

Teaching Computer Programming to High School Students.

By Edward Naillon, Computer Programming Instructor

Oroville High School

naillone@chopaka.wednet.edu

2011 all rights reserved please ☺

Table of Contents

Preface.....	2
Cautions	2
The history of modern computing	2
Principals of operation of modern computers and software.....	4
Vocabulary to integrate into your lessons	4
High Level Language Options for an introductory course	8
Tools of the trade, the options are endless.....	8
Using the wxDevC++ open source development environment.....	9
Debugging your code	16
Structure of a simple program.....	19
Logic and problem definition.....	21
Main routine	27
Variables	28
Arrays.....	32
Input/Output	32
Strings	33
Pointers	36
Flow Control and Decision Structures	39
Subroutines.....	46
Event loop	48
C Operators for comparison, evaluation and processing.....	50
Bitwise Operations	52
Structures.....	52
Files.....	54
Just for fun.....	57
Windows Programming.....	61
Creating a simple Window	61
The Windows Procedure	66
The WinMain Routine.....	69
Adding a Menu.....	72
Controlling Window Background Color	79
Handling Messages and Commands.....	81
The Windows Create message.	83
Mouse Commands.....	84
Dialogs.....	93
Modifying Menu Items	97
Conclusion and resources.....	106

Preface

The computer programming instructor has a unique opportunity to greatly influence a high school student's mindset. Very few classes require the level of perfection and detail that exists in computer programming. Programming computers is a natural avenue to reinforce mathematical concepts, analytical thinking, imagination and problem solving. Although difficult for many students, programming allows you to build a very positive culture, a culture of pride, accomplishment, and teamwork. Students relive the memories created in such a class years after the final mark is given. Programming computers gives us the opportunity to promote absolute perfection in craftsmanship. It also demands individual problem solving and organizational skills, confidence, persistence and a devout sense of humor.

Cautions

Computer Programming can be a very tedious task. The slightest error in convention can render your program useless, or an error in logic can yield unexpected results. Some students have a hard time with this, and can be prone to feeling like a failure. A programmer will fail 10 times to succeed once. Most secondary students are impatient, and want to feel like they grasp the concept NOW. Waiting a week or two for it to click is almost unbearable for some. The student must be encouraged and realize that finding the solution sometimes depends on finding the methods that do not work. Approached correctly, the failures can help guide you to the correct solution.

Do not grade on perfection early on in the course. Work each problem until you get perfection, but do not grade on it. Programmers need confidence. You will pull the rug out from under your own feet if you take that away from them with your grading policy.

Grade on effort, participation and attitude until the basics have been established. This can take 9 weeks, so do not rush it.

The history of modern computing

Although a high school computer programming class is not a history class, an introduction to the significant people and developments that led up to today's micro computing technology will be beneficial to the student. Not only will it help the student understand the mysterious world inside the box, but will also allow them to engage in reflective conversation with peers and prospective customers!

I recommend exposing the student to a basic time line and some of the main contributors to the industry. Then allow them to research and report on one individual or technological advancement of choice. I do not spend more than a few days on this. I can then point them towards resources that are applicable if they are interested in further study of computing history. I think it is important to have the student make the computer do something cool right away so I move quickly.

Relevant topics for independent student research

- Al-Khwarizmi, the father of the algorithm (named after him).
- Blaise Pascal, a French mathematician who designed the first mechanical adding machine. He had a programming language named after him.
- Gottfried Wilhelm von Leibniz, a German mathematician, created a mechanical calculating machine that could multiply and divide.
- Joseph Jacquard developed a loom for weaving cloth with patterns controlled by punched cards.
- George Boole invented a form of algebra that used relational operators to solve problems using conditional logic.
- Herman Hollereth, an American engineer designed the punch card machines for processing census data.
- Alan Turing pioneered a conceptual model of what would be computable by mechanical means. He conceived the Turing Machine, which was used to predict limits of computability. He is considered by many to be the father of modern computing.
- John Von Neumann introduced the concept of a stored program, where instructions and data are stored in memory and can be executed again and again.
- John V. Atanasoff who created the first electronic special purpose computer.
- Konrad Zuse developed the first general purpose electronic computers.

Exercises

1. Please answer the following question, you may have to do a bit of reading to form an opinion, but write your opinion down in a one paragraph statement. Bill Gates, Steve Jobs, Michael Dell are all current industry leaders. Are they making historical contributions? What do you think? Or do they stand on the shoulders of other contributors?
2. Research and write a 1 page article on one of the historical contributors listed on pages 2 and 3. Speak about personal aspects of the contributor, explain the contribution, and also explain why you feel the contribution was important to the development of modern computing.
3. Research the following topics, and answer the following questions, and explain why you feel each answer is correct. You may have to sort through different opinions and articles to form what YOU feel is the right answer.

Who invented the GUI (Graphical User Interface)

Who invented the Internet?

Who invented the first Portable Computer?

Who invented Internet Chat?

Principals of operation of modern computers and software

Basic computer structure is important to cover. I will bring an old machine in the class during the first week of class and disassemble it with the students. I will then go over each part with the student and explain the function of each, and how it correlates to the other parts. I find that when students look at a part, it inspires them to ask questions that validate what I am trying to teach. I liken the relationship between the components of a computer to the relationship between the components of a conventional office. The hard drive is like the file cabinet, allowing you to retrieve and store information, the desk top is like your system RAM, allowing you to place retrieved information out so you can work on it, and you are the processor, as you go through and process the files.

Vocabulary to integrate into your lessons

Input devices

An input device is any device that sends a signal to the computer. The signal can then be interpreted and handled by hardware and software in the machine. Mice, keyboards, joysticks, bar code readers, sensors, scanners are all examples of input devices.

Displays

Devices that display information to the user fall into this category. Monitors and projectors are two types of displays in common use. Many type of textural displays for the blind also exist, using vibration or dynamic Braille characters as output.

Output devices

A piece of hardware that takes data from the computer and uses it to provide information or have control over an outside environment is an output device. Monitors are not only displays but also output devices. A force feedback joystick is not only an input but also an output device. Printers, speakers, card punchers, electronic valves and switches are also output devices. Manufacturing today uses giant machines as output devices to produce consumer products.

Converters

Converters typically translate analog signals to digital signals and vice versa. Video production often uses A/D converters to translate from VHS tape to DVD. Consumers use it often when working with Digital and Super 8 video cameras. A joystick is an excellent example of an analog to digital conversion. Please note that no REAL conversion from digital to analog signal types exists. Because a digital signal contains only samplings, and analog is infinitely continuous down to the atomic level, any conversion is actually only simulated with averaging and interpolation. You cannot make something out of nothing.

Processing

Processing is the act of taking information from a variety of sources and manipulating it inside of a computer's CPU through mathematical operation and presenting the result in the expected or perhaps an unexpected form.

Electronic memory

Through the use of solid state transistors, information can be held in chips, with no moving parts. Some solid state chips, such as memory cards and sticks, solid state hard drives offer persistent storage. This means data can be held even if no power is supplied to the device. Others offer temporary storage of

information, such as random access memory and cache chips. These devices offer much faster performance, but when power is removed the information is lost.

Hard Drive

The hard drive is conventionally a mechanical/magnetic device containing aluminum platters covered with magnetic material. A moving head magnetizes or senses the magnetic condition of particles at very precise locations along the rapidly spinning platter to read and write information. New forms of hard drives are emerging that utilize electronic rather than magnetic storage, and some research has been put into organic data storage.

BUS

A BUS is simply a data path or conduit between devices within the computer. Typically they connect CPUs with memory, data storage and expansion devices. They can be in the form of cabling or printed circuitry. External buses exist in the form of FIREWIRE and USB cables.

PCI

PCI stands for Peripheral Connect Interface, and is a standard developed by the INTEL Corporation. PCI is a system BUS standard used for connecting expansion devices to the motherboard of the computer system through the use of plug in slots on the motherboard. It is a 64 bit bus, running at speeds up to 66 MHz.

ISA

The Industry Standard Architecture for expansion devices was developed by INTEL to give computers standard plug in access. It was developed in the early 1990s and in 1993 became the first system BUS to utilize Plug and Play technology. It is slow by today's standards with a 16 bit data path. Some motherboards still include the large black ISA connectors for backward compatibility.

AGP

The Accelerated Graphics Port is a variation of the INTEL PCI standard, and is designed particularly for graphics expansion cards. Standard AGP is a 32 bit path running at 66 MHz, so it does not provide a throughput advantage over PCI. Optional variations exist which do exceed the limitations of PCI, providing throughput of up to 1.07 GB per second. It is designed to use main system memory to store textures. This increases the speed at which objects can be rendered to the screen resulting in increased performance.

USB

The Universal Serial Bus was designed to connect external devices with plug and play functionality. It is well suited for data transfer to hard drives, cameras, printers, scanners and many other devices. USB can daisy chain up to 127 devices to a single port. Transferring information at 12 Mbps, USB has largely replaced the serial and parallel BUS. The new USB 2.0 will transfer information at up to 480 Mbps and is fully compatible with the older 1.1 standard devices.

Serial

Serial is the most primitive external data transfer BUS. Serial transfers information 1 bit at a time over a single data channel.

Parallel

The Parallel external BUS was developed to allow several bits to be transferred concurrently....like multiple serial cables in Parallel. It was the standard for printing devices through the 1990s.

Cache

Because CPUs run at faster speeds than system memory, it is required to pool information that is waiting to be written to RAM to prevent bottlenecks. CACHE has a tremendous effect on system performance for machines with otherwise equal specifications. CACHE can be incorporated in to the processor or placed on the motherboard.

Removable Storage

Floppy disk drives were the first form of easily removable magnetic storage. The thin plastic 5 ½ disk was flexible and earned the technology the floppy name. Prior to that digital data was stored on punch cards and paper tape. Now we have Compact disk, Digital Video Disk, memory cards of various formats, pluggable USB devices and even hot swappable hard drives that can be easily removed and transferred to other computers.

BITS

A bit is the simplest form of computer data. It is accomplished by testing the on or off state of a transistor in memory. Bits in succession can be used to represent characters, integers and virtually any type of information because they can represent numerical values based on the base 2 system.

Machine/Assembly Language

Machine language is the lowest and simplest level of computer programming code. Machine language is understood by the central processing unit of the computer without any form of translation. Programmers will use a more intuitive language or High Level Language to program, and then that code will be converted into machine language by the compiler. Machine language is almost impossible for humans to understand and code because it is written only in numbers. Every type of CPU uses its own machine language so a program must be re-compiled to run on different platforms. Assembly language is very similar to machine language, except that Assembly uses variable names and instruction names instead of all numbers. Assembly is then converted to machine code by an assembler.

High Level Languages

A high level language is an intuitive programming interface where the code is written in a form very similar to the English language. They are very structured with many complicated functions already written and then accessed by the programmer through the use of pre-built library function calls. A high level language is independent of any hardware platform, and will be converted to the proper type of machine code by a compiler. There are many High Level Languages in use today. First implemented in the 1950's the list has grown to include Cobol, Fortran, Basic, Pascal, C, C++, Java, C#, Perl, ADA , Algol, prolog and many others.

Compiler

A compiler will take the code written in a high level language and convert it to machine code that can then be executed by a specific microprocessor.

Linker

A Linker will link source code files together with the appropriate runtime code for the targeted operating system and works with the compiler to generate a running executable program.

Debugger

The Debugger will catch errors in a running program and present the error to the user and the location in code that generated the error so that the programmer can troubleshoot and solve the problem.

WYSIWYG

What You See Is What You Get is an Object Oriented IDE design method that allows a user to design graphical interfaces by dragging and positioning pre built interface components such as buttons, editors and list boxes into the desired configuration without and code having to be written. The WYSIWYG IDE then generates the proper API calls to reproduce the interface in code.

GUI

A graphical user interface is a system of visual controls that are presented to the programs user and allow a program to be run and information to be modified and accessed through the use of the mouse, scroll bars, buttons, text boxes, list boxes and much more.

IDE

The Integrated Development Environment ties a high level language to a compiler, specific prewritten libraries and linkers. It provides all necessary tools to produce a running program.

Console

The console is the level of an operating system beneath the Graphical User Interface. It is characterized by a text only display, and typed commands to control the operating system.

Linux, UNIX, Mac OSX and all flavors of Microsoft Windows give access to the OS console.

WIN32

Windows software after Windows 3.11 featured an Applications Programming Interface for developing 32 bit applications. 32 bit apps can access and store more data at higher rates than the older 16 bit standard.

Hardware

Hardware comprise the physical components of a computer system that you can touch and feel.

Monitors, keyboards, CPU chips, motherboards, hard drives, mice, etc are all examples of hardware.

Software

Computer instructions or data that exists in electronic or magnetic storage are considered software. In of itself, software has no form or substance.

High Level Language Options for an introductory course

Most of the core concepts that should be addressed in an entry level programming course can be delivered in any programming language. I feel that the students on occasion should be exposed to devising the same solution in more than one language. Programmers must use resources well. They need to know how to find information to solve a problem from any available source. Sometimes the solution will be coded in another language. A simple conversion will then be very rewarding for the student.

As a matter of fact, all programming is a conversion, from your native language to one that adheres to the strict conventions of the computer. Your thoughts and decisions will be executed in the processing unit of the machine. Every computer program is a glimpse in to the very mind of its programmer.

I do the bulk of my lessons in the C and C++ languages. I feel that I address the largest need with these languages, as many other languages are supersets or subsets of C. Perl, Java, C#, C and C++ are all very similar.

