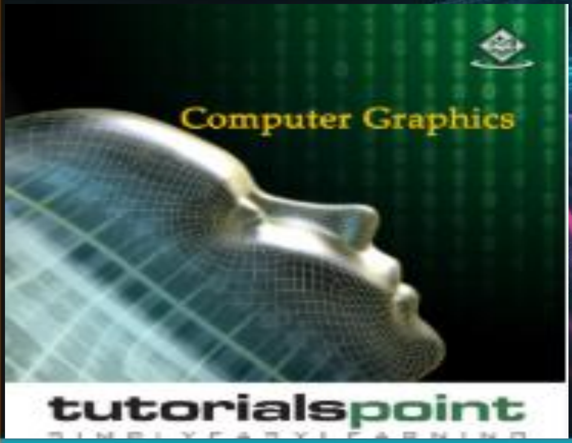
UNIVERSITY OF DIYALA

College of Education and Pure Ssciences

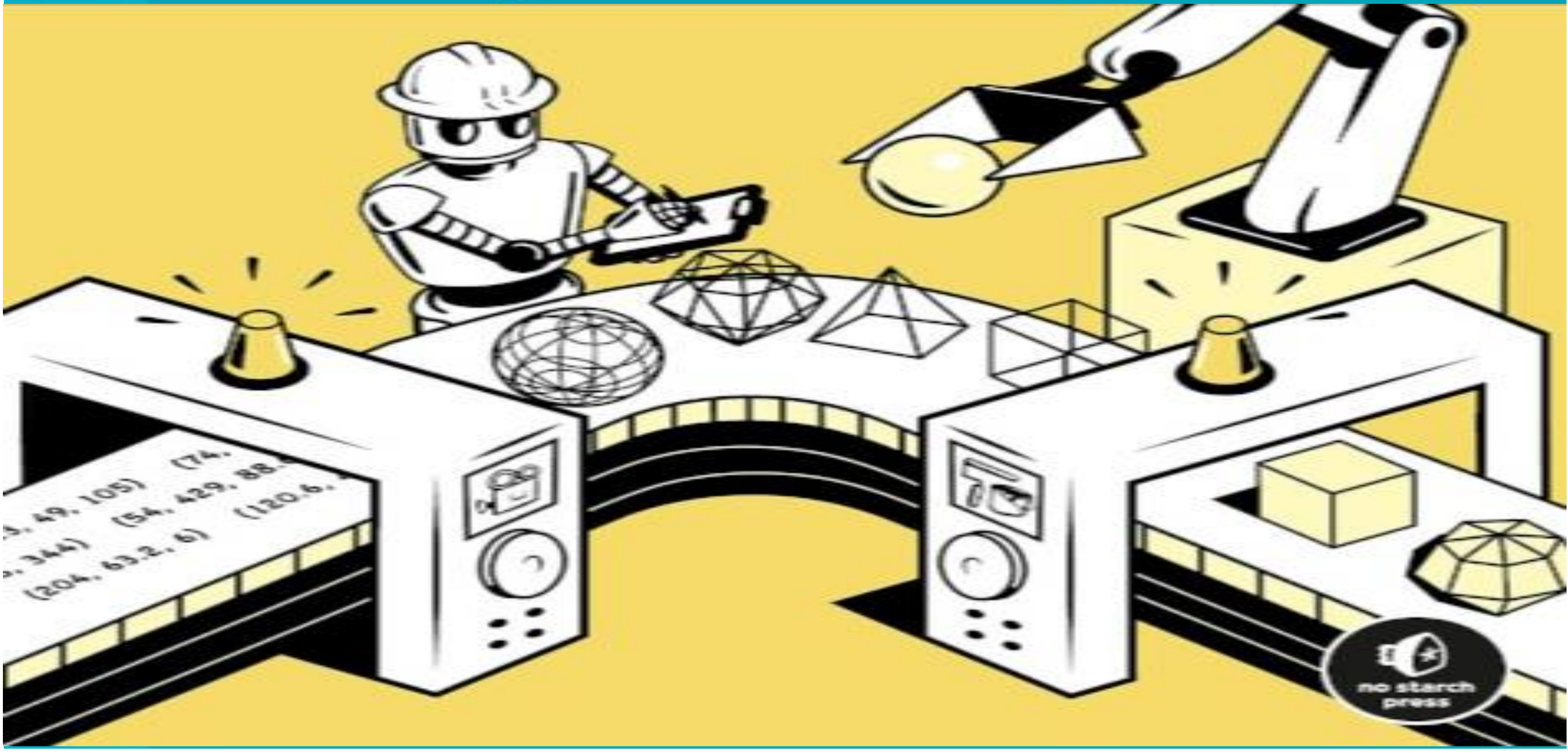
Lecture 1



Dr. Qasim Al-Obaidi

Third class

Introduction to Computer Graphics

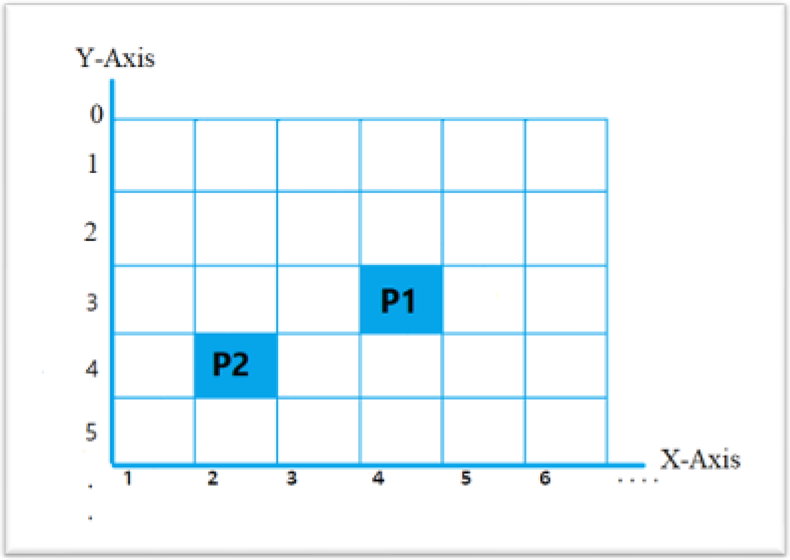


Introduction to Computer Graphics

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. The phrase "Computer Graphics" was coined in 1960 by William Fetter, a graphic designer for Boeing. Computers have become very significant in today's modern world. Computer graphics are pictures and movies created using computers usually referring to image data created by a computer specifically with help from specialized graphical hardware and software. It is a vast and recently developed area of computer science.

On the other words, 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. The picture or graphics object may be an engineering drawing, business graphs, architectural structures, a single frame from an animated movie or a machine part illustrated for a service manual. 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 and color of the pixel which composed the screen. Computer Graphics 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, 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, translation, scaling, 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.

There are two types of graphics that are used nowadays, interactive and passive.

1- Interactive Computer graphics (IGU): it involves a two-way communication between the computer and user. Here, the displayed picture is modified appropriately to signals, the computer receives from input device e.g. (keyboard, mouse, joystick... etc.). It appears that picture is changing instantaneously in response to the observer's commands. Example: computer games (video game controller).

2- Non-Interactive Computer graphics: (passive Computer graphics). Here, the user does not have any kind of control over the image. The image is totally under the control of program instructions not under the user. Example: screen savers. The differences between active and passive graphics are shown in Table 1.

Table 1: The differences between Active and Passive Graphics.

	Passive ***Computer graphics***	**Active** ***Computer graphics***
Control	No control	Control (dynamic nature)
Communication	One way	Two way
Interaction	No interaction.	High interaction.
Motion and Updation	No facility	2-D,3-D transformations
Prone to Problems	Less	More

Choosing between interactive graphics and passive graphics depends on whether you are aiming for simplicity or features. Interactive graphics are more entertaining than passive graphics but is also more difficult to deal with. Some sites choose a compromise by creating two versions for their site; one with interactive graphics, and another with passive graphics.

There are many advantages of interactive graphics over passive graphics:

• Higher quality of images

• Low cost

• Higher productivity

• Low analysis

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

Application of Computer Graphics

Today, computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, for example in their weather reports, or in all kinds of medical investigation and surgical procedures. Some people even make computer graphics as art. We can classify applications of computer graphics into four main areas:

- Display of information
- Design
- User interfaces
- Simulation

CAD (Computer Aided Design): use of a computer to aid in the design of product.

CAM (Computer Aided Manufacturing): use of a computer to control the manufacturing of products.

CAI (Computer Aided Instructing): use of computer to display animated pictures to illustrate educational concept.

CAE (Computer Aided Engineering): use of a computer as engineer work. **Presentation Graphics**: To produce illustrations which summarize various kinds of data.

Entertainment: Motion pictures, Music videos, and TV shows, Computer games. Education and

Training: Training with computer-generated models of specialized systems such as the training of ship captains and aircraft pilots.

Visualization: For analyzing scientific, engineering, medical and business data or behavior. Image Processing: Image processing is to apply techniques to modify or interpret existing pictures.

Graphical User Interface: Multiple window, icons, menus allow a computer setup to be utilized more efficiently.

Simulation: use of a computer to experience circumstance that otherwise would be too expensive or catastrophic to experience in reality E.g.: flight simulators, simulating unclear reactors.

Graphics Display Devices

A display device is a device for visual or tactile presentation of images (including text) acquired, stored, or transmitted in various forms. As we are familiar that we can interact with the computer system through the Display Devices. These display devices enable us to communicate with the computer through the graphical interface.

There are different types of Display Devices which are used to interact with the computer system or any other device. Some of them are following:

• Monitor

This is mostly used as an output display device. There are further two types of Monitors available which are discussed following 1- Cathode Ray Tube (CRT): A CRT monitor is heavier than flat-panel display (FPD) because it contains a large cathode ray tube. It uses electron guns to activate phosphors behind the screen, causing each pixel on the monitor to generate red, green, or blue.

Figure 1: A CRT Monitor.

How the cathode ray tube (CRT) works:

A CRT is an evacuated glass tube. The CRT consists of three electron guns: red, green and blue. They emit a beam of negative charged electrons towards a positively charged phosphor-coated screen. Along the way the electron beam passes through the focusing system that concentrates the beam so that by the time the electrons reach the screen they have converged to a small dot. The beam then passes through the deflection system (horizontal and vertical) which deflect the beam to strike any point on the screen. When this focused electron beam strikes the screen, the phosphor emits as dot of visible light. There are two techniques used for producing image on the CRT screen: Vector scan / random scan and Raster scan. The video display is divided in to very small dots called "pixels" (picture elements) each pixel is composed of a triangular pattern of red, green and blue phosphor dot. The CRT has three electron guns one for each of the three primary colors: red, green and blue each electron gun hits its corresponding phosphor dot causing that particular color to appear on the screen the light on the display screen starts to fade as soon as the beam moves to another location. So, the beam must refresh the screen by illuminating pixels 30 to 60 times each second.

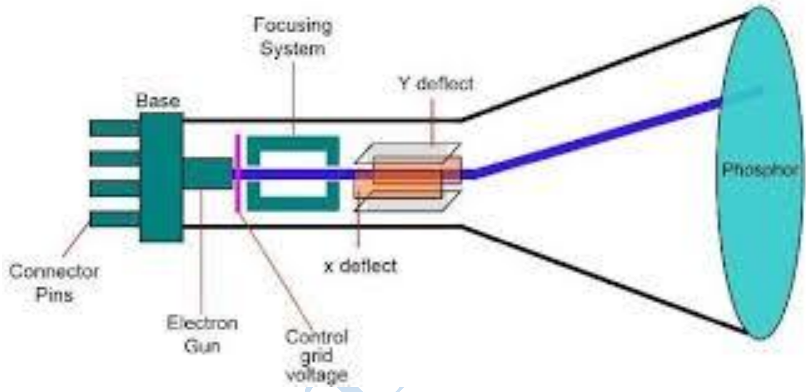


Figure 2: Cathode ray tube (CRT)

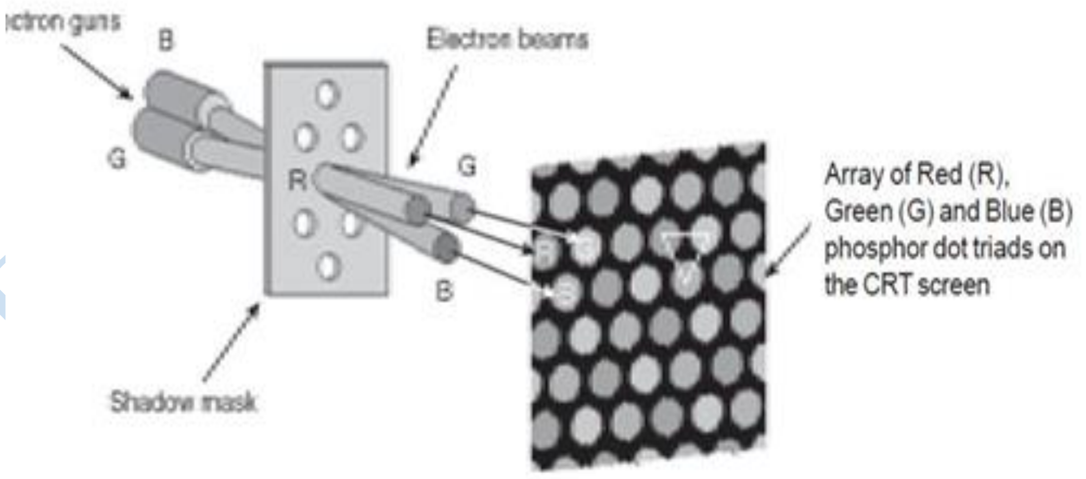

ctron guns
B
G
Electron beams
R
G
B
Shadow mask
Array of Red (R), Green (G) and Blue (B) phosphor dot triads on the CRT screen

Figure 3: Electron Guns Passing Through CRT.

Generating Color on a RGB monitors:

Each electron gun in an RGB monitor, has an assigned number of bit that determines the intensities of the red, green and blue phosphors, with one bit per gun eight color are possible (2^3=8).

R	G	B	Binary value	Color value
0	0	0	0	black
0	0	1	1	blue
0	1	0	2	Green
0	1	1	3	Cyan
1	0	0	4	red
1	0	1	5	Magenta
1	1	0	6	Brown
1	1	1	7	White

If a fourth bit is used for the brightness' or intensity of the displayed color it results in (2^4=16) possible colors. When the brightness bit equals one another set of eight bright colors is produced.

Note: if True Color is 24 bit then displayed color = 0 to 2^{24}-1. In RGB 24 bit => Red (8 bit = 0 to 2^{8}-1), Green (8 bit = 0 to 2^{8}-1), Blue (b bit= 0 to 2^{8}-1).

2- Flat Panel Display (FPD):

A flat-panel display (FPD) is very thin, lightweight, and very little power device. A flat panel display has common types like LCD (Liquid Crystal Display), LED (Light Emitting Diode) and OLED (Organic LED). LCD uses a liquid crystal molecule for displaying, LED uses light emitting diodes for display while OLED uses a special organic compound for display. CRT Display and Flat Panel Display (FPD) are shown in following Table3:

Table 3: The Differences between CRT and FPD.

	CRT	***FPD***
Stands For	For "Cathode Ray Tube ".	For "Flat Panel Display ".
Examples	Televisions and Was Used in Old Computer Monitors.	LCD, Plasma, OLED, And LED
Cost	It Is Less Expensive.	It Is More Expensive.
Power Consumption	It Consumes High Power.	It Consumes Low Power.
Weight	Heavier	Less Heavy
Image Retension	Not There in CRT.	There in FPD.
Size	Cannot Be Used for Smaller Displays Ex: Watches	Can Be Used for Smaller Displays
Thickness	Much Larger	Less Large
Dead Pixels	Do Not	Suffer

All computer art is digital, but there are two very different ways of drawing digital images on a computer screen, known as raster and vector graphics.

Raster Graphics:

Describes a picture using many small dots of color. Each dot is called a pixel, which is an abbreviation for "picture element". If the dots are small enough and close enough together, a person does not sees the individual dots, but rather sees a "picture".

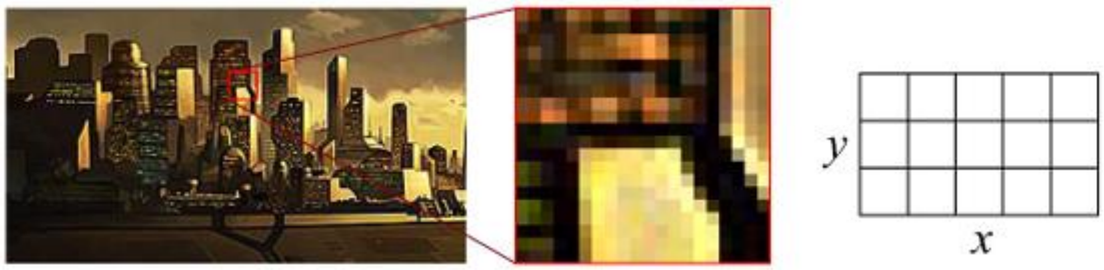


Figure 3: Raster graphics lose quality as they are scaled up (or down!).

One of 2N intensities or colors are associated with each pixel, where N is the number of bits per pixel. Greyscale typically has one byte per pixel, for 28 = 256 intensities. Color often requires one byte per channel, with three color channels per pixel: red, green, and blue.

A bitmap (also called "raster") graphic is created from rows of different colored pixels that together form an image. In their simplest form, bitmaps have only two colors, with each pixel being either black or white. With increasing complexity, an image can include more colors; photograph-quality images may have millions. Examples of bitmap graphic formats include GIF, JPEG, PNG, TIFF, XBM, BMP, and PCX as well as bitmap (i.e., screen) fonts.

Color data is stored in a frame buffer. This is sometimes called an image map or bitmap.

Scan conversion is the process of converting basic, low level objects into their corresponding pixel map representations. "The process of representing continuous graphics objects as a collection of discrete pixel is called scan conversion".

• Vector Graphics:

Describes objects as geometric shapes using mathematical equations. A picture is created from the mathematical descriptions through a process called rendering. The results of a rendering is a 2-dimensional raster image. In below figure we adopt a standard notation and show vector quantities in **bold** type scalar quantities (like real number) in standard type. Thus, the arrow from A to B will written AB and the arrow from A to B will be written BA; in both cases the length of the line (the size of the vector) is the same. (In handwriting, we usually underline vector quantities).

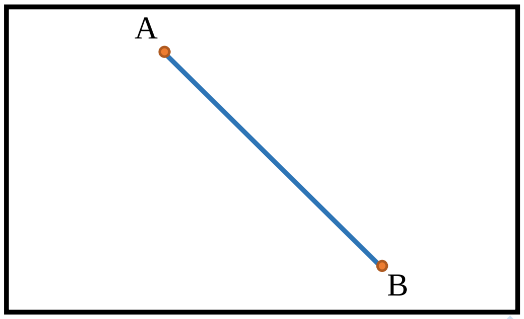
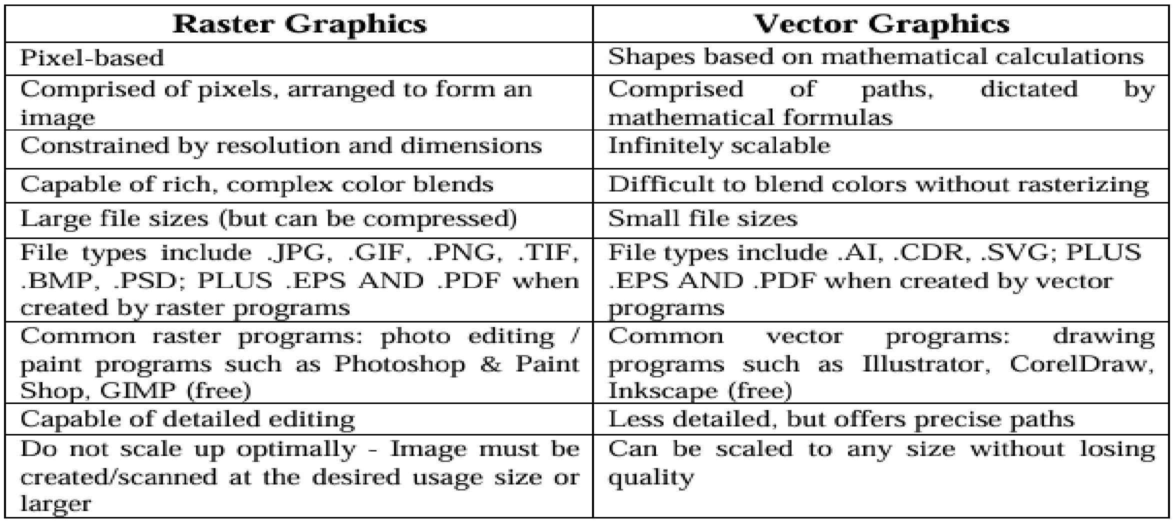


Figure 4: lines Connection.

As shown in above figure, vector scan CRT display directly traces out only the desired lines on CRT i.e. If we want a line connecting point A with point B on the vector graphics display, we simply drive the beam deflection circuitry, which will cause beam to go directly from point A to B. if we want to move the beam from point A to point B without showing a line between points, we can blank the beam as we move it. To move the beam across CRT, the information about, magnitude and direction is required. This information is generated with the help of vector graphics generator. The figure bellow shows the typical vector display architecture.

Table 4: The differences, advantage (pros), and disadvantages (cons) between Raster and vector Graphics.



Raster Graphics	Vector Graphics
Pixel-based	Shapes based on mathematical calculations
Comprised of pixels, arranged to form an image	Comprised of paths, dictated by mathematical formulas
Constrained by resolution and dimensions	Infinitely scalable
Capable of rich, complex color blends	Difficult to blend colors without rasterizing
Large file sizes (but can be compressed)	Small file sizes
File types include .JPG, .GIF, .PNG, .TIF, .BMP, .PSD; PLUS .EPS AND .PDF when created by raster programs	File types include .AI, .CDR, .SVG; PLUS .EPS AND .PDF when created by vector programs
Common raster programs: photo editing / paint programs such as Photoshop & Paint Shop, GIMP (free)	Common vector programs: drawing programs such as Illustrator, CorelDraw, Inkscape (free)
Capable of detailed editing	Less detailed, but offers precise paths
Do not scale up optimally - Image must be created/scanned at the desired usage size or larger	Can be scaled to any size without losing quality

Important Characteristics of Video Display Devices:

Persistence: The major difference between phosphors is their persistence. It decides how long they continue to emit light after the electron beam is removed.

Resolution: Resolution indicates the maximum number of points that can be displayed without overlap on the CRT. Resolution depends on the type of phosphor, the intensity to be displayed and the focusing and deflection systems used in the CRT.

Aspect Ratio: It is the ratio of vertical points to horizontal points to produce equal length lines in both directions on the screen. An aspect ratio of 4/5 means that a vertical line plotted with four points has the same length as a horizontal line plotted with five points.

