Using the WinBGIm Graphics Library with Dev-C++

September 2008

Description:

This page provides a mechinism for using the WinBGIm Graphics Library with the Dev-C++ development environment.. Additional documentation for the WinBGIm graphics library is available at www.cs.colorado.edu/~main/bgi.

Installation Notes:

1. Install Dev-C++. I installed from the Version 4.9.9.2 Setup File.
2. Download graphics.h to the include/ subdirectory of the Dev-C++ directories.
3. Download libbgia. to the lib/ In order to use the WinBGIm subdirectory of the Dev-C++ directories.
4. Whenever you #include <graphics.h> in a program, you must instruct the linker to link in certain libraries. The command to do so from Dev-C++ is Alt-P. Choose the Parameters tab from the pop-up window and type the following into the Linker area:

```
5. -lbgi
6. -lgdi32
7. -lcomdlg32
8. -luuid
9. -loleaut32
10.  -lole32
```

You can now compile and run programs that use the WinBGIm graphics library, such as this one that opens a small window, draws a circle and waits for the user to press a key:

```
#include <graphics.h>

int main( )
{
    initwindow(400, 300, "First Sample");
    circle(100, 50, 40);
    while (!kbhit( ))
    {
        delay(200);
    }
    return 0;
}
```

NOTE 2

Dev-C++ Tutorial for CSC 161 Students (Maintained by Michael Serrano)

Update (9/12/2006):

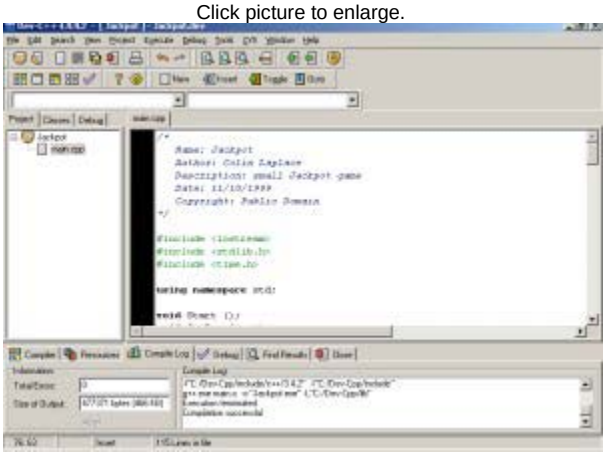
It's been quite some time since I've updated this page; in fact, I lost track of it for awhile. In the hopes that it may still be useful, I've updated this tutorial to be current with the latest version of Dev-C++ (as of this writing, Version 5 Beta 9--a.k.a. Version 4.9.9.2). ‹‹ Mike ››

What is Dev-C++?

Dev-C++, developed by Bloodshed Software, is a fully featured graphical IDE (Integrated Development Environment), which is able to create Windows or console-based C/C++ programs using the MinGW compiler system. MinGW (Minimalist GNU* for Windows) uses GCC (the GNU g++ compiler collection), which is essentially the same compiler system that is in Cygwin (the unix environment program for Windows) and most versions of Linux. There *are*, however, differences between Cygwin and MinGW; link to Differences between Cygwin and MinGW for more information.

**GNU is a recursive acronym for "GNU's Not Unix"; it is pronounced "guh-NEW". The GNU Project was launched in 1984 to develop a free and complete Unix-like operating system.*

Click picture to enlarge.



Bloodshed!?

I'll be the first to say that the name Bloodshed won't give you warm and fuzzies, but I think it's best if the creator of Bloodshed explains:

First I would like to say that I am not a satanist, that I hate violence/war and that I don't like heavy metal / hard-rock music. I am french, but I do know the meaning of the "Bloodshed" word, and I use this name because I think it sounds well. If you are offended by the name, I am very sorry but it would be a big mess to change the name now.

There's also a reason why I keep the Bloodshed name. I don't want people to think Bloodshed is a company, because it isn't. I'm just doing this to help people.

