(&gd, &gm, "C:\\TC\\BGI");
  setbkcolor(15);cleardevice(); setcolor(0);
```

```
  settextstyle(BOLD_FONT,HORIZ_DIR,2);
  outtextxy(220,10,"PIE CHART");
  // Setting cordinate of center of circle
  x = getmaxx()/2;    y = getmaxy()/2;
  settextstyle(SANS_SERIF_FONT,HORIZ_DIR,1);
  setfillstyle(SOLID_FILL, RED);
  pieslice(x, y, 0, 60, 120);
  outtextxy(x + 140, y - 70, "FOOD");
//************************************
  setfillstyle(SOLID_FILL, YELLOW);
  pieslice(x, y, 60, 160, 120);
  outtextxy(x - 30, y - 170, "RENT");
//************************************
  setfillstyle(SOLID_FILL, GREEN);
  pieslice(x, y, 160, 220, 120);
  outtextxy(x - 250, y, "ELECTRICITY");
//************************************
  setfillstyle(SOLID_FILL, BROWN);
  pieslice(x, y, 220, 360, 120);
  outtextxy(x, y + 150, "SAVINGS");
   getch();   closegraph();   return 0;}
```

27.Sector() Function

sector() function draws and fills an **elliptical pie slice** with (x, y) as center, (start_angle, end_angle) as starting and ending angle and (x_radius, y_radius) as x and y radius of sector.

Syntax :

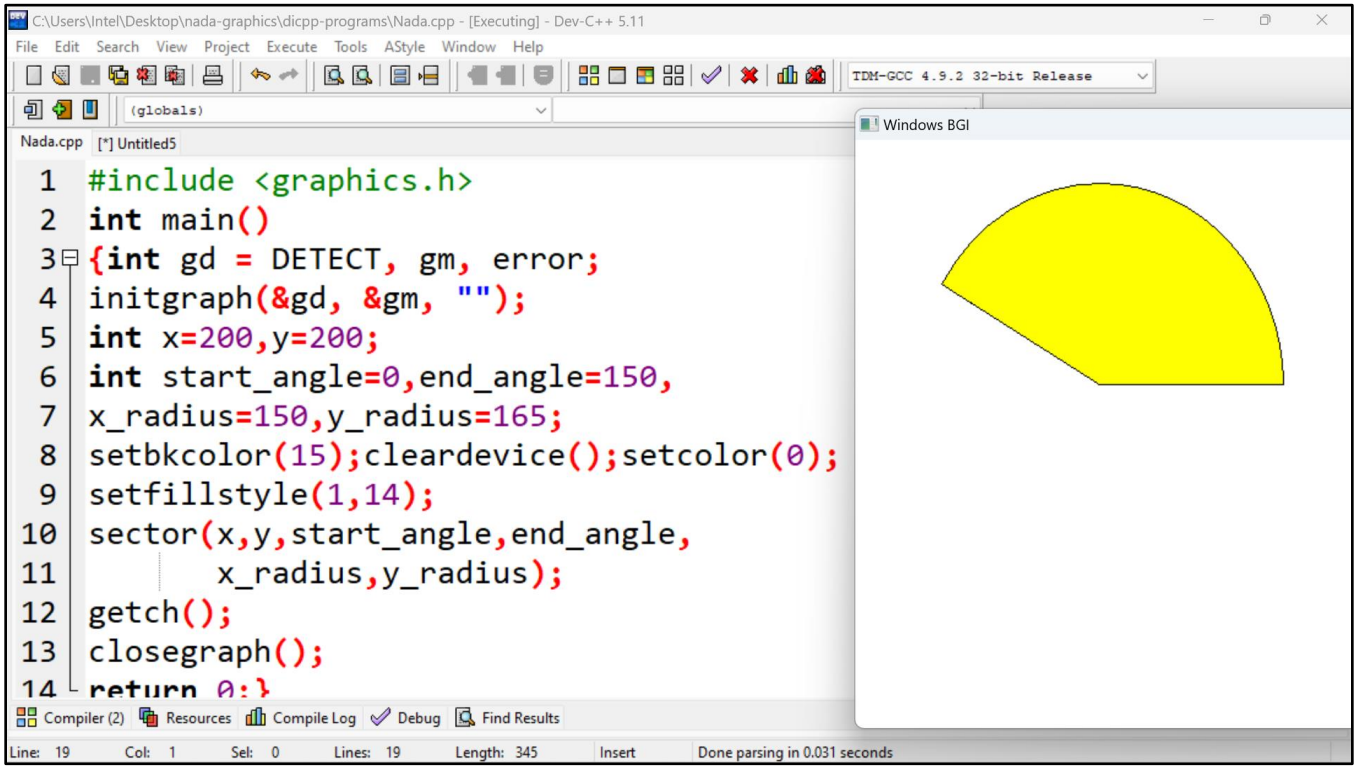
```
void sector(int x, int y, int s_angle,int e_angle, int x_radius, int y_radius);
```

where, (x, y) is center of the sector.

(s_angle, e_angle) are starting and ending angles.

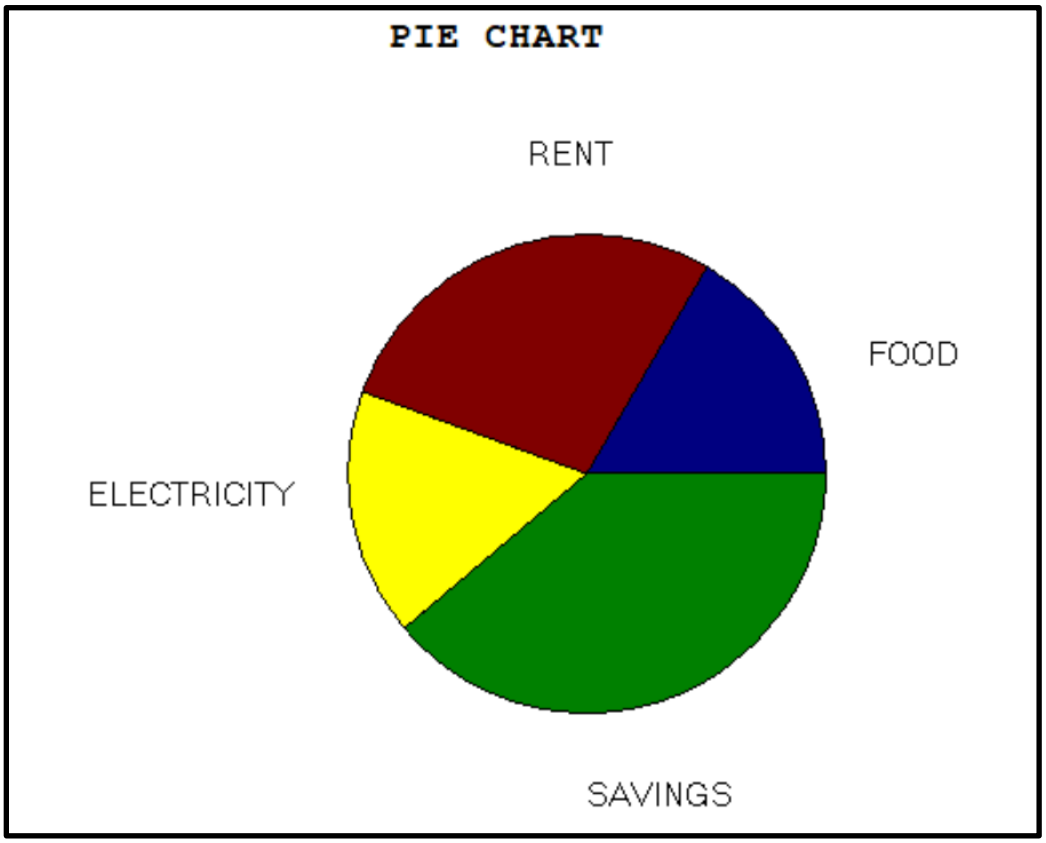
(x_radius, y_radius) are x and y radius of sector.

Examples:



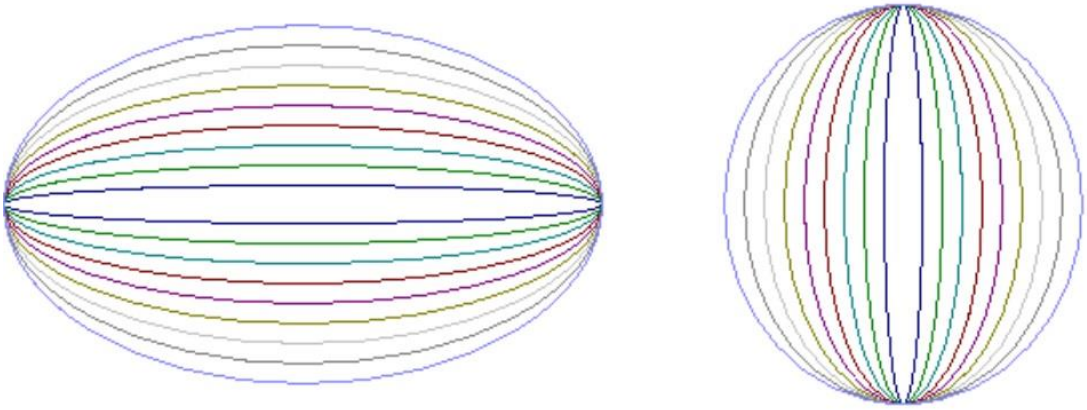
HOMEWORK:

Write c++ program to output the following figure using Sector() function.



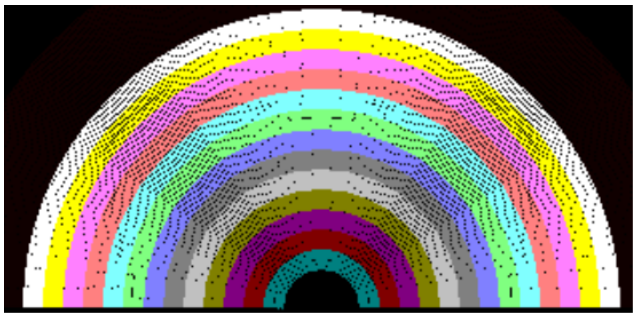
HOMEWORK:

Write c++ program to output the following figure using ellipse() function.



HOMEWORK:

Write c++ program to output the following figure using arc() function.



28.Drawpoly() Function

Drawpoly function is used to draw polygons i.e., triangle, rectangle, pentagon, hexagon etc.

Syntax: **void drawpoly(int num, int *polypoints);**

num indicates (n+1) number of points where n is the number of vertices in a polygon, **polypoints** points to a sequence of (n*2) integers. Each pair of integers gives x and y coordinates of a point on the polygon. We specify (n+1) points as first point coordinates should be equal to $(n+1)^{th}$ to draw a complete figure.

To understand more clearly we will draw a triangle using drawpoly, consider for example, the array:-
int points[] = { 320, 150, 420, 300, 250, 300, 320, 150};

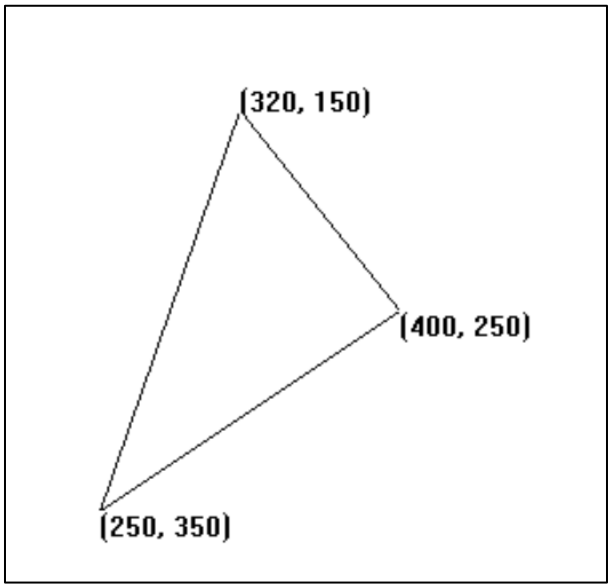
points array contains coordinates of triangle which are (320, 150), (420, 300) and (250, 300). Note that last point (320, 150) in array is same as first.

Example1: Drawing a triangle using drawpoly() function. triangle heads are :(320, 150), (400, 250), (250, 350).

Solution