Lecture 2

UNIVERSITY OF DIYALA

College of Education an

Pure Ssciences

Dr. Qasim Al-Obaidi

Third class

Introduction to Computer Graphics

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”);

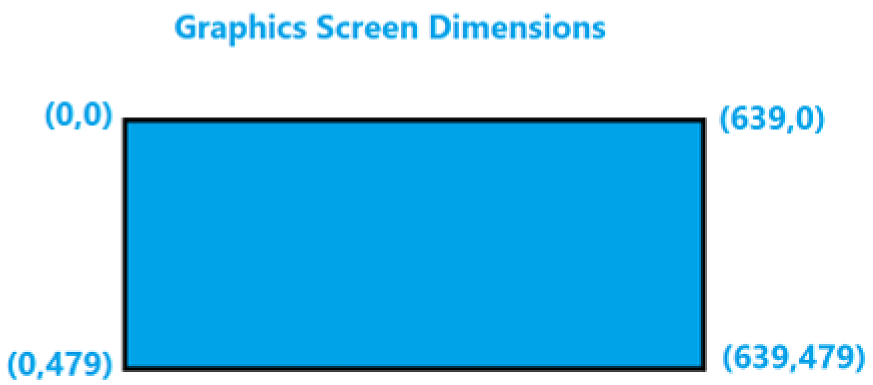
Path to driver:

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,“”);



3. **Graphresult ():**

This function is used to return the error code at any 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();

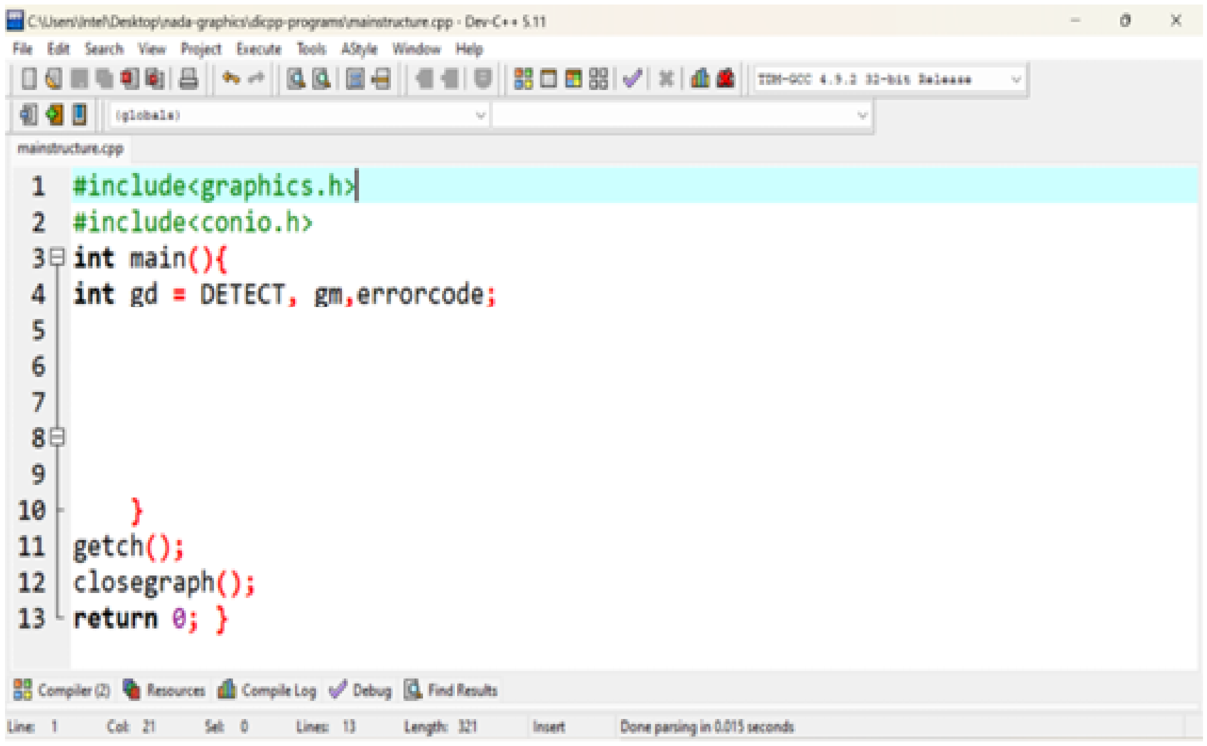
closegraph();

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++:


C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\mainstructure.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 32-bit Release
(globals)
mainstructure.cpp
Compiler (2) Resources Compile Log Debug Find Results
Line: 1 Col: 21 Sel: 0 Lines: 13 Length: 321 Insert Done parsing in 0.015 seconds

```
#include<graphics.h>
#include<conio.h>
int main(){
int gd = DETECT, gm,errorcode;






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

HOMEWORK

What is the output of this code?

```
#include<iostream.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();
}
```

Lecture 3

UNIVERSITY OF DIYALA

College of Education an

Pure Ssciences

Dr. Qasim Al-Obaidi

Third class

Introduction to Computer Graphics

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.

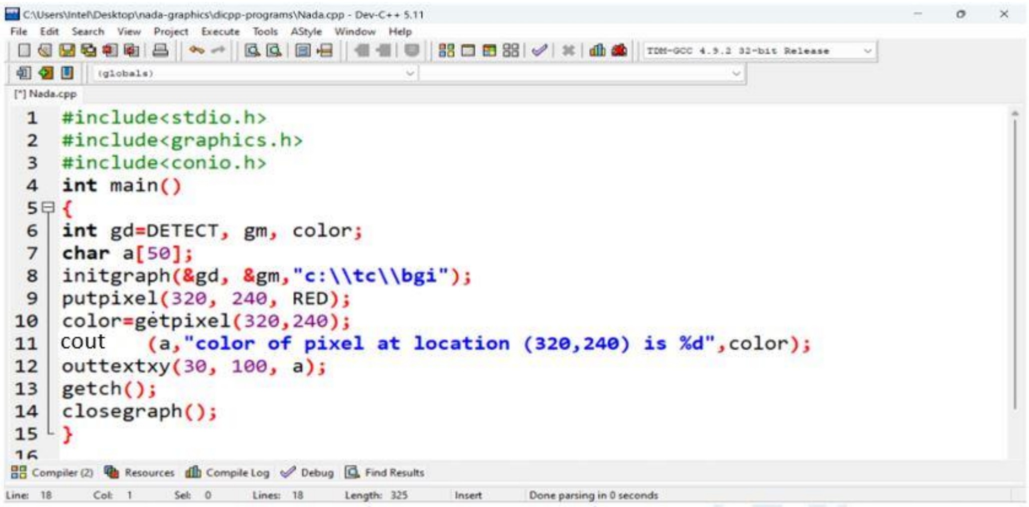
2. Getpixel () returns the color of the pixel present at the location (x, y).

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

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

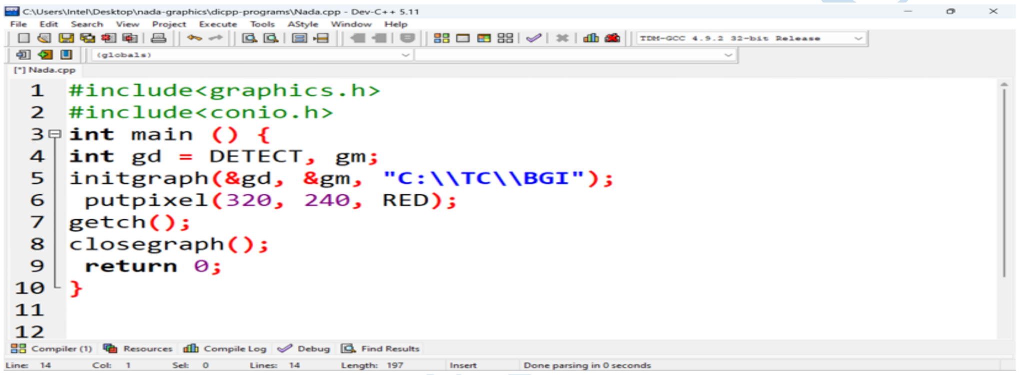
```
char array[50]; int color = getpixel(0, 0);
cout<<(array,"color of pixel at (0,0) = %d",color);
outtext(array);
```

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



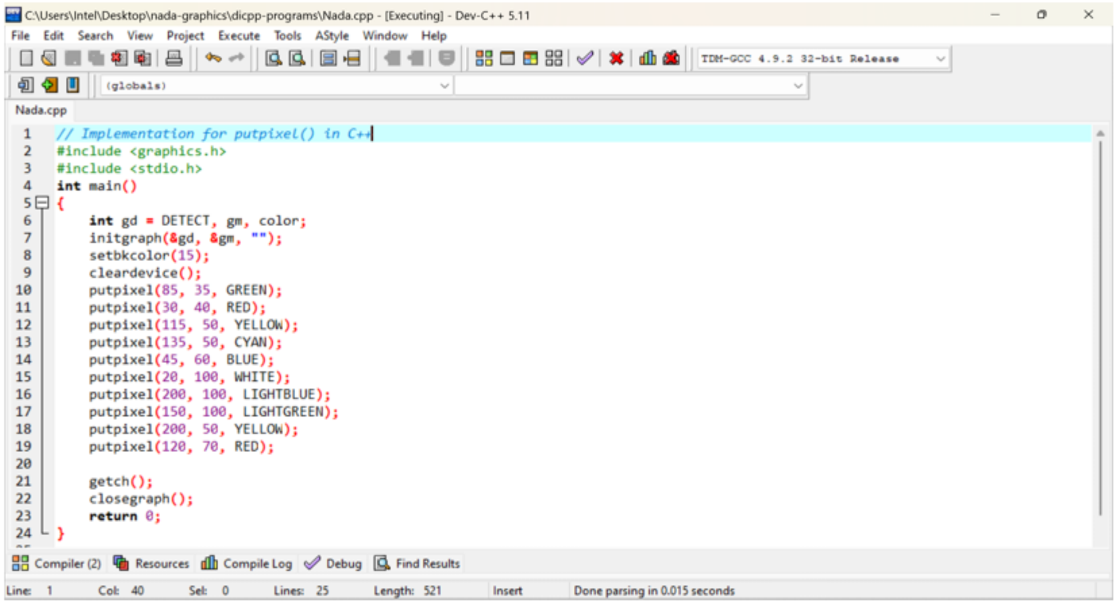
```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
int main()
{
int gd=DETECT, gm, color;
char a[50];
initgraph(&gd, &gm,"c:\\tc\\bgi");
putpixel(320, 240, RED);
color=getpixel(320,240);
cout     (a,"color of pixel at location (320,240) is %d",color);
outtextxy(30, 100, a);
getch();
closegraph();
}
```

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



```
#include<graphics.h>
#include<conio.h>
int main () {
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
 putpixel(320, 240, RED);
getch();
closegraph();
 return 0;
}
```

HOMEWORK What is the output of this code?

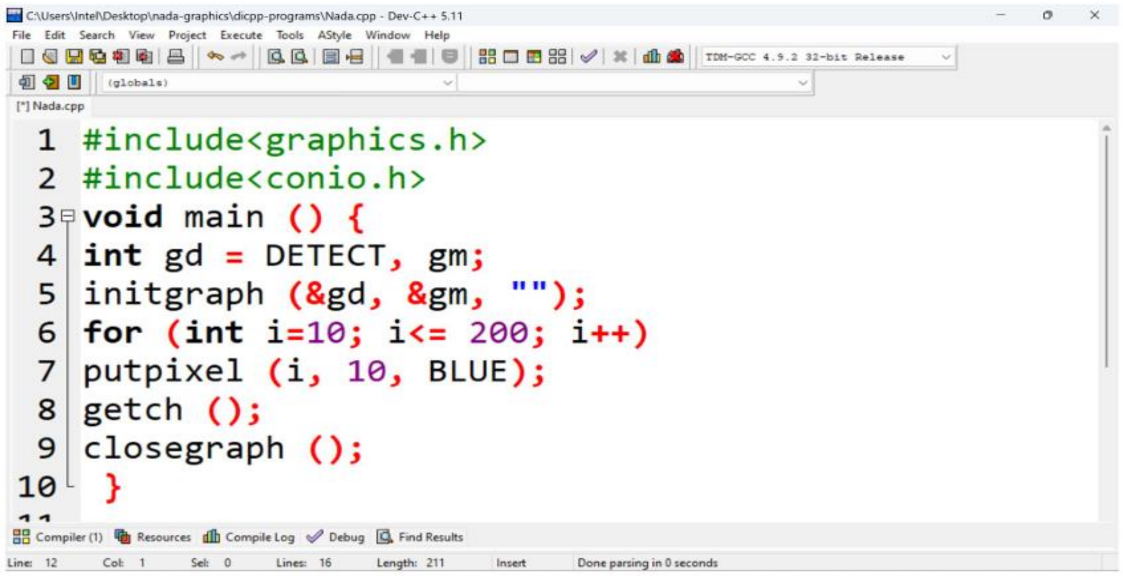


```
// Implementation for putpixel() in C++
#include <graphics.h>
#include <stdio.h>
int main()
{
    int gd = DETECT, gm, color;
    initgraph(&gd, &gm, "");
    setbkcolor(15);
    cleardevice();
    putpixel(85, 35, GREEN);
    putpixel(30, 40, RED);
    putpixel(115, 50, YELLOW);
    putpixel(135, 50, CYAN);
    putpixel(45, 60, BLUE);
    putpixel(20, 100, WHITE);
    putpixel(200, 100, LIGHTBLUE);
    putpixel(150, 100, LIGHTGREEN);
    putpixel(200, 50, YELLOW);
    putpixel(120, 70, RED);

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

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).


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
Compiler (1) Resources Compile Log Debug Find Results
Line: 12 Col: 1 Sel: 0 Lines: 16 Length: 211 Insert Done parsing in 0 seconds

```
#include<graphics.h>
#include<conio.h>
void main () {
int gd = DETECT, gm;
initgraph (&gd, &gm, "");
for (int i=10; i<= 200; i++)
putpixel (i, 10, BLUE);
getch ();
closegraph ();
 }
```

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

Syntax : int getmaxy();

example:

// C++ Implementation for getmaxx() and getmaxy()

#include <graphics.h>

#include <stdio.h>

#include<iostream.h>

Void main ()

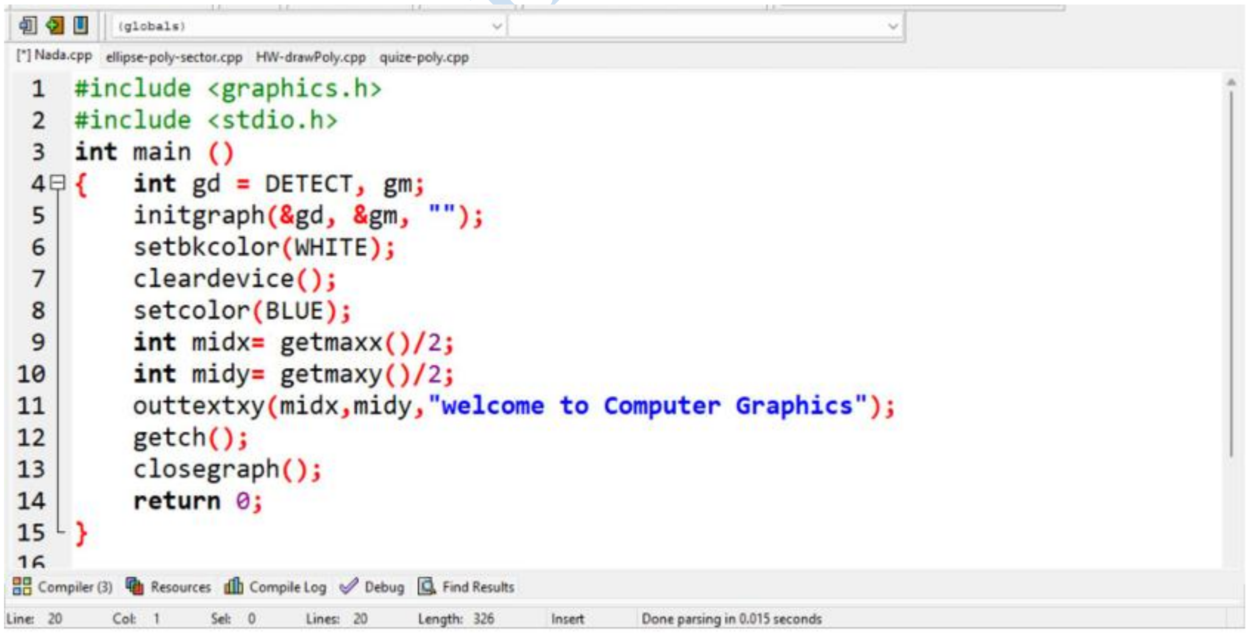
{ int gd = DETECT, gm;

```
char array [100];
initgraph(&gd, &gm, "");
      int maxx= getmaxx();
      int maxy= getmaxy();
cout(array, "Maximum X coordinate and Maximum Y coordinate for current
    Graphics Mode and Driver  = %d,%d", maxx,maxy);
outtext(array);
getch();
closegraph();
}
```

THE OUTPUT:

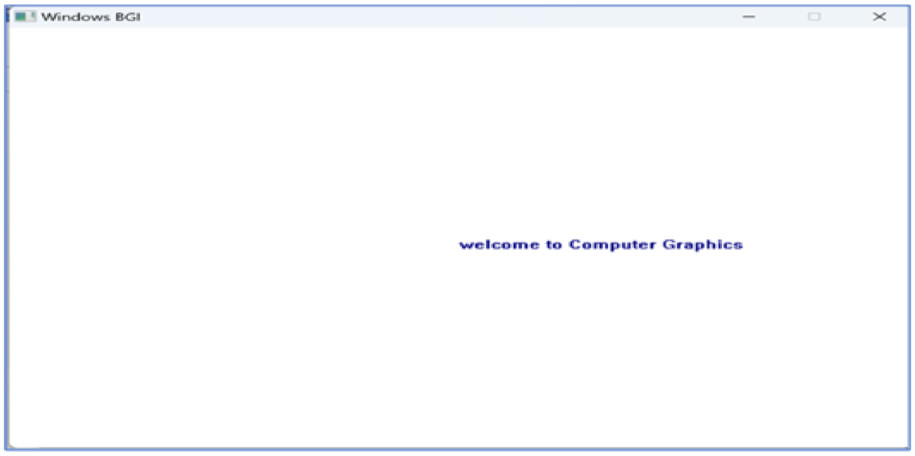
```
Maximum X coordinate and Maximum coordinate for current
Graphics Mode and Driver = 639, 479
```

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



```
#include <graphics.h>
#include <stdio.h>
int main ()
{    int gd = DETECT, gm;
     initgraph(&gd, &gm, "");
     setbkcolor(WHITE);
     cleardevice();
     setcolor(BLUE);
     int midx= getmaxx()/2;
     int midy= getmaxy()/2;
     outtextxy(midx,midy,"welcome to Computer Graphics");
     getch();
     closegraph();
     return 0;
}
```

THE OUTPUT:



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

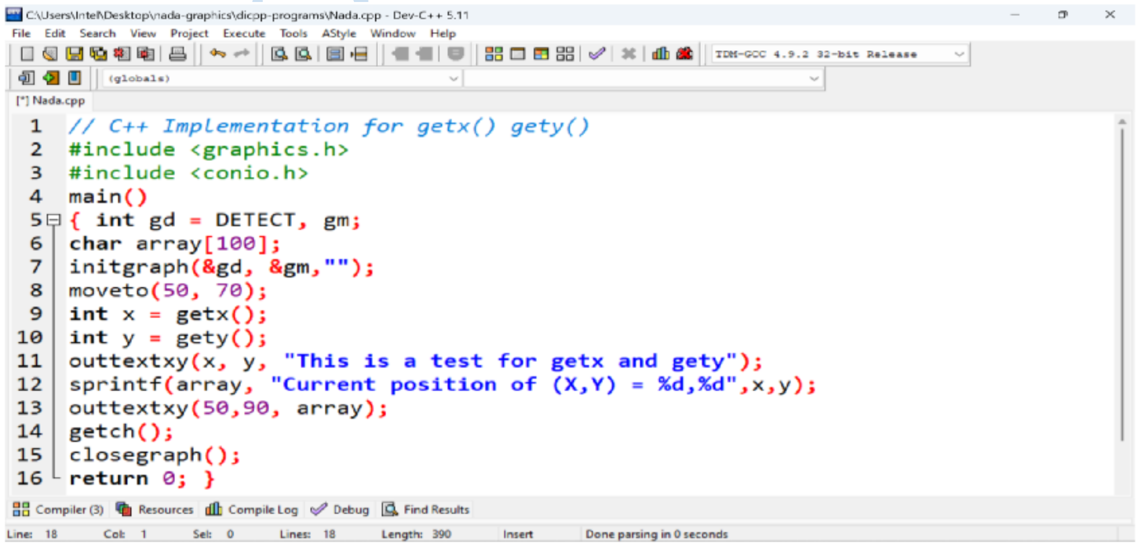
Syntax : int getx();

➢ 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).