Here is a good remark on the Bloodshed name I received from JohnS:
I assumed that this was a reference to the time and effort it requires of you to make these nice software programs, a la "Blood, Sweat and Tears".

Peace and freedom,

Colin Laplace

Getting Dev-C++

The author has released Dev-C++ as free software (under GPL) but also offers a CD for purchase which can contain all Bloodshed software (it's customizable), including Dev-C++ with all updates/patches.

Link to Bloodshed Dev-C++ for a list of Dev-C++ download sites.

You should let the installer put Dev-C++ in the default directory of C:\Dev-Cpp, as it will make it easier to later install add-ons or upgrades.

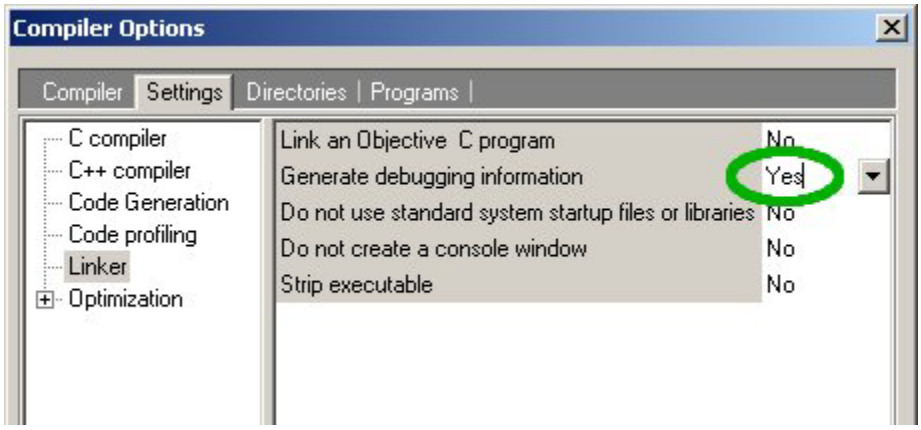
Using Dev-C++

This section is probably why you are here.

All programming done for CSC161 will require separate compilation projects (i.e. class header file(s), class implementation file(s) and a main/application/client/driver file). This process is relatively easy as long as you know what Dev-C++ requires to do this.

Step 1: Configure Dev-C++.
We need to modify one of the default settings to allow you to use the debugger with your programs.

- Go to the "`Tools`" menu and select "`Compiler Options`".
- In the "`Settings`" tab, click on "Linker" in the left panel, and change "`Generate debugging information`" to "`Yes`":



- Click "`OK`".

Step 2: Create a new project.
A "project" can be considered as a container that is used to store all the elements that are required to compile a program.

- Go to the "`File`" menu and select "`New`", "`Project...`".
- Choose "`Empty Project`" and make sure "`C++ project`" is selected.
  Here you will also give your project a name. You can give your project any valid filename, but keep in mind that the name of your project will also be the name of your final executable.
- Once you have entered a name for your project, click "`OK`".
- Dev-C++ will now ask you where to save your project.

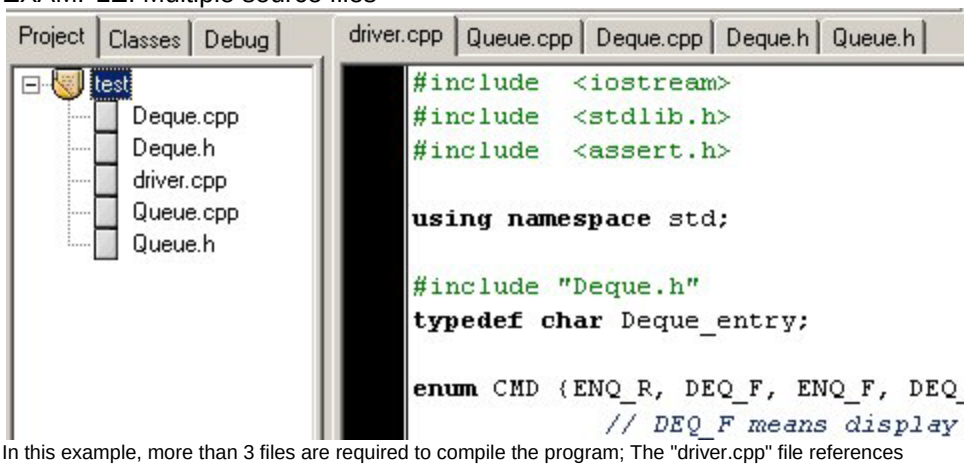
Step 3: Create/add source file(s).
You can add empty source files one of two ways:

- Go to the "`File`" menu and select "`New Source File`" (or just press CTRL+N) OR
- Go to the "`Project`" menu and select "`New File`".
  Note that Dev-C++ will not ask for a filename for any new source file until you attempt to:
  1. Compile
  2. Save the project
  3. Save the source file
  4. Exit Dev-C++

You can add pre-existing source files one of two ways:

- Go to the "`Project`" menu and select "`Add to Project`" OR
- Right-click on the project name in the left-hand panel and select "`Add to Project`".

EXAMPLE: Multiple source files



In this example, more than 3 files are required to compile the program; The "driver.cpp" file references "Deque.h" (which requires "Deque.cpp") and "Deque.cpp" references "Queue.h" (which requires "Queue.cpp").

Step 4: Compile.
Once you have entered all of your source code, you are ready to compile.

- Go to the "`Execute`" menu and select "`Compile`" (or just press CTRL+F9).

  It is likely that you will get some kind of compiler or linker error the first time you attempt to compile a project. Syntax errors will be displayed in the "`Compiler`" tab at the bottom of the screen. You can double-click on any error to take you to the place in the source code where it occurred. The "`Linker`" tab will flash if there are any linker errors. Linker errors are generally the result of syntax errors not allowing one of the files to compile.

Once your project successfully compiles, the "`Compile Progress`" dialog box will have a status of "`Done`". At this point, you may click "`Close`".

Step 5: Execute.
You can now run your program.

- Go to the "`Execute`" menu, choose "`Run`".

Note: to pass command-line parameters to your program, go to the "`Execute`" menu, choose "`Parameters`" and type in any paramaters you wish to pass.

Disappearing windows
If you execute your program (with or without parameters), you may notice something peculiar; a console window will pop up, flash some text and disappear. The problem is that, if directly executed, console program windows close after the program exits. You can solve this problem one of two ways:

- *Method 1 - Scaffolding:*
  Add the following code before any `return` statement in `main()` or any `exit()` or `abort()` statement (in any function):

  ```
  /* Scaffolding code for testing purposes */
  cin.ignore(256, '\n');
  cout << "Press ENTER to continue..." << endl;
  cin.get();
  /* End Scaffolding */
  ```

  This will give you a chance to view any output before the program terminates and the window closes.

- *Method 2 - Command-prompt:*
  Alternatively, instead of using Dev-C++ to invoke your program, you can just open an MS-DOS Prompt, go to the directory where your program was compiled (i.e. where you saved the project) and enter the program name (along with any parameters). The command-prompt window will not close when the program terminates.

For what it's worth, I use the command-line method.

Step 6: Debug.
When things aren't happening the way you planned, a source-level debugger can be a great tool in determining what really is going on. Dev-C++'s basic debugger functions are controlled via the "`Debug`" tab at the bottom of the screen; more advanced functions are available in the "`Debug`" menu.

Using the debugger:
The various features of the debugger are pretty obvious. Click the "`Run to cursor`" icon to run your program and pause at the current source code cursor location; Click "`Next Step`" to step through the code; Click "`Add Watch`" to monitor variables.
Setting breakpoints is as easy as clicking in the black space next to the line in the source code. See the Dev-C++ help topic "Debugging Your Program" for more information.