```
// coordinates of polygon
   int array[] = {320, 150, 400, 250,250, 350, 320, 150};
   drawpoly(4, array);
```

The output:



Exercise:

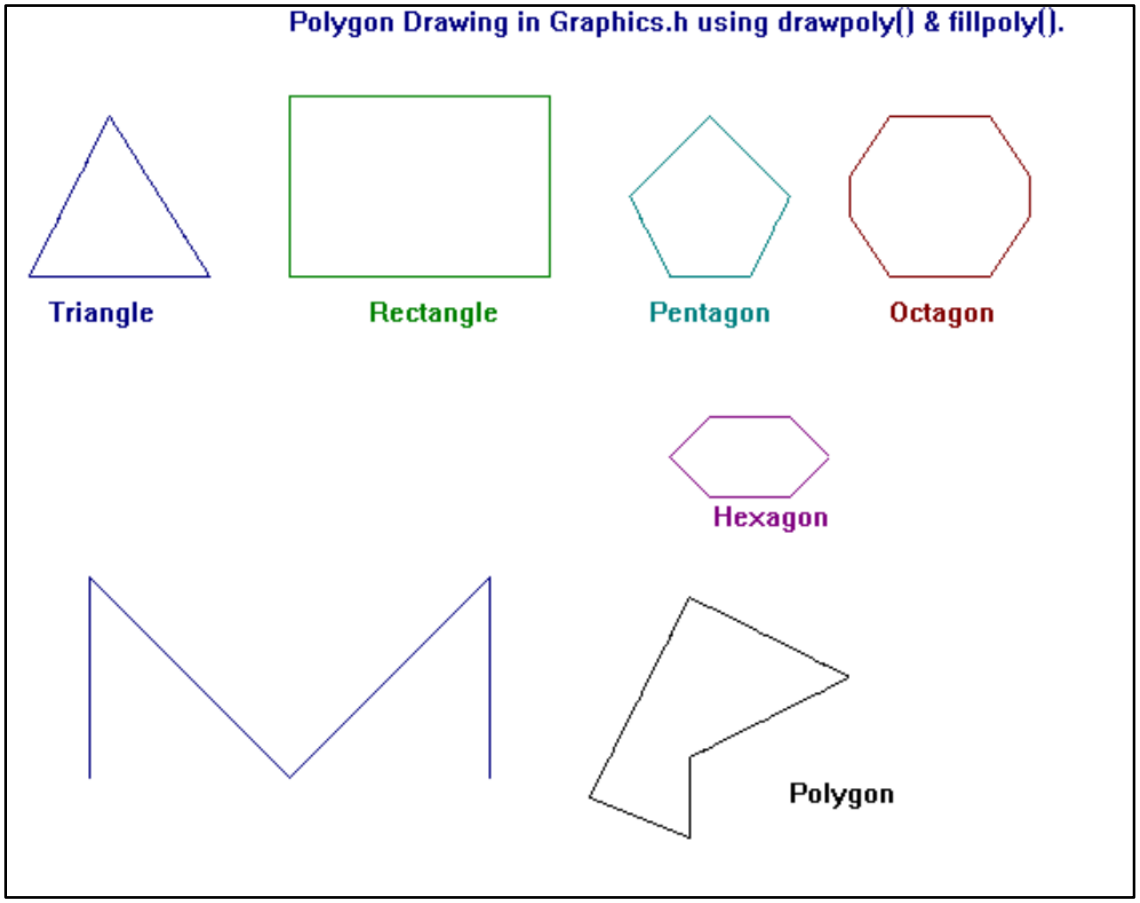
What is the output of this code?

```
#include<graphics.h>

int main(){

int gd=DETECT, gm;

int triangle[8]={20,150, 60,70, 110,150, 20,150};

int rect[10]={150,60, 280,60, 280,150, 150,150, 150,60};
```

```
int pentagon[12]={340,150, 320,110, 360,70, 400,110, 380,150, 340,150};

int hexagon[14] ={ 360,260, 340,240, 360,220, 400,220, 420,240, 400,260, 360,260};

int octagon[18]={450,150, 430,120, 430,100, 450,70, 500,70, 520,100, 520,120, 500,150,
        450,150};

int poly1[10]={50,400, 50,300, 150,400, 250,300, 250,400 };

int poly2[12]={350,430, 350,390, 430,350, 350,310, 300,410, 350,430 };

initgraph(&gd, &gm,NULL);

setbkcolor(15);cleardevice();

setcolor(1);

outtextxy(150,15, "Polygon Drawing in Graphics.h using drawpoly() & fillpoly().");

drawpoly(4,triangle);

outtextxy(30,160, "Triangle");

setcolor(2);

drawpoly(5,rect);

outtextxy(190, 160, "Rectangle");

setcolor(3);

drawpoly(6,pentagon);

outtextxy(330, 160, "Pentagon");

setcolor(4);

drawpoly(9,octagon);

outtextxy(450, 160, "Octagon");