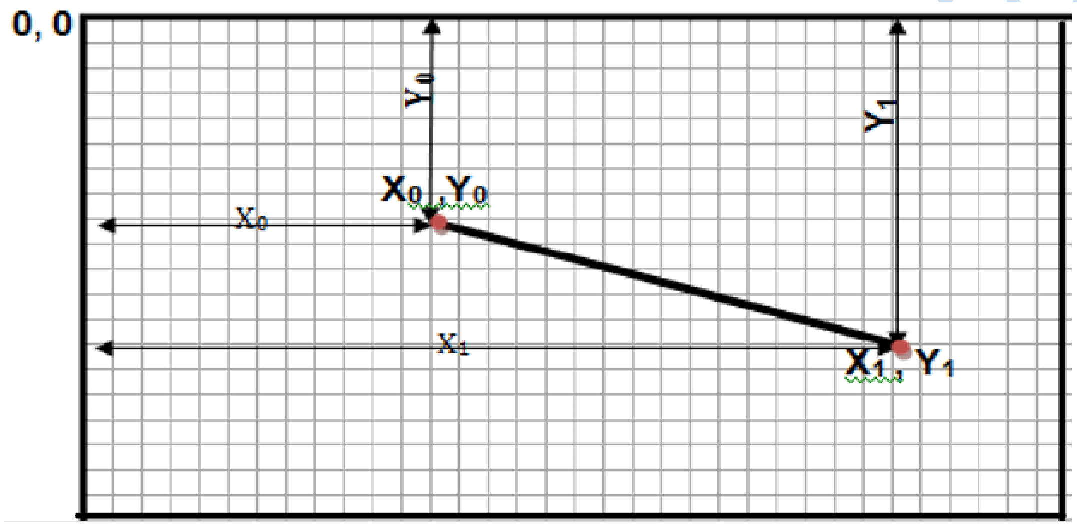
```
// C++ Implementation for getx() gety()
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm;
char array[100];
initgraph(&gd, &gm,"");
moveto(50, 70);
int x = getx();
int y = gety();
outtextxy(x, y, "This is a test for getx and gety");
sprintf(array, "Current position of (X,Y) = %d,%d",x,y);
outtextxy(50,90, array);
getch();
closegraph();
return 0; }
```

THE OUTPUT: This is a test for getx and gety Current position of **(X, Y) =50,70**

Line functions in C++

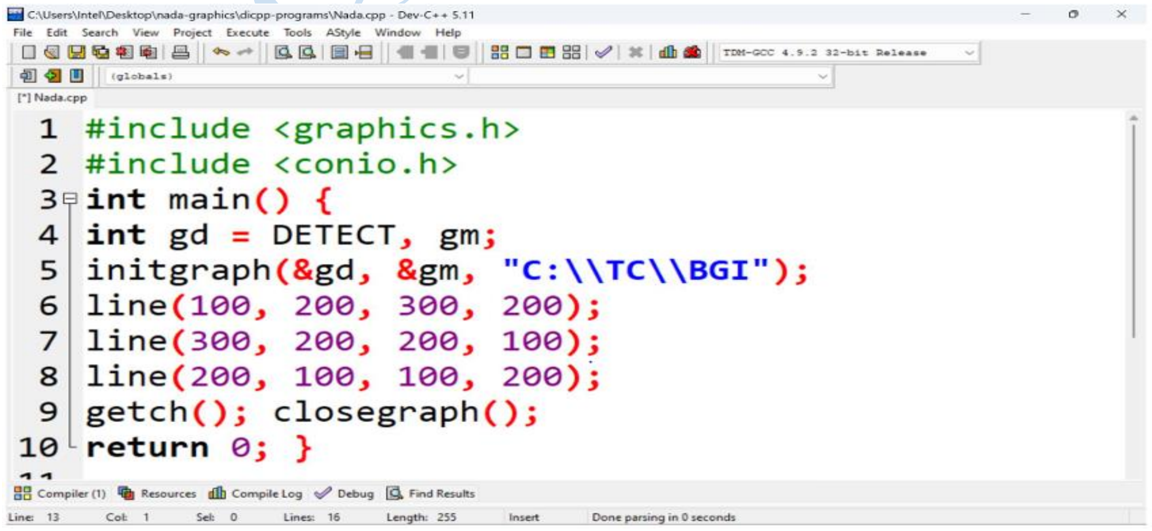
3-line () function:

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);



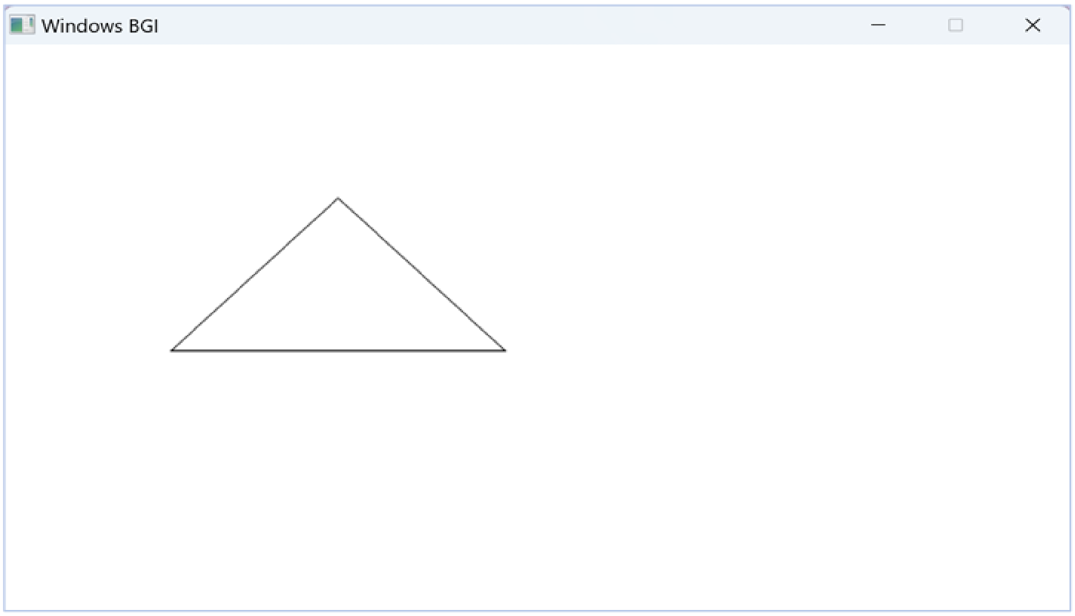
Example: if we want to draw a line from point (10,10) to point (120,120), write the code: **Line(10,10,120,120);**

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).

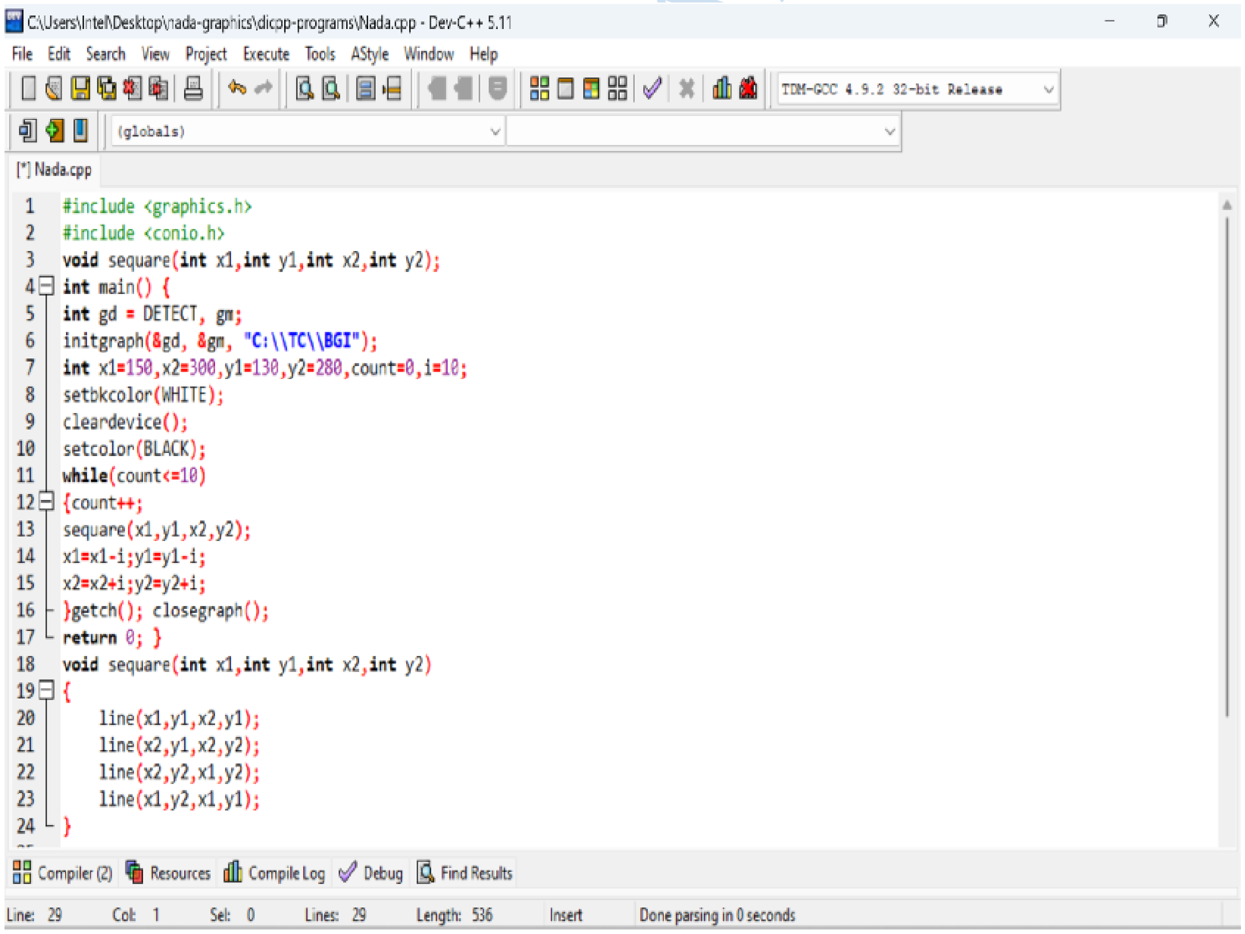

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 32-bit Release
(globals)
[*] Nada.cpp
Compiler (1) Resources Compile Log Debug Find Results
Line: 13 Col: 1 Sel: 0 Lines: 16 Length: 255 Insert Done parsing in 0 seconds

```
#include <graphics.h>
#include <conio.h>
int main() {
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
line(100, 200, 300, 200);
line(300, 200, 200, 100);
line(200, 100, 100, 200);
getch(); closegraph();
return 0; }
```

THE OUTPUT:

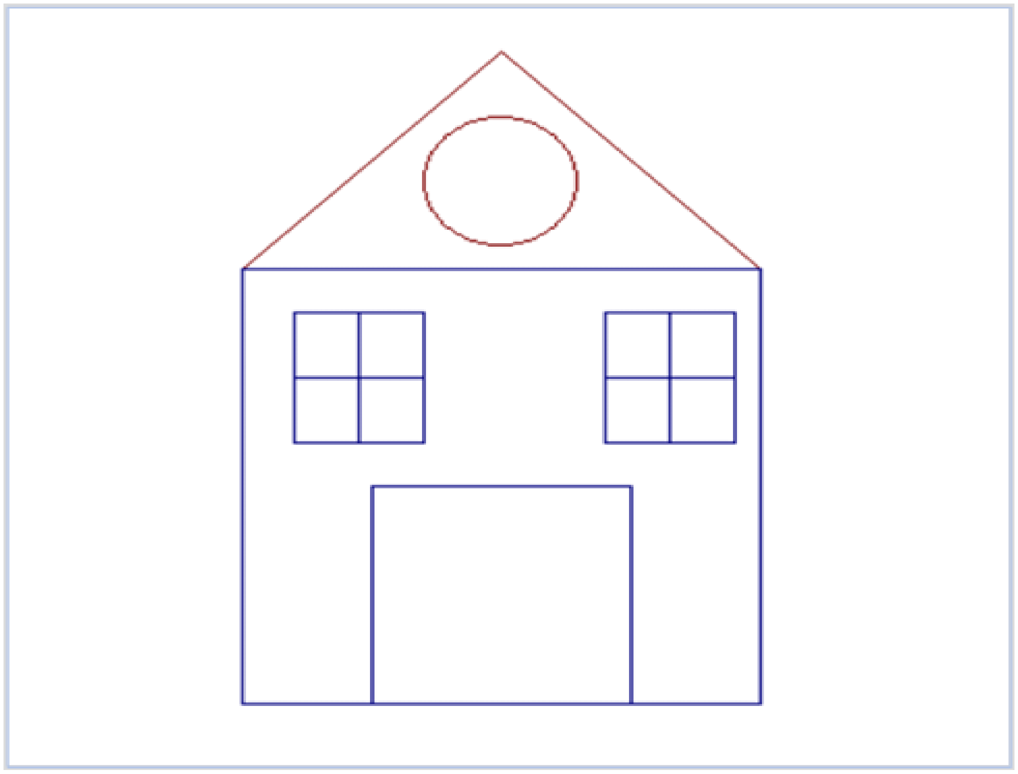


Exercise: write a C++ program to draw the following figure using Line () function.


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
Compiler (2) Resources Compile Log Debug Find Results
Line: 29 Col: 1 Sel: 0 Lines: 29 Length: 536 Insert Done parsing in 0 seconds

```
#include <graphics.h>
#include <conio.h>
void square(int x1,int y1,int x2,int y2);
int main() {
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
int x1=150,x2=300,y1=130,y2=280,count=0,i=10;
setbkcolor(WHITE);
cleardevice();
setcolor(BLACK);
while(count<=10)
{count++;
sequare(x1,y1,x2,y2);
x1=x1-i;y1=y1-i;
x2=x2+i;y2=y2+i;
}getch(); closegraph();
return 0; }
void sequare(int x1,int y1,int x2,int y2)
{
    line(x1,y1,x2,y1);
    line(x2,y1,x2,y2);
    line(x2,y2,x1,y2);
    line(x1,y2,x1,y1);
}
```

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);

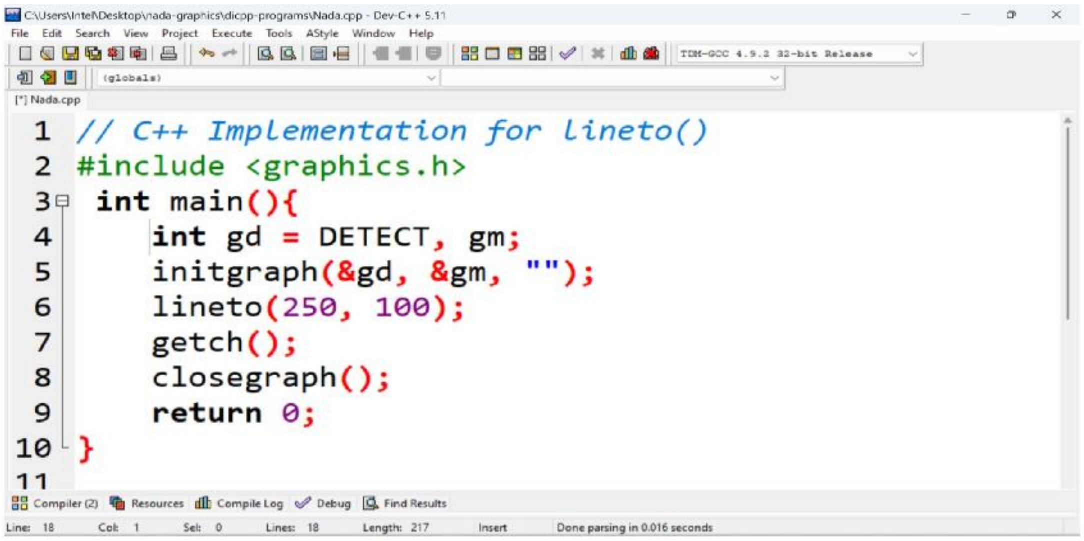
Where, (x, y) are the coordinates up to which the line will be drawn from the previous point.

Example: If we want to draw a line from (100,100) to (320,240) using Moveto() and lineto() as follow:

Moveto(100,100);

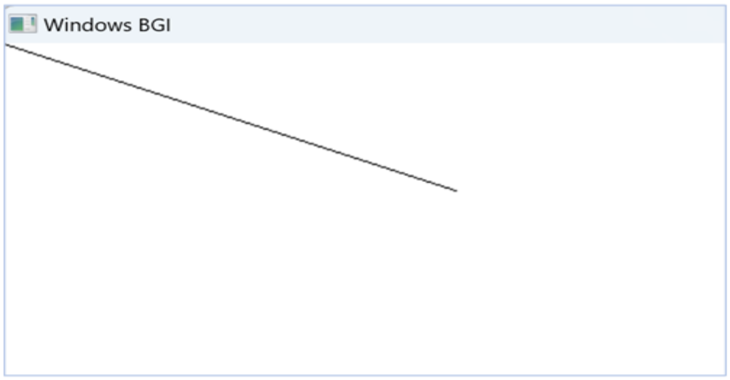
lineto(320,240);

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

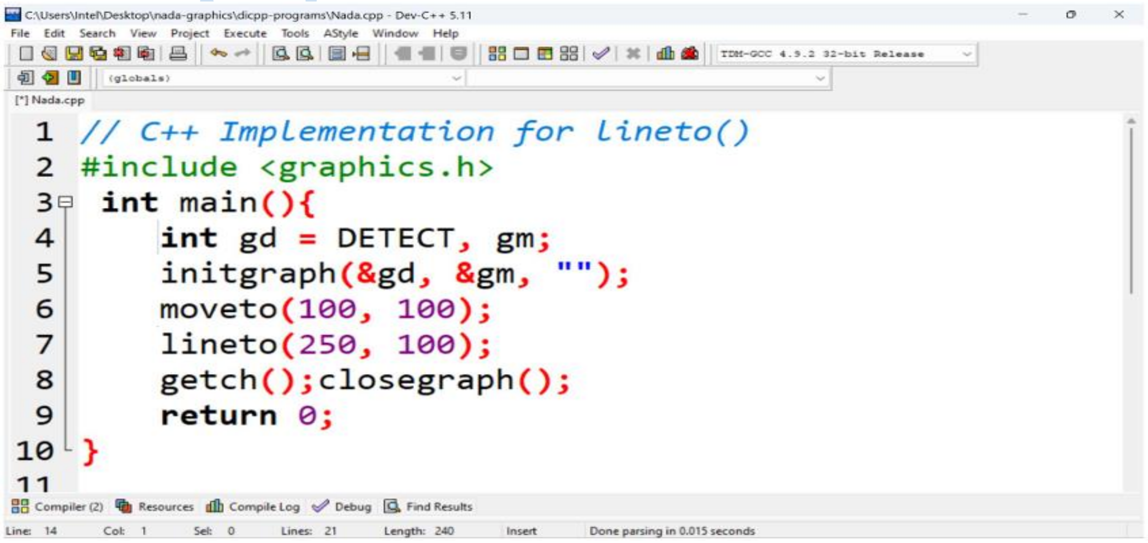


```
// C++ Implementation for lineto()
#include <graphics.h>
int main(){
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    lineto(250, 100);
    getch();
    closegraph();
    return 0;
}
```

THE OUTPUT:

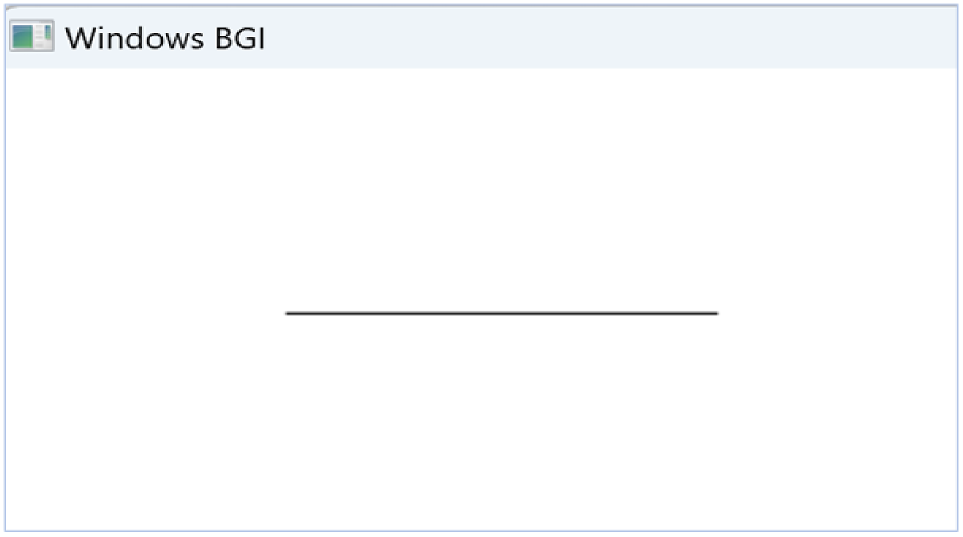


Example: Draw a line from position (100,100) to point (250,100).

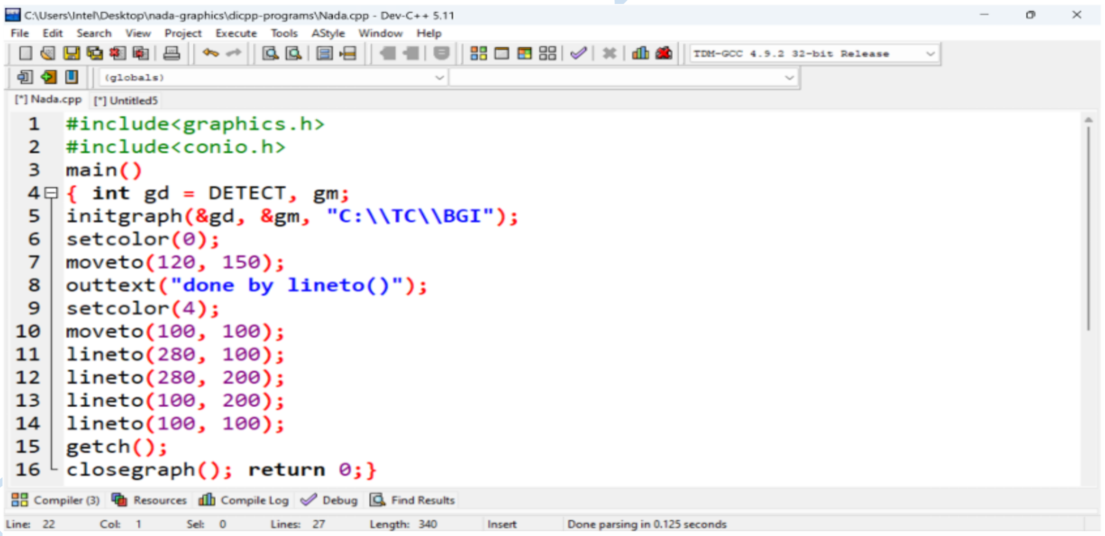


```
// C++ Implementation for lineto()
#include <graphics.h>
int main(){
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "");
    moveto(100, 100);
    lineto(250, 100);
    getch();closegraph();
    return 0;
}
```

THE OUTPUT:

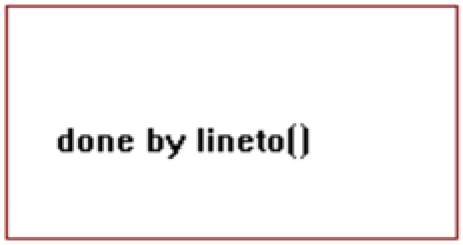


Exercise: Write a C++ program to draw a red rectangle using lineto() function, its coordinates from the top left angle are (100,100) and the coordinates from the bottom right angle are (280,200).