Basic is also very popular, and adequate for general programming. You do give up a certain level of control with Basic, and C really is not more difficult to code. In fact, a C programmer will use fewer keystrokes to solution, as the language is more refined and concise.

Why do I use C at all and not just use C++? I like the low level control that some C functions give me. A C++ compiler will compile C and C++ in the same block of code, as C++ contains all of the C syntax as well.

Tools of the trade, the options are endless

Today's programmer has many options, determined mostly by the type of programming that they intend to do. A person that programs only for Windows Based computers and devices is likely to choose a professional solution such as Microsoft Visual Studio. A person wanting to program for Macintosh will likely use Apple's Xcode system.

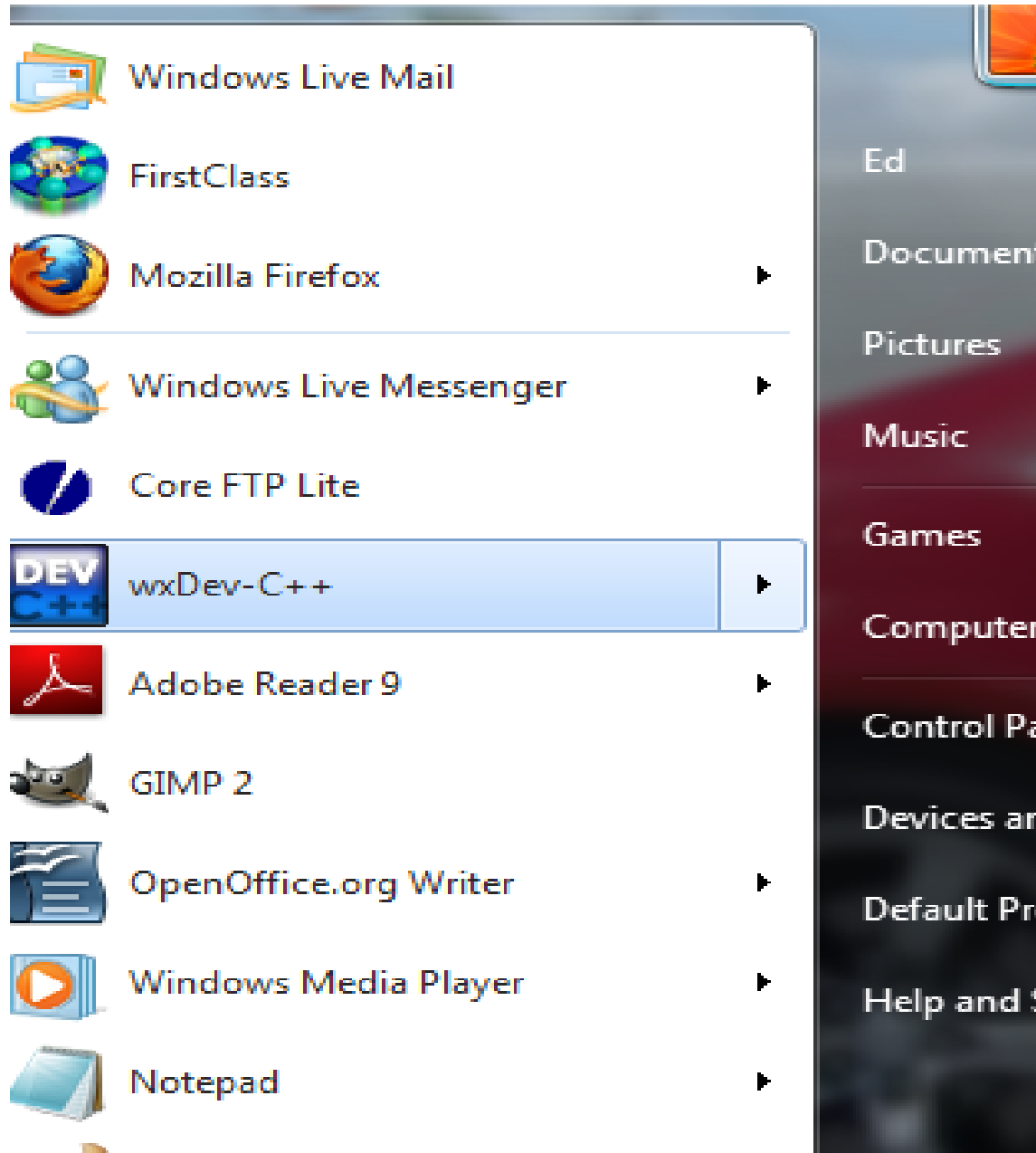
Development kits can be downloaded for more specific tasks such as programming for mobile devices and phones. If you intend to develop for the iPhone or iPad for instance, you can download the iPhone developer's kit from apple for use with Xcode. Android developers will use the android software development kit and the Eclipse development environment.

There are more tools available for specific tasks such as game programming, such as Flash, Game Studio or Game Maker. One think that they all have in common is that they can use a language that is based on C or C++. If a person knows this language, they can use the various tools available to them to create solutions.

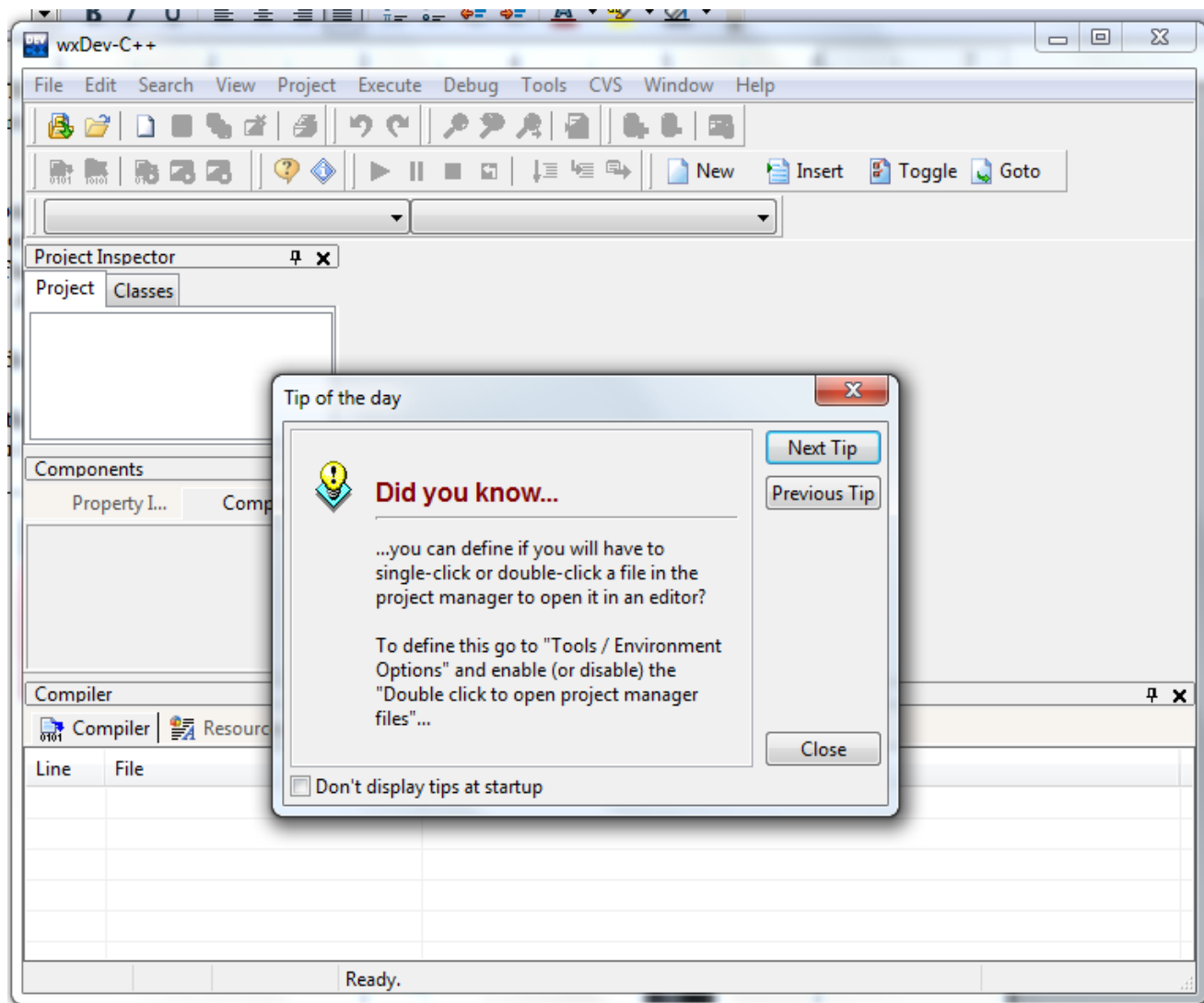
Using the wxDevC++ open source development environment

For the purpose of this course I have chosen the wxDevC++ Integrated development Environment. It is small and easy to install, provides many organizational and code assistant functions, and has a look and feel similar to many of the professional solutions available. And it is free and easy to get online.

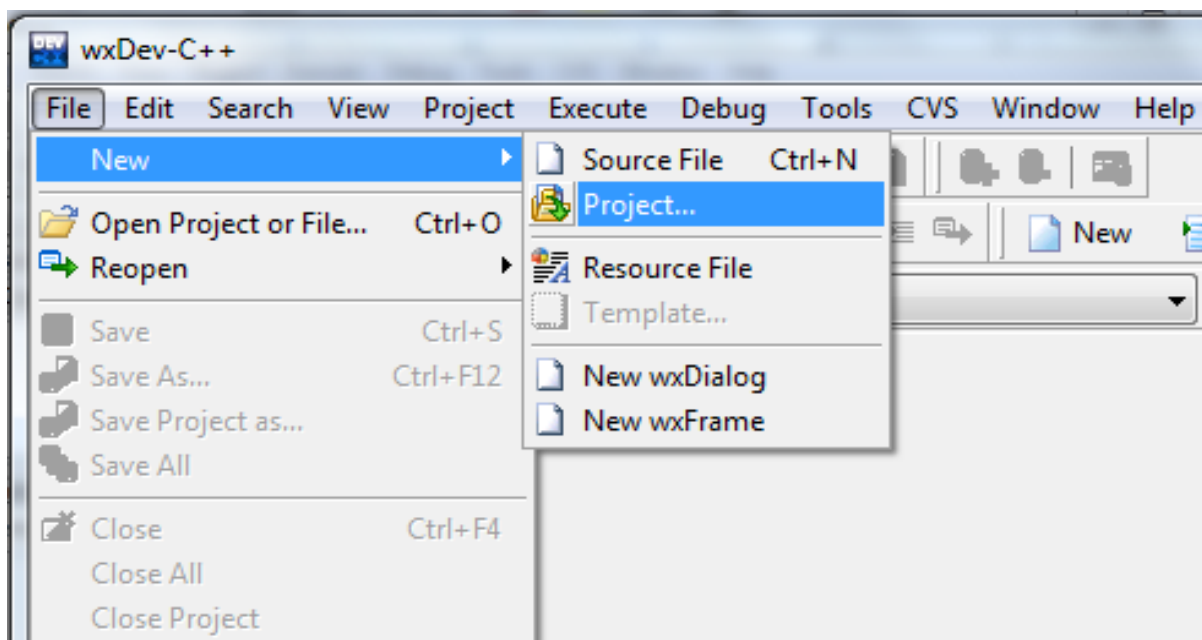
You can download and install this program from <http://wxdsgn.sourceforge.net/>. I will leave the installation instructions up to the documentation on this site, as they are clear and concise. After the installation, opening the DevC++ program is done by clicking on the icon on your desktop or the start menu item.



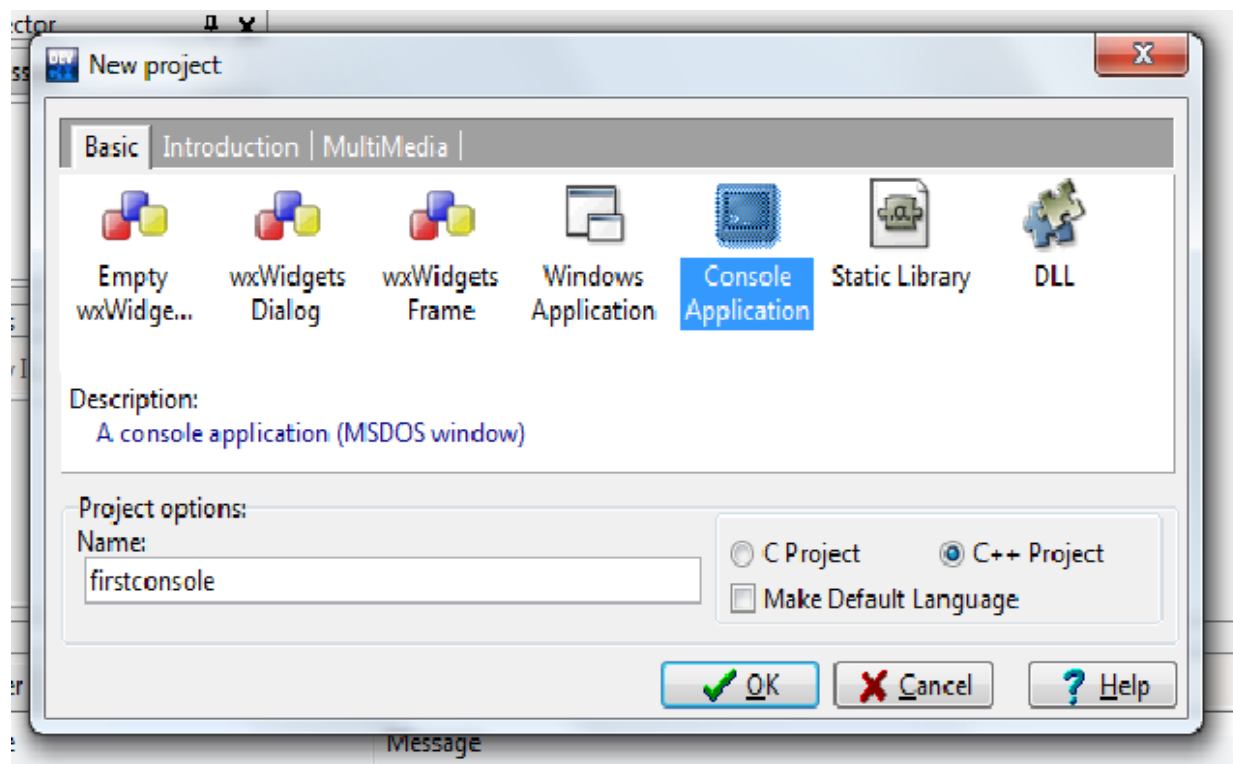
The program will come up in the default state, showing the tip of the day. You can turn this feature off if it annoys you, or you can read the tip before you close it each day to learn a little about the program.