setcolor(5);
```

```
fillpoly(7,hexagon);
outtextxy(362, 262, "Hexagon");
setcolor(1);
drawpoly(5,poly1);
setcolor(0);
drawpoly(6,poly2);
outtextxy(400, 400, "Polygon");
getch();closegraph();return 0;}
```

Theoutput:



29.Fillpoly() Function

Fillpoly() function is used to **draw and fill** a polygon i.e. triangle, rectangle, pentagon, hexagon etc. It require same arguments as drawpoly().

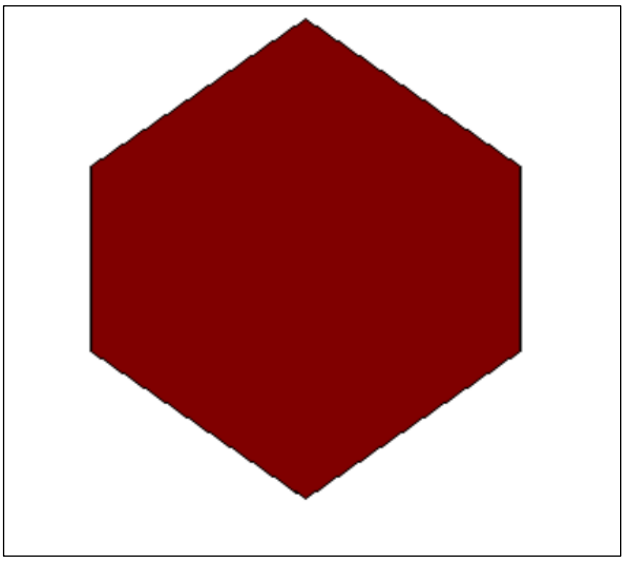
Syntax: **void fillpoly(int number, int *polypoints);**

The declaration of fillpoly() contains two arguments: **number** indicates (n + 1) number of points where n is the number of vertices in a polygon. The second argument, i.e., **polypoints** points to a sequence of (n * 2) integers. Each pair of integers gives x and y coordinates of a point on the polygon. We specify (n + 1) points because first point coordinates should be equal to (n + 1)th to draw a complete figure.

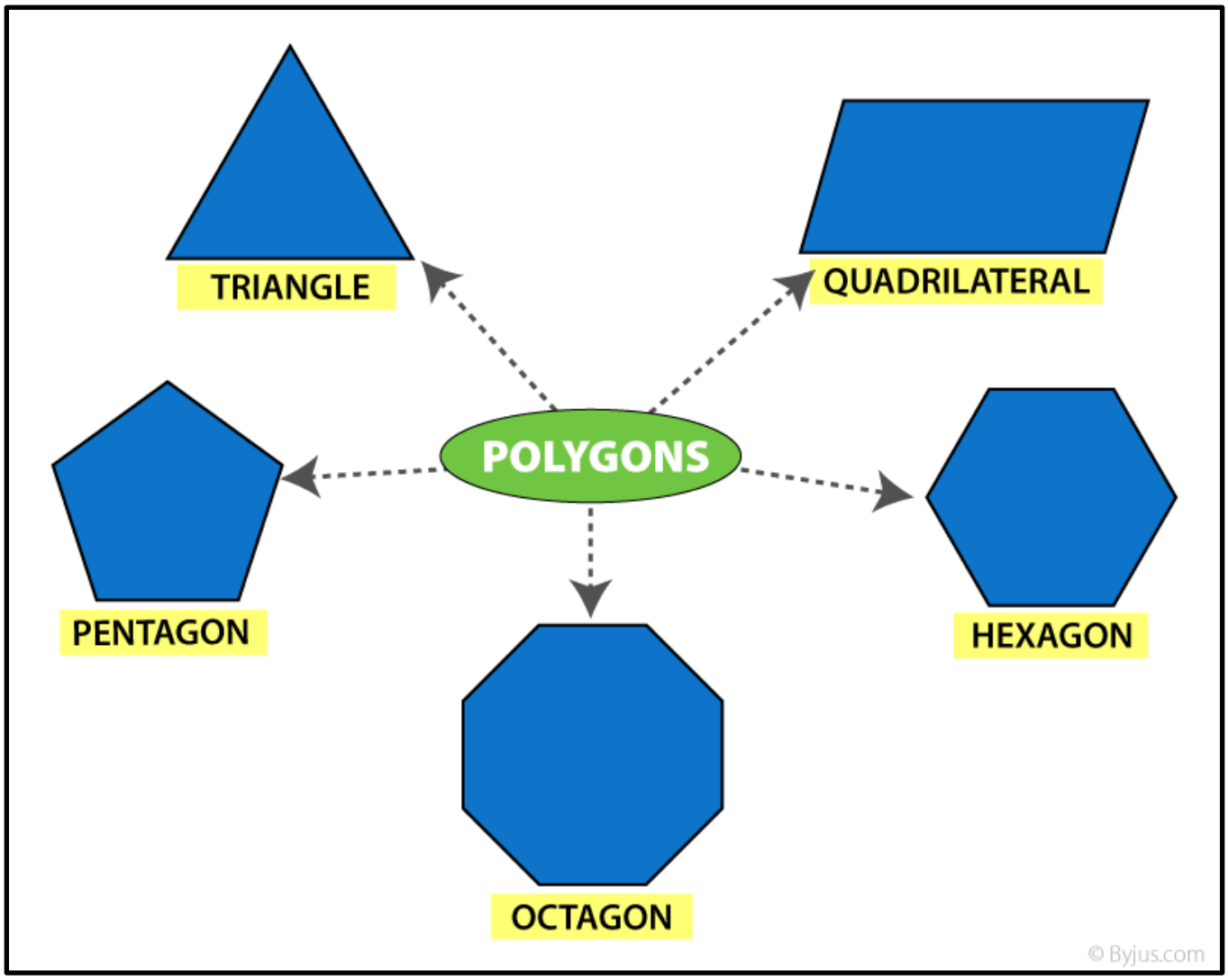
Example: **int arr[] = {220, 120, 130, 200, 130, 300,220, 380,310, 300,310, 200,220, 120};**

setfillstyle(1,4);

fillpoly(7, arr);



HOMEWORK:Draw the following figure using poly()or fillpoly()



Line generation Algorithm in Computer Graphics

There are four cases to draw line:

First: Horizontal Line where **x1≠x2** and **y1=y2** .

Example: Line (10,50,70,50);

Second:Vertical Line where **x1=x2** and **y1≠y2 .**

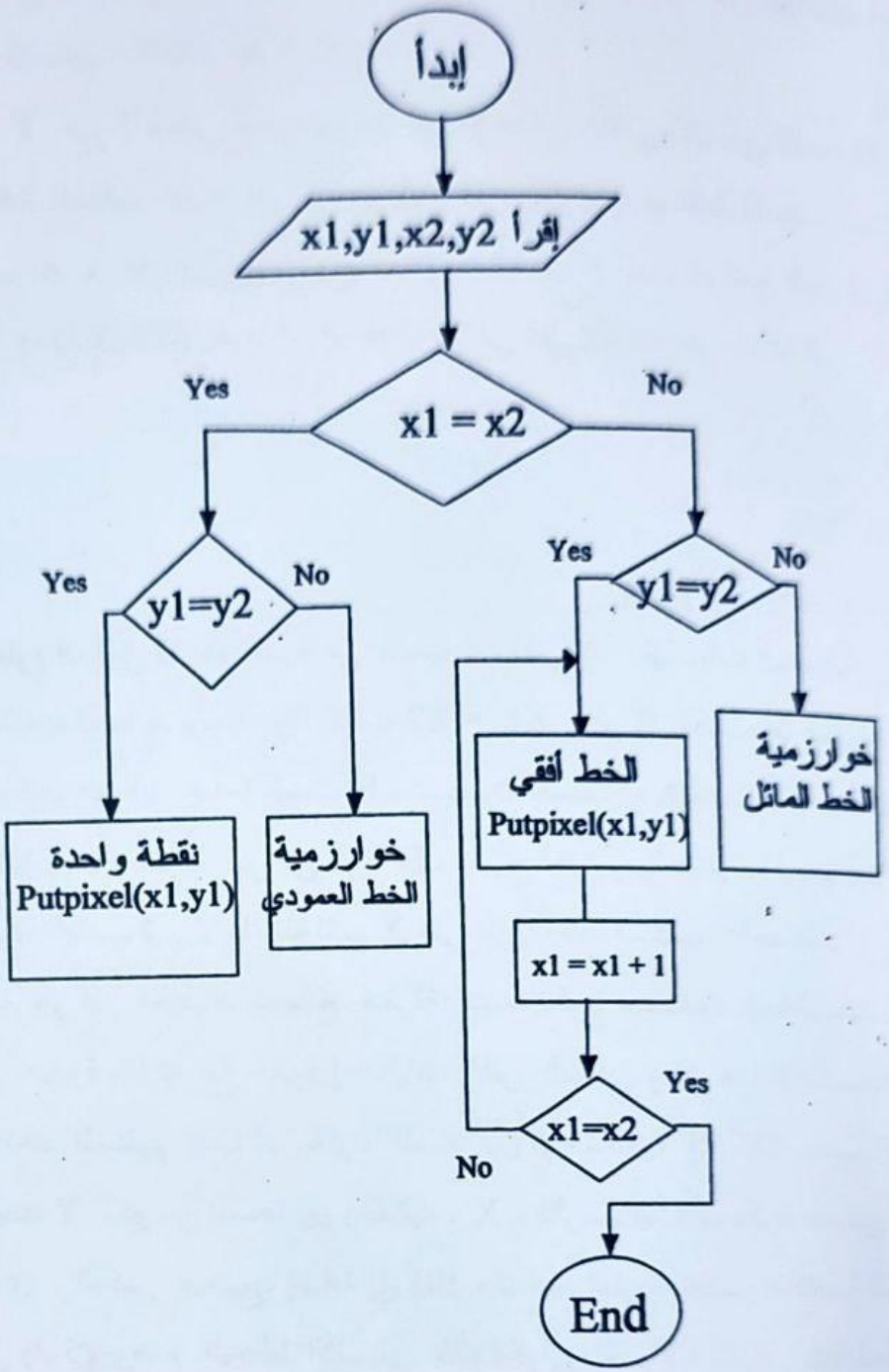
Example: Line (10,30,10,60);

Third: Diagonal Line where **x1≠x2** and **y1≠y2 .**

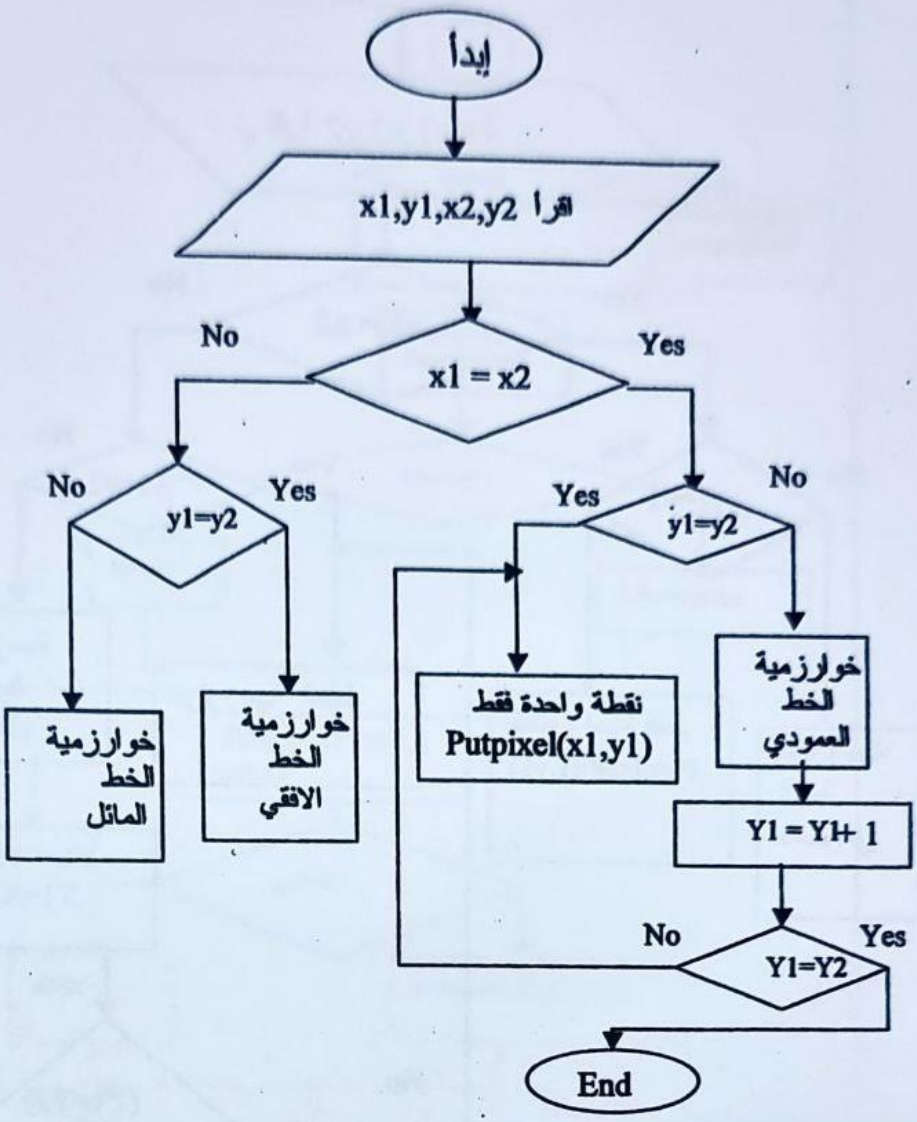
Example: Line (10,20,30,40)

Forth: ×Only One Pixel where **x1=x2** and **y1=y2 .**