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
Compiler (3) Resources Compile Log Debug Find Results
Line: 22 Col: 1 Sel: 0 Lines: 27 Length: 340 Insert Done parsing in 0.125 seconds

```
#include<graphics.h>
#include<conio.h>
main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
setcolor(0);
moveto(120, 150);
outtext("done by lineto()");
setcolor(4);
moveto(100, 100);
lineto(280, 100);
lineto(280, 200);
lineto(100, 200);
lineto(100, 100);
getch();
closegraph(); return 0;}
```

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);

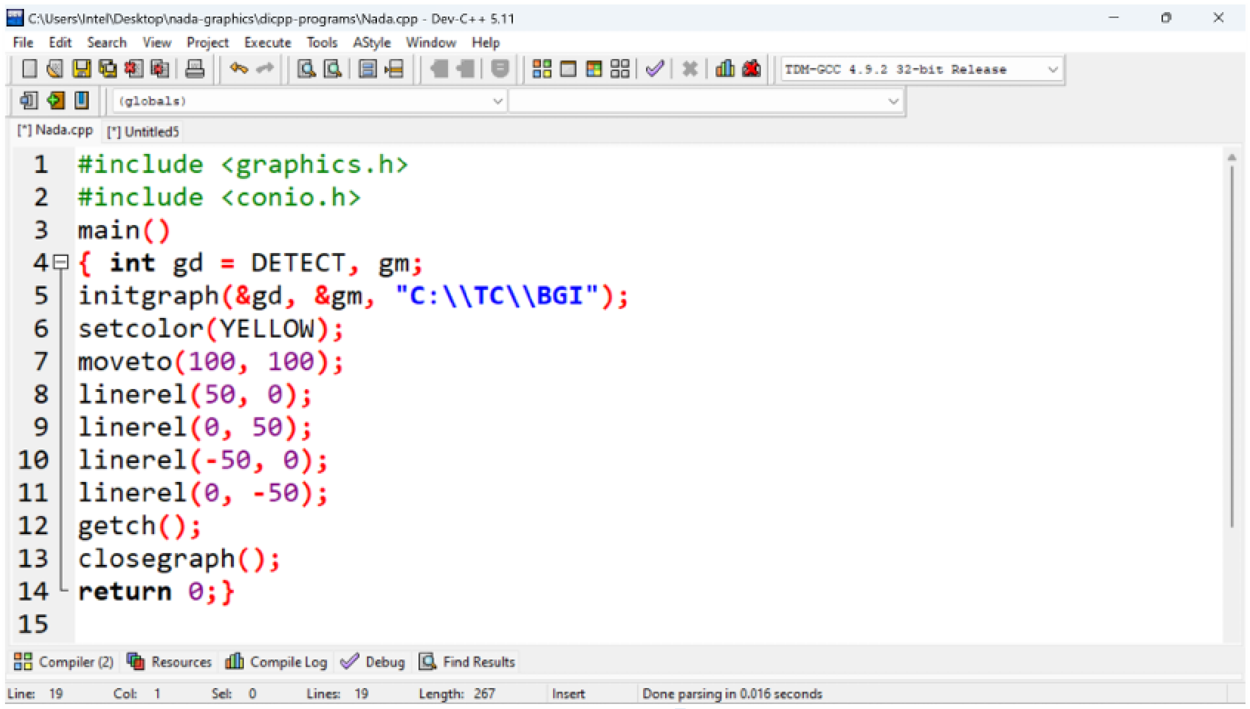
Moverel(100,30);

Linerel(320,240);

Solution:

Line(120,60,440,300);

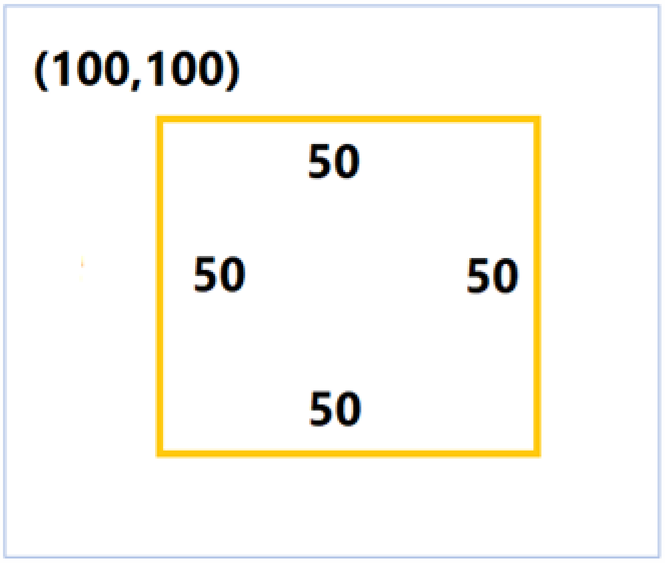
Exercise: Write a C++ program to draw a yellow square with a radius of (50), its coordinates start from (100,100). Using lineto() function.


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

```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
setcolor(YELLOW);
moveto(100, 100);
linerel(50, 0);
linerel(0, 50);
linerel(-50, 0);
linerel(0, -50);
getch();
closegraph();
return 0;}
```

Line: 19 Col: 1 Sel: 0 Lines: 19 Length: 267 Insert Done parsing in 0.016 seconds

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”);

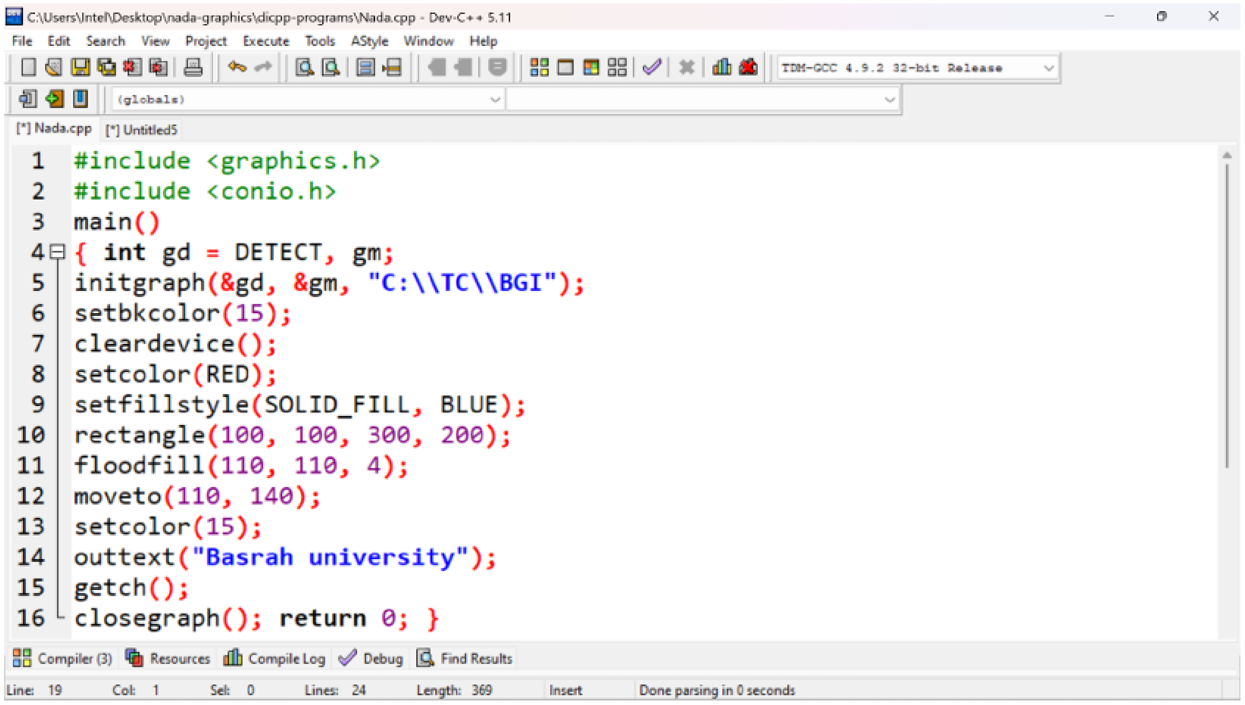
Example: if we want to write “Computer Graphics” on the position

(100,100).

moveto(100,100);

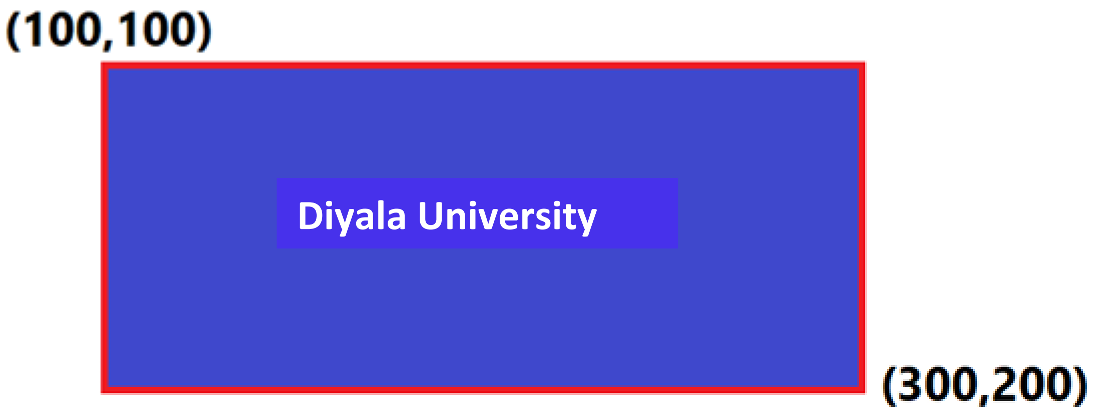
outtext(“Computer Graphics”);

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).

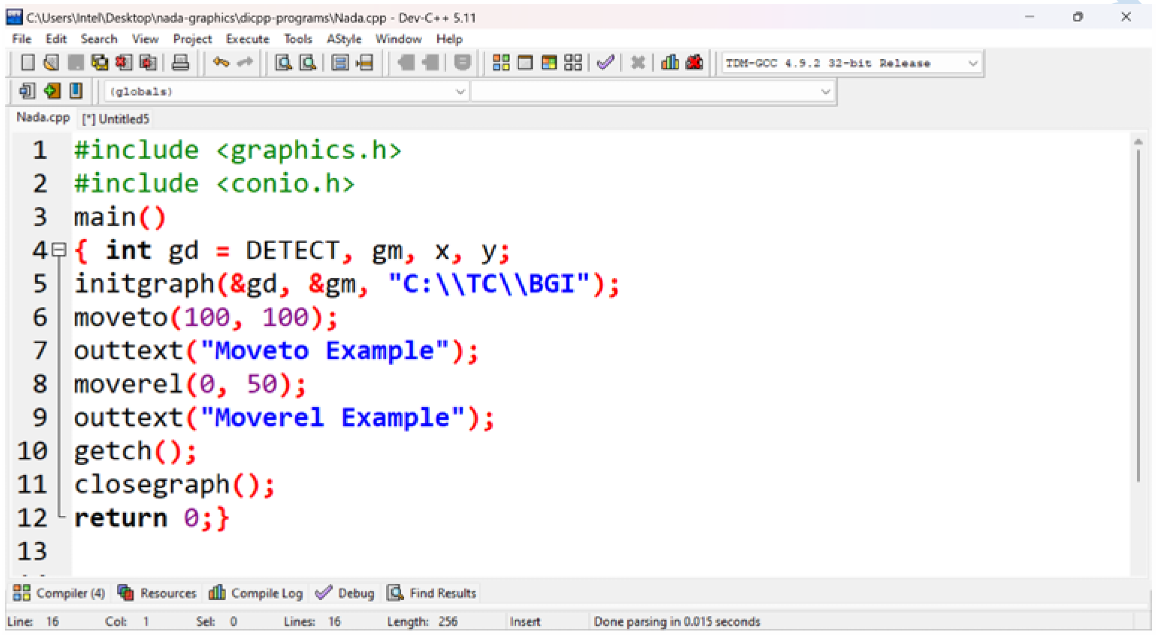
Note: getx and gety can be used to find the current position.

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);

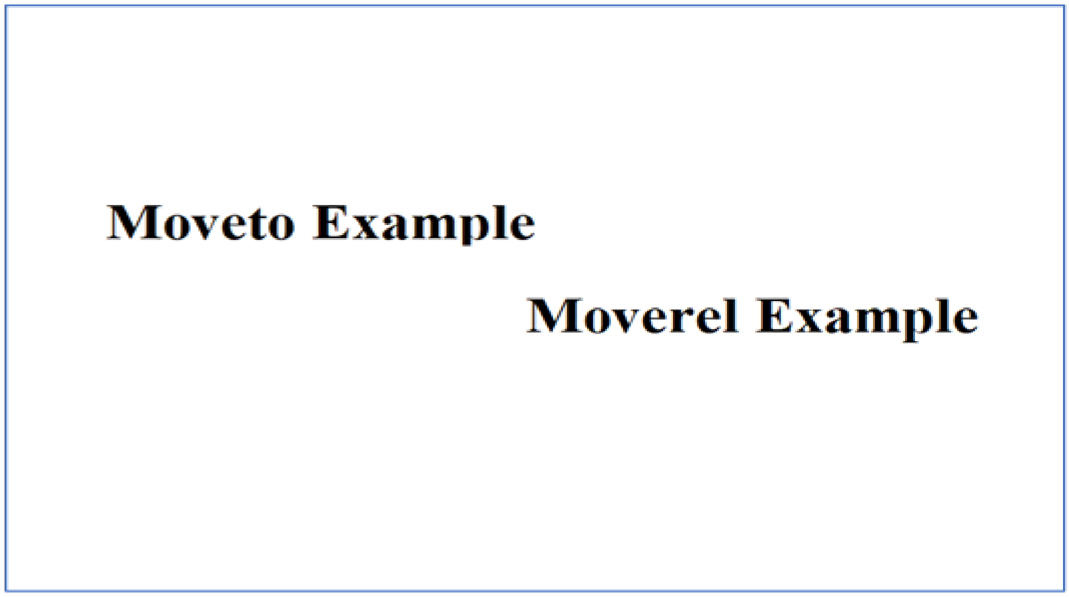
Exercise: write a C++ program to print " Moveto Example" at the point(100,100).then move the cursor to the new position(100,150) using the function moverel() then write the sentence "Moverel Example".



```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm, x, y;
initgraph(&gd, &gm, "C:\\TC\\BGI");
moveto(100, 100);
outtext("Moveto Example");
moverel(0, 50);
outtext("Moverel Example");
getch();
closegraph();
return 0;}
```

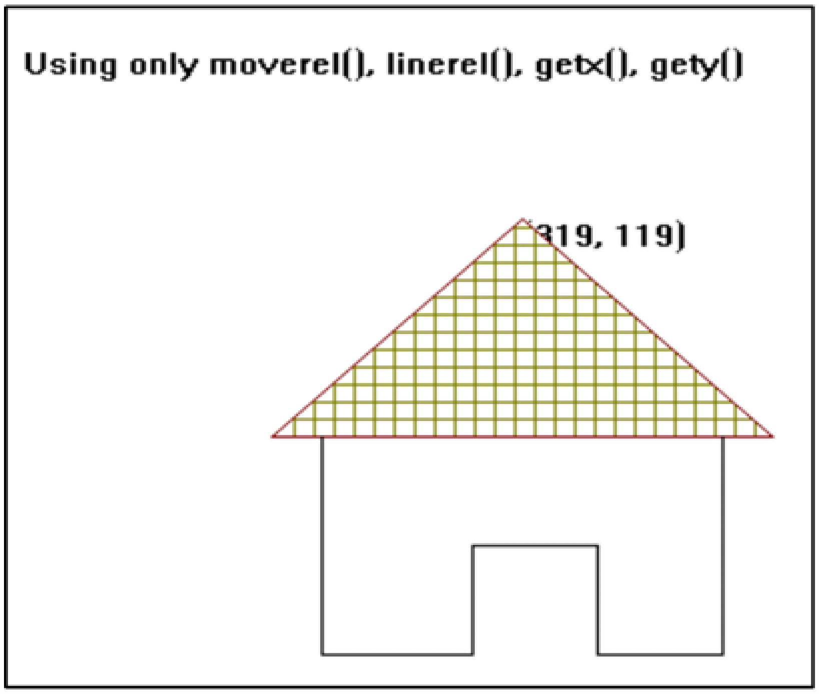
Line: 16 Col: 1 Sel: 0 Lines: 16 Length: 256 Insert Done parsing in 0.015 seconds

THE OUTPUT:



```
Moveto Example
                    Moverel Example
```

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



```
#include <graphics.h>
#include<iostream.h>
#include<conio.h>
void 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();
cout<<(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();
}
```

C o l o r s F u n c t i o n s دوال الالوان

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.

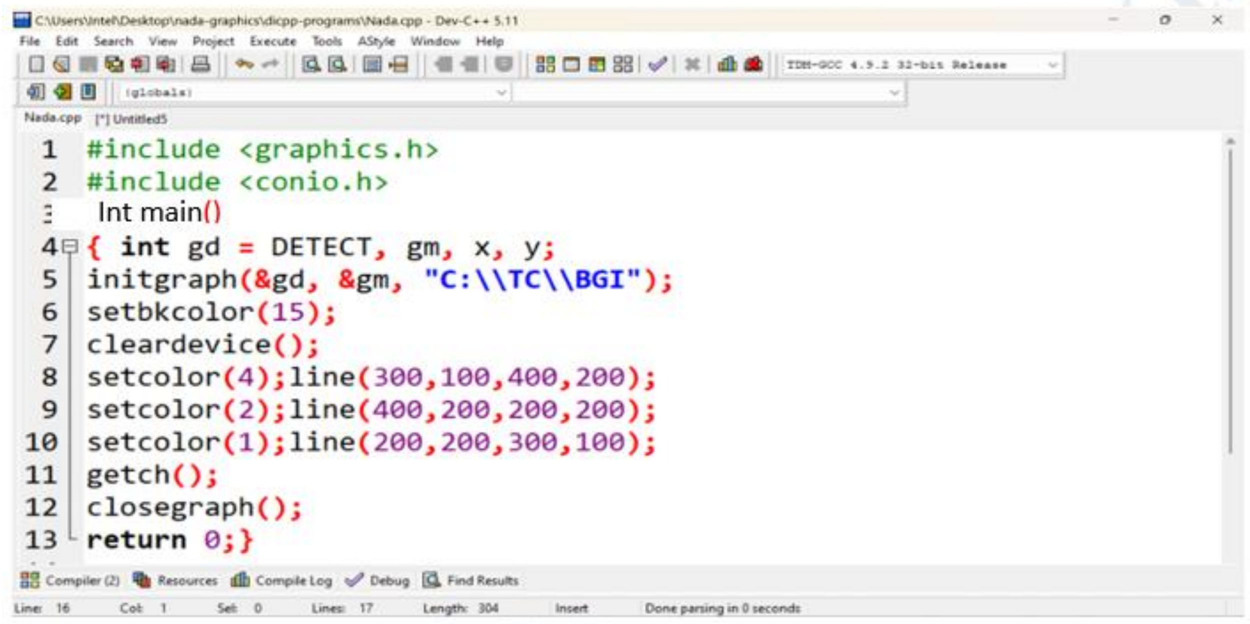
For example, setcolor(RED) or setcolor(4) changes the current drawing color to RED. Remember that the default drawing color is WHITE. The Colors table is given below.

COLORS (VALUES)

BLACK (0)	LIGHTBLUE (9)
BLUE (1)	LIGHTGREEN (10)
GREEN (2)	LIGHTCYAN (11)
CYAN (3)	LIGHTRED (12)
RED (4)	LIGHTMAGENTA (13)
MAGENTA (5)	YELLOW (14)

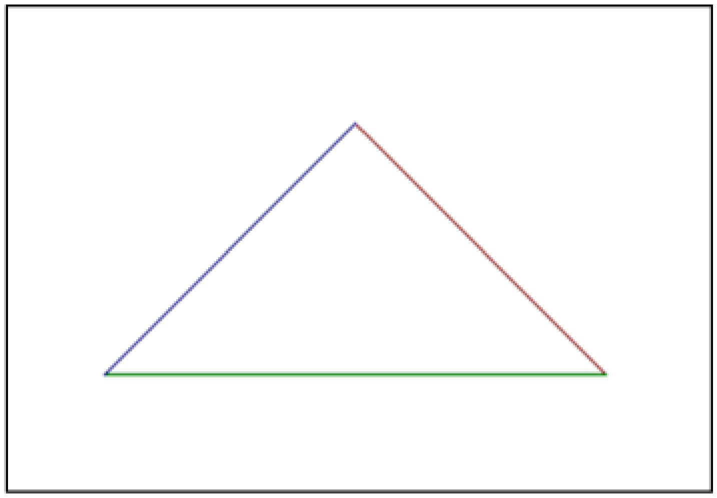
BROWN (6)	WHITE (15)
LIGHTGRA (7)	

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



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

THE OUTPUT:

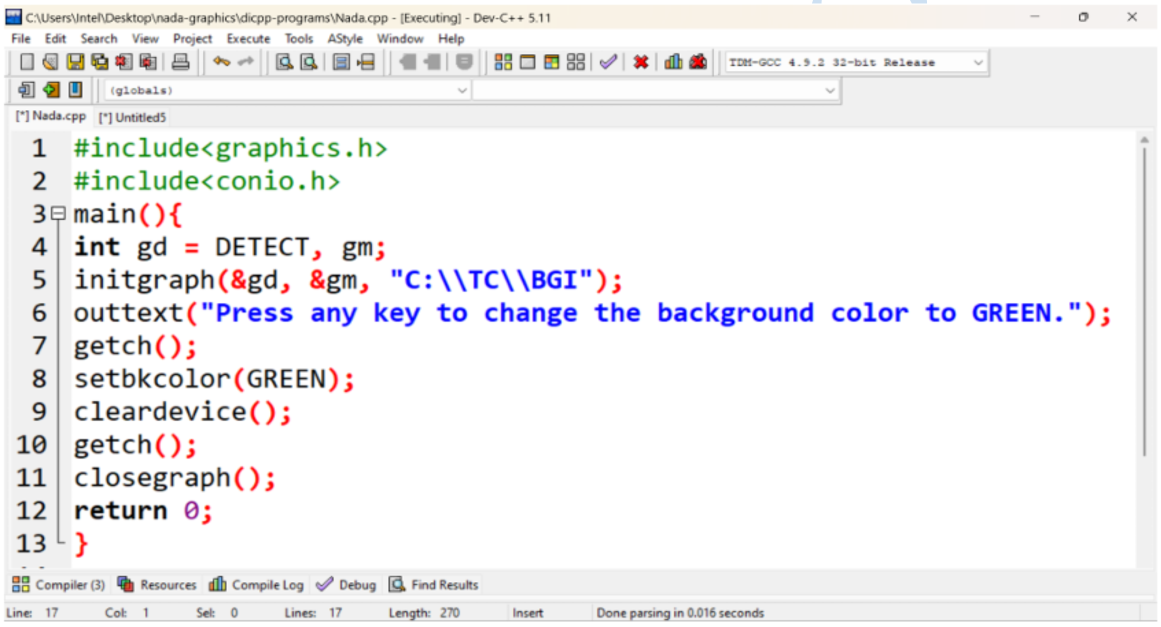


10. Setbkcolor() function

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);