Let's start a new project. To get things started, close the tips window and from the menu at the top of the application choose the File|New|Project item as shown below.



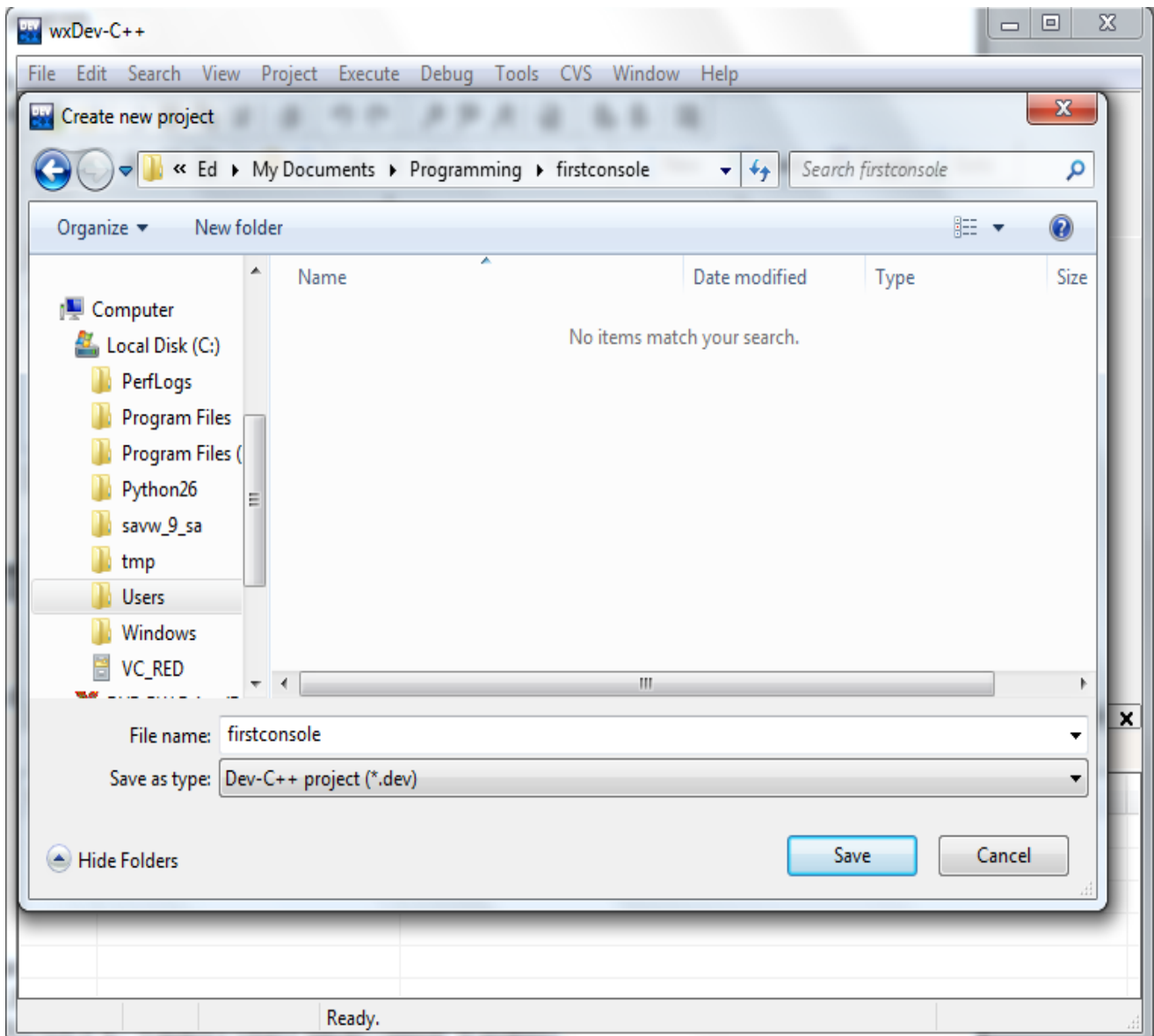
The project window will come up, choose the Console Application type, and under the project options type firstconsole for the name as shown below. Click OK to close the dialog.



Now you will be asked for a location to save your project. When you are saving your projects, it is very important to save each one of them in their own folder. This is because a project is usually a group of files that are created, that all work together to create the finished program.

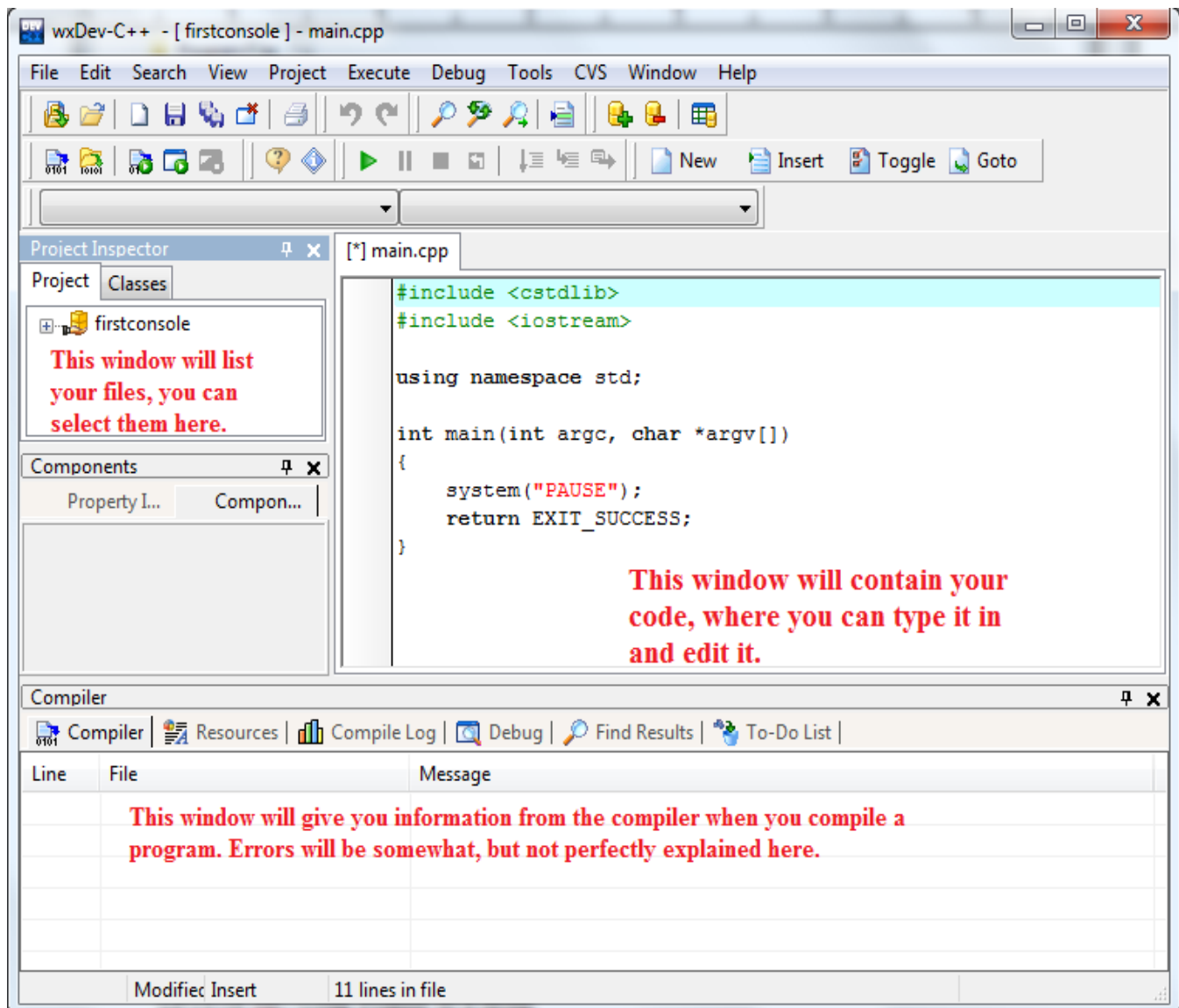
If you do not give each one a folder of its own, after you make a few projects you will end up so disorganized that it will start becoming difficult to access and modify your code.

Always choose to save your projects in the My Documents folder, create a folder called programming, and then create another folder with the same name as your project. In this case, firstconsole. Look at the top of the file dialog below and you will see what I mean.



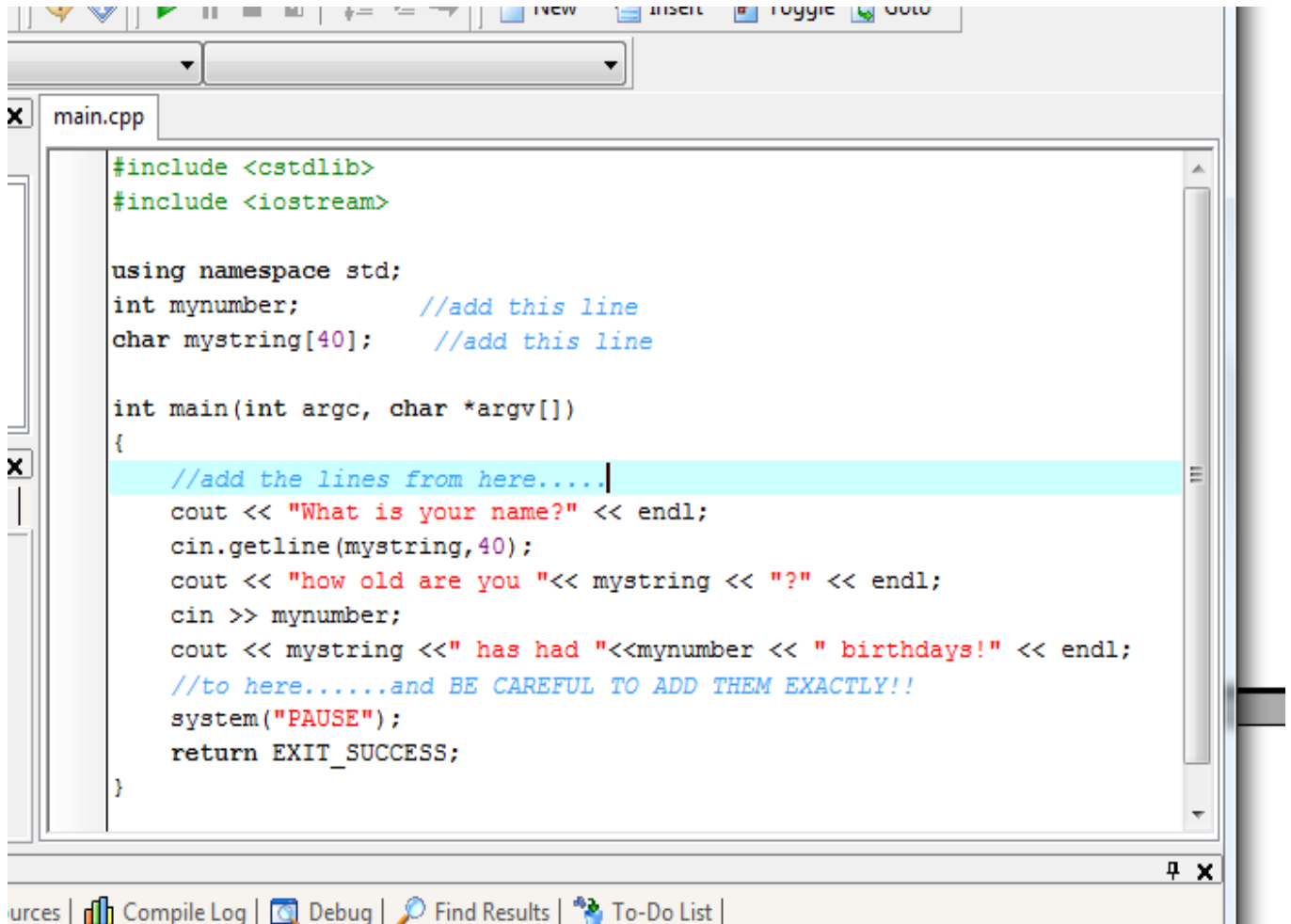
When you click OK, the Integrated Development Environment will come up, creating a basic project and source code file for a simple console application. I explain the three main windows that you will use in the image below. The project inspector is very important, as it will be the way that you manage and edit different files as you add them. You can see that your firstconsole project is listed there. If you expand that by clicking the little plus sign to the left it will show your main.cpp source code file, the one that you see in the code editor window. Use the File|Save All menu item from the top of the application, and save the files that are created into your firstconsole project folder. For now it will only save the main.cpp file. This is your source code file, the .cpp stands for C ++.