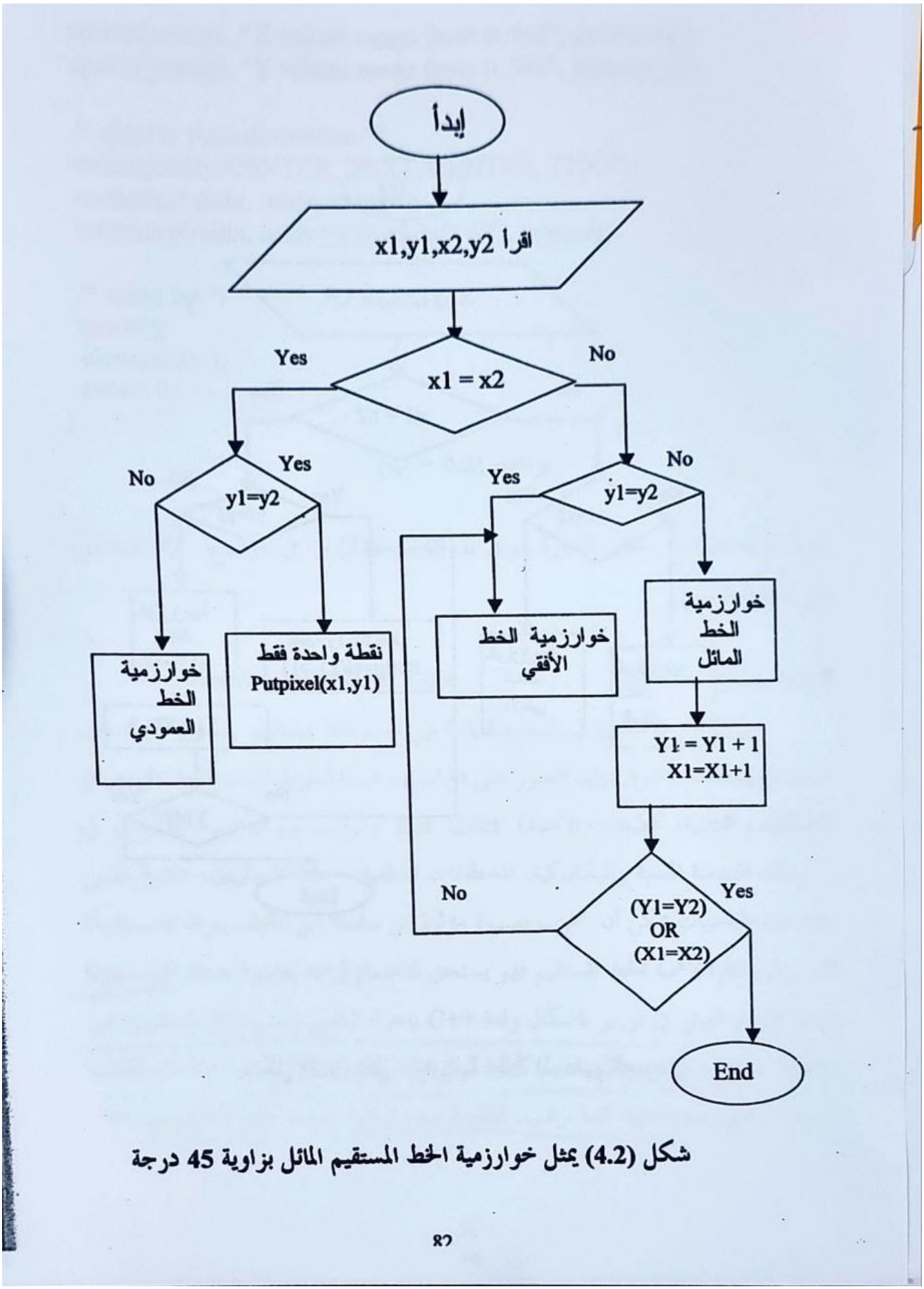
Example: Line (50,50,50,50);



شكل (2.2) يمثل خوارزمية الخط المستقيم الأفقي



شكل (3.2) يمثل خوارزمية الخط المستقيم العمودي



شكل (4.2) يمثل خوارزمية الخط المستقيم المائل بزاوية 45 درجة

Digital Differential Analyzer (DDA) Line generation Algorithm in Computer Graphics

In any 2-Dimensional plane, if we connect two points (x1, y1) and (x2, y2), we get a line segment. But in the case of computer graphics, we cannot directly join any two coordinate points, for that, we should calculate intermediate points' coordinates and put a pixel for each intermediate point, of the desired color with the help of functions like **putpixel(x, y, color)** , where (x,y) is our coordinate and its color.

Examples:

Input: *For line segment between (2, 2) and (6, 6) :*
Output: *we need (3, 3) (4, 4) and (5, 5) as our intermediate points.*

Input: *For line segment between (0, 2) and (0, 6) :*
Output: *we need (0, 3) (0, 4) and (0, 5) as our intermediate points.*

For using graphics functions, our system output screen is treated as a coordinate system where the coordinate of the top-left corner is (0, 0) and as we move down our y-ordinate increases, and as we move right our x-ordinate increases for any point (x, y). Now, for generating any line segment we need intermediate points and for calculating them we

can use a basic algorithm called DDA(Digital differential analyzer) line generating algorithm.

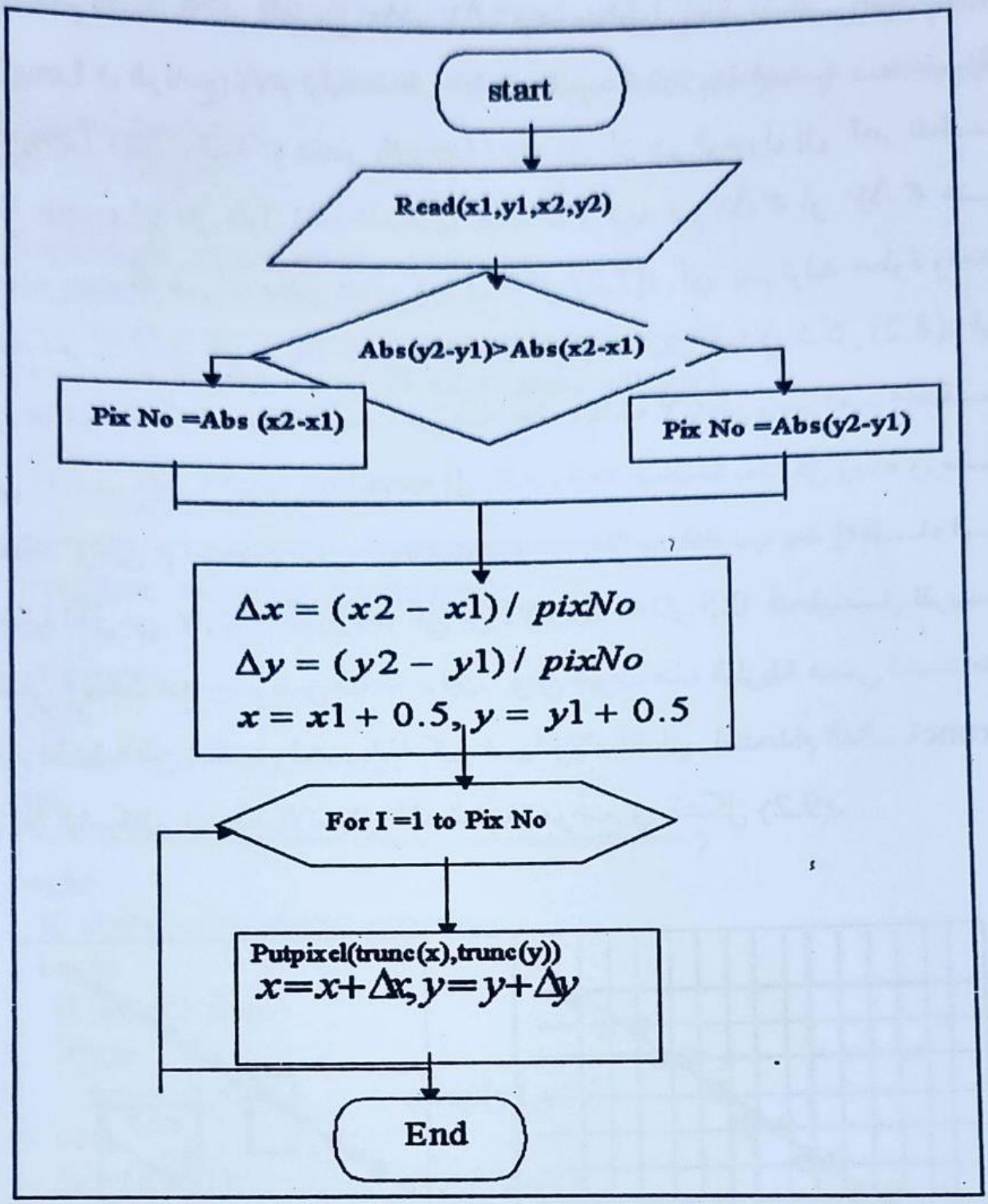
DDA Algorithm:

Consider one point of the line as (X1, Y1) and the second point of the line as (X2, Y2).