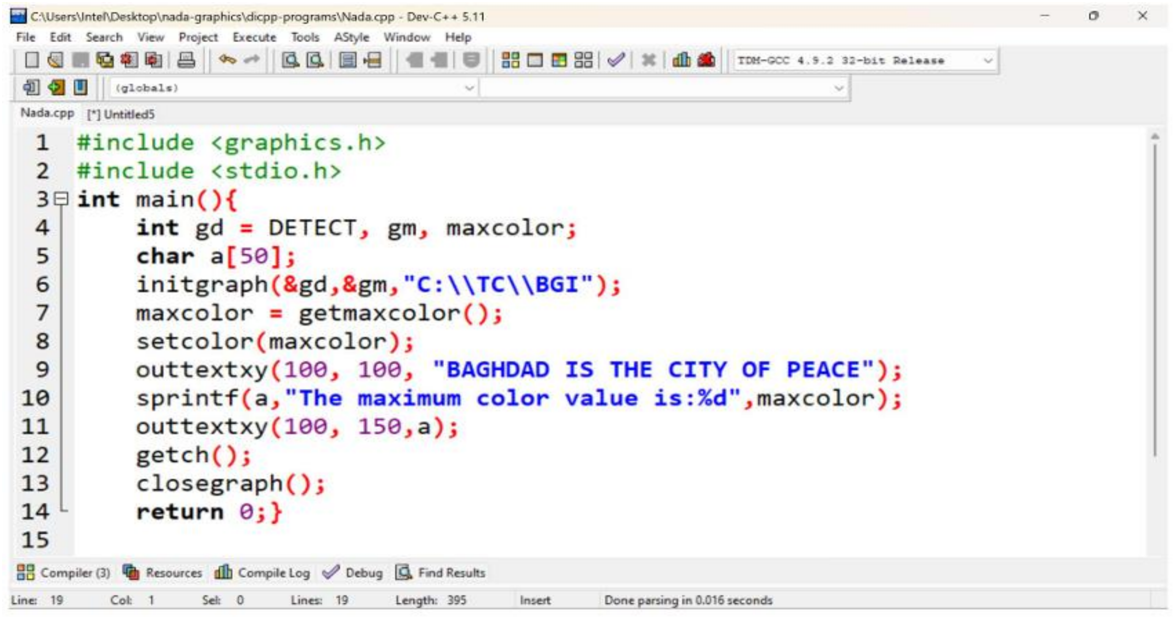
Example: in this program when we Press any key the background color change to GREEN.



```
#include<graphics.h>
#include<conio.h>
main(){
int gd = DETECT, gm;
initgraph(&gd, &gm, "C:\\TC\\BGI");
outtext("Press any key to change the background color to GREEN.");
getch();
setbkcolor(GREEN);
cleardevice();
getch();
closegraph();
return 0;
}
```

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - [Executing] - 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
Compiler (3) Resources Compile Log Debug Find Results
Line: 17 Col: 1 Sel: 0 Lines: 17 Length: 270 Insert Done parsing in 0.016 seconds

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).



```
#include <graphics.h>
#include <stdio.h>
int main(){
    int gd = DETECT, gm, maxcolor;
    char a[50];
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    maxcolor = getmaxcolor();
    setcolor(maxcolor);
    outtextxy(100, 100, "BAGHDAD IS THE CITY OF PEACE");
    sprintf(a,"The maximum color value is:%d",maxcolor);
    outtextxy(100, 150,a);
    getch();
    closegraph();
    return 0;}
```

THE OUTPUT:

```
BAGHDAD IS THE CITY OF PEACE
```

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

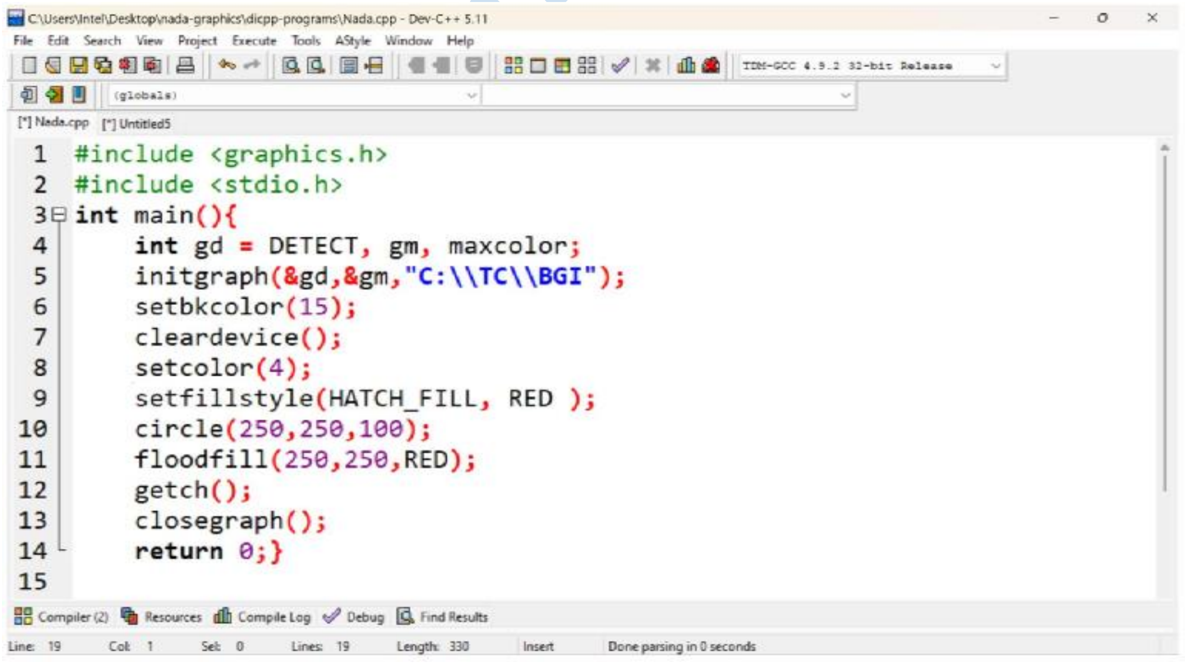
Exercise:

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

pattern = HATCH_FILL, Color = RED

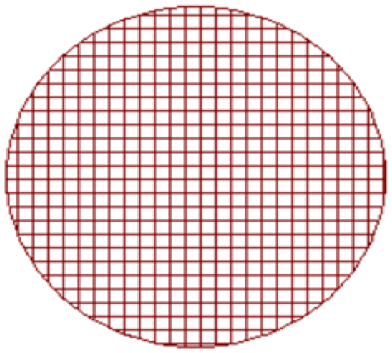
circle : x = 250, y = 250, radius = 100

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
Compiler (2) Resources Compile Log Debug Find Results
Line: 19 Col: 1 Sel: 0 Lines: 19 Length: 330 Insert Done parsing in 0 seconds

```
#include <graphics.h>
#include <stdio.h>
int main(){
     int gd = DETECT, gm, maxcolor;
     initgraph(&gd,&gm,"C:\\TC\\BGI");
     setbkcolor(15);
     cleardevice();
     setcolor(4);
     setfillstyle(HATCH_FILL, RED );
     circle(250,250,100);
     floodfill(250,250,RED);
     getch();
     closegraph();
     return 0;}
```

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().

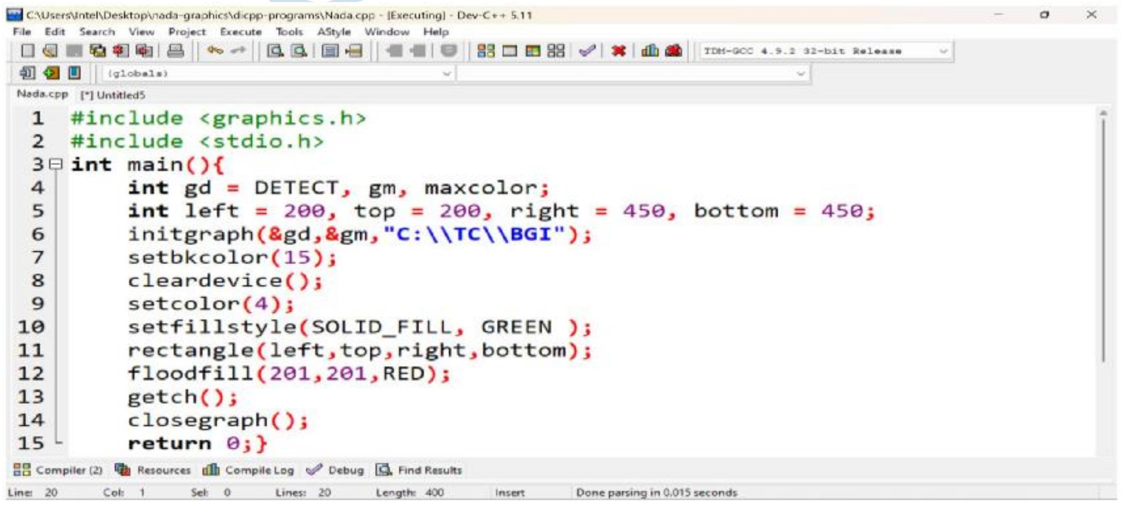
Exercise:

Write a c++ program to draw a rectangle filled with:

pattern = SOLID_FILL, Color = GREEN.

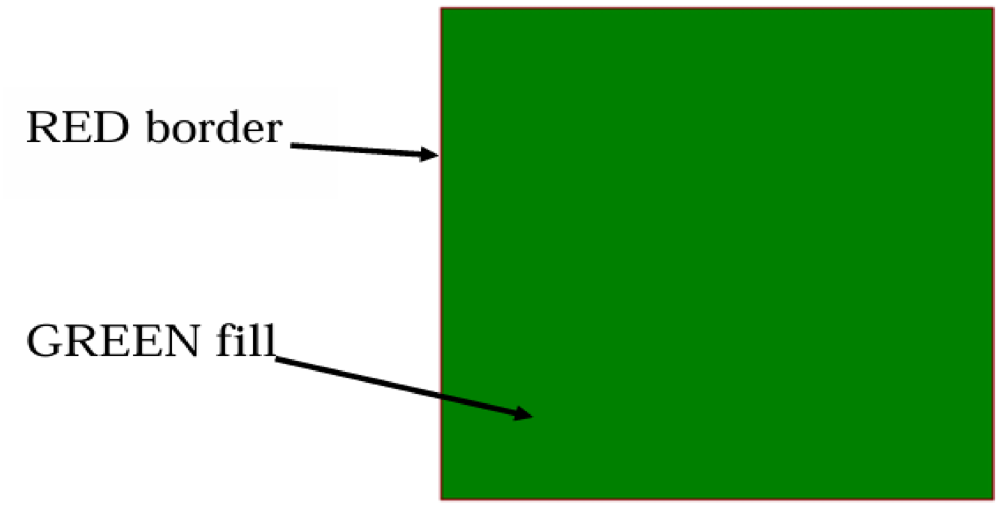
rectangle: left = 200, top = 200, right = 450, bottom = 450.

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

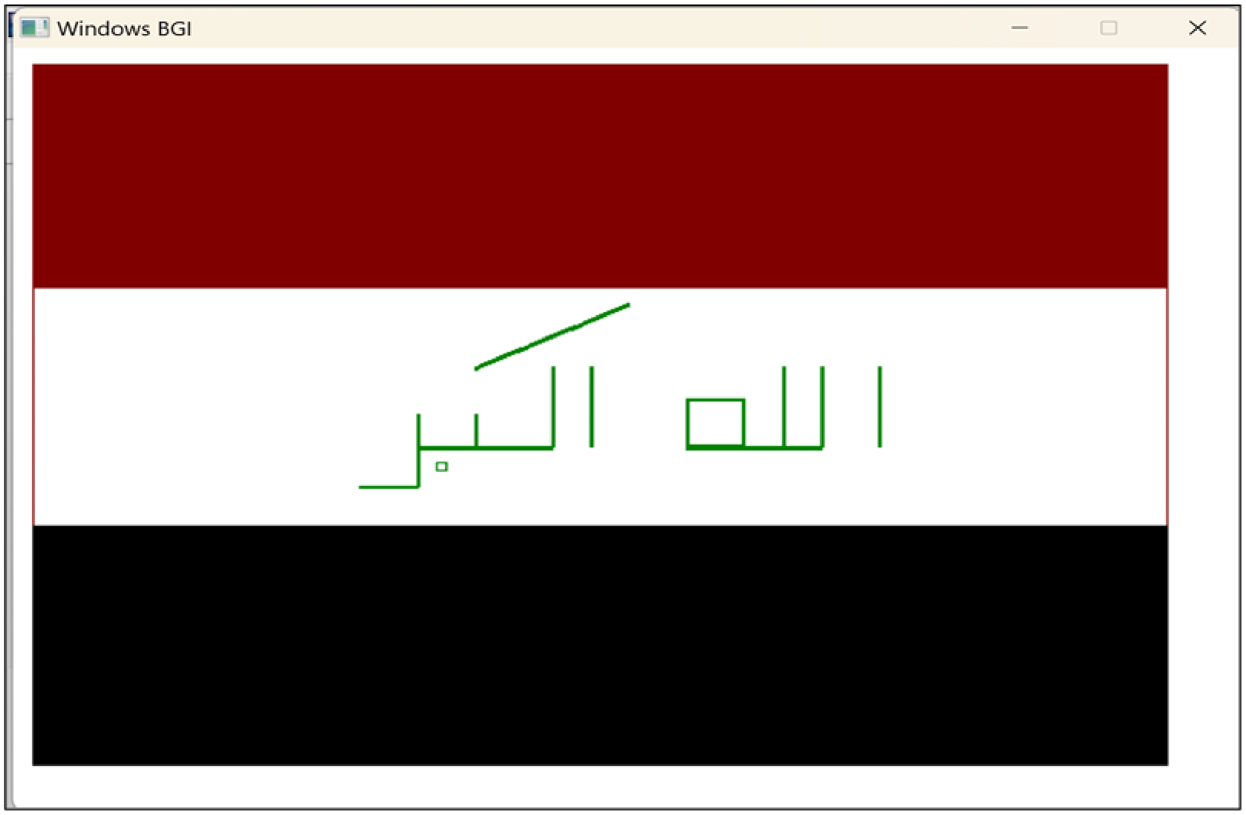


```
#include <graphics.h>
#include <stdio.h>
int main(){
    int gd = DETECT, gm, maxcolor;
    int left = 200, top = 200, right = 450, bottom = 450;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    setbkcolor(15);
    cleardevice();
    setcolor(4);
    setfillstyle(SOLID_FILL, GREEN );
    rectangle(left,top,right,bottom);
    floodfill(201,201,RED);
    getch();
    closegraph();
    return 0;}
```

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.

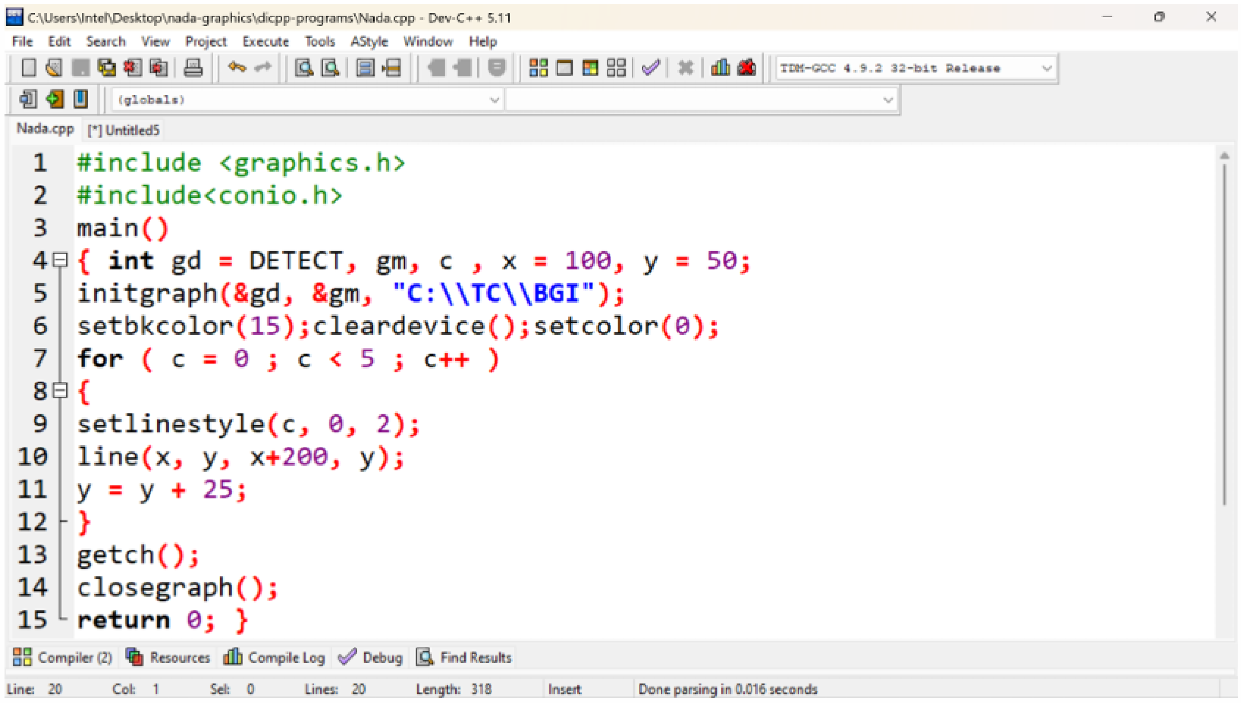
Syntax: **Void setlinestyle(int linestyle,unsigned pattern,int thickness);**

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

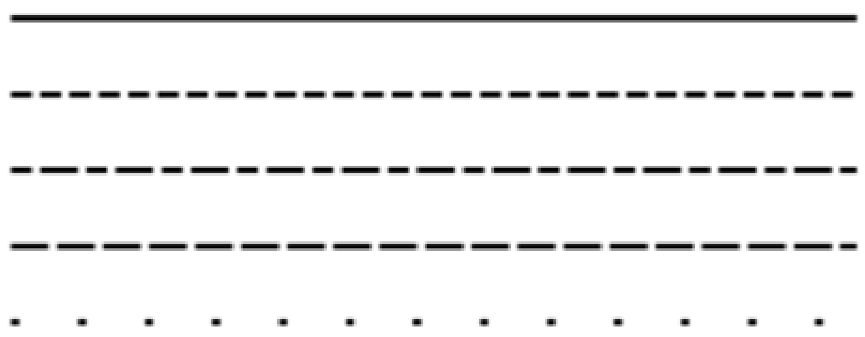
An unsigned pattern is a 16-bit pattern that applies only if linestyle is USERBIT_LINE (4). In that case, whenever a bit in the pattern word is 1, the corresponding pixel in the line is drawn in the current drawing color. A value for an 'unsigned pattern' must always be supplied. It is simply ignored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



```
#include <graphics.h>
#include<conio.h>
main()
{ int gd = DETECT, gm, c , x = 100, y = 50;
initgraph(&gd, &gm, "C:\\TC\\BGI");
setbkcolor(15);cleardevice();setcolor(0);
for ( c = 0 ; c < 5 ; c++ )
{
setlinestyle(c, 0, 2);
line(x, y, x+200, y);
y = y + 25;
}
getch();
closegraph();
return 0; }
```

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();
cout<<(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.

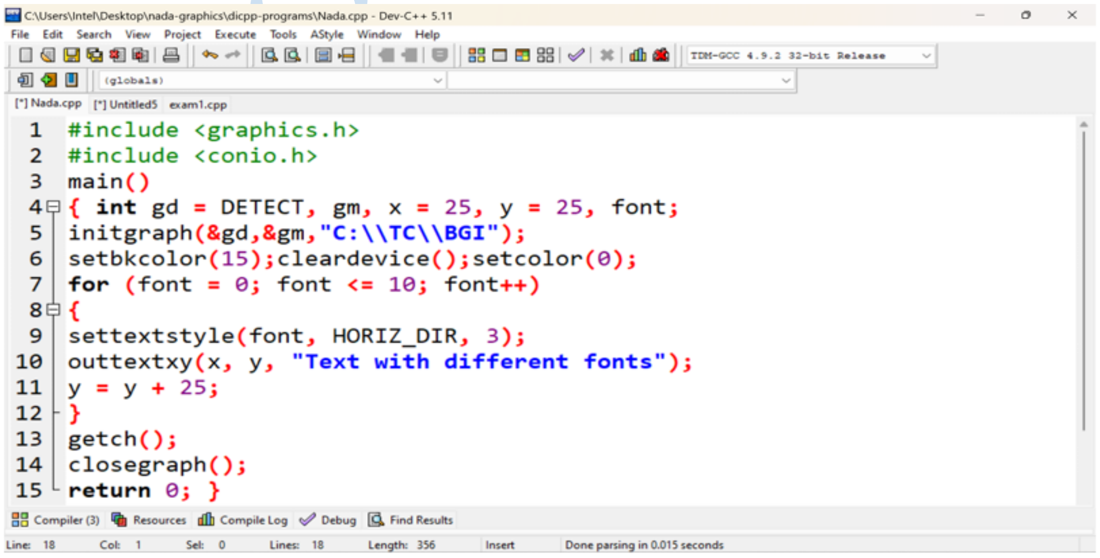
Syntax: void settextstyle(int font, int direction, int font_size);

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

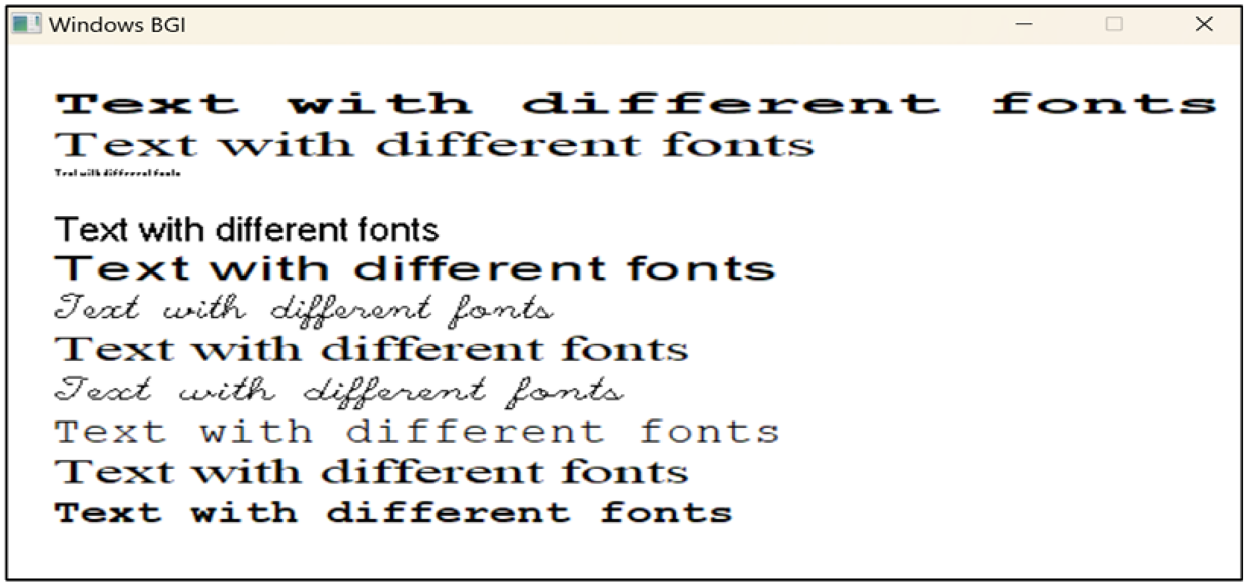
Exercise: Write a C++ program to display "Text with different fonts", with horizontal direction, and different fonts and sizes at position(x,y).



```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm, x = 25, y = 25, font;
initgraph(&gd,&gm,"C:\\TC\\BGI");
setbkcolor(15);cleardevice();setcolor(0);
for (font = 0; font <= 10; font++)
{
settextstyle(font, HORIZ_DIR, 3);
outtextxy(x, y, "Text with different fonts");
y = y + 25;
}
getch();
closegraph();
return 0; }
```

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 exam1.cpp
Compiler (3) Resources Compile Log Debug Find Results
Line: 18 Col: 1 Sel: 0 Lines: 18 Length: 356 Insert Done parsing in 0.015 seconds

THE OUTPUT:



```
Text with different fonts
Text with different fonts
Text with different fonts

Text with different fonts
Text with different fonts
Text with different fonts
Text with different fonts
Text with different fonts
Text with different fonts
Text with different fonts
Text with different fonts
```

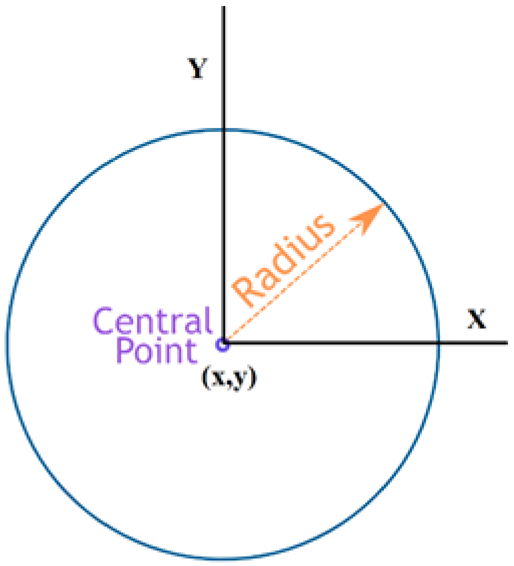
Draw 2D Shapes

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.