The compiler window at the bottom will give you errors, warnings and status messages as you try to compile your programs. If you see errors they must be fixed before your program can run. We will get to all that later. But for now we are going to add a couple lines of code, and then compile and run the code to show you your first computer program.



Edit your main.cpp file in the code editor. The code editor is just like a word processing program or notepad, it allows you to type in text that will become your computer code. You can see that it uses colors to indicate certain things. Type in the new lines that I am showing below, you will see my comments. They are preceded by // symbols. We are adding two variables, mynumber and mystring. The variable mynumber is an integer, and the variable mystring is a string of forty characters that can be used to hold a word or sentence. Variables hold information in a computer program, just like your pocket holds your cell phone!

We then add some basic input and output functions to fill the variables and display them back to you.

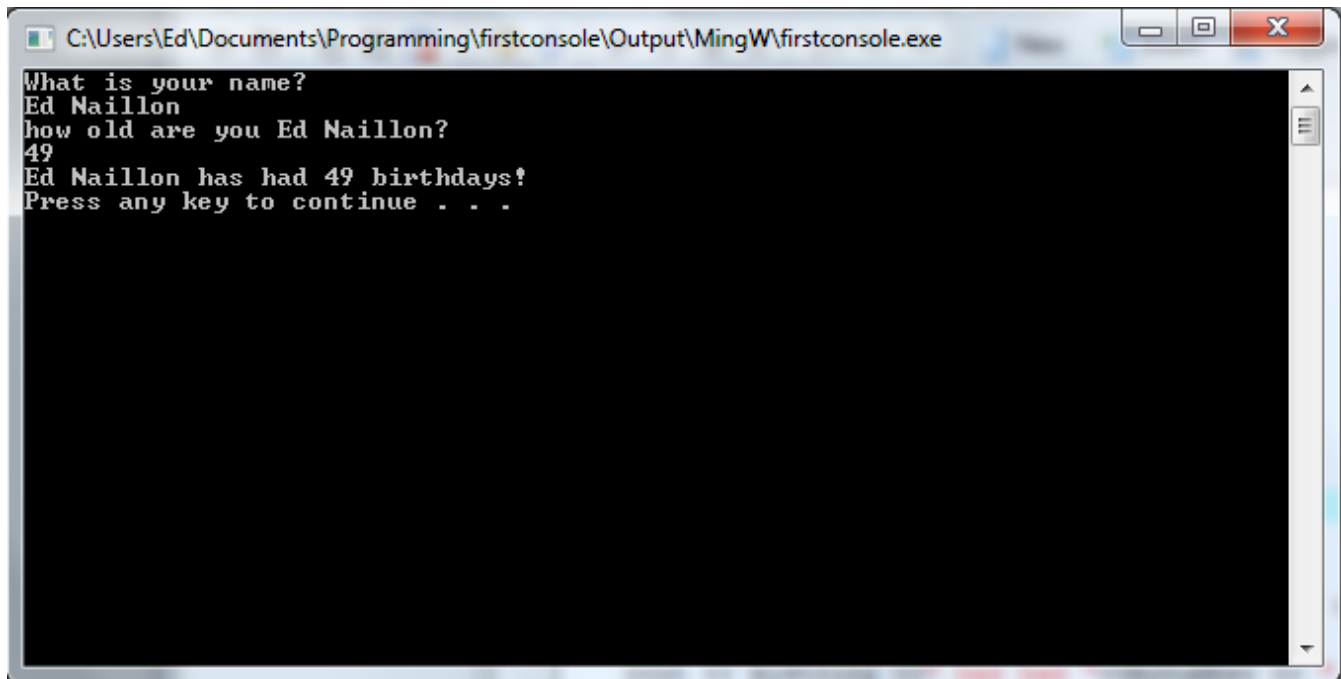


```
#include <cstdlib>
#include <iostream>

using namespace std;
int mynumber;           //add this line
char mystring[40];      //add this line

int main(int argc, char *argv[])
{
    //add the lines from here.....
    cout << "What is your name?" << endl;
    cin.getline(mystring,40);
    cout << "how old are you " << mystring << "?" << endl;
    cin >> mynumber;
    cout << mystring << " has had " << mynumber << " birthdays!" << endl;
    //to here.....and BE CAREFUL TO ADD THEM EXACTLY!!
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

When you are done editing your source code, and you are SURE that it is EXACTLY as above, including spaces and capitalization, press F9 to compile and run your program. If all is well the result will look like this...



```
C:\Users\Ed\Documents\Programming\firstconsole\Output\MingW\firstconsole.exe
What is your name?
Ed Naillon
how old are you Ed Naillon?
49
Ed Naillon has had 49 birthdays!
Press any key to continue . . .
```

Debugging your code

What if this does not work, and your code has errors? I will make a few errors in the code and attempt to compile. Here is the code with the errors in place, can you see the errors?

```
#include <cstdlib>
```

```
#include <iostream>
```

```
using namespace std;
```

```
int mynumber    //add this line
```

```
char mystring[40]; //add this line
```

```
int main(int argc, char *argv[])
```

```
{
```

```
    //add the lines from here.....
```

```
    cout << "What is your name?" < endl;
```

```
    cin.getline(mystring,40);
```



```

cout << how old are you "<< mystring << "?" << endl;
cin >> mynumber;
cout << mystring << " has had "<< mynumber << " birthdays!" << endl;
//to here.....and BE CAREFUL TO ADD THEM EXACTLY!!
system("PAUSE");
return EXIT_SUCCESS;
}

```

When we compile the code above, we definitely get some feedback from DevC++.

In the compiler window at the bottom of the screen, you will see many clues about what is wrong with your code.. For instance on my code below it says that on line 6 there is an initializer expected before “char”. This might not mean much to you now, but it certainly means that right before the word “char” on line 6 there is a problem. But there is nothing before “char” on line 6, so I will look at the end of line 5. Do you see what is wrong? There is a missing semicolon!

Compiler		
Property Inspector Components To-Do List Find Results Debug Compil		
Line	File	Message
6	C:\Users\Ed\Documents\Program...	expected initializer before "char"
	C:\Users\Ed\Documents\Program...	In function `int main(int, char**):
11	C:\Users\Ed\Documents\Program...	no match for 'operator<' in 'std::operator<< [with _Trait
12	C:\Users\Ed\Documents\Program...	`mystring' was not declared in this scope
13	C:\Users\Ed\Documents\Program...	`how' was not declared in this scope
13	C:\Users\Ed\Documents\Program...	expected ';' before "old"
13	C:\Users\Ed\Documents\Program...	missing terminating " character
15	C:\Users\Ed\Documents\Program...	`mynumber' was not declared in this scope
	C:\Users\Ed\Documents\Program...	[Build Error] [Objects/MingW/main.o] Error 1

Now in line 11 it states that there is no match for the < operator. So look in line 11 for <, and you will see if you compare it to the correct code example it should be <<, not < .

Usually if you correct the error at the top, the list of errors in the compiler window will change, as they all depend on the lines of code that successfully run ahead of them. I will correct these two errors. We will look at the compiler window again when I re-compile with F9. Just as I suspected, correcting 2 errors actually got rid of four errors. Look at the images below, and you will see what the compiler says now. It says that there is a problem right before “how”. If you look closely at the correct code on page 15, you will see we forgot a quote in front of how. We are printing “how old are you “ to the screen with

cout, so it needs to be quoted. I Now I will continue correcting line by line, and re-compile with F9 after each correction until my code runs correctly again.

```
main.cpp
#include <cstdlib>
#include <iostream>

using namespace std;
int mynumber;          //add this line
char mystring[40];      //add this line

int main(int argc, char *argv[])
{
    //add the lines from here.....
    cout << "What is your name?" << endl;
    cin.getline(mystring,40);
    cout << how old are you "<< mystring << "?" << endl;
    cin >> mynumber;
    cout << mystring << " has had "<< mynumber << " birthdays!" << endl;
    //to here.....and BE CAREFUL TO ADD THEM EXACTLY!!
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Compiler		
Property Inspector Components To-Do List Find Results Debug		
Line	File	Message
	C:\Users\Ed\Documents\Program...	In function 'int main(int, char**)':
13	C:\Users\Ed\Documents\Program...	'how' was not declared in this scope
13	C:\Users\Ed\Documents\Program...	expected ';' before "old"
13	C:\Users\Ed\Documents\Program...	missing terminating " character
	C:\Users\Ed\Documents\Program...	[Build Error] [Objects/MingW/main.o] Error 1

After correcting that error, the code compiles and runs again.

These basics will allow you to debug the code that I ask you to type in. When looking for errors in your code with the debugger, refer to the sample code that I have given you to help you find the errors. It will be difficult at first, but will become easier. Remember to take the time to learn as you go. Troubleshoot and learn from it. Do not give up. If you take the time to learn while establishing the basics you will not hit a wall later on. So be stubborn, ask for help, but do NOT ask for someone to do it for you. That really helps no one in the long run.

Exercise

1. After making sure that you are logged into your workstation as program, in your “My Documents” Folder, create a new folder called programming, and within that folder create a new Folder called firstconsole2. Remember that each new project must have its own folder!
2. Open DevC++, and create a new project, select Console Application as the project type and name it firstconsole2, making sure it is saved in the firstconsole2 folder. Type in the code on page 15 (The same as in firstconsole) and modify it so that it asks you who your best friend is, and how old they are. Press F9 to compile your code. Work together to find any errors you may encounter.

Structure of a simple program

Before you can ever hope to program a computer, you must learn the basic structure of a computer program. Only then can you define solutions that fit within that structure. All sophisticated interactive programs have a similar structure, whether written for the console or for a graphical GUI. They will have;

Start

The start will be either a main function in a console app or a winmain function in a windows program. It happens when the user types in an executable filename and presses enter at the prompt, or when they double click on a shortcut or program in Windows.

Event loop

This is a structure with in the program that loops continuously waiting for input. Many console apps and Windows apps contain this feature, unless they are a specialized program designed to do one thing then quit. When input is received, it sends control to a different specialized section of code. When that code is complete, control is passed back to the event loop. Predefined events such as pressing Q or pressing a Quit menu item will cause the loop exit and end the program.

Input Facility

An input facility would be a place in the program that will accept key presses or windows that are configured to process mouse movements and clicks.

Decision/Redirect Facility

These structures test values of variable created through user input or programmatic calculation and make decisions to execute specific code based on the value.

Data Storage Retrieval Facility

To be fully useful, programs need the ability to read and write information to files on the drives of a computer or to memory. These facilities provide that capability.

Subroutines

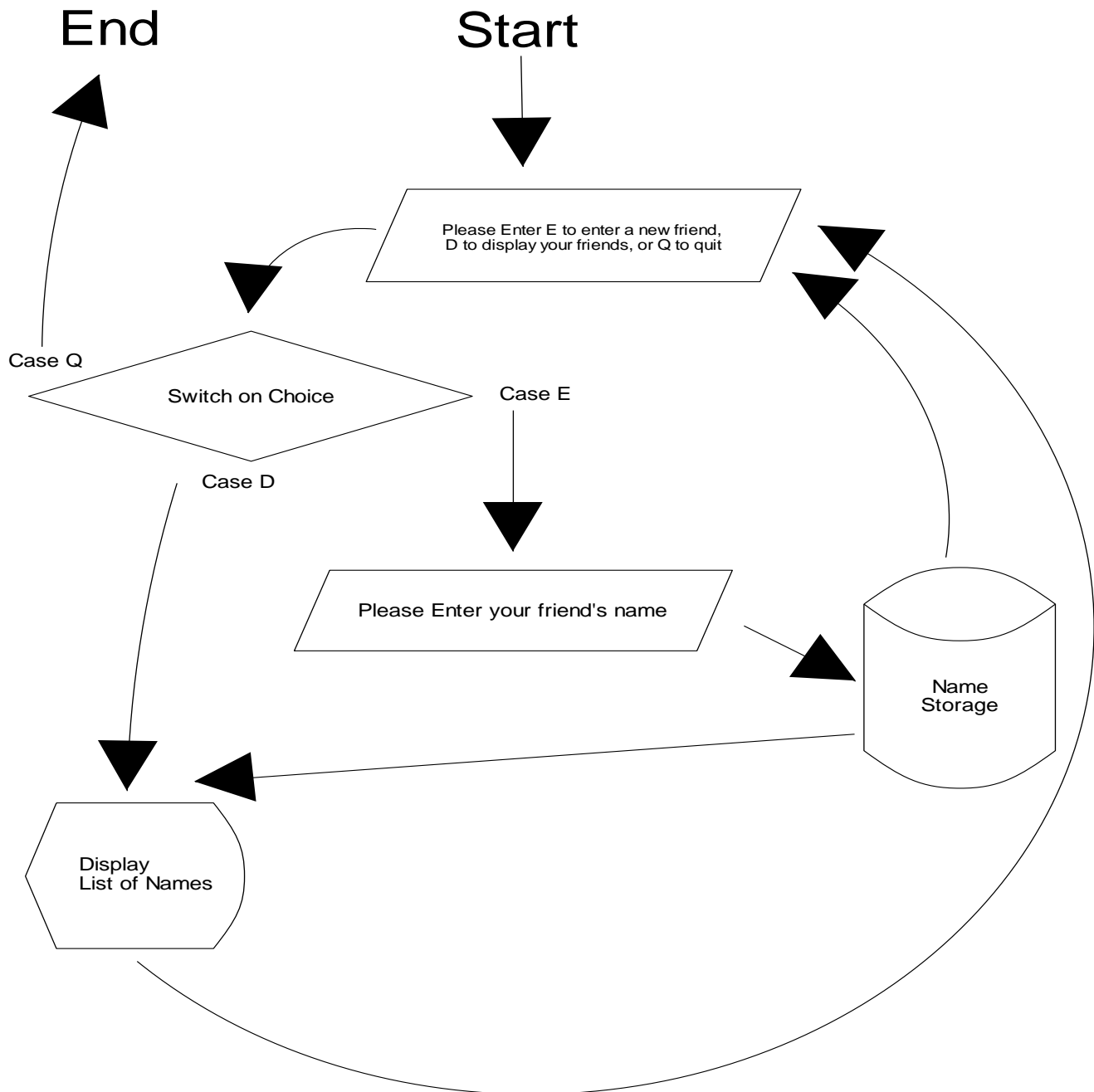
Subroutines are specific blocks of code that deal with specific data and produce prescribed results. This allows a complicated program to be broken down into smaller, simpler code structures that can be called on repeatedly. This saves typing, executable size and helps prevent errors.