```
// calculate dx , dy

dx = X2 – X1;
dy = Y2 – Y1;

// Depending upon absolute value of dx & dy
// choose number of steps to put pixel as

// if abs(dx) > abs(dy) then steps= abs(dx) else steps= abs(dy)
steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

// calculate increment in x & y for each steps

Xinc = dx / (float) steps;
Yinc = dy / (float) steps;

// Put pixel for each step

X = X1;
Y = Y1;

for (int i = 0; i <= steps; i++)
{
   putpixel (round(X),round(Y),WHITE);
   X += Xinc;
   Y += Yinc;
}
```

```
//Implement Digital Differential Analyzer (DDA) Algorithm in C++
#include <bits/stdc++.h>
#include <graphics.h>
#include <conio.h>
// function for rounding off the pixels
int round (float n)
{  if (n - (int)n < 0.5)
   return (int)n;
   return (int)(n + 1);
}
// DDA Function for line generation
void DDALine(int x1, int y1, int x2, int y2)
{      // Calculate dx and dy
       int dx = x2 – x1;
       int dy = y2 – y1;
       int step;
       // If dx > dy we will take step as dx
       // else we will take step as dy to draw the complete line
       if (abs(dx) > abs(dy))
              step = abs(dx);
       else
```

```
        step = abs(dy);
    // Calculate x-increment and y-increment for each step
    float x_incr = (float)dx / step;
    float y_incr = (float)dy / step;
    // Take the initial points as x and y
    float x = x1;
    float y = y1;
    for (int i = 0; i < step; i++) {
        putpixel(round(x), round(y), WHITE);
        cout << round(x) << " " << round(y) << "\n";
        x += x_incr;
        y += y_incr;        }
}
int main(){
int gd = DETECT, gm;
initgraph(&gd,&gm,"C:\\TC\\BGI");
int x1 = 200, y1 = 100, x2 = 400, y2 = 100;
DDALine(x1, y1, x2, y2); // Function call
getch();
closegraph();
return 0; }
```

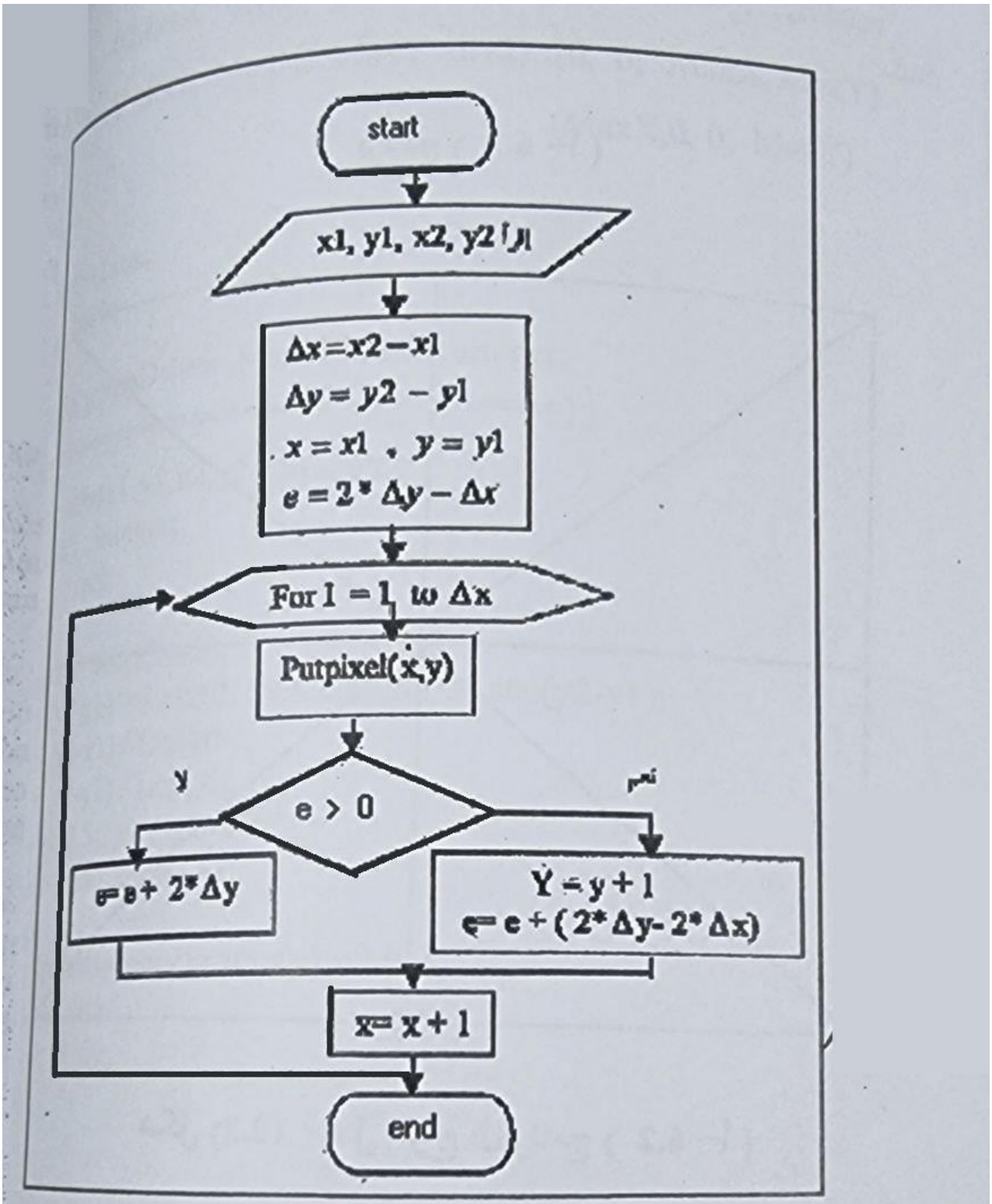


شكل (9.2) يوضح المخطط الإنسيابي لطريقة SDDA لرسم الخط المستقيم

Digital Differential Integer Analyzer (DDIA) Bresenham's Line Generation Algorithm

```
// C++ program for DDIA line generation
#include <bits/stdc++.h>
#include <graphics.h>
#include <conio.h>
using namespace std;
void bresenhamdrawline(int x0, int y0, int x1, int y1)
{ int dx, dy, e, x, y;
 dx=x1-x0;
 dy=y1-y0;
 x=x0;
 y=y0;
 e=2*dy-dx;
 for(int i=1;i<dx;i++) {
   putpixel(x,y,15);
 if(e>=0)
 {   y=y+1;
```

```
    e=e+2*dy-2*dx;
  }
  else
  {  e=e+2*dy;  }
 x=x+1;
}}
 int main(){
int gdriver=DETECT, gmode,x0, y0, x1, y1;
cout<<"Enter co-ordinates of first point:x0,y0 ";
cin>>x0>>y0;
cout<<"Enter co-ordinates of second point:x1,y1 ";
cin>>x1>>y1;
initgraph(&gdriver, &gmode, "c:\\turboc3\\bgi");
bresenhamdrawline(x0, y0, x1, y1);
getch();
closegraph();
return 0;
}
```



شكل (12.2) المخطط الإنسيابي لطريقة DDIA لرسم الخط المستقيم

The equation of a regular polygon معادلة المضلع المنتظم

A regular polygon is a shape in which all its angles are equal. Previously, if we want to draw a polygon, we would enter the values of the polygon vertices directly into a matrix of points, and then draw the polygon using the function **drawpoly().**

example:

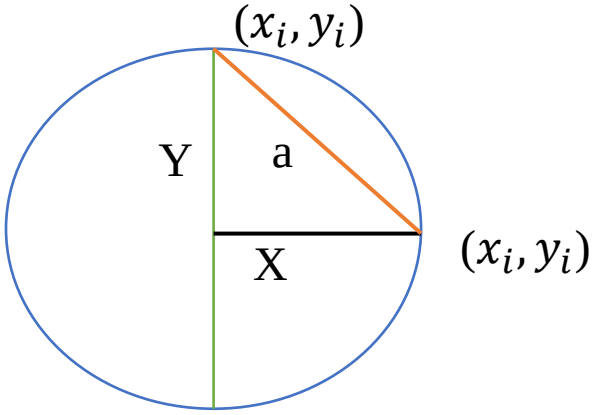
points[]={320,150,420,300,250,300,320,150};

drawpoly(4, points);

There is another and better way to determine the points of a regular polygon using the circle equation with the help of the general properties of the regular polygon.

All the polygon sides are equal in length and all its angles are equal in measure, additionally, all the vertices(رؤوس المضلع) of a regular polygon lie on the circumference(محيط الدائرة) of a circle. This means that each regular polygon has a circle around it and a circle enclosed inside it.

It is possible to construct a regular polygon with n sides using the circle equation to find the points or vertices of the polygon as follows:



The general formula of the circle is:

$$a^2 = X^2 + Y^2$$

We can detail the formula as

$$(x_i - X_0)^2 + (y_i - Y_0)^2 = a^2$$

Where:

(X_0, Y_0)=Center of the circle.

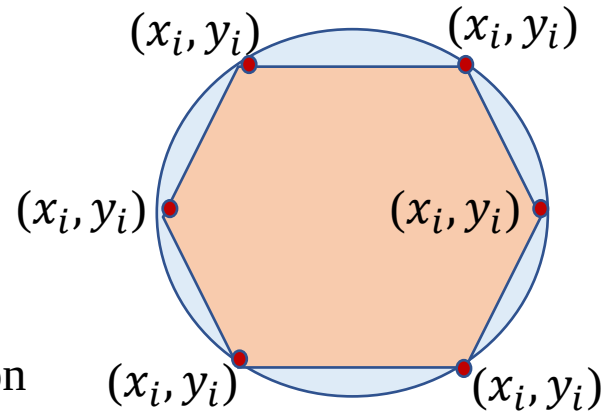
a= radius.

(x_i, y_i)= Points on circle circumference.

From the above equation, we can derive the equation of the points of a regular polygon as follows:

$$x_i = x_0 + a * \cos(2 * Pi * i/n)$$

$$y_i = y_0 + a * \sin(2 * Pi * i/n)$$



Where:

(x_i, y_i)= Vertices of a regular polygon

(x_0, y_0)=the center of the regular polygon

a =radius.

Pi=3.14

n=The number of sides of a regular polygon.

The equation of a regular polygon برنامج معادلة المضلع المنتظم

البرنامج التالي يجد رؤوس المضلع باستخدام معادلة المضلع المنتظم ثم يرسم المضلع باستخدام دالة**drawpoly()**

```
#include<graphics.h>
#include<math.h>
int main(){
 int gd = DETECT, gm;
initgraph(&gd, &gm, "");
int x0=300,y0=250,r=200,
n=8,lx[100],j=0;
float xi,yi,pi=3.14;
for(float i=0;i<=n;i++)
{ xi=x0+r*cos((2*pi)*(i/n));
  yi=y0+r*sin((2*pi)*(i/n));
  lx[j]=xi;
  lx[j+1]=yi;
  j+=2;
}
drawpoly(n+1,lx);
getch();
closegraph();
return 0; }
```

ملاحظه:

(x0,y0) مركز المضلع المنتظم

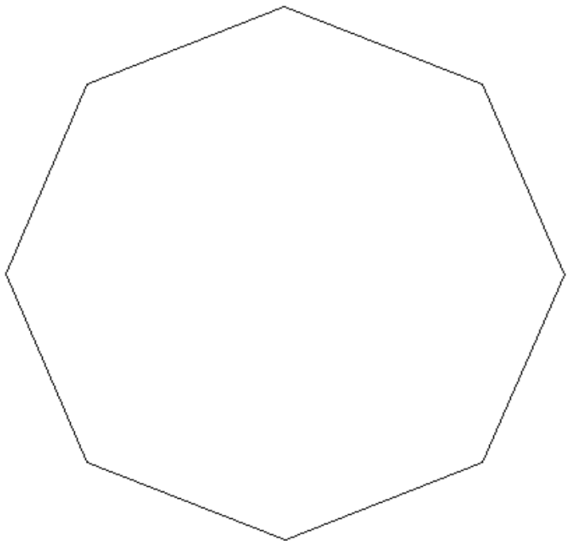
r= نصف القطر

Lx[] مصفوفة لخزن نقاط رؤوس المضلع

(xi,yi) نقاط رؤوس المضلع

n= عدد اضلاع المضلع المنتظم

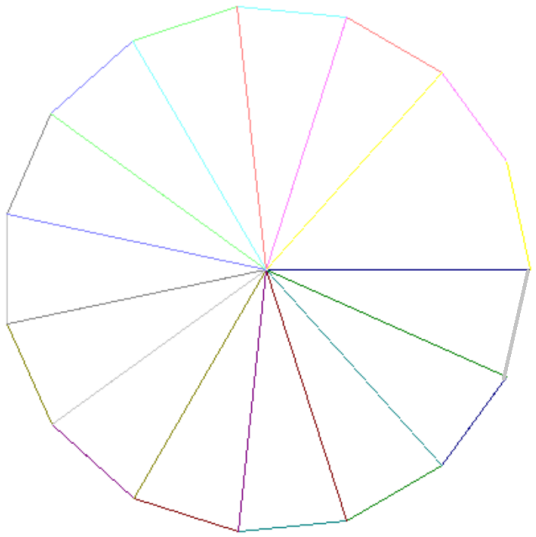
The Output:



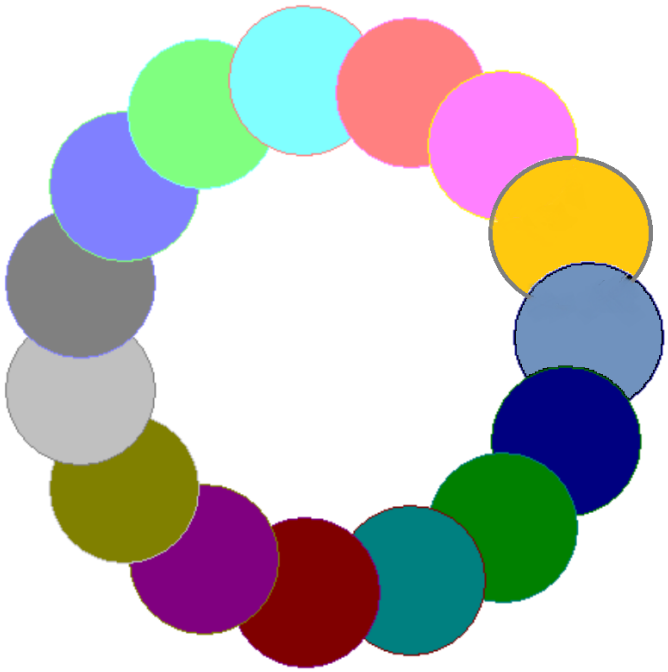
البرنامج التالي يجد رؤوس المضلع باستخدام معادلة المضلع المنتظم ثم يرسم باستخدام دالة الخط