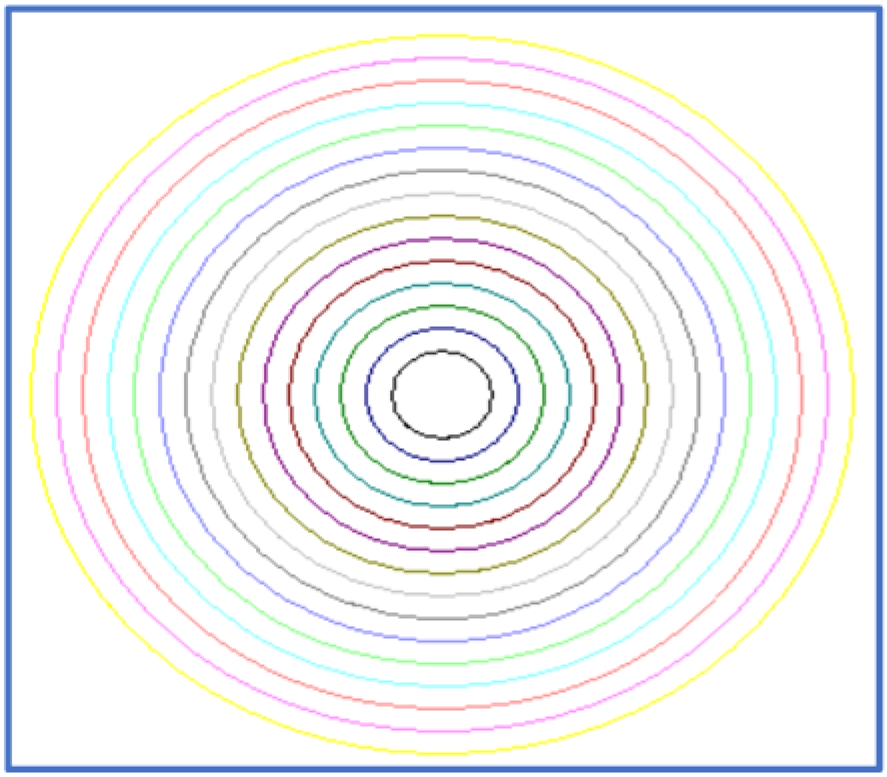
19.Circle() Function Circle function in c++ draws a circle with center at (x, y) and given radius.

Syntax: circle(int x,int y,int radius);

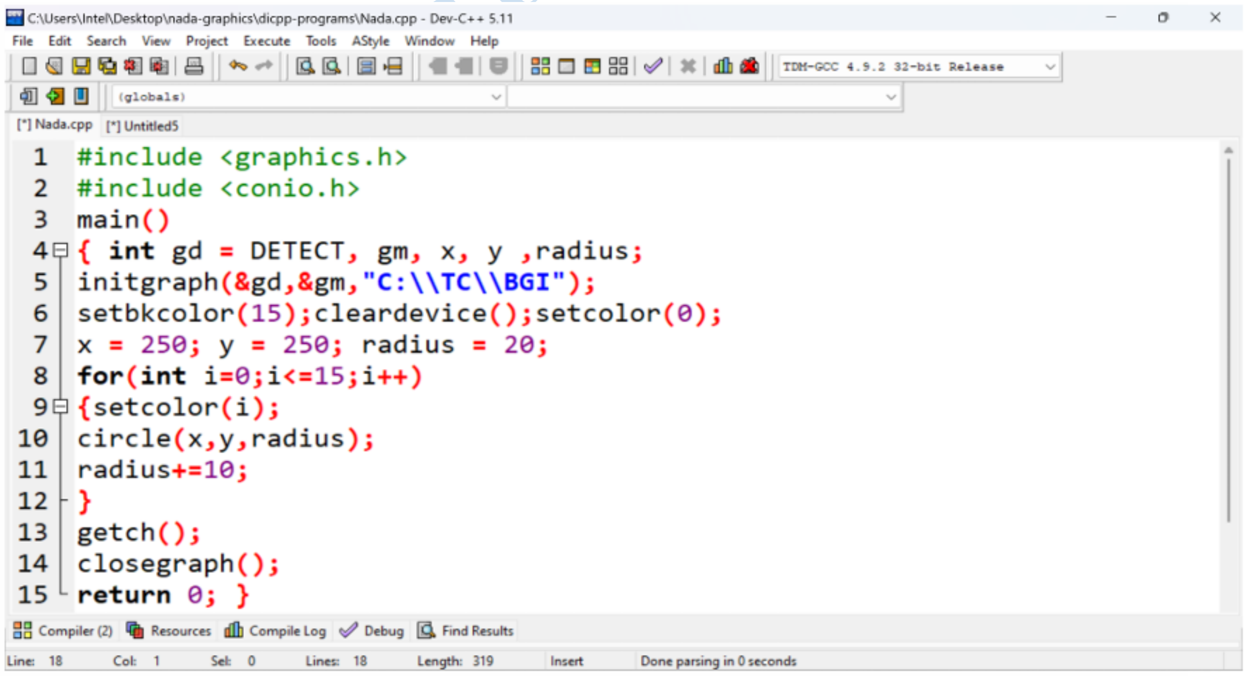
Where, (x, y) is center of the circle, and 'radius' is the Radius of the circle.



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



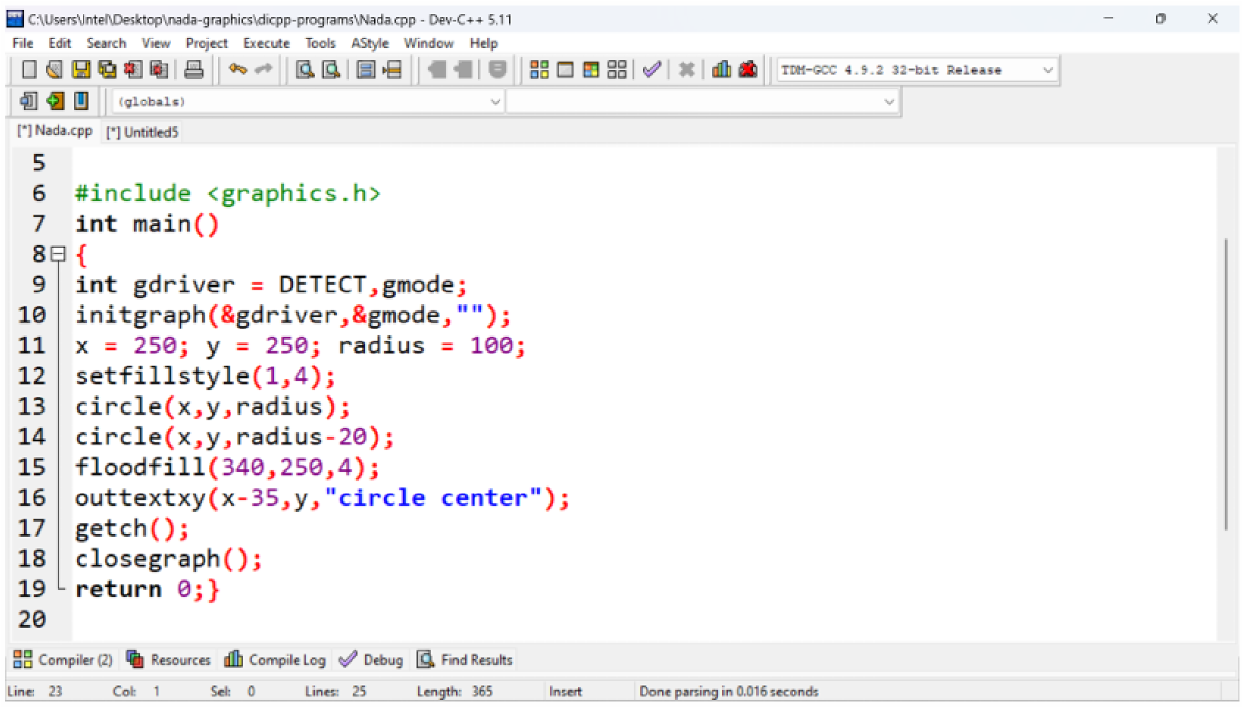
Solution:


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

```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm, x, y ,radius;
initgraph(&gd,&gm,"C:\\TC\\BGI");
setbkcolor(15);cleardevice();setcolor(0);
x = 250; y = 250; radius = 20;
for(int i=0;i<=15;i++)
{setcolor(i);
circle(x,y,radius);
radius+=10;
}
getch();
closegraph();
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:

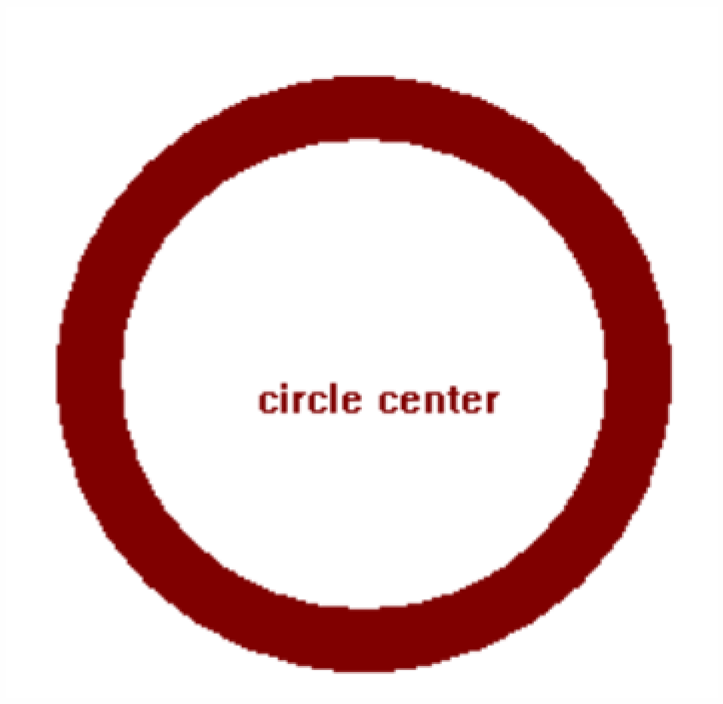

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 32-bit Release
(globals)
[*] Nada.cpp [*] Untitled5
Compiler (2) Resources Compile Log Debug Find Results
Line: 23 Col: 1 Sel: 0 Lines: 25 Length: 365 Insert Done parsing in 0.016 seconds

```

#include <graphics.h>
int main()
{
int gdriver = DETECT,gmode;
initgraph(&gdriver,&gmode,"");
x = 250; y = 250; radius = 100;
setfillstyle(1,4);
circle(x,y,radius);
circle(x,y,radius-20);
floodfill(340,250,4);
outtextxy(x-35,y,"circle center");
getch();
closegraph();
return 0;}

```

Solution:



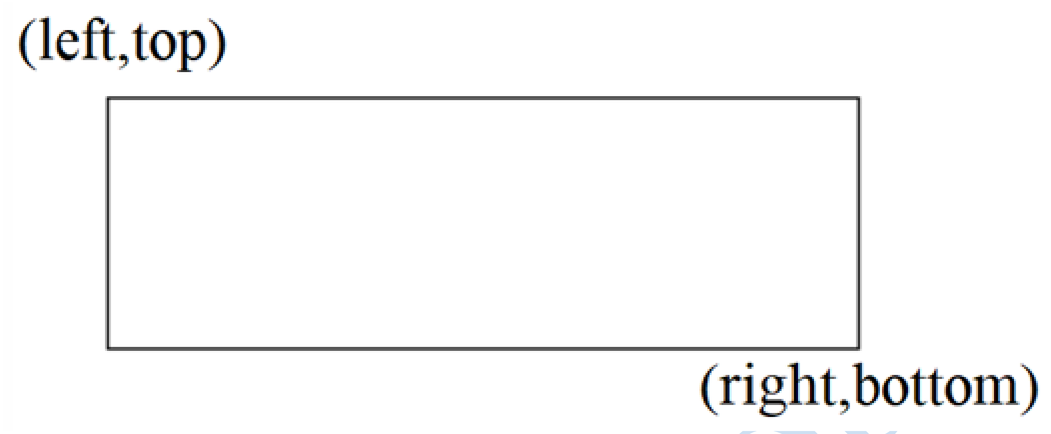
20. Rectangle() Function

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);

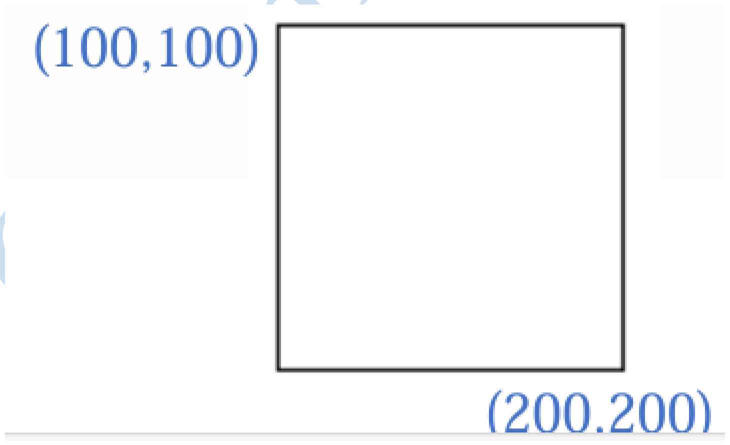


Exercise: use the rectangle () function to draw a square.

Solution:

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

rectangle (left, top, right, bottom);



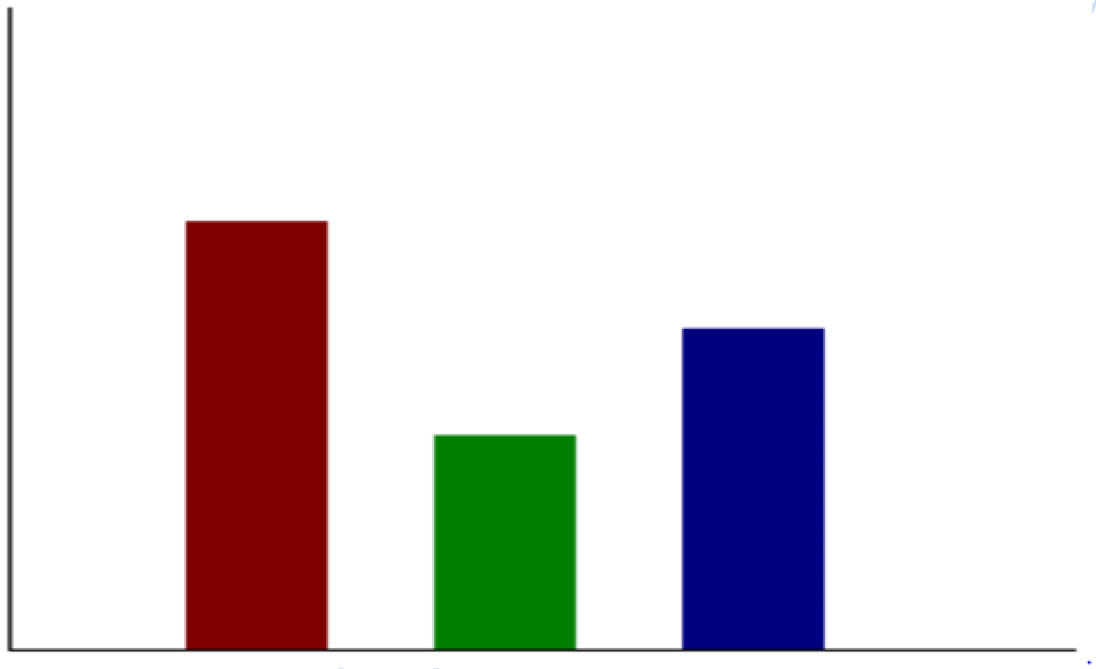
21. Bar() Function

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 ().

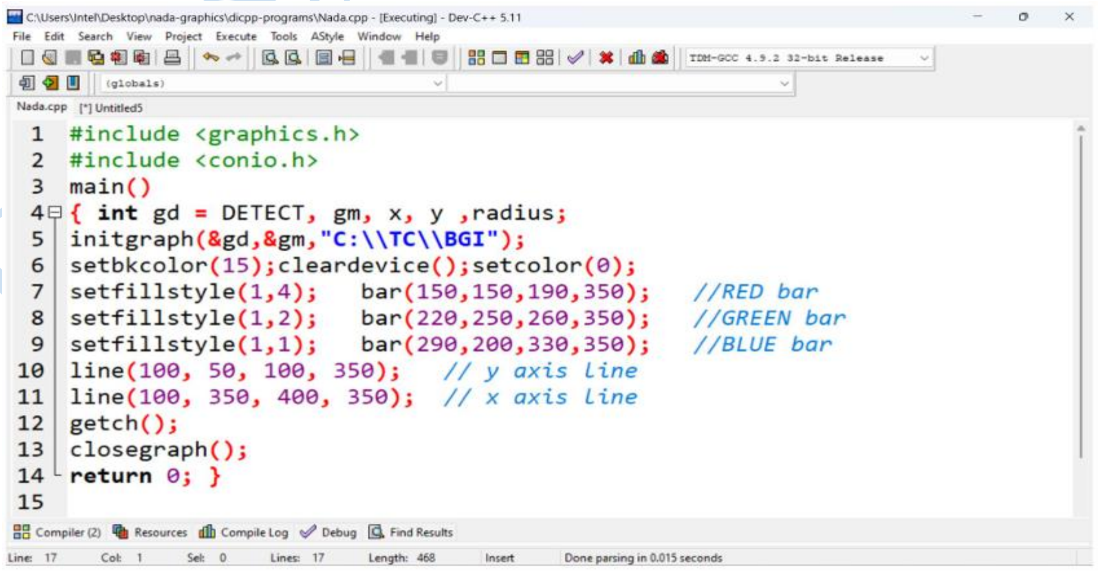
Syntax: void bar (int left, int top, int right, int bottom);

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



```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm, x, y ,radius;
initgraph(&gd,&gm,"C:\\TC\\BGI");
setbkcolor(15);cleardevice();setcolor(0);
setfillstyle(1,4);   bar(150,150,190,350);   //RED bar
setfillstyle(1,2);   bar(220,250,260,350);   //GREEN bar
setfillstyle(1,1);   bar(290,200,330,350);   //BLUE bar
line(100, 50, 100, 350);   // y axis line
line(100, 350, 400, 350);   // x axis line
getch();
closegraph();
return 0; }
```

22. Bar3d () Function

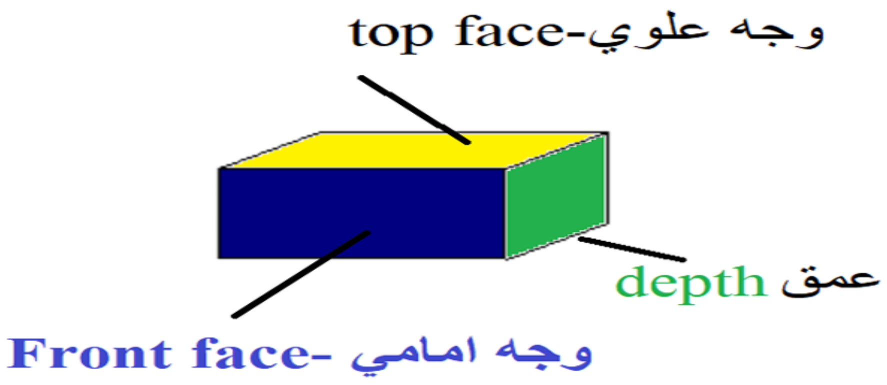
Bar3d function is used to draw a 2-dimensional, rectangular filled in bar.

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

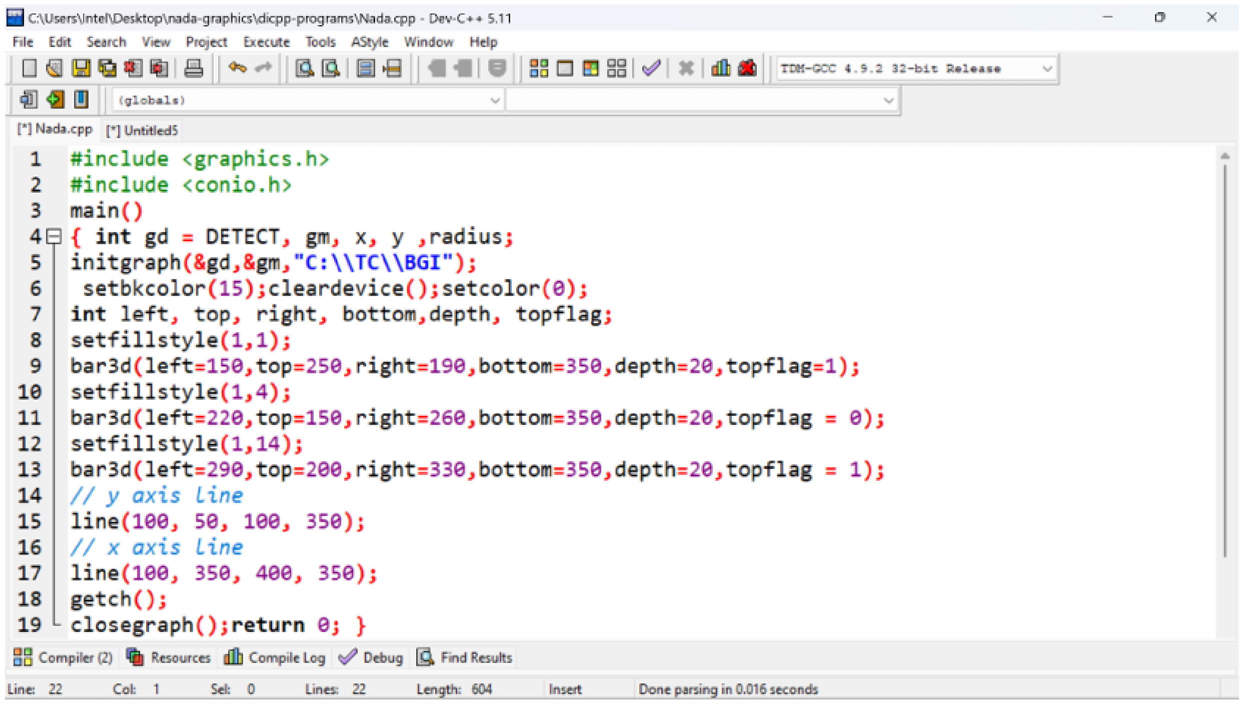
Coordinates of left top and right bottom corner of bar are required to draw the bar. **left** specifies the X-coordinate of top left corner, **top** specifies the Y-coordinate of top left corner, right specifies the X coordinate of right bottom corner, **bottom** specifies the Y-coordinate of right bottom 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);
```