Output Facility

If you cannot present the information to the user through a monitor, printer, or other specialized device, your program had no reason for being. These functions take the values of variables, format them correctly and send it to the required device.

End

The program will end and return control to the operating system when a predefined event is intercepted in the event loop. This flowchart will demonstrate how all of the above parts flow together to create a running application.



Notice the definite loop that waits for a choice from the user after every action. This is the event loop. It has two input facilities, where you choose E, D or Q, and where you enter your friend's name, it has a decision/redirect facility where it branches to the input or display subroutine (output facility) depending on your choice, the processing facility would be nested within the display, input and output facility, it both writes to and reads from the storage facility (name storage) and finally an End.

In a Windows program the structure is similar, but buried under Objects and properties of Objects (Object oriented programming ...OOP). The event loop is actually a Window Message handler where your form waits for things to happen such as mouse clicks, button pushes or keyboard entry. The message handler then sends the data to decision and processing structures.

Logic and problem definition

An instructor cannot teach a student how to think. He or she can however, teach a student how to organize thought, and teach them tools and methods for defining and simplifying solutions and problems. If a student does not have this ability, then teaching them a language will be a wash. Spend some time on this first before any code hits the screen.

When defining a solution one needs to ask what types of data am I providing? And what type of data and formatting am I expecting to receive? You then model your loop and decision structure around that. Notice in the above illustration no detail is provided about the processing. This is not usually done in a flow diagram. Think of a flow diagram as an outline of your program just as you would create before writing an essay.

Let's try another example, and go through the whole problem solving procedure. Every program begins with a problem, or a task to accomplish. The problem is a story problem of sorts, and the programmer converts it into a story problem that can solve itself, once the variables of the problem are provided.

Problem

The user wants a program that will take a whole number, and convert it into its equivalent bits. Then the program is to display the resulting bits. Remember that bits are base two, and are represented by 1s and 0s.

Step 1, Simplify the problem

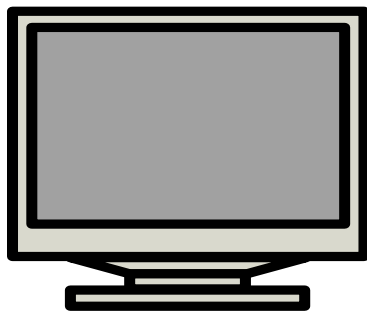
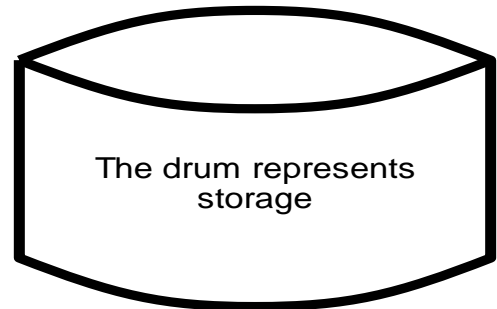
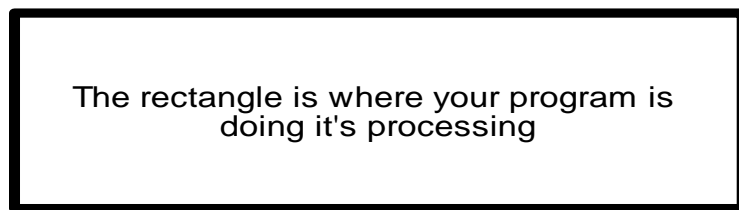
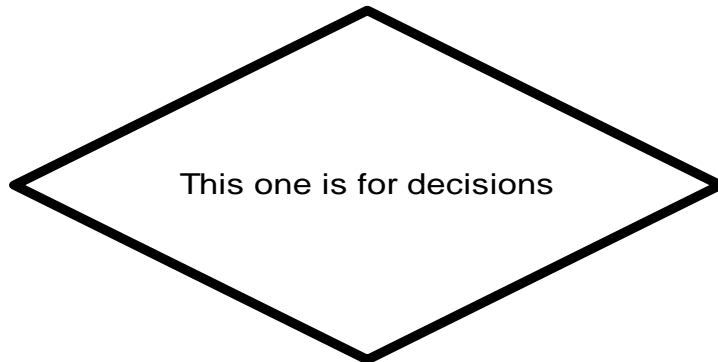
We are entering 1 number, and expecting 1 result. No simplification can be done for this problem.

Step 2, Determine the program's structure

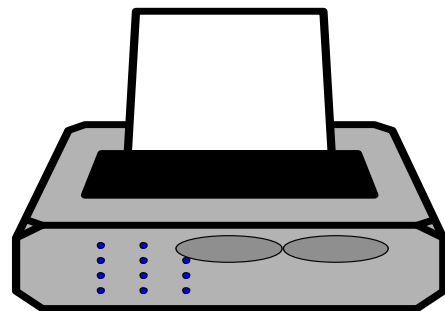
The best way to do this is through the use of a flow chart like I used in the example above. These flowcharts can be hand drawn, or drawn using Paint or a specialized flow-charting application such as Microsoft Visio, or an open source product such as yEd. For the intent of this class, I suggest paper and pencil for your flowcharts.

Here is a run down of the common symbols that your students should use in their flowcharts.

Flowcharts

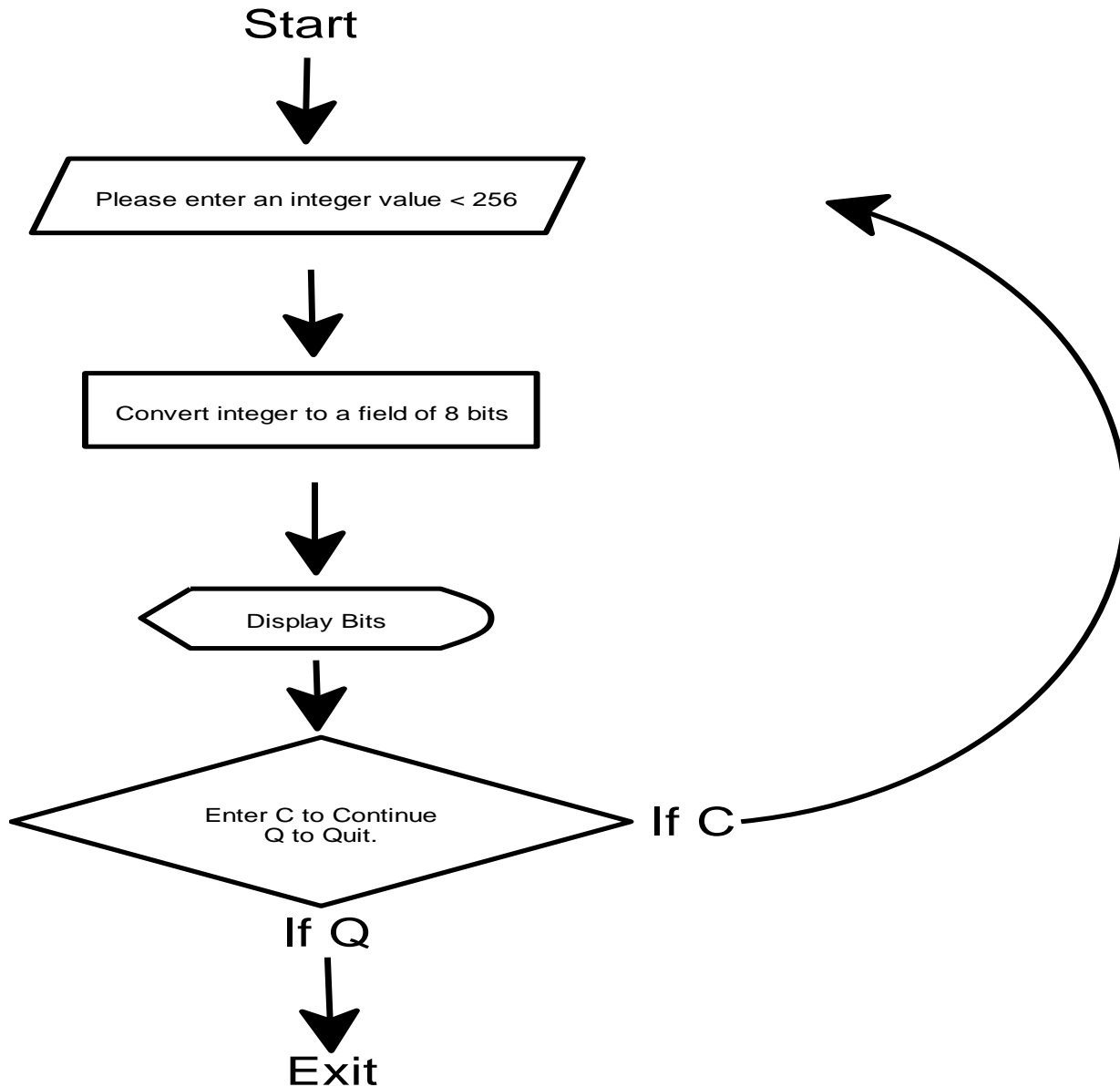


Monitor



Printer

These symbols can be easily drawn by hand. A programmer needs to learn that most of the work is done between the ears, in conceptualization and brainstorming. This simple integer to bit program can be represented in the flowchart below.



Step 3, solving the problem in plain English

The next step of the problem solving stage is to describe the program in plain English, how it operates, what values it needs, etc. This is where the thought goes into the processing facility of the program.

It is required for this program that the student understands bits and bytes. A bit is simply a 1 or a 0; a byte is a sequence of 8 bits that together can represent a maximum value of 255. A computer enumerates with a base 2 system. The diagram below shows a byte of computer information and the placeholder values of each bit in the byte across the top, and how they were calculated along the bottom.

A byte of memory

128	64	32	16	8	4	2	1
1	0	0	1	1	0	1	1
2^7	2^6	2^5	2^4	2^3	2^2	2^1	1

The highest bit can represent a maximum value of 128, or 2 to the 7th power. The next highest bit can represent 64, or 2 to the power of 6. The low bit is used to represent 0 or 1, so that the represented number can be even or odd.

A byte of information can be used to represent a short integer, with a maximum value of 255. If all of the bits in a byte are set to 1, you would add all of the placeholder values together;
 $128+64+32+16+8+4+2+1 = 255$.

The byte in the box above can be converted to a decimal value by adding the placeholders that carry a bit with a value of 1 together. Zeros are ignored. $128+16+8+2+1$ or 155.

Integer to Bits in plain English

A loop can be set up that will step through the bits 7 times. If the bit is positive, or 1, you add the placeholder value that corresponds with the step that you are on. After the loop is done, you test the rightmost bit and add 1 to the total if it is positive. Here is one way to convert a number to bits in plain English.

Start the program; ask the user for an integer less than 256

Set a counter = to 7

Start a loop that loops while the counter is greater than zero

If the integer is at least equal to 2 to the counter's power, print a 1 to the screen, then subtract 2 to the counter's power from the integer. Otherwise print a 0.

Subtract 1 from the counter

Execute the loop again if the counter is greater than zero

Now if the integer is equal to 1, print a final 1 to the screen, otherwise print 0.

Ask the user if they want to continue, or quit

If they want to continue, execute the program again.

The idea behind the plain English solution is to organize the solution, and to make sure that the programmer has a good concept of how to solve the problem before they start coding. The answer lays in the mind, not in the editor, so train the mind to think. After training, the programmer will start thinking in the framework of loops, decision trees and conditional statements. The functions that you know are not as important as how you put them together. Logical thought is the most important aspect of programming.

