Using the Graphics Library

When starting a new project under Microsoft Visual Studio C++, remember to download the two files `library.h` and `library.obj` from `http://rabbit.eng.miami.edu/class/library`, and add them to the project. Be sure to get the version of `library.obj` that matches the version of Visual Studio you are using. If unsure, check the "About Microsoft Visual Studio" item under the "Help" menu. The files should be saved in the folder where Visual Studio automatically creates the other project files.

The only requirements on a program using the graphics library are to include the line

```
#include "library.h"
```

at the beginning, and must define the function "main" with void return type and no parameters

```
void main()
{ ...
  ... }
```

The paradigm for graphics programming is to think of a window as a sheet of graph paper with an invisible pen associated with it (the pen itself is invisible, not the ink). The system remembers exactly where the pen is, what colour it is, and how thick it is. A program may change any of those things at any time. The pen may be moved from one position to another either invisibly or drawing a straight line as it goes. The pen may be moved either by stating the x-y co-ordinates it should move to, or by specifying a direction and distance. Curved lines must be programmed as a succession of very short straight lines.

The programmer may command the creation of a graphics window of any size and at any position within the computer's monitor display. The size of the window is given in terms of pixels in width and height - a modern monitor typically has over 1,000 pixels in each direction, packed at around 100 to the inch. Positions within a window, the x-y co-ordinates, are also measured in terms of pixels. The x direction is horizontal and the y direction is vertical.

It is an unfortunate feature of computer displays that they are nearly always "upside-down" compared to the way we normally expect graph paper to be. The point (0, 0) is at the top left, not the usual bottom left: Y values are small at the top and large at the bottom, but X values still increase in their traditional left-to-right way. If you forget this, your results will come out upside-down, and you may be annoyed.

Programs do not *need* to use graphics at all. The traditional telex or typewriter-like interface is also provided. This "character mode" input and output takes place in a "dos shell window", which usually has a black background and initially appears behind any graphics window. Remember how the Windows operating system understands "input focus": to type input using the Dos shell window, that window must be selected and on top. If a graphics window is selected when you type, your input will be processed differently.

When your program is running, the title bar of the dos window should show "Running...", and when your program is finished, it should show "Terminated". The windows themselves remain visible until you get rid of them, either by closing them, or by typing ctrl-C, or by pressing the stop button in Visual Studio. If you want to stop a program that is still running, the same methods work (except that if it is waiting for input, you may have to close the dos window separately). A running program may be paused and un-paused by pressing the ESC button.

Essential Basics

To Create a Graphics Window 800 pixels wide and 500 pixels high:

```
make_window(800, 500);
```

Of course, the 800 and 500 may be replaced by whatever width and height you want. The system will attempt to put the window as near to the centre of the screen as possible. The invisible pen starts at the centre of the window, it is one pixel wide, and contains black ink.

You can select the position of the window by providing two more inputs, one specifying the distance between the left edge of the screen and the left edge of the window, and the second specifying the distance between the top edges of the screen and window. Both are measured in pixels. To make the window appear at the top of the screen, but with a 150 pixel space to its left:

```
make_window(800, 500, 150, 0);
```

Windows may be moved and resized at any time. See the *Changing the Window* section later.

To Move the Pen to a new position, without making any mark:

```
move_to(25, 300);
```

This would set the invisible pen's position to 25 pixels from the left edge of the window and 300 pixels down from the top. The pen moves invisibly without making any marks. The position of the pen, as x-y co-ordinates is always measured from the top left corner of the window. It makes no difference where that window is on the screen.

There are alternative ways to move the pen, see the *Moving the Pen* section later.