```
#include<graphics.h>
#include<math.h>
int main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "");
int x0=300,y0=250,r=200,n=15,lx[100],ly[100],k=0;
float xi,yi,pi=3.14;
for(float i=0;i<n;i++)
{ xi=x0+r*cos((2*pi)*(i/n));
  yi=y0+r*sin((2*pi)*(i/n));
  lx[k]=xi;
  ly[k]=yi;
  k+=1;
  putpixel(lx[k],ly[k],14); }
int c=1;
for(int j=0;j<n;j++)
{line(lx[j],ly[j],lx[(j+1)%n],ly[(j+1)%n]);
setcolor(c);
line(x0,y0,lx[j],ly[j]);
c=c+1;}getch();closegraph();return 0; }
```

The Output:



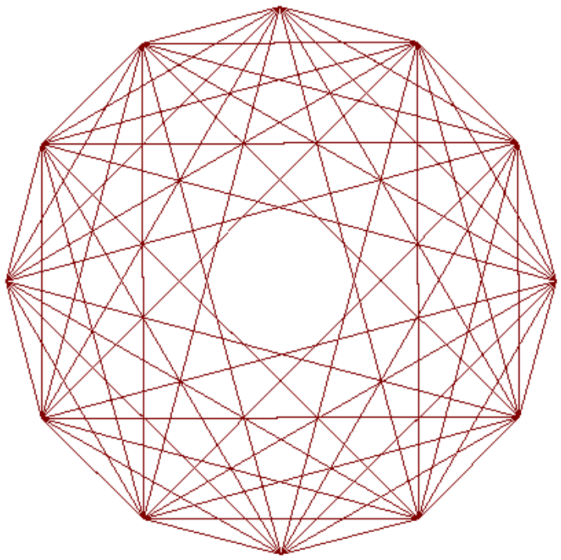
Exercise: Write a C++ program to Draw the following figure:



```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
Int main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "");
int x0=300,y0=250,r=170,n=15,lx[1000],ly[1000],k=0;
float xi,yi,pi=3.14;
setbkcolor(WHITE);
cleardevice();
```

```
for(float i=0;i<n;i++)
{ xi=x0+r*cos((2*pi)*(i/n));
  yi=y0+r*sin((2*pi)*(i/n));
  lx[k]=xi;
  ly[k]=yi;
  k+=1;
  putpixel(lx[k],ly[k],14);
}
for(int j=0;j<n;j++)
{
setcolor(j+1);
setfillstyle(1,j);
circle(lx[j],ly[j],50);
floodfill(lx[j],ly[j],j+1);
circle(x0,y0,80);

}
getch();
closegraph();
return 0;
}
```

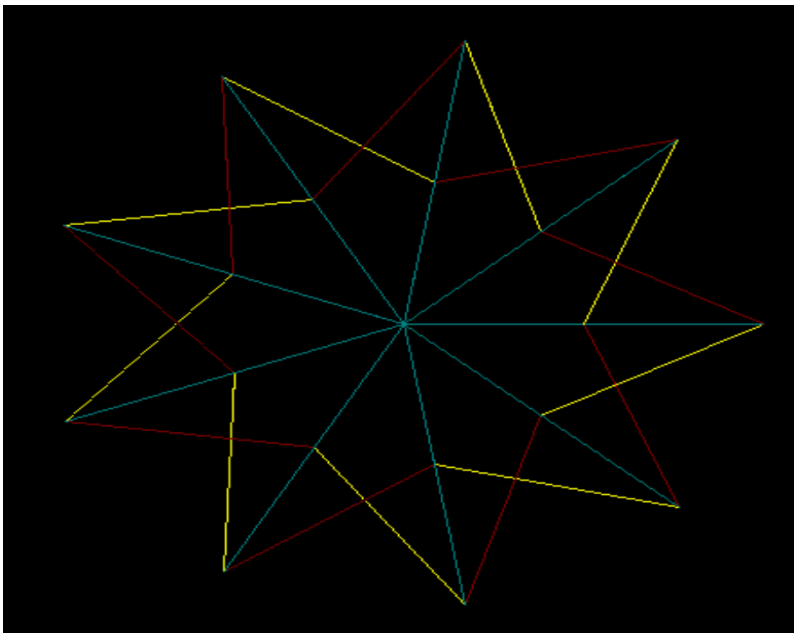
Exercise: Write a C++ program to Draw the following figure:



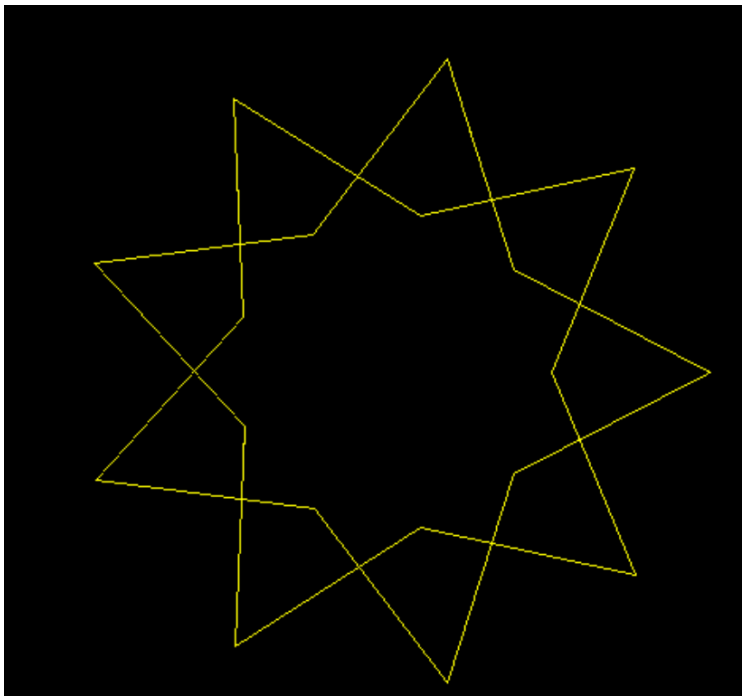
```
#include<graphics.h>
#include<math.h>
main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "");
int x0=300,y0=250,r=200,r2=100,
n=12,lx[1000],ly[1000],k=0,lx2[1000],ly2[1000],k2=0;
float xi,yi,pi=3.14,xi2,yi2;
setbkcolor(15);
cleardevice();
setcolor(4);
for(float i=0;i<n;i++)
```