Here is the actual C code of the same program into bits I inserted it here so that you could look for the similarities between the plain English, the flow chart and the actual programming language.

```
#include <cstdlib> //standard C library
#include <iostream> //standard input and output library
#include <math.h> //lets us do math!
using namespace std; // the program will look in the standard library
//to find the functions that we use

int main (void)
{
    char c; //creates a character variable called c
    int Intmycounter; //creates an integer variable
    double Dmynumber; //creates a double variable
    do
    {
        cout << "Please enter a number from 0 to 255" << endl; //asks you for a number
        cin >> Dmynumber; //puts your answer into the double variable we created

        Intmycounter = 7; //sets a counter equal to 7
        do //starts a loop
        {
            if (Dmynumber >= pow(2,Intmycounter)) //checks to see if the number is
            { //greater than 2 to the power of counter
                cout << "1"; //if it is print a 1
                Dmynumber -=pow(2,Intmycounter); //and subtract 2 to the power of counter
            }
            else
                cout << "0"; //otherwise print a 0 bit to the screen
            Intmycounter--; //subtract one from the counter
        }while(Intmycounter > 0); //do the loop until the counter is used up
        if (Dmynumber == 1) //look to see if the remainder of your number is one
            cout << "1" << endl; //if it is, print the final bit as a 1
        else
            cout << "0" << endl; //otherwise print 0

        cout << "Press Enter Key to continue, q to quit" << endl;
        while (getchar() != '\n');
        c = getchar(); //we are getting a single character as your choice
    }while(c != 'q');//do the loop over and over until you decide to quit //by
        //pressing q
    return 0;
}
```

I have taken you through the process of designing, documenting in plain English, and charting the solution to a problem. Always include these steps to some degree in your class work. Also bear in mind that the completed code may not follow the flowchart and initial concepts completely. This is a natural evolution of the problem solving process. Some streamlining occurs when the solution is being coded. For example, here is another C solution that has been streamlined from the first C code. It produces the same result using fewer mathematical operations.

```
main.cpp
#include <cstdlib>
#include <iostream>

using namespace std;

int main(void)
{
    char result[9];
    int i, opval, placemat;
    do
    {
        cout << "Please enter a number between 0 and 255" << endl;
        cin >> opval;
        strcpy(result, "00000000");
        placemat = 128;
        for(i=0; i<=7; i++)
        {
            if (opval >= placemat)
            {
                result[i] = '1';
                opval -= placemat;
            }
            placemat /=2;
        }
        result[8] = '\0';
        cout << result << endl;
        cout << "Please Press q to quit, Enter key to continue" << endl;
        while(getchar() != '\n');
    }while(getchar() != 'q');
    return 0;
}
```

Exercises

1. Start a new console project, call it inttobits and save it into your programming folder in a folder called inttobits
2. Type the above code into a source code file and compile. Debug your code until you get your project to run.
3. The code on page 25 contains comments. Comments are preceded by // and can be used to “explain” your code. Look closely at the code and comments on page 25, then on the program that you just wrote using the code on page 26, comment as many lines as you can, explaining what each line does.
4. Can you identify the various elements of the program structure? How many of the following can you identify? Print out your code from your program and identify as many of the following as you can.
 - **Start**
 - **Event loop**
 - **Input Facility**
 - **Decision/Redirect Facility**
 - **Data Storage Retrieval Facility**
 - **Subroutines**
 - **Output Facility**
 - **End**

Main routine

Every console app has a main routine. This is the routine that is essentially called by the operating system when the program starts. A C console app will include at very minimum the following code. This program will compile and run, but will do absolutely nothing except return control to the operating system.

```
int main (void)
{

return 0; //returns 0 to the operating system, indicating all is well
}
```

What does a program need to do to be useful? It needs to get information, process information and display or send information. It is also useful to be able to store information in memory or on the drive for later retrieval. The C language includes many functions for performing these tasks. They are pre written for you so that you can call them by name and use them. In order to access these functions you must include the pre written code in your source code. We are going to add the pre written code for the standard standard C++ functions and for input and output. This code is contained in the files `cstdlib` and `iostream`. Include statements should be among the very first in your code. The third line, “using namespace std;” allows us to call the functions conveniently without first referencing the standard library in code.

```
#include <cstdlib>
#include <iostream>
using namespace std;
```

```
int main (void)
{
return 0;
}
```

The <> symbols tell the compiler to look in its own root directory for the files. If the source code files that you want to include are contained in your program's directory, use quotes instead.

```
#include <cstdlib>
#include <iostream>
using namespace std;
```

```
int main (void)
{

return 0;
}
```

All right, we can access the prewritten input output functions, but we still are not doing anything. The cout statement is a member of iostream, so we will use it in our program to print a message. The endl in the cout statement causes the running program to print a carriage return after the text. Notice how the << operators point to the cout function, they “point” the text to the console!

```
#include <cstdlib>
#include <iostream>
using namespace std;
```

```
int main (void)
{
cout << "Hello Free world!" << endl;
system("PAUSE");
return 0;
}
```

Variables

A variable in C is a named value that can be stored in memory, accessed and changed. A variable name can contain upper and lowercase letter, the underscore, and numbers. A variable name cannot start with a number, or be named the same as any other C reserved keywords or function names. Variables must be declared prior to use according to one of the C data types.

Standard types of data that can be represented in C include;

Boolean (bool) 1 byte

This value can be set to true or false. (0 or 1) .

Character (char) 1 byte

This can store a small integer or character. It is stored in 1 byte and has a maximum value of 255.

Short Integer (short) 2 bytes

This is an integer or Unicode character(wide char) stored in 16 bits. It can be signed or signed. An unsigned short can represent a maximum value of 32,767.

Integer (int) 4 bytes

An int is a 32 bit signed or unsigned number. An unsigned in can hold a maximum value of 4294967295

Long Integer (long) 4 bytes

Same as integer.

Long Long 8 bytes

A 64 bit integer. BIG. 18446744073709551615

Floating Point (float) 4 bytes

This type is different for different compilers. About the same value as an int, but can represent decimal precision as well.

Double (double) 8 bytes

This type is different for different compilers. About the same value as an long long, but can represent decimal precision as well.

Long Double (long double) 12 bytes

A Long Double is a floating point with extended precision displaying approximately 19 digits.

Variables must be declared as <type> <name>, where type is one of the standard C data types, and name follows variable naming conventions. A variable name is case sensitive. Myint is not the same as MyInt.

An integer variable can be declared like this;

```
int myint;
```

Also, multiple variables of the same type can be declared on the same line like this;

```
int myint1,myint2;
```

A char would be declared like this;

```
char mychar;
```

A variable can be filled with a value to, like this;

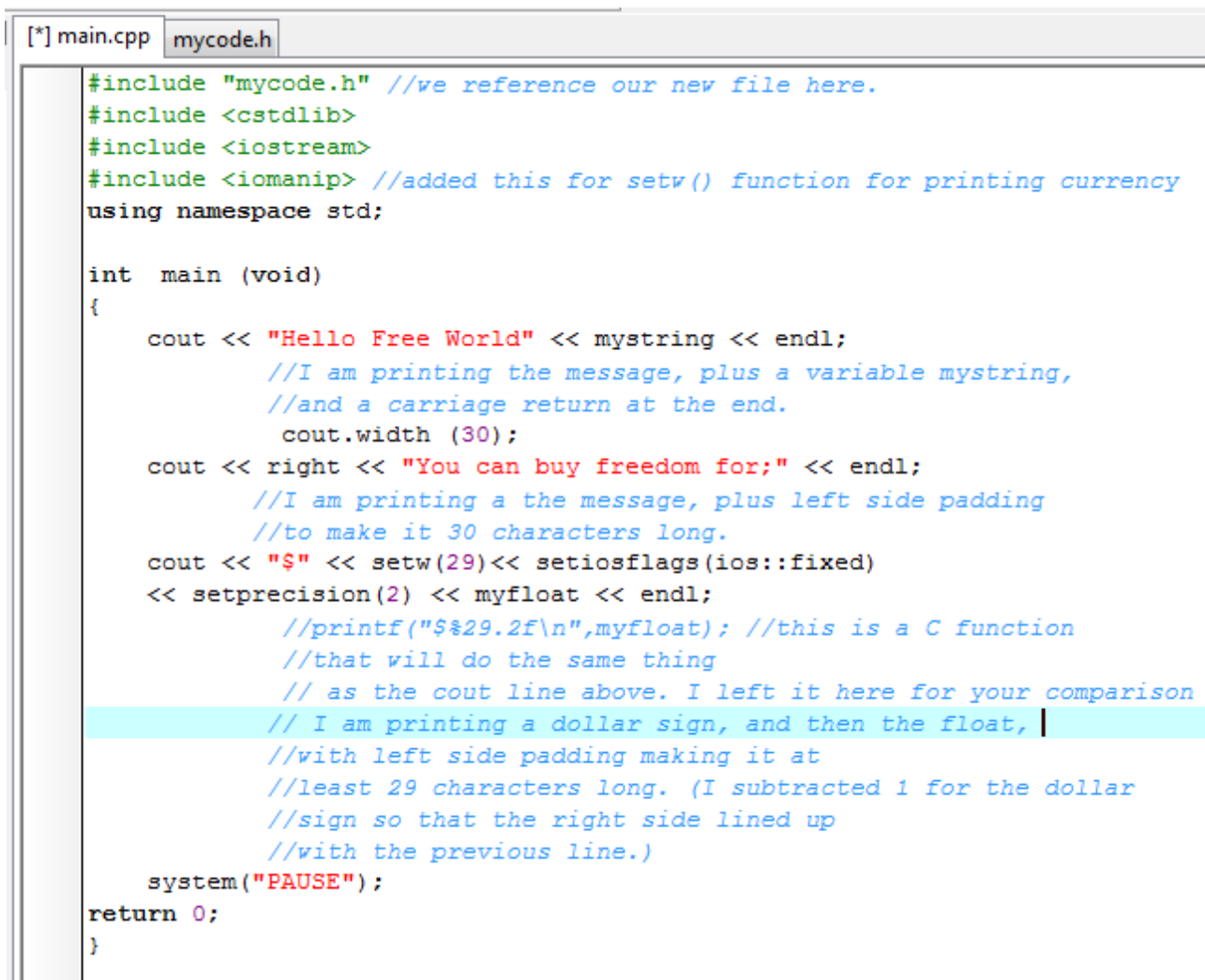
```
int myint = 5;
```

or

```
char mystring[] = "I am a line of text";
```

Let's use one in our simple little app. This lesson will help you understand include files, integers, cin , cout and printf statements. I have two source files, main.cpp and the include file mycode.h. Notice how it is referenced in the top line of main.cpp.

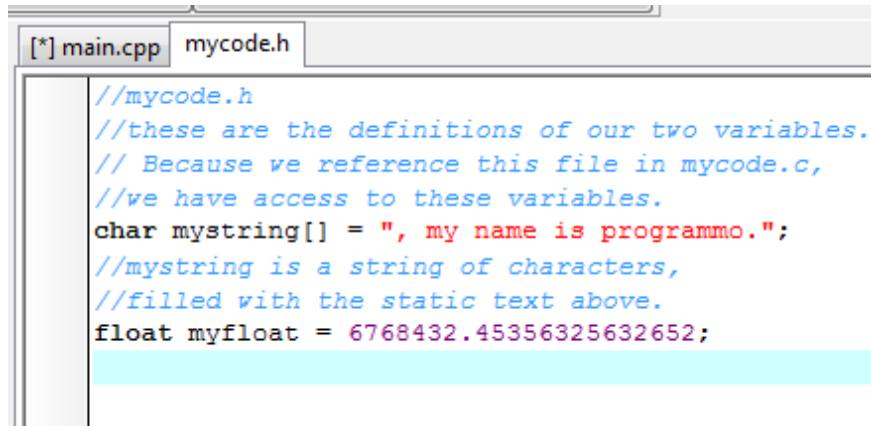
Create a new console project, call it variables1, create a folder for it in your programming folder. Type this code in the main.cpp file.



```
[*] main.cpp mycode.h
#include "mycode.h" //we reference our new file here.
#include <cstdlib>
#include <iostream>
#include <iomanip> //added this for setw() function for printing currency
using namespace std;

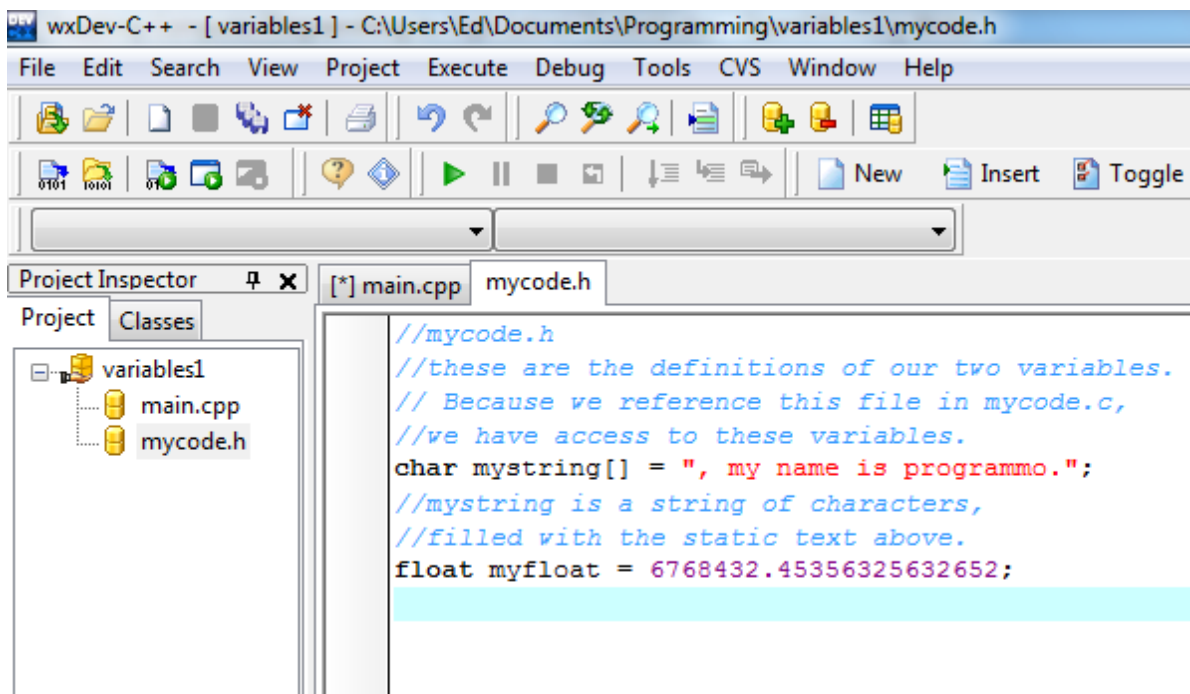
int main (void)
{
    cout << "Hello Free World" << mystring << endl;
        //I am printing the message, plus a variable mystring,
        //and a carriage return at the end.
        cout.width (30);
    cout << right << "You can buy freedom for;" << endl;
        //I am printing a the message, plus left side padding
        //to make it 30 characters long.
    cout << "$" << setw(29)<< setiosflags(ios::fixed)
    << setprecision(2) << myfloat << endl;
        //printf("$%29.2f\n",myfloat); //this is a C function
        //that will do the same thing
        // as the cout line above. I left it here for your comparison
        // I am printing a dollar sign, and then the float, |
        //with left side padding making it at
        //least 29 characters long. (I subtracted 1 for the dollar
        //sign so that the right side lined up
        //with the previous line.)
    system("PAUSE");
    return 0;
}
```

From the file menu, choose New|Source File. Type the code below into that new blank file that appears in the editor, and save it as mycode.h.