To Draw a Straight Line:

```
draw_to(25, 300);
```

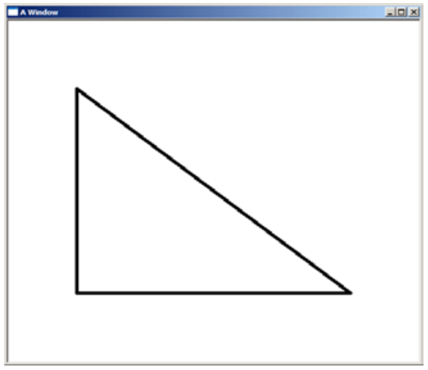
The `draw_to` function works exactly like the `move_to` function, except that as well as moving the pen, it draws a straight line from the pen's original position to its new one. The line will of course be as wide as, and the same colour as, the pen.

A Dull Example:

To draw Pythagoras' favourite right angled triangle (3:4:5), but 100 times bigger in terms of pixels:

```
#include "library.h"

void main()
{ make_window(600, 500);
  move_to(100, 100);
  draw_to(100, 400);
  draw_to(500, 400);
  draw_to(100, 100); }
```



To Change the Pen’s Width to 10 pixels:

```
set_pen_width(10);
```

That’s all it takes. The 10 may be replaced by any width (measured in pixels) you desire, but keep in mind that the smallest resolution a monitor can accurately produce is one whole pixel.

A one pixel pen very often does not produce very nice looking results. In the dull example, I cheated a bit so that the picture would still look OK when shrunk to fit on the page. The program I really ran was this:

```
#include "library.h"

void main()
{ make_window(600, 500);
  set_pen_width(5);
  move_to(100, 100);
  draw_to(100, 400);
  draw_to(500, 400);
  draw_to(100, 100); }
```

To Change the Pen’s Colour to red:

```
set_pen_color(color::red);
```

The graphics library understands 26 different colours by name, but allows you to specify any of 16,777,216 shades by other means (see the *Specifying Colours* section later). The identifiers for these named colours all begin with “`color::`” to make them stand out.

Once a program changes the pen’s width or colour, it stays at the new settings until explicitly changed later. Of course, the change is not retro-active: whatever was already drawn before the pen change stays exactly as it was.

These are the 26 named colours. Bear in mind that some colours look quite different depending on whether they are printed or on screen.

⬛	`color::black`	🟩	`color::dark_green`
⬛	`color::dark_grey`	🟩	`color::green`
⬜	`color::grey`	🟩	`color::light_green`
⬜	`color::light_grey`	🟩	`color::lime_green`
	`color::white`	🟨	`color::yellow`
🟥	`color::dark_red`	🟨	`color::sodium_d`
🟥	`color::red`	🟧	`color::orange`
🟥	`color::light_red`	🟫	`color::brown`
🟥	`color::pink`	🟪	`color::indigo`
🟦	`color::dark_blue`	🟪	`color::mauve`
🟦	`color::blue`	🟪	`color::violet`
🟦	`color::light_blue`	🟪	`color::purple`
🟦	`color::cyan`	🟪	`color::magenta`

To Make Text Appear in the Graphics Window:

```
write_string("six times nine is ");
write_string(6*9);
```

This sample would display "six times nine is 54". Text is written to the graphics window starting at the current pen position, and appears in the current pen colour. The exact alignment of the text is so that the left edge base-line of the first character printed is at the original pen position. When the text is written, the pen position moves up so that a further use of `write_string` will start in the expected place.

The pen's width has no effect on how text appears, but you may change the font to produce whatever effect is desired.

Changing the Font

Initially, the Times New Roman font is selected, at a size that would result in about 4 lines of text fitting comfortably per inch.

To change the size of the font, decide how tall a line of text should be, in terms of the number of pixels between the top of one line and the top of the next. On screen, there are usually about 100 pixels per inch. Use the `set_font_size` function with that size as its input. For example, to produce letters that are about two inches tall, use:

```
set_font_size(200);
```

To select a font by name, use the set_font function. The name you provide must match the name on one of the fonts installed on your system exactly. It is usually safe to expect at least "Arial", "Times New Roman", and "Courier New" to be available.

```
set_font("arial");
```

You may set the font type and size all at once, with this version:

```
set_font("arial", 32);
```

There are many other variations available, see the *Fonts* section later.