```
{ xi=x0+r*cos((2*pi)*(i/n));
  yi=y0+r*sin((2*pi)*(i/n));
  xi2=x0+r2*cos((2*pi)*(i/n));
  yi2=y0+r2*sin((2*pi)*(i/n));
  lx[k]=xi;
  ly[k]=yi;
  k+=1;
  putpixel(lx[k],ly[k],14);
  lx2[k2]=xi2;
  ly2[k2]=yi2;
  k2+=1;
  putpixel(lx2[k2],ly2[k2],BLUE);
}
int c=1;
for(int j=0;j<n;j++)
{
      for(int m=1;m<=5;m++)
      { line(lx[j],ly[j],lx[(j+m)%n],ly[(j+m)%n]);
}}
getch();
closegraph();
return 0; }
```

Home Work: Draw the following figure using regular polygon equation where x0=300,y0=250,radius1=200,radius2=100,n=9.



Home Work: Draw the following figure using regular polygon equation



Chapter Four

Transformation

التحويلات

Transformation

All the shapes that we studied previously were a group of points whose coordinates are determined on the screen and then drawn, like lines, circles, arcs, ellipses, polygon, etc.

It is possible to perform the conversion/transformation process (عملية التحويل)on each point of the shape by applying the conversion function(دالة التحويل) to each point, and then using these points after the conversion process to form another shape of the geometry drawn(الشكل الهندسي المرسوم), these transformations are :

1- Translation. (الازاحة)
2- Scaling (التقييس)
3- Rotation (الدوران)
4- Reflection(الانعكاس)
5- Shear(القص)

1-Translation (الازاحة)

To apply translation to a shape, each of the points(x,y) that represent the shape must be shifted to a new location($\bar{x}$,$\bar{y}$)

(x,y)→ ($\bar{x}$,$\bar{y}$)

Where:

$\bar{x}$=x+t_x

$\bar{y}$=x+t_y

ملاحظة

(x,y) قيمة احداثي النقطة في الموقع القديم

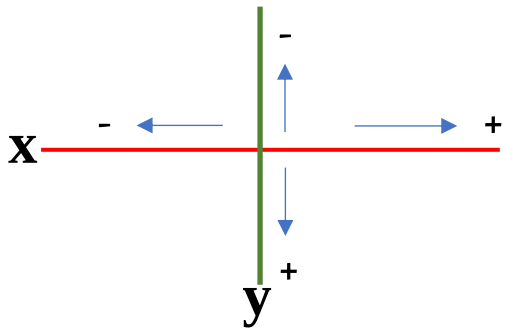
($\bar{x}$,$\bar{y}$)قيمة احداثي النقطة في الموقع الجديد

t_x مقدار إزاحة الشكل على محور السينات

t_y مقدار إزاحة الشكل على محور الصادات

Note 1: If the value of (t_x) is positive, then the point is shifted to the right, but if it is negative, then the point is shifted to the left.

Note 2: if the value of (t_y) is positive, the point is shifted downwards (increases(تزداد)), and if it is negative, the shape is shifted upwards (decreases(تقل)).

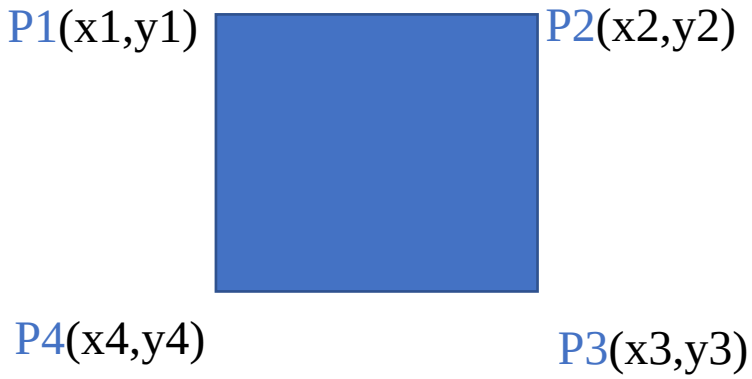


Note 3: when a shape is shifted, it must first be shifted to the origin point (نقطة الاصل) and then moved to the new location.

When moving the point to the origin point (0,0), this is done by subtracting the point from itself **P(x-x, y-y) .**

When displacing(ازاحة) a shape, it must be moved to the origin point relative (نسبة الى) to one of its points.

This is done by subtracting (طرح)the coordinates of that point from all the points of the shape. For example, if we want to displace a square shape, the coordinates of its points are as follows:



The new points are:

P1(x1-x1,y1-y1),P2(x2-x1,y2-y1),P3(x3-x1,y3-y1),P4(x4-x1,y4-y1).

After its displacement (إزاحة الشكل)to the origin point, the amount of displacement (مقدار الازاحة)is added to each point of the shape.

Note 4: the variable (**Flag**) takes one of the two values, one(1) or negative one(-1). When its value is **(1)**, the shape is shifted to the **origin point**, and when its value is**(-1)**, the shape is shifted from the origin point to the **new location**.

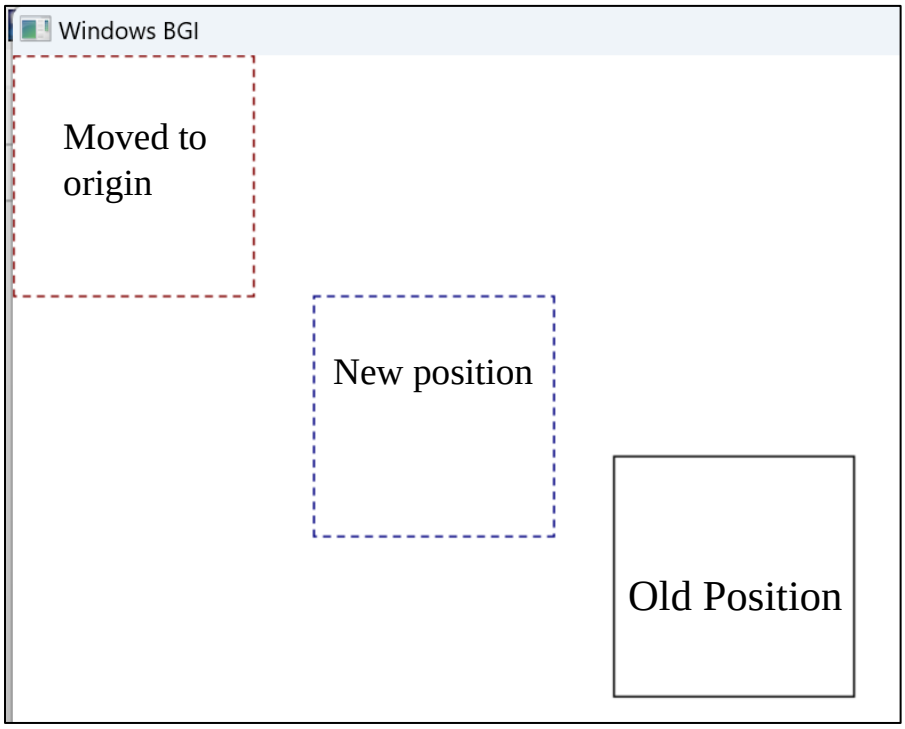
When the shape is moved, the general shape(الشكل العام) or size(الحجم) of the drawn shape is not changed.

Exercise: Write a C++ program to translate the following shape to the new location p(150,120).

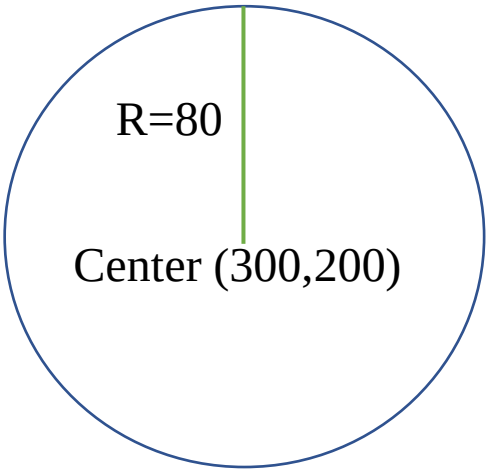
```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
void DrawSquare(int px1,int py1,int px2,int py2,int px3,int
py3,int px4,int py4)
 { line(px1,py1,px2,py2);
 line(px2,py2,px3,py3);
 line(px3,py3,px4,py4);
 line(px4,py4,px1,py1);
 }
 void Translate(int flag,int tx,int ty,int &px1,int &py1,int
&px2,int &py2,int &px3,int &py3,int &px4,int &py4)
 {
  px1=px1-(tx*flag);    py1=py1-(ty*flag);
  px2=px2-(tx*flag);    py2=py2-(ty*flag);
  px3=px3-(tx*flag);    py3=py3-(ty*flag);
  px4=px4-(tx*flag);    py4=py4-(ty*flag);
 }
Int main()
{ int gd = DETECT, gm;
```

```
initgraph(&gd, &gm, "");
int x1=300,y1=200,
    x2=420,y2=200,
    x3=420,y3=320,
    x4=300,y4=320,
    tx=300,ty=200;
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
//translate the square to the original point (0,0) with respect to point
//(x1,y1) by subtracting this point from each point of the square
Translate(1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
//Redraw the square in the new position (0,0)
setlinestyle(1,1,1);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
tx=150;ty=120;
//translate the square to the new position
Translate(-1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
//Redraw the square in the new position
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
getch();closegraph();return 0; }
```

The output



Exercise: Write a C++ program to translate the following shape to the new location p(480,350), where (x1=300,y1=200),r=80.



```
#include<graphics.h>
#include<conio.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
void Drawcircle(int px1,int py1,int r)
 { circle(px1,py1,r);
 }
 void Translate(int flag,int tx,int ty,int &px1,int &py1)
 {
   px1=px1-(tx*flag);     py1=py1-(ty*flag);
 }
int main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm,"");
int x1=300,y1=200,r=80;
int tx=300,ty=200;
setcolor(15);
outtextxy(x1-50,y1,"old position");
Drawcircle (x1,y1,r);
Translate(1,tx,ty,x1,y1);
```