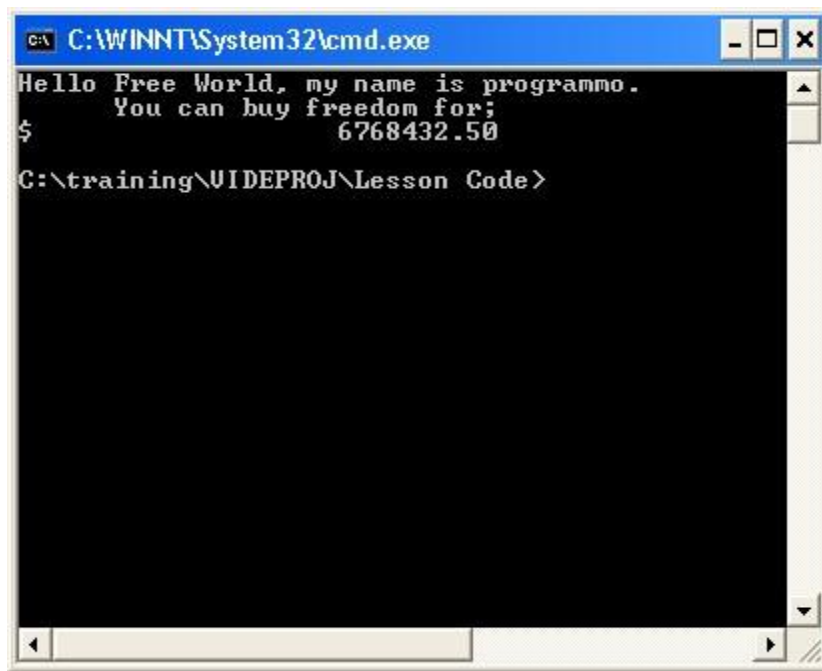
```
[*] main.cpp mycode.h
//mycode.h
//these are the definitions of our two variables.
// Because we reference this file in mycode.c,
//we have access to these variables.
char mystring[] = ", my name is programmo.";
//mystring is a string of characters,
//filled with the static text above.
float myfloat = 6768432.45356325632652;
```

Your project explorer will now look like this. Notice the new source file!



Press F9 to compile your program and run it. You may have to debug the code, use your compiler messages and refer to the code above to correct your typing errors. When you run the program you get the following aligned output. Notice to the right of the decimal, .50 prints instead of .45356325632652.

I wonder if any one of you watched the old movie "Office Space" They had a plan to get rich from that 7 tenths of a cent!!



```
C:\WINNT\System32\cmd.exe
Hello Free World, my name is programmo.
You can buy freedom for;
$ 6768432.50
C:\training\VIDEPROJ\Lesson Code>
```

Arrays

In the above code, we defined a variable in the mycode.h file named mystring. It was of the type char, and was followed by two brackets.

The definition `char mystring[]` creates an array of characters linked together. When a character array is filled when it is created like in the above code, you do not have to specify how many elements are in the array because the compiler counts the characters in the assignment statement;

```
char mystring[] = ", my name is programmo."
```

The variable mystring is actually 24 characters long. The compiler adds a null character to the end of the array to terminate the array in memory. That way a program will stop reading data in before it goes beyond the end of the mychar array. Each element of the array can be accessed or set individually. The statement `mystring[0] = 'A'` would set the first character of the string to A.

The array mystring could be declared as `char mystring[24]`. If you do not know when you are writing the code what will be contained in the array, then make sure it is more than large enough to hold the data you want to get. The null character at the end of the data will truncate the array. You can have multi-dimensional arrays too, declaring them like this, `arrayname[number of elements][number of elements]`;

Input/Output

A computer program needs to be able to prompt the user for data, and assign that data to variables, manipulate that data and pass that information back to the user. I will show several ways of doing this,

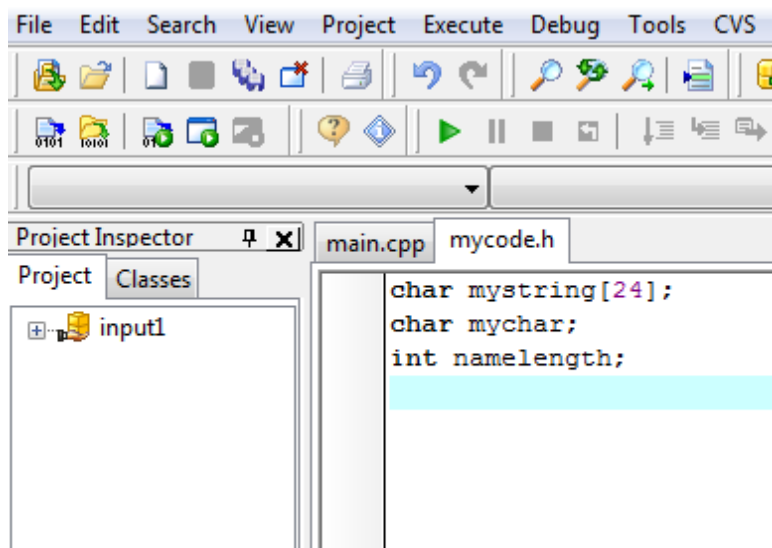
and methods to get different types of data. I will introduce strings to you in the next program, and how to get them from and display them to the user.

Strings

I will have you type in a program that will ask you for your first name, then your last initial. It will splice the two together and give you some information about the variables in the output of the program.

Here I introduce a new include file at the top of main.cpp. The include file string.h is a standard string library which will give me some new string handling functions. I will access specific elements of the mystring array and splice data to it. I will introduce the strlen function, which gets the length of a string for the programmer, and also the sizeof variable which returns the actual allocated space in memory that is afforded a variable. Another function introduced here is the getchar () function, I am using it to clear the keyboard buffer of the carriage returns.

In the mycode.h file you will see that I added two new variables. I created a char variable that I call mychar to hold a single character that I will retrieve from the keyboard. I then create a new integer variable to hold the length of the name that I will retrieve with the strlen function.



In the main.cpp file I have added lots of new code, so that you can see additional formatting techniques, learn to access specific elements of array variables, and become exposed to a few new functions.

main.cpp	mycode.h
<pre>#include <cstdlib> #include <iostream> #include <string.h> //thisline gives me access to C functions for string handling #include "mycode.h" using namespace std; int main(int argc, char *argv[]) { cout << "Please Enter your first name -> "; //print a prompt to the screen cin >> mystring; //simple input, only gets single words while (getchar() != '\n');// in case you space out and enter //more than just the first name, this clears the cin buffer cout << "Thank you " << mystring << "!" << endl; //Thanks the user with a multiple cout, and moves to teh next line down cout << "Oops I forgot to ask for your Last initial! -> ";//nothing new here cin >> mychar; //gets a single letter, your initial and puts it in mychar namelength = strlen (mystring);//Here I pass the strlen function mystring as a //parameter. The strlen function then fills the integer I created in the // mycode.h file with the value. cout << "Your first name is " << namelength << " characters long.\n"; // prints the value of namelength, telling you how long your name is cout << "The variable mystring is " << sizeof(mystring) << " bytes long.\n"; //in this cout statement I am printing the return value of the sizeof function. //I did not create a variable to hold this information because I just want to //display the value for information. This is how many letters you could // put in mystring</pre>	

//Continued below.....

```
mystring[namelength] = ' '; //adds a space right after your first name
//Here I am accessing a specific element of the character array.
//The index (position) is in between the brackets and I am using the
//variable namelength which I set equal to the length of the string above.
// Now I want to add to it, but in order
// to do that I have to overwrite the value with a space so that there is
//a space between your first name and initial
// A character array //starts at position 0, so the length of the string
//actually point to the
//next position past the last character.

mystring[namelength +1] = mychar;//because I overwrote the terminating
//character, I can append the Last Initial here,just past the space.
//we got your initial up above, with the cin >> mychar; statement

namelength = strlen (mystring);//Here I pass the strlen function mystring as a
//parameter. The strlen function then fills the integer namelength with the
//length of the string

mystring[namelength+2] = '\0';
//I have to manually append a terminating character after the initial.

cout << "Your name and initial are " << namelength << " characters long.\n";
cout << "The variable mystring is still " << sizeof(mystring)<< " bytes long.\n";
//

cout << "Thank you " << mystring << ", I won't trouble you further.\n";
system("PAUSE");
}
```

Exercise

1. Create a new console project called input1, creating a folder for it in your programming folder. You are remembering to have a new folder for each project aren't you?
2. Use the file menu to create a new source file, naming it mycode.h. And type the code shown on page 33 into it, then in your main.cpp file, type in the code on pages 34 and 35. Save, debug, and run your program.

Getting lines of text.

The program above gets single words and characters from the user, but that may not be very useful if we want to get a complete sentence from the user. Here I will introduce the `cin.getline` function. This function needs the name of the char array variable and the maximum length of characters to “get” from the user. So for the program above we would have used `cin.getline(mystring,23)`; We need to remember that a character array will hold one less character than the size that it is declared at, to allow for a terminating character at the end. Think of it as a plug on the end of a pipe full of letters. The last one is the plug.

```
#include <iostream>
using namespace std;

int main () {
    char name[256], title[256];

    cout << "Enter your name: ";
    cin.getline (name,256);

    cout << "Enter your favorite movie: ";
    cin.getline (title,256);

    cout << name << "'s favorite movie is " << title <<endl;
    system("PAUSE");
    return 0;
}
```

Pointers

Another cool way to get complete lines of text is with the use of pointers.

Pointers allow the programmer to point to a specific numerical address in memory. Every variable is contained in memory during the program’s operation, and if a programmer knows exactly where the data is he can access it directly. This allows for addresses to be passed to functions, and the changes to data made in that function do not have to be passed back because they have been written to memory.

The following program will demonstrate the creation and use of pointer variables, and also introduce two new input functions, `getchar ()` and `gets ()`. I will also give you a preview of the flow control routine `while`.

```

#include <cstdlib>
#include <iostream>
using namespace std;

char mystring[24]; //this string holds the name
char myinfo[40]; //this string will hold the information
char *c;          //this is a pointer to a character
int charcount = 1; //this is an integer variable initialized to 1

int main(int argc, char *argv[])
{
    c = &mystring[0]; //I point the character pointer to the first
                       //element of mystring
    cout << "Please Enter your full name. -> ";
    gets(c); //This neat string function gets the whole line of text,
             //spaces and everything.
    cout << "Thank you " << mystring << endl;
    cout << "Could you tell me about yourself (under 40 characters)? \n->";
    c = myinfo; //I am pointing the character pointer to the first character of
                //myinfo. Note that on a string, I do not have to use the
                //& operator, or specify the first element with [0]
                //This statement is equivalent to c = &myinfo[0]

                //Now I will execute the while flow control routine. Every time
                //it loops it executes the code within it's parenthesis and also
                //the code between the brackets { and }. It will check to make
                //sure that you have not pressed enter or the count has not
                //exceeded 40 characters.
    while((( *c = getchar()) != '\n') && (charcount < 40)) //the while loop!!!!
    { //loops from here, getting a char in the line above
        c++; //each time the loop executes, it adds one to the pointer, so
             //that that character from getchar() is written to the next
             //location in the string
        charcount++; //Lets add one to count with each character.
    } //to here and does the stuff between the brackets!!

    c++; //after the loop is done, we have to move the pointer ahead
         //because we need to write the terminating character
    *c = '\0'; //we terminate the string with character zero
              //this is the plug on the end of the pipe
    cout << myinfo << endl; //no matter how much information we typed, only
                          //40 characters are stored.
    system ("PAUSE");
    return 0; //this returns a 0 to the operating system notice at top it now
             //says int main(void) instead of void main(void).
}

```

There is a `string.h` function that is specifically designed to splice two strings together. I did not use it because I would rather show you how it worked, and also I wanted to show you what you can do when you can manipulate data one element at a time. The `strcat` function will take two strings, put them together and place the result in the first parameter. It is called like this;

```
strcat (char *dest, char *src);
```

Here is a small program that declares a char array, puts a string in it, and then splices a string to the end of it. I introduce two new string functions, the `strcpy` function, which copies a string to another string, and the `strcat`, which adds or splices one string to another. I made `deststring` long enough to hold extra characters. When you do this type of splicing, make sure your destination variable is plenty big.

```
#include <cstdlib>
#include <iostream>
#include <string>
using namespace std;

int main(int argc, char *argv[])
{
    char deststring[50]; //new 50 character string , make sure it is long enough
                        //for anything we want to splice into it

    strcpy (deststring, "Hey Howdy Hey!");
    //this functions copys data to a string, puting it to the first
    //posistion (element 0)

    strcat(deststring, " I'm a yo yo!");
    //This function takes the second parameter (the " I'm a yo you!")
    //and appends it to the first string, automatically taking care of
    //the terminating characters.
    cout << deststring << endl;
        system("PAUSE");
        return EXIT_SUCCESS;
}
```

Exercise

1. Using what you have learned so far, using `cin`, `getline`, `strcpy`, `strcat`, variables, arrays, pointers , input and output, create a new console project name `myfirstprogram`. Make sure it has its own folder.
2. Create a unique program that
 - Gets a number from the user
 - Gets a name from the user and stores it in a variable
 - Gets a full sentence from the user and stores it in a variable
 - Uses `strcat` to create a unique message that you display to the user.