Dev-C++ User F.A.Q.

Why do I keep getting errors about "cout", "cin", and "endl" being undeclared?
It has to do with namespaces. You need to add the following line after the includes of your implementation (.cpp) files:

```
using namespace std;
```

How do I use the C++ string class?
Again, it probably has to do with namespaces. First of all, make sure you "#include <string>" (not string.h). Next, make sure you add "using namespace std;" after your includes.

Example:

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
    string s;
    s = "This is a test";
    cout << s << endl;
    system("PAUSE");
    return 0;
}
```

How do I use Borland Graphics Interface (graphics.h)?
For those of you migrating from Borland, you may be wondering where graphics.h is. Unfortunately, graphics.h is a Borland specific library and cannot be used with Dev-C++. Fortunately, a benevolent soul by the name of Michael Main has modified a BGI emulation

library for Windows applications to be used under MinGW (and therefore Dev-C++) which he has aptly named WinBGIm.

The files we need are:

graphics.h (download to C:\Dev-Cpp\include)
libbgi.a (download to C:\Dev-Cpp\lib)

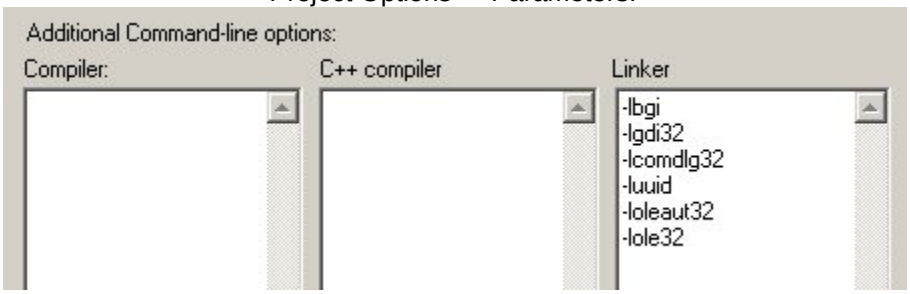
After you have downloaded the files to the correct locations, you can now use WinBGIm's graphic.h as you would Borland's graphics.h with a few caveats.

Using library files:
First, you have to tell Dev-C++ where to find the library functions that WinBGIm references--this is done in the "`Project Options`" dialog box.
Here are instructions on how to do this with a new project:

- Follow step 2 and step 3 of "Using Dev-C++".
- Go to "`Project`" menu and choose "`Project Options`" (or just press ALT+P).
- Go to the "`Parameters`" tab
- In the "`Linker`" field, enter the following text:

```
-lbgi
-lgdi32
-lcomdlg32
-luuid
-loleaut32
-lole32
```

Project Options -> Parameters:



- Click "`OK`".
- Follow step 4, step 5 and step 6 of "Using Dev-C++".

BGI and WinBGIm differences:
WinBGIm is a superset of BGI and as such may have functions and features with which you are unfamiliar. See Michael Main's BGI Documentation for more information.

Test code:
Just to make sure you've got everything set up correctly, try this test code in a new Dev-C++ WinBGIm project:

```
#include <graphics.h>

int main()
{
    initwindow(400,300); //open a 400x300 graphics window
    moveto(0,0);
    lineto(50,50);
    while(!kbhit());     //wait for user to press a key
    closegraph();        //close graphics window
    return 0;
}
```

If you've done everything correctly, you should get a simple graphic that closes when the user presses a key.

Where is <sstream> (string streams)?

In previous versions of Dev-C++, the included compiler did not implement sstream. Fortunately, new versions of Dev-C++ do not have this problem. Upgrade to the latest version of Dev-C++.

Bloodshed Software's Dev C++
Instructions for Installation and Use

Introduction
There are several separate files you will need to install for your home computer to match the C++ environment we have in the school lab: The DevC++ *IDE*, and the *winbgim.h graphics library*. You can borrow a CD that has all the files. These files are also all available for free on the internet:

1. Dev C++: http://www.bloodshed.net/ (We use version 4.0.0.0, it seems to work better with winbgim graphics than some later versions)
2. winbgim graphics: winbgim.h, winbgim.cpp, libbgi.a, (Right click and choose *save target as*)

Installation

1. Run the Dev C++ Setup program (the one with the icon of a computer). Choose "Typical" and accept all the default choices. This will create a folder called **Dev-C++** on your C: drive.
2. Copy the following winbgim graphics files to the **C:/Dev-C++/include** folder: *winbgim.h, winbgim.cpp*
3. Copy the following winbgim graphics file to the **C:/Dev-C++/lib** folder: *libbgi.a*.

Creating and running a simple (non-graphics) program

1. Start Dev C++. Under the start menu choose *Programs* | *Dev C++* | *Dev C++*
2. Choose *File* | *New Source File*. Dev C++ will automatically generate the following code:

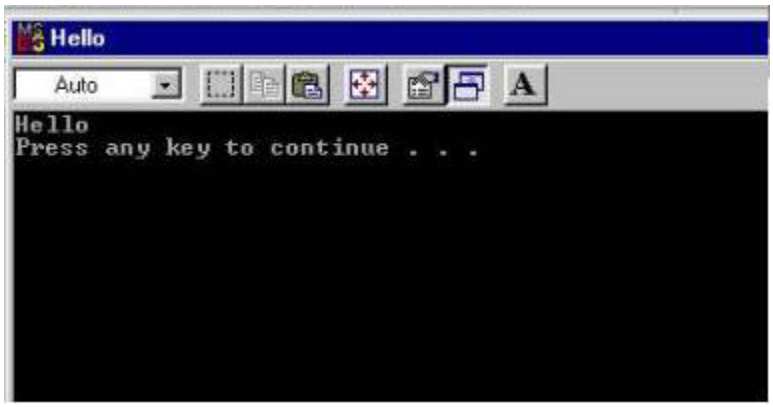
```
#include <iostream.h>
#include <stdlib.h>

int main()
{

      system("PAUSE");
       return 0;
}
```

11. Add your code to the program before `system("PAUSE");` (for example: `cout<<"hello"<<endl;`)
12. Save your program by choosing *File* | *Save Unit As*.
13. To compile and run your program choose *Execute* | *Compile and Run*.

14. A DOS window should that looks similar to the one below should appear.



15. Pressing any key will return you to the Dev C++ window. Click continue in the "Compilation completed" window.

Copying sample output from the output window.

1. While the output window is still open, Click on the icon in the top left corner and choose: *Edit* | *Mark*.
2. Click and drag over the output you wish to copy.
3. Choose *Edit* | *Copy Enter*. You can now paste the sample output back into your program.

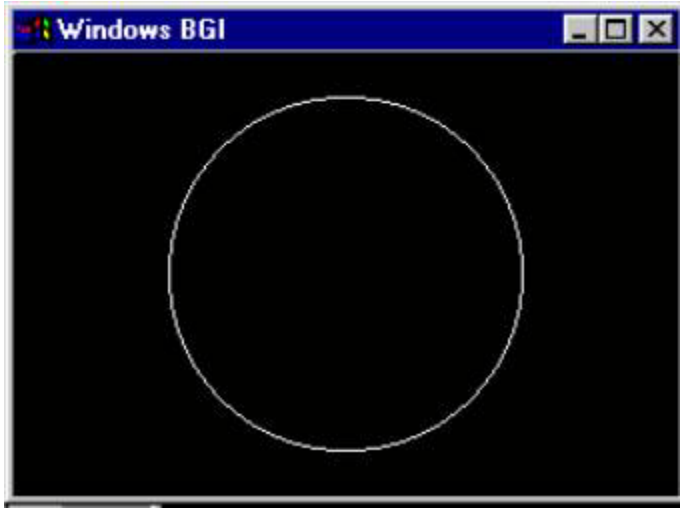
Creating and running a graphics program.

1. Choose *File* | *New Project* | *Empty Project* | *OK*. Choose a name for your new project.
2. Type code in the untitled window. For example:

```
#include "winbgim.h"
int main()
{
  initwindow(640,480); //open a 640x480 graphics window
  setcolor(WHITE);
  circle(320,240,100); //Draw a circle
  getch();   //wait for user to press a key
         closegraph();        //close graphics window
         return 0;
}
```

12. Save your program by choosing *File* | *Save Unit As*.
13. **IMPORTANT:** Choose *Project* | *Project Options*. In the Further object files or linker options field, enter *-lbgi -lgdi32*

14. Choose *Execute* | *Compile and Run*. You should see a window with a white circle similar to the one below:



15. If the window above doesn't appear, and you followed the above instructions correctly, go back and make sure that you correctly saved the *winbgim.h, winbgim.cpp* and *libbgi.a* files. See the installation instructions above.

NAME	FUNCTION	SYNTAX
1.arc	• arc draws a circular arc.	• void far arc(int x,int y,int stangle,int endangle,int radius);
2.circle	• circle draws a circle.	• void far circle(int x,int y,int radius);
3.bar	draws a bar.	• void far bar(int left,int top,int right,int bottom);
4.closegraph	shutdown the graphic system.	• void far closegraph(void);
5.ellipse	ellipse draws an elliptical arc.	• void far ellipse(int x,int y,int stangle,int endangle,int xradius,int yradius);
6.floodfill	• floodfills a bound region.	• void far floodfill(int x,int y,int radius);
7.getbkcolor	getbkcolor returns the current back ground color.	int far getbkcolor(void);
8.getgraphmode	getgraphmode returns the current graphic mode.	• int far getgraphmode(void);
9.getmaxcolor	returns the maximum color value.	• int far getmaxcolor(void);
10.getmaxx	returns maximum x screen coordinate.	• int far gatmaxx(void);
11.getmaxy	returns maximum yscreen coordinate.	int far getmaxy(void);
12.gety	returns the current positions y coordinate.	int far gety(viod);
13.getx	• returns the current	• int far getx(void);

	positions xcoordinate.	
14.detectgraph	determaines graphic driver and mode to use by checking the hardware.	• void far detectgraph(int far *graphdriver,int far *graphmode);
15.fillellipse	fillellipse draws and fill an ellipse.	• void far fillellipse(int x,int y,int xradius,int yradius);
16.getarccoords	gets coordinate of the last call to arc.	• void far getarccoords(struct arccoords type far *arccoords);
17.getcolor	returns the current drawing color.	• int far getcolor(void);
18.getfillpattern	copies a userefined fill pattern into memory.	• void far getfillpattern(char far *pattern);
19.getmaxmode	returns maximum graphics mode number for current driver.	• int far getmaxmode(void);
20.drawpoly	draws the outline of a polygon.	• void far drawpoly(int numpoints,int far *polypoints);
21.fillpoly	fillpoly draws and fills a polygon.	• void far fillpoly(int numpoints,int *polypoints);
22.clearviewpoint	clear the current view port.	• void far clearviewport(void);
23.getpixel	getpixel gets the color of a specified pixel.	• unsigned far getpixel(int x,int y);
24.grapherrormsg	returns a pointer to an error	• char *far

	message string.	grapherrormsg(int errorcode);
25.lineto	draws a line from th current position cp to (x,y).	• void far lineto(int x,int y);
26.line	draws a line between two specified points.	• void far line(int x1,int y1,int x2,int y2);
27.initgraph	initialize the graphic system.	• void far initgraph(int far *graphdrive,int far *graphmode,int far *pathtodrive);
28.rectangle	draws a rectangle.	• void far rectangle(int left,int top,int right,int bottom);
29.putpixel	plots a pixel at a specified point.	• void far putpixel(int x,int y,int color);
30.imagesize	returns the number of bytes required to store a bit image.	• Unsigned far imagesize(int left,int top,int right,int bottom);
31.moveto	Moves the cp to (x,y).	• void far moveto(int x,int y);
32.setcolor	sets the current drawing color.	void far setcolor(int color);
33.setgraphmode	sets the system to graphics mode,clear the screen.	void far setgraphmode(int mode);
34.textwidth	returns the width of string in pixels.	int far textwidth(char far *textstring);
35.textheight	returns the height of string in pixels.	int far textheight(char far *textstring);

Line function is used to draw a line from a point(x1,y1) to point(x2,y2) i.e. (x1,y1) and (x2,y2) are end points of the line.The code given below draws a line.

Declaration: - void line(int x1, int y1, int x2, int y2);

C programming code for line