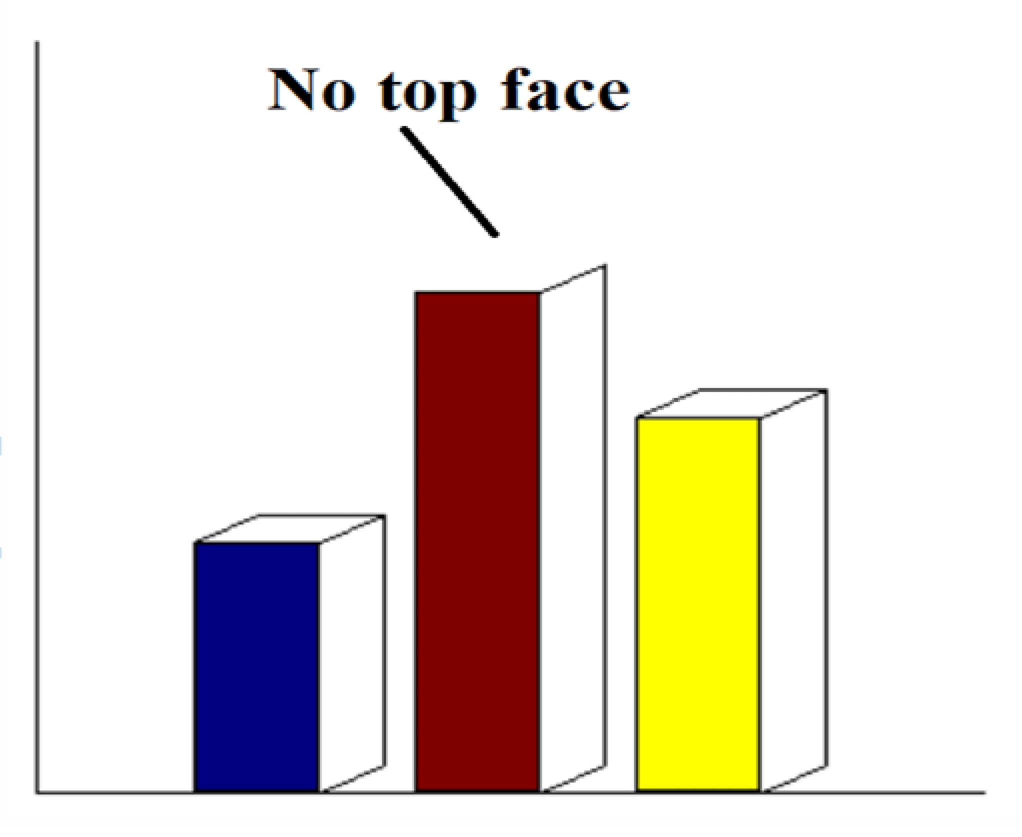


Exercise: What is the output of this code.

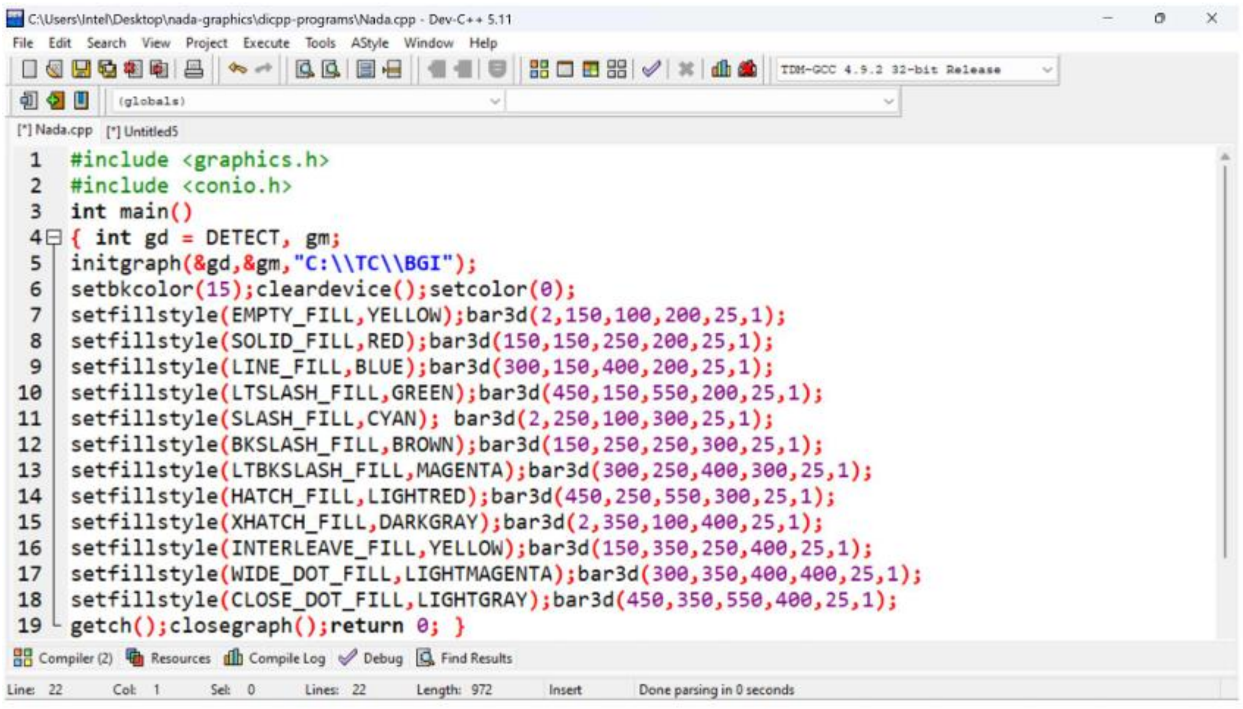


```
#include <graphics.h>
#include <conio.h>
main()
{ int gd = DETECT, gm, x, y ,radius;
initgraph(&gd,&gm,"C:\\TC\\BGI");
 setbkcolor(15);cleardevice();setcolor(0);
int left, top, right, bottom,depth, topflag;
setfillstyle(1,1);
bar3d(left=150,top=250,right=190,bottom=350,depth=20,topflag=1);
setfillstyle(1,4);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

The output:

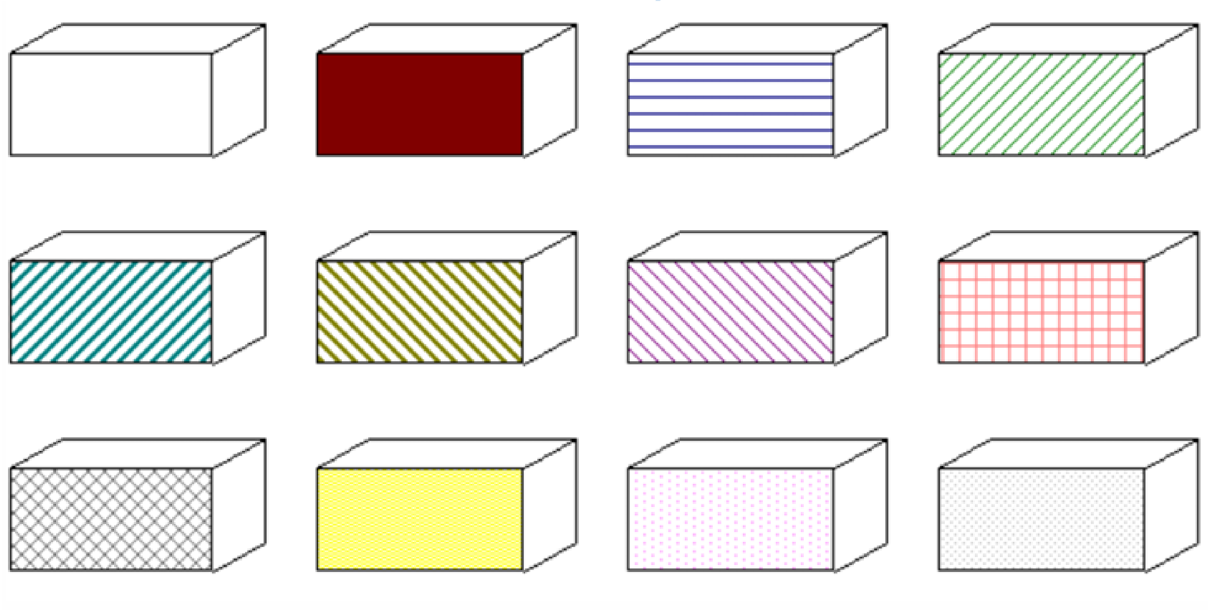


Example: draw multiple bar3d() with different style.



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

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);

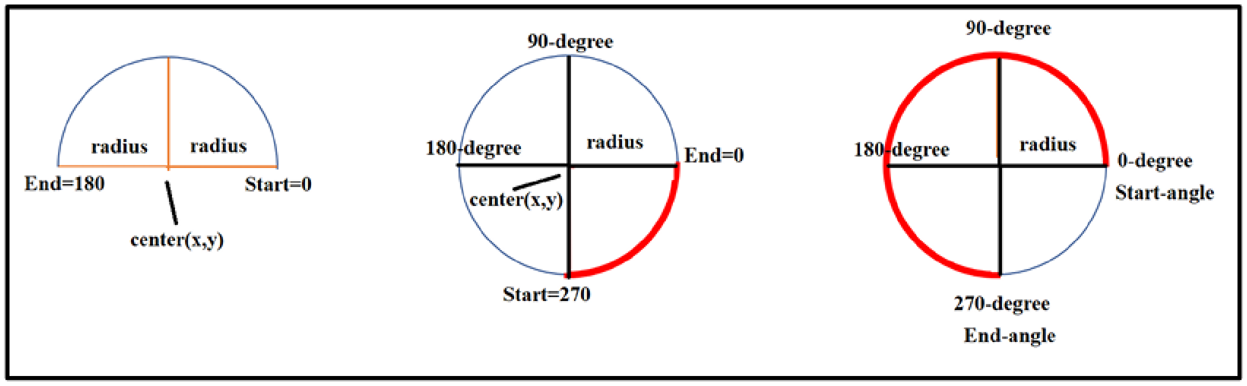
Where:

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

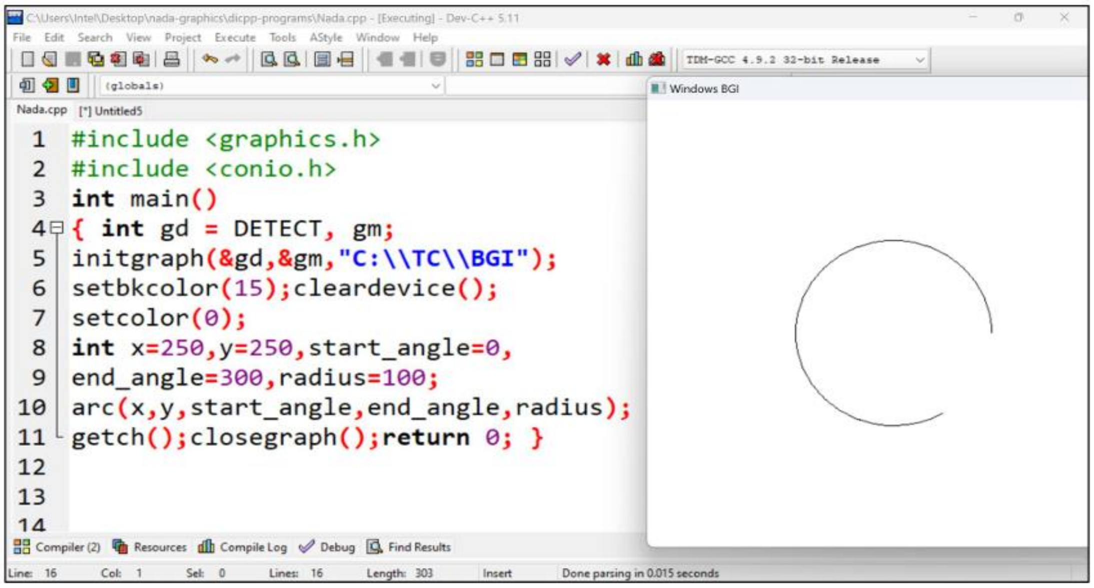
start_angle: is the starting angle and

end_angle: is the ending angle.

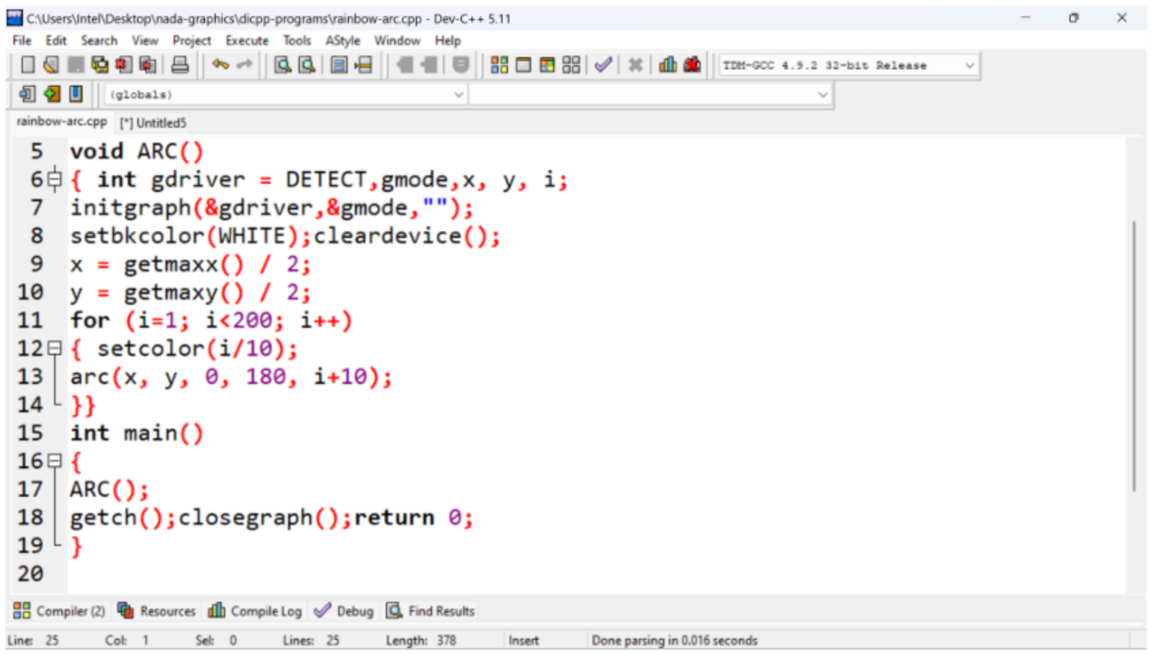
Radius: is the Radius of the arc.



Example:


```
#include <graphics.h>
#include <conio.h>
int main()
{ int gd = DETECT, gm;
initgraph(&gd,&gm,"C:\\TC\\BGI");
setbkcolor(15);cleardevice();
setcolor(0);
int x=250,y=250,start_angle=0,
end_angle=300,radius=100;
arc(x,y,start_angle,end_angle,radius);
getch();closegraph();return 0; }
```
C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - [Executing] - 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
Windows BGI
Compiler (2) Resources Compile Log Debug Find Results
Line: 16 Col: 1 Sel: 0 Lines: 16 Length: 303 Insert Done parsing in 0.015 seconds

HOMEWORK: what is the output of the following code?


C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\rainbow-arc.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 32-bit Release
(globals)
rainbow-arc.cpp [*] Untitled5
Compiler (2) Resources Compile Log Debug Find Results
Line: 25 Col: 1 Sel: 0 Lines: 25 Length: 378 Insert Done parsing in 0.016 seconds

```
void ARC()
{ int gdriver = DETECT,gmode,x, y, i;
initgraph(&gdriver,&gmode,"");
setbkcolor(WHITE);cleardevice();
x = getmaxx() / 2;
y = getmaxy() / 2;
for (i=1; i<200; i++)
{ setcolor(i/10);
arc(x, y, 0, 180, i+10);
}}
int main()
{
ARC();
getch();closegraph();return 0;
}
```

24. Ellipse() Function

Ellipse is used to draw an ellipse on graphics screen.

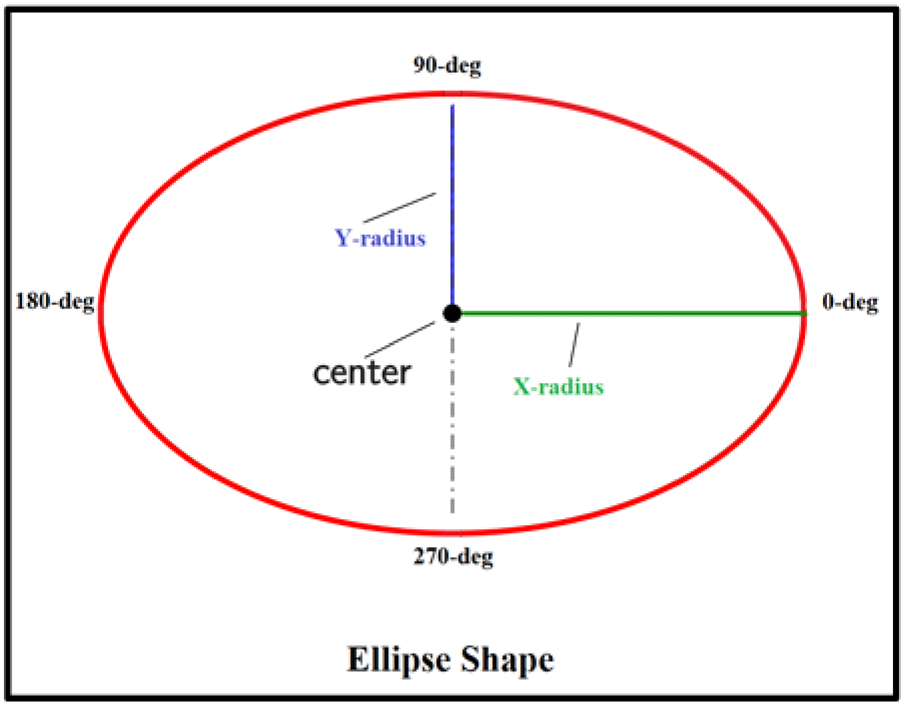
Suntax: void ellipse(int x, int y, int start_angle, int end_angle, int x_radius, int y_radius)

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.

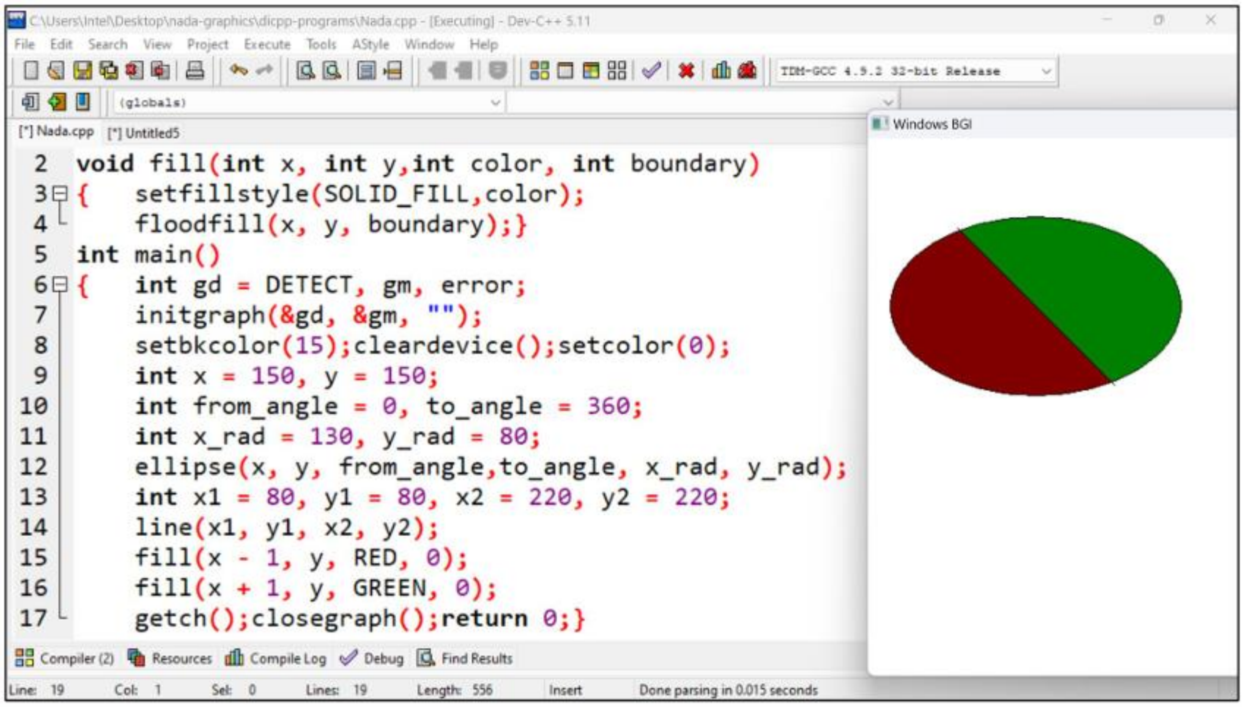
start_angle is the starting point of angle and end_angle is the ending point of angle.

The value of angle can vary from 0 to 360 degree.



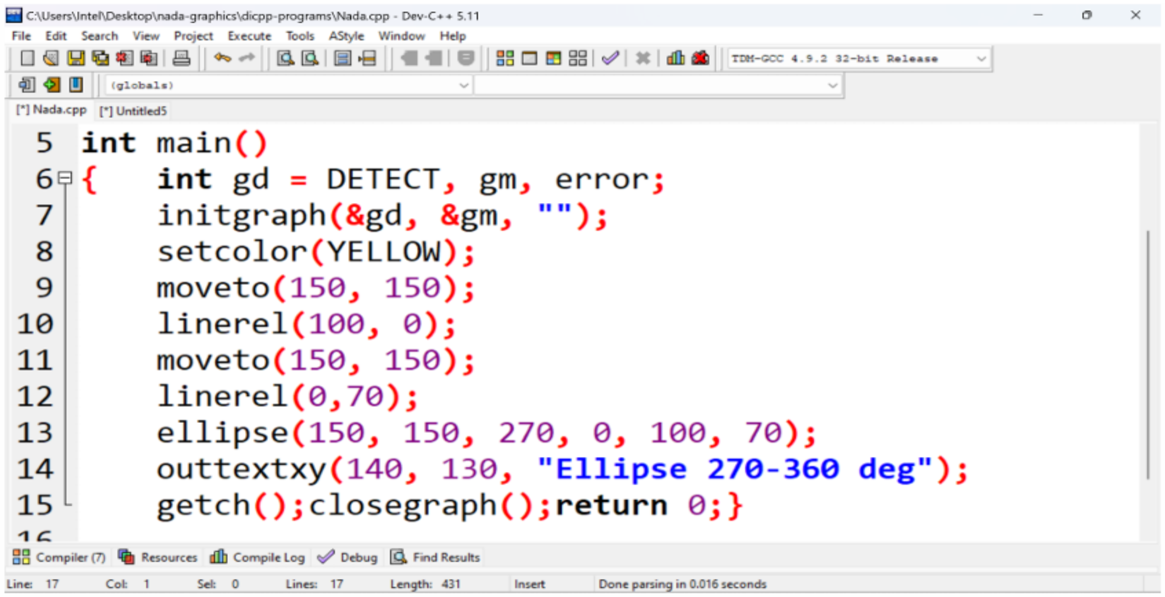
Exercise: Draw an ellipse divided by straight line into two colored parts in C++:

Solution:


C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - [Executing] - 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
Windows BGI
Compiler (2) Resources Compile Log Debug Find Results
Line: 19 Col: 1 Sel: 0 Lines: 19 Length: 556 Insert Done parsing in 0.015 seconds