```
//Redraw the circle in the new position (0,0)
setlinestyle(1,1,1);
setcolor(4);
Drawcircle (x1,y1,r);
tx=480;ty=350;
//translate the circle to the new position
Translate(-1,tx,ty,x1,y1);
//Redraw the circle in the new position
setcolor(14);
Drawcircle (x1,y1,r);
outtextxy(x1-50,y1,"new position");
getch();closegraph();return 0; }
```

2- Scaling of Graphics Objects تقييس الاشكال الرسومية

The process of scaling is to change the size of the drawn shape, in terms of enlarging or reducing each of the points **p(x,y)** that represent the shape is moved to a new location $\bar{p}$ $(\bar{x}, \bar{y})$

عملية التقييس هي تغيير حجم الشكل المرسوم من ناحية التكبير او التصغير لكل نقطة من النقاط التي تمثل الشكل يتم تحريكها الى موقع جديد وحسب المعادلات التالية:

$$\bar{x}=S_x * x$$

$$\bar{y}=S_y * y$$

Where S_x, S_y are the scaling factor.

Notes:

If the scaling factors are equal to one, it means the shape maintains the same size, and if the scaling factors are greater than one, it means increasing the size of the shape. If the scaling factors are less than one, it means reducing the size of the shape.

إذا كان معامل التقييس يساوي واحد يعني الحفاظ على الشكل المرسوم بنفس الحجم وإذا كان أكبر من واحد يعني زيادة حجم الشكل اما إذا كان هذا المعامل أصغر من واحد فيعني تصغير حجم الشكل.

If the scaling factors are **not equal**, this leads to a distortion in the shape resulting from the original shape. These transformations, whether enlarging or reducing the shape, are called distortions with the origin point, but if the scaling factors are related to another point called the **Pivot Point** (x_p, y_p) other than the origin point (0,0), then the scaling process is done according to the following points:

إذا كانت معامل التقييس غير متساوية فهذا يؤدي الى تشويه في الشكل الناتج عن الشكل الاصلي.
هذه التحويلات سواء كانت تكبير او تصغير للشكل تسمى التحريفات بالنسبة لنقطة الاصل اما إذا كان التقييس بالنسبة لنقطه اخرى تسمى نقطه الارتكاز غير نقطة الاصل فيتم التقييس حسب النقاط التالية:

1- The displacement of the shape to the origin point relates to that point using the previous displacement methods, where the coordinates of each point of the shape become as follows:

ازاحه الشكل الى نقطه الاصل نسبه الى تلك النقطة باستخدام طرق الإزاحة السابقة حيث تصبح احداثيات كل نقطه من نقاط الشكل كما يلي

$$\bar{x} = x - x_p$$

$$\bar{y} = y - y_p$$

2- Scaling the displacement point according to the following:

$$\bar{\bar{x}} = \bar{x} * s_x \quad \Rightarrow \quad (x - x_p)s_x$$

$$\bar{\bar{y}} = \bar{y} * s_y \quad \Rightarrow \quad (y - y_p)s_y$$

3- Return each scaled point to its original position by re-displacing it by the same amount as it was displaced in the first step, but in the opposite direction.

$$\bar{\bar{\bar{x}}} = \bar{\bar{x}} + x_p$$

$$\bar{\bar{\bar{y}}} = \bar{\bar{y}} + y_p$$

From the above three steps, we get the result:

$$\bar{\bar{x}} = (x - x_p)s_x + x_p$$

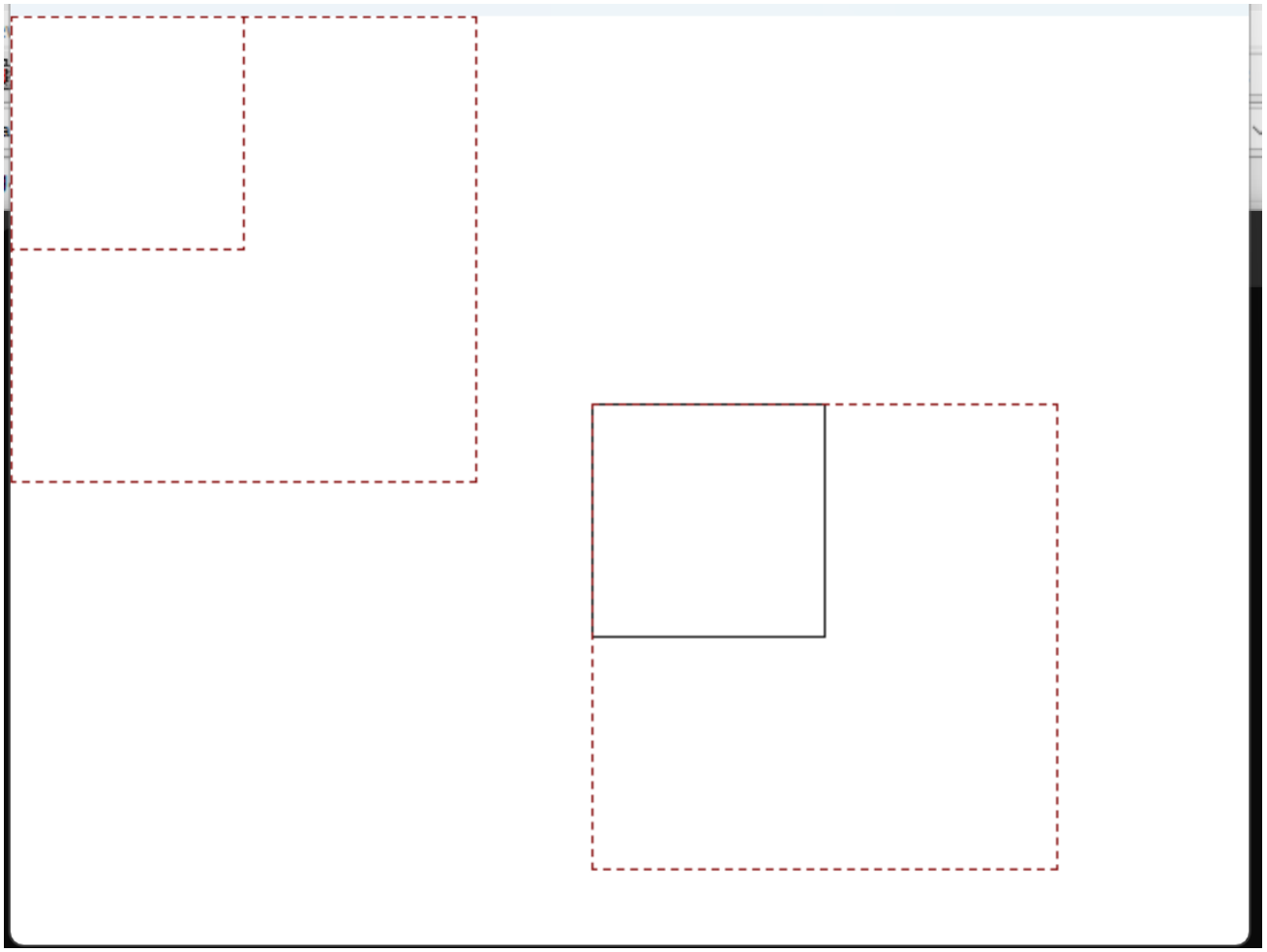
$$\bar{\bar{y}} = (y - y_p)s_y + y_p$$

The Following program is the implementation of Scaling Process on a square with points (x1,y1), (x2,y2), (x3,y3), (x4,y4) to the point(x1,y1).

```
#*****************************************
#include<graphics.h>
#include<stdlib.h>
#include<stdio.h>
#include<math.h>
void DrawSquare(int px1,int py1,int px2,int py2,
                int px3,int   py3,int px4,int py4)
 { line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
 }
 void Translate(int flag,int tx,int ty,int &px1,int &py1,
  int &px2,int &py2,int &px3,int &py3,int &px4,int &py4)
 { px1=px1-(tx*flag);     py1=py1-(ty*flag);
   px2=px2-(tx*flag);     py2=py2-(ty*flag);
   px3=px3-(tx*flag);     py3=py3-(ty*flag);
   px4=px4-(tx*flag);     py4=py4-(ty*flag);
 }
```

```
 void Scale(int sx,int sy,int &x,int &y)
 {     x=x*sx;     y=y*sy;
 }
main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "");
setbkcolor(15);
cleardevice();
int x1=300,y1=200,
    x2=420,y2=200,
    x3=420,y3=320,
    x4=300,y4=320,
    tx=300,ty=200;
setcolor(0);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
setlinestyle(1,1,2);
Translate(1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
setlinestyle(1,1,1);
setcolor(4);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
Scale(2,2,x1,y1);Scale(2,2,x2,y2);
Scale(2,2,x3,y3);Scale(2,2,x4,y4);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
Translate(-1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
getch();closegraph();
return 0;
 }
```

The OUTPUT



HOMEWORK:

By using the above program how can we scale a CIRCLE Where its Radius =50 , CenterPoint=(100,100), and the scaling factor(0.5)

3-Rotation التدوير

Sometimes it is useful to rotate a graphic figure around a specific point called the Pivot Point or a specific axis called the center at a specific angle $\boldsymbol{\theta}$. In order to rotate around this point, the figure must return to the origin point (We have already taken the displacement methods with the required routines), then the shape is rotated relative to the origin point by rotating each of the figure points. If we assume that one of the points P(x,y) is to be rotated at a certain angle $\boldsymbol{\theta}$, then:

من المفيد احيانا تدوير شكل رسومي حول نقطه معينه تسمى نقطه الارتكاز او محور معين يسمى المركز بزاوية معينه. ولأجل ان يتم التدوير حول هذه النقطة يجب ان يرجع الشكل الى نقطه الاصل، اي يزاح الشكل الى نقطه الاصل بالنسبة ل نقطه الارتكاز (سبق ان اخذنا طرق الإزاحة مع الروتينات المطلوبة لذلك) ثم يتم تدوير الشكل بالنسبة الى نقطه الاصل وذلك بتدوير كل نقطة من النقاط التي تمثله .لو فرضنا ان احدى النقاط يراد تدويرها بزاوية معينة فأن:

$$(x, y) \longrightarrow (\bar{x}, \bar{y})$$

After translate to the origin point where:

$$\bar{x} = x - x_p$$

$$\bar{y} = y - y_p$$

Then rotate P($\bar{x}, \bar{y}$) to give us the new point P($\bar{\bar{x}}, \bar{\bar{y}}$) where:

$$\bar{\bar{x}} = (x - x_p)\cos\theta - (y - y_p)\sin\theta$$

$$\bar{\bar{y}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta$$

Where the angle $\boldsymbol{\theta}$ must be in radial angles (**زوايا نصف قطرية**) and be a multiple of the number (مضاعفات العدد) **$2\pi/2$**, then it will translate to its original position (الموقع الأصلي للشكل) after rotation to give us the point P ($\bar{\bar{\bar{x}}}, \bar{\bar{\bar{y}}}$)where:

$$\bar{\bar{\bar{x}}} = (x - x_p)\cos\theta - (y - y_p)\sin\theta + x_p$$

$$\bar{\bar{\bar{y}}} = (y - y_p)\cos\theta + (x - x_p)\sin\theta + y_p$$

Where the rotation is clockwise (*مع عقارب الساعة*) and if we want to rotate the shape to counterclockwise (*عكس عقارب الساعة*), we should just change the angle to (- $\boldsymbol{\theta}$) instead of ($\boldsymbol{\theta}$).

The Following program implement the Rotation Process on a square with points (x1,y1), (x2,y2), (x3,y3), (x4,y4).

```
#*************************************************
#include<graphics.h>
#include<math.h>
```

```
void DrawSquare(int px1,int py1,int px2,int py2,
                int px3,int py3,int px4,int py4)
 {
   line(px1,py1,px2,py2);
   line(px2,py2,px3,py3);
   line(px3,py3,px4,py4);
   line(px4,py4,px1,py1);
 }
 void Translate(int flag,int tx,int ty,int &px1,
           int &py1,int &px2,int &py2,int &px3,
           int &py3,int &px4,int &py4)
 {
  px1=px1-(tx*flag);    py1=py1-(ty*flag);
  px2=px2-(tx*flag);    py2=py2-(ty*flag);
  px3=px3-(tx*flag);    py3=py3-(ty*flag);
  px4=px4-(tx*flag);    py4=py4-(ty*flag);
 }

 void Rotatepoint(int alpha,int &x,int &y)
 {
  float xtemp,ytemp;
  float sine,cosine;
```

```
    float Pi=22/7;
    xtemp=x;ytemp=y;
    cosine=cos(alpha*Pi/180);
    sine=sin(alpha*Pi/180);
    x=floor(cosine*xtemp-sine*ytemp);
    y=floor(sine*xtemp+cosine*ytemp);
  }
Int main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "");
setbkcolor(15);cleardevice();
int x1=300,y1=200,
    x2=420,y2=200,
    x3=420,y3=320,
    x4=300,y4=320,
    tx=300,ty=200,
    alpha=30;
setcolor(0);
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
Translate(1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
setlinestyle(1,1,1);
setcolor(4);
```

```
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
//Rotate the square in the position (0,0)
Rotatepoint(alpha,x1,y1);Rotatepoint(alpha,x2,y2);
Rotatepoint(alpha,x3,y3);Rotatepoint(alpha,x4,y4);
//Redraw the square after the rotation
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
Translate(-1,tx,ty,x1,y1,x2,y2,x3,y3,x4,y4);
//Redraw the square after the rotation and translation
DrawSquare(x1,y1,x2,y2,x3,y3,x4,y4);
getch();closegraph();
return 0;
}
```

The OUTPUT