```
#include <graphics.h>
#include <conio.h>

main()
{
   int gd = DETECT, gm;

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

   line(100, 100, 200, 200);

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

Above program draws a line from (100, 100) to (200, 200).

Declaration :- void floodfill(int x, int y, int border);

floodfill function is used to fill an enclosed area. Current fill pattern and fill color is 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 boundary of area. To change fill pattern and fill color use setfillstyle. Code given below draws a circle and then fills it.

C programming code

```
#include <graphics.h>
#include <conio.h>

main()
{
   int gd = DETECT, gm;

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

   setcolor(RED);
   circle(100,100,50);
   floodfill(100,100,RED);

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

In the above program a circle is drawn in RED color. Point (100,100) lies inside the circle as it is the center of circle, third argument to floodfill is RED which is color of boundary of circle. So the output of above program will be a circle filled with WHITE color as it is the default fill color.

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

Declaration :- 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. See the program below and then its output, it will further clear your understanding.

C program for drawpoly

```
#include <graphics.h>
#include <conio.h>

main()
{
   int gd=DETECT,gm,points[]={320,150,420,300,250,300,320,150};

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

   drawpoly(4, points);

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

Graphics Tutorial

In this chapter you will learn about basics of C++ Graphics. Before getting more information about c++ graphics first we starting from the header file:*graphics.h.* This header contains many graphics functions, if you will not declare this header file the graphics function will not work.
it means you will have to add *#include<graphics.h>* in header file section in your program.

Note : the important thing is ***BGI Error in c++****:*

```
BGI Error: Graphics not initialized (use 'initgraph')
```

If you see the above error in your output screen, you have to use the following code in *main()* function

```
int gd=DETECT,gm;
initgraph(&gd,&gm,"c:\\tc\\bgi\\");
```

We are starting from line() function in C++ graphics:

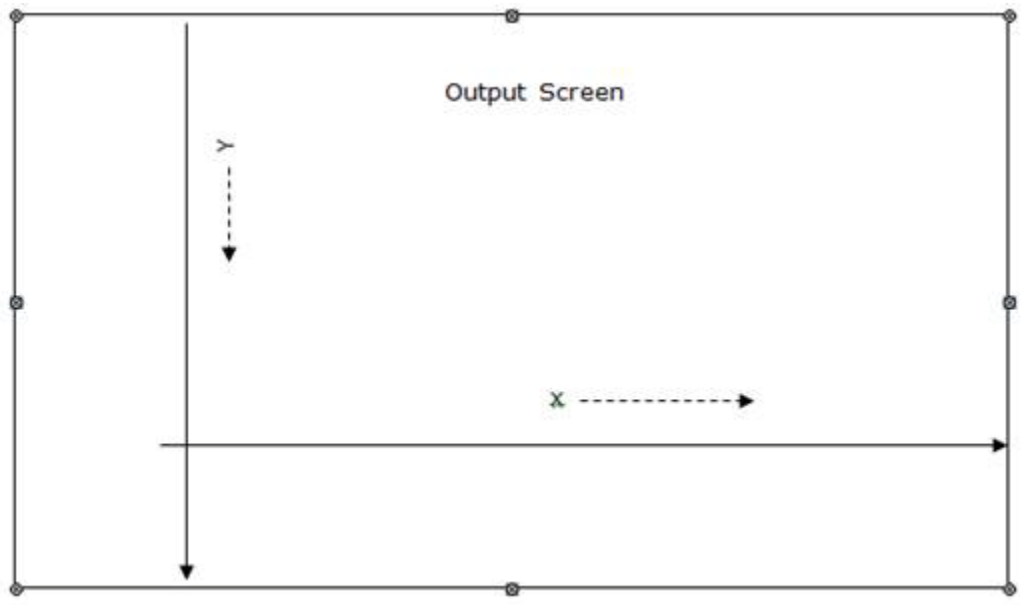
line() Function

function line is used to draw a line between two specified points.

its format is:

```
line(int x1, int y1, int x2, int y2);
```

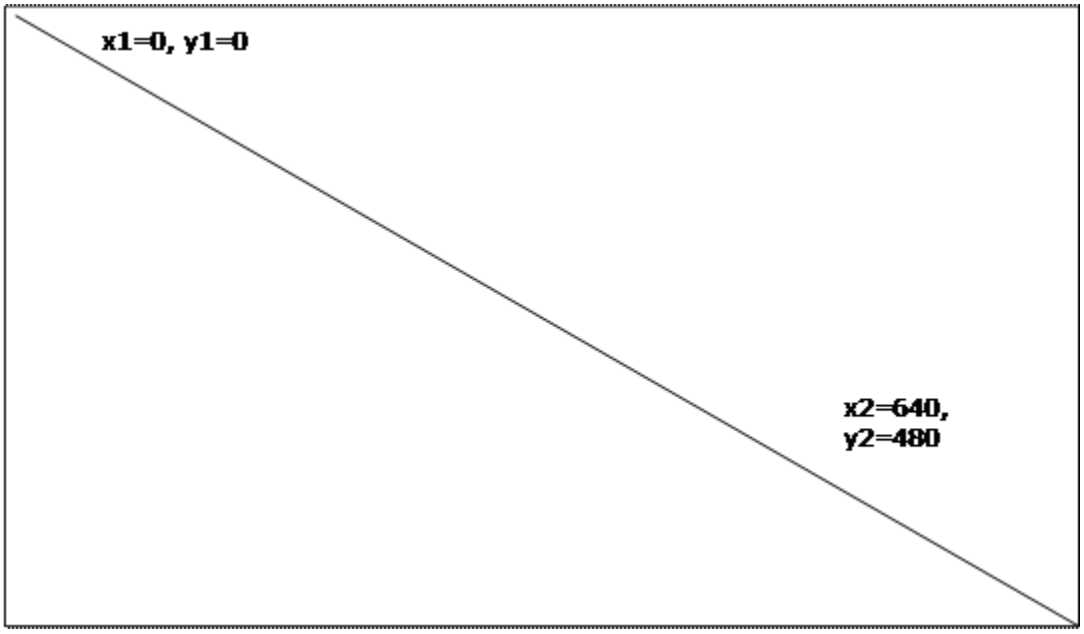
where *int x1* and *int y1* is starting point and *int x2* and *y2* is ending point of line, it will draw a line from *x1, y1 to x2, y2;*
Normally the maximum distance of x is 640, and y is 480.




```
Example to draw a line
```

```
include<graphics.h>
include<conio.h>
void main()
{
int gd=DETECT,gm;
initgraph(&gd,&gm,"c:\\tc\\bgi\\");
line(0,0,640,480);
getch();
}
```

Output



linerel()

The linerel function is used to draw a line up to x2 and y2 from the current position

Format:

```
linerel(int x2, int y2);
```

The *int x2* and *int y2* is ending point value of line,

e.g-

```
linerel(640,480);
```