```
void fill(int x, int y,int color, int boundary)
{    setfillstyle(SOLID_FILL,color);
     floodfill(x, y, boundary);}
int main()
{    int gd = DETECT, gm, error;
     initgraph(&gd, &gm, "");
     setbkcolor(15);cleardevice();setcolor(0);
     int x = 150, y = 150;
     int from_angle = 0, to_angle = 360;
     int x_rad = 130, y_rad = 80;
     ellipse(x, y, from_angle,to_angle, x_rad, y_rad);
     int x1 = 80, y1 = 80, x2 = 220, y2 = 220;
     line(x1, y1, x2, y2);
     fill(x - 1, y, RED, 0);
     fill(x + 1, y, GREEN, 0);
     getch();closegraph();return 0;}
```

Homework: what is the output of this code?



```
int main()
{   int gd = DETECT, gm, error;
    initgraph(&gd, &gm, "");
    setcolor(YELLOW);
    moveto(150, 150);
    linerel(100, 0);
    moveto(150, 150);
    linerel(0,70);
    ellipse(150, 150, 270, 0, 100, 70);
    outtextxy(140, 130, "Ellipse 270-360 deg");
    getch();closegraph();return 0;}
```

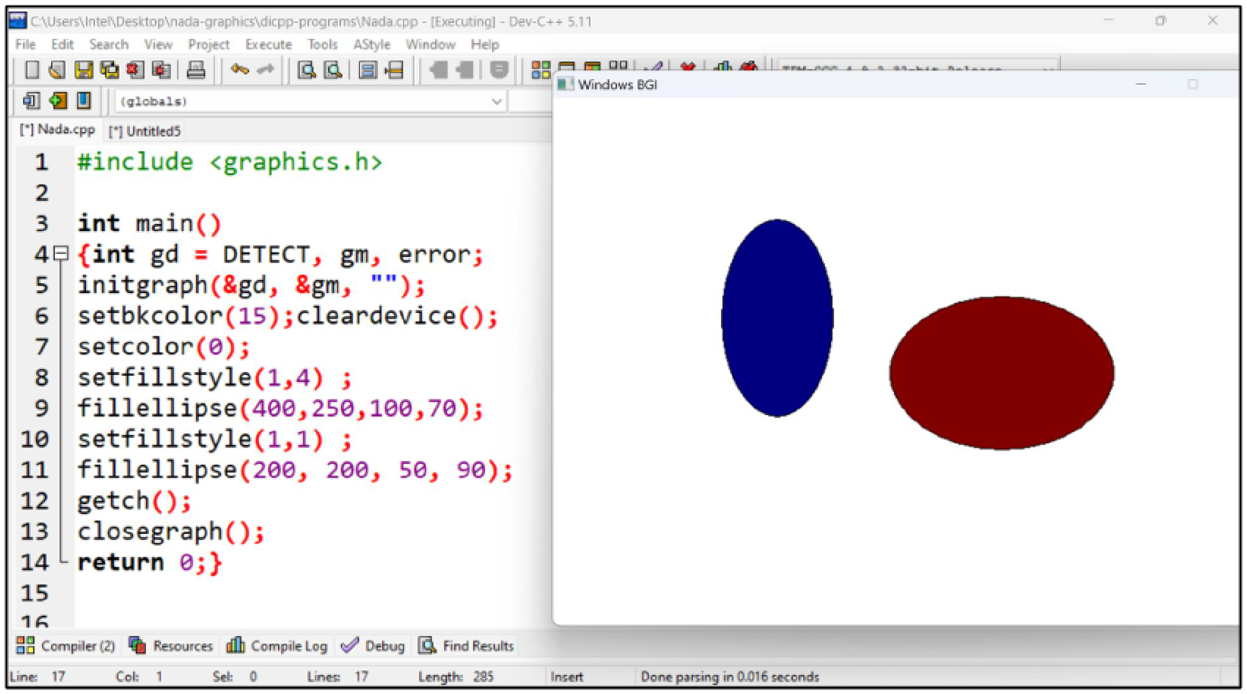
25.fillellipse () Function

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:


C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
[*] Nada.cpp [*] Untitled5
Windows BGI
Compiler (2) Resources Compile Log Debug Find Results
Line: 17 Col: 1 Sel: 0 Lines: 17 Length: 285 Insert Done parsing in 0.016 seconds

```
#include <graphics.h>

int main()
{int gd = DETECT, gm, error;
initgraph(&gd, &gm, "");
setbkcolor(15);cleardevice();
setcolor(0);
setfillstyle(1,4) ;
fillellipse(400,250,100,70);
setfillstyle(1,1) ;
fillellipse(200, 200, 50, 90);
getch();
closegraph();
return 0;}
```

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.

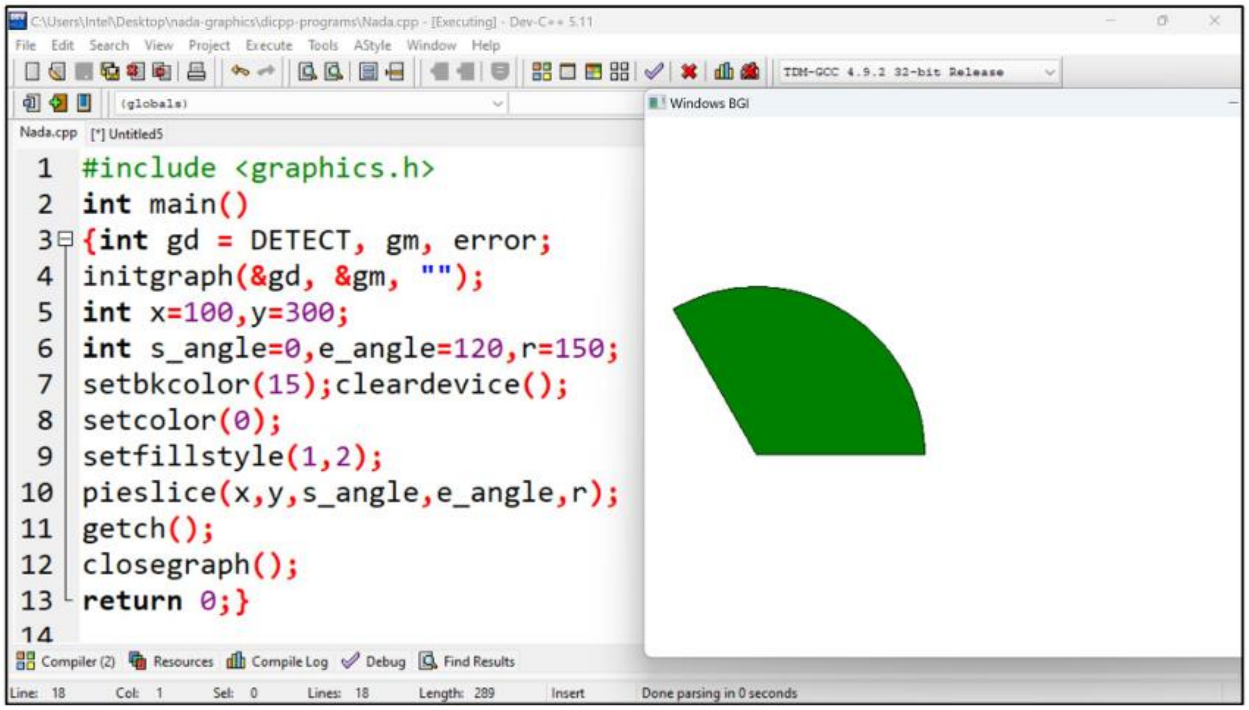
Syntax: void pieslice (int x, int y, int start-angle,int end-angle, int r);

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.

Examples:

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

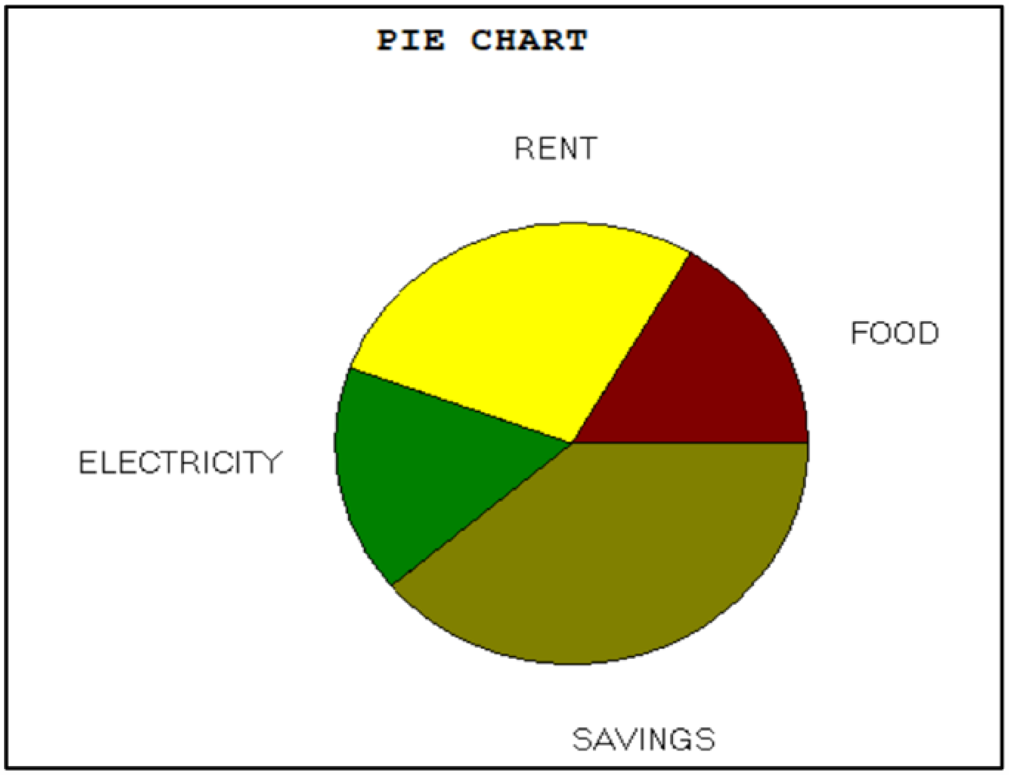
Output:



```
#include <graphics.h>
int main()
{int gd = DETECT, gm, error;
initgraph(&gd, &gm, "");
int x=100,y=300;
int s_angle=0,e_angle=120,r=150;
setbkcolor(15);cleardevice();
setcolor(0);
setfillstyle(1,2);
pieslice(x,y,s_angle,e_angle,r);
getch();
closegraph();
return 0;}
```

Windows BGI
Line: 18 Col: 1 Sel: 0 Lines: 18 Length: 289 Insert Done parsing in 0 seconds

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;}
```

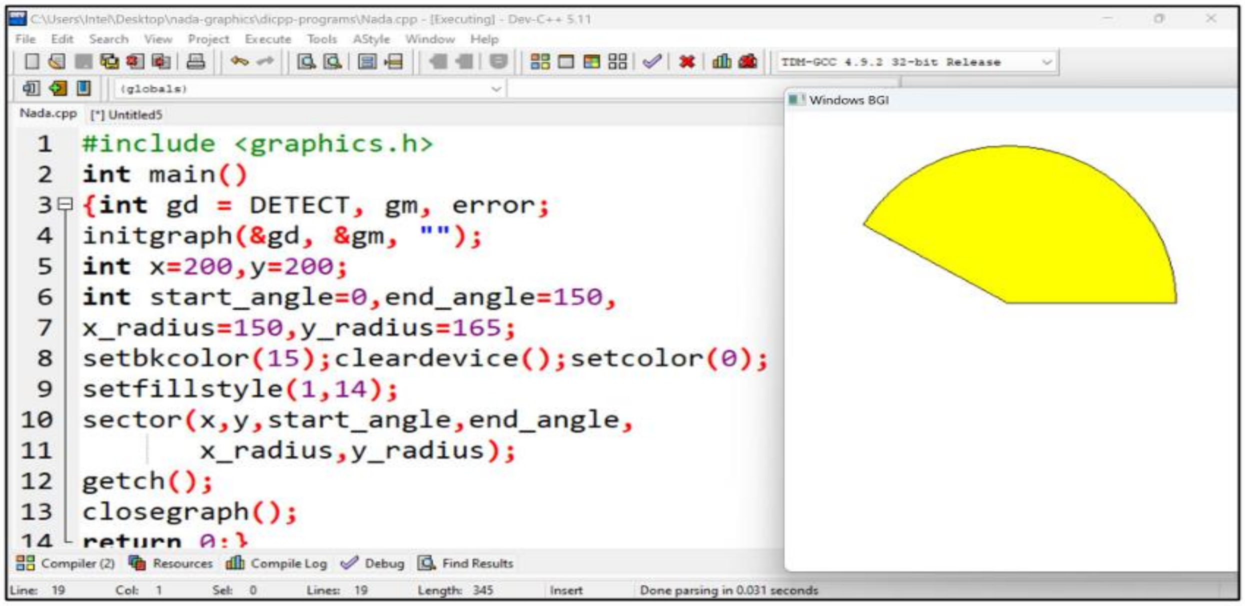
27. Sector() Function

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.

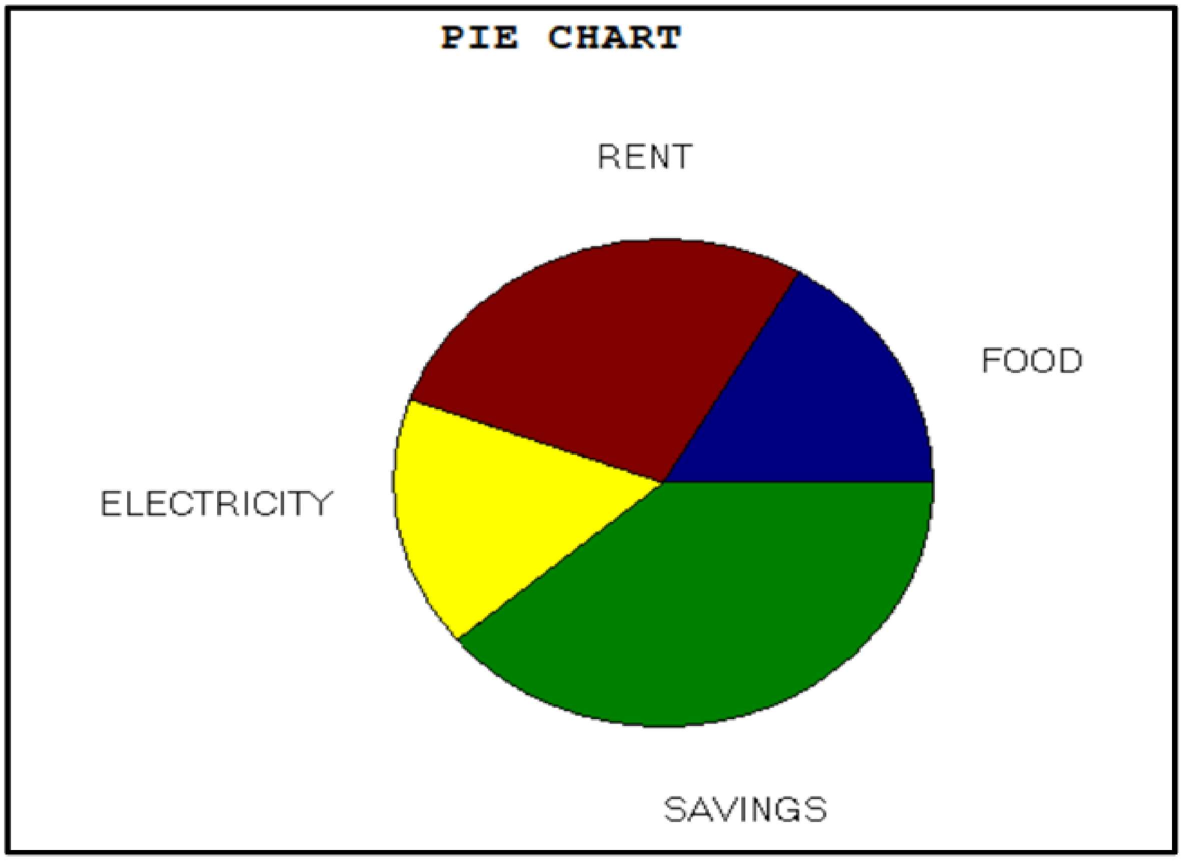
Examples:



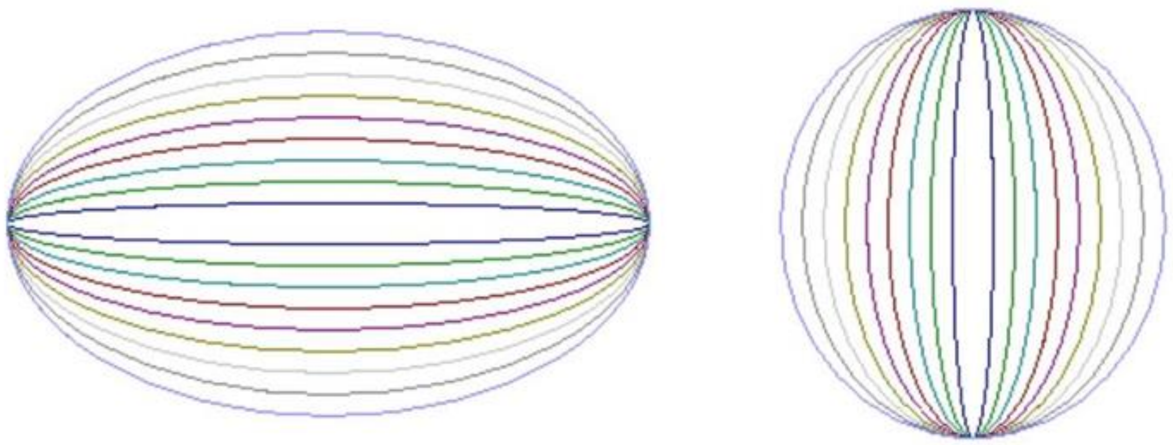
```
#include <graphics.h>
int main()
{int gd = DETECT, gm, error;
initgraph(&gd, &gm, "");
int x=200,y=200;
int start_angle=0,end_angle=150,
x_radius=150,y_radius=165;
setbkcolor(15);cleardevice();setcolor(0);
setfillstyle(1,14);
sector(x,y,start_angle,end_angle,
        x_radius,y_radius);
getch();
closegraph();
return 0;}
```

C:\Users\Intel\Desktop\nada-graphics\dicpp-programs\Nada.cpp - [Executing] - 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
Windows BGI
Compiler (2) Resources Compile Log Debug Find Results
Line: 19 Col: 1 Sel: 0 Lines: 19 Length: 345 Insert Done parsing in 0.031 seconds

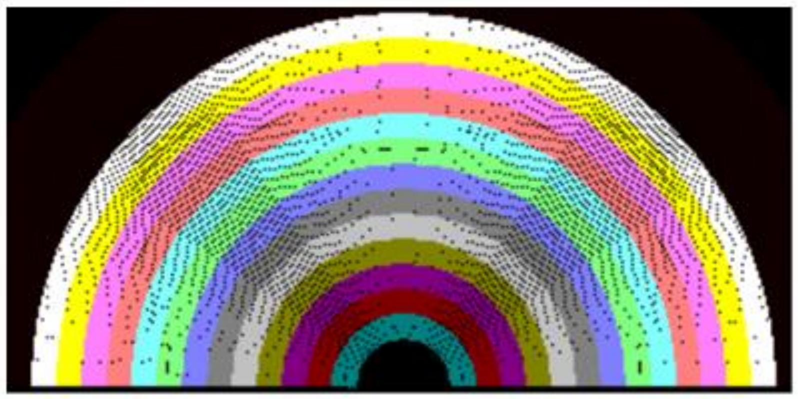
HOMEWORK: Write c++ program to output the following figure using Sector() function.



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



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



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; }
```

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; }
```