GOOD LUCK THIS IS YOUR FIRST REAL PROGRAM!!

Flow Control and Decision Structures

A computer program will have a hard time doing much work if it cannot compare values, make decisions and direct the execution of the code based on conditions. That is why all high level languages have flow control routines. These control routines are used in conjunction with functions and comparative operators to make decisions in an organized way.

Conditional Statements

The workhorse if () statement evaluates a condition, and then can execute different code depending on the value of that condition. For example, look at the following code.

```
#include <cstdlib>
#include <iostream>

using namespace std;
int age;
int main(int argc, char *argv[])
{
    cout << "Please Enter your age -> ";
    cin >> age;
    if (age <= 12)
        cout << "What a youngster!\n";
    else if (age <= 19)
        cout << "Ahh those teenage years!\n";
    else if (age <= 40)
        cout << "Not over the hill yet.\n";
    else
        cout << "Let's just keep quiet about your age.\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

You can have as many else if statements as you want, and the final else is what executes if any other condition is true.

The if, while and switch statements

The next program will use the if () statement, the while () statement that was introduced before, and will show you a new one, the switch () statement. I am also using some of the string functions again for your review. We will also demonstrate nested flow control with the while () being within an else, and another if () within that while () statement

```
#include <stdio.h>
#include <iostream.h>
#include <string.h>

char authorizedname[] = "Edward";//This is a static string, initialized at
//declaration It is the only user that will be allowed
char username[40];
char password[20];
char authorized = 'y';//I preset this character to yes. If you give the wrong
//credentials,I will set it to n

int main(void)//notice I am using int main(void) now
{
cout << "Please Enter your name \n->";
cin >> username;//we are scanning in a string by filling username with the cin

if (strcmp(authorizedname,username))//we compare the preset username with the
//name you just entered. If it returns a
//value other than 0 (0 differences)
//there is a difference.
{ // there is a difference if this executes
    cout << "Hey, you are not Ed!\n";//notice my use of brackets {} in
    //the if statement to group code.
} //end of code that executes if there is a difference
else

{ //strcmp returned 0, so all is good
    //there is no difference between the username and the authorizedname
    //this bracket groups all this code
    //to the else statement.
    cout << "Hey Ed. how are you?\n"; //welcome
```

//continued on the next page...


```

while (strcmp(password,"topsecret") != 0) //this starts a while loop that
//keeps asking for the password untill you get it right by comparing
//what you enter to "topsecret" that is the power
//of loops, and while loops in particular
{ //this bracket starts all the code to be executed with the
  //while statement
  cout << "please enter password,q to quit\n"; //ask
  cin >> password; //get it from the user
  if (strcmp(password,"q") == 0) //this lets the user give up
    //and quit
  { //this bracket starts the if statement conditional block
    authorized = 'n'; //set the authorized flag to no
    break; //this will force the code execution to exit
    //while statement
  } //ends the if block
} //this bracket ends the while block of code
} //this bracket is the end of the else statement code segment

switch (authorized) //the switch statement takes the value of a variable
//and will direct code based on it's value. The switch
//can only test integers and single character chars
{ //there is always a bracket after the switch to group the case statements
case 'n':
  cout << "Sorry you are not granted access.";
break; //if it was n, then leave the switch statement after printing
case 'y':
  cout << "Congratulations, you have full access to nothing! \n";
break;
default: //if it was neither y or n, this would happen but the way this
//program is written, it will always be one or the other.
  cout << "You will never see this message";
}
system ("PAUSE");
return 0;
}

```

Exercise

1. Create a new project called flowcontrol2, and create a new console project called flowcontrol 2 and save it into the new folder. Type in the code above, and pay close attention to the comments as you type them in.
2. Press F9 to run your code. I predict you will spend some time debugging. Do not give up, just compare it line by line to the code above and make sure each line is exact as you go through the compiler messages!.

Another very powerful flow control routine that deal with repetitive tasks is the for () loop. Of all the high level languages, the C language provides the most powerful for () loop. The for loop executes a block of code for a fixed number of repetitions. Its syntax is as follows.

```
for (initialize the variable; test for conditions; increment the variables)
{
statements, multiple lines in braces...
}
```

Make sure you always place a semicolon between the initialize, condition and increment statements.

Look at the following program. Can you predict what it does?

```
#include <cstdlib>
#include <iostream>
#include <iomanip>
using namespace std;

int main(int argc, char *argv[])
{
    int i; //local variable for our loop
    cout << " " << (char) 201; //this line prints two spaces, followed by character
                                //201, which is an upper left corner of a double
                                //line border.
    for (i = 1; i < 63; i++) cout << (char)205; //this single line for loop prints
                                                //63 character 205s
                                                //which are double line border tops.
    cout << (char) 187 << endl; //this prints a right upper corner, followed by a
                                //carriage return.

    for(i = 32; i <= 79 ;i++) //this loop starts at 32, executes until i is equal to
                                //79, and adds one to i every time it executes
    { //so the loop runs from here
        cout << " " << (char) 186 << " Character " << i << " is a -> " << (char) i
        << "\t\tCharacter " << setw(3) << i+48 << " is a -> " << setw(3) << (char) i+48
        << (char)186 << endl;
        //This complicated cout statement executes every time the for loop executes.
        //It uses the value of i as set by the for statement on each iteration.
        //This print statement will print two spaces, followed by character 186
        //(left double border), two more spaces, then the word "Character", then a
        // space and the value of i, then a space and the words "is a ->"
        //then it prints another space and then the character represented by the value
        //of i. It then prints two tabs, then the word "Character" followed by a
```

```
//space and then the value of i + 48 (because I want two columns of
//information, with the first column containing 48 rows), then it prints a
//space and the words "is a ->", another space and then the
//character represented by I +48. Then it prints one final space and then
//character 186, or //the right double border. That is a powerful line of code.
//the setw(3) makes sure what I print is at least 3 characters long for
//alignment
} //and the loop ends right here

cout << " " << (char) 200; //now I print two spaces and a lower left corner
for (i = 1; i < 63; i++) cout << (char)205; //I print 63 lower double borders
cout << (char)188 << endl; //finally I print a lower right corner.

system("PAUSE");
}
```

Here is the output of the above program.

```

Character 32 is a ->
Character 33 is a -> !
Character 34 is a -> "
Character 35 is a -> #
Character 36 is a -> $
Character 37 is a -> %
Character 38 is a -> &
Character 39 is a -> '
Character 40 is a -> (
Character 41 is a -> )
Character 42 is a -> *
Character 43 is a -> +
Character 44 is a -> ,
Character 45 is a -> -
Character 46 is a -> .
Character 47 is a -> /
Character 48 is a -> 0
Character 49 is a -> 1
Character 50 is a -> 2
Character 51 is a -> 3
Character 52 is a -> 4
Character 53 is a -> 5
Character 54 is a -> 6
Character 55 is a -> 7
Character 56 is a -> 8
Character 57 is a -> 9
Character 58 is a -> :
Character 59 is a -> ;
Character 60 is a -> <
Character 61 is a -> =
Character 62 is a -> >
Character 63 is a -> ?
Character 64 is a -> @
Character 65 is a -> A
Character 66 is a -> B
Character 67 is a -> C
Character 68 is a -> D
Character 69 is a -> E
Character 70 is a -> F
Character 71 is a -> G
Character 72 is a -> H
Character 73 is a -> I
Character 74 is a -> J
Character 75 is a -> K
Character 76 is a -> L
Character 77 is a -> M
Character 78 is a -> N
Character 79 is a -> O
Character 80 is a -> P
Character 81 is a -> Q
Character 82 is a -> R
Character 83 is a -> S
Character 84 is a -> T
Character 85 is a -> U
Character 86 is a -> V
Character 87 is a -> W
Character 88 is a -> X
Character 89 is a -> Y
Character 90 is a -> Z
Character 91 is a -> [
Character 92 is a -> \
Character 93 is a -> ]
Character 94 is a -> ^
Character 95 is a -> _
Character 96 is a -> `
Character 97 is a -> a
Character 98 is a -> b
Character 99 is a -> c
Character 100 is a -> d
Character 101 is a -> e
Character 102 is a -> f
Character 103 is a -> g
Character 104 is a -> h
Character 105 is a -> i
Character 106 is a -> j
Character 107 is a -> k
Character 108 is a -> l
Character 109 is a -> m
Character 110 is a -> n
Character 111 is a -> o
Character 112 is a -> p
Character 113 is a -> q
Character 114 is a -> r
Character 115 is a -> s
Character 116 is a -> t
Character 117 is a -> u
Character 118 is a -> v
Character 119 is a -> w
Character 120 is a -> x
Character 121 is a -> y
Character 122 is a -> z
Character 123 is a -> {
Character 124 is a -> |
Character 125 is a -> }
Character 126 is a -> ~
Character 127 is a -> ^
C:\training\VIDEPROJ\lesson8>

```

That is a lot of formatted output for very few lines of code. It speaks to the power of the for () loop.

Exercise

1. Create a new project called flowcontrol3, and create a new console project called flowcontrol3 and save it into the new folder. Type in the code above, and pay close attention to the comments as you type them in.
2. Press F9 to run your code. I predict you will spend some time debugging. Do not give up, just compare it line by line to the code above and make sure each line is exact as you go through the compiler messages!
3. Learning what you did from the lessons above, create a folder called nameinabox, create a new console project called nameinabox and write a program that draws a box to the screen, with your name in it. The box will be similar to the one shown around the output in the program above.

Hint. The values 188, 205, 201, 187, 200 and 186 print different parts of the "box".

To print part of the box for example you would write;

```
cout << (char) 188;
```

Good luck and do not give up!

Subroutines

A computer program can get large difficult to debug when things go wrong. It is also required that the same operation sometimes needs to be carried out time and time again during the execution life of the program. If a program is written in a purely linear fashion, then it would be required to write and rewrite the same blocks of code again and again to duplicate their functionality. As a result of this design necessity, all high level languages incorporate the ability to write code in modular, reusable segments called subroutines. All of the functions that we have used so far, the strcmp (), strcpy (), printf (), scanf () etc have all been subroutines contained in prewritten libraries.

A subroutine is declared a lot like a variable. One difference is that not only can you declare it as a standard variable type, but you can also pass information to it. The function then returns (or essentially becomes) the value of operation that it performed. Let's take a look. READ THE COMMENTS!!

```
#include <cstdlib>
#include <iostream>
using namespace std;
int number1 = 5;
int number2 = 7;
int myanswer;
int mylittleaddfunction(int num1,int num2);//function declaration
//always above the main routine or in an include file. It specifies (from left
// to right) the type of variable the function returns the name of the function,
//(follow variable naming rules) and within parenthesis the types and names of
//the variables that you will pass it.
int main(int argc, char *argv[])
{
    cout << "Number 1 = " << number1 << " and Number 2 = " << number2 << endl;
    //see how I can print two variables with the two %d specifiers
    myanswer = mylittleaddfunction(number1,number2);
    //I am sending the function both numbers, and getting one back
    //in the form of myanswer. I do not have to state variable types
    //here, just variable names of the same type as declared in the
    //top declaration
    cout <<"I have added number 1 and number 2.\n";
    cout << "The answer is -> " << myanswer << endl;
    //now I can print the return value of mylittleaddfunction
    system("PAUSE");
    return EXIT_SUCCESS;
}

int mylittleaddfunction(int num1,int num2)//here I am writing the sub-
//routine. Identical to the definition except no semicolon at the end
{
    return num1 + num2;//number1 becomes num1, number2 becomes num2
    //they are added and the return statement
    //sends the answer back like a boomerang
    //to the routine that called it.
}
}
```

Check out the output;

```
Number 1 = 5.Number 2 = 7.  
I have added number 1 and number 2.  
The answer is ->12  
  
C:\training\UIDEPROJ\lesson7>_
```

Exercise

1. Create a new folder called subroutines in your programming folder. Create a new console project called subroutines and type the code above into your main.cpp file. Press F9 to compile and run. Debug until your program runs and produces the output shown above.
2. Modify your code so that the program asks you for number1, then asks you for number2, then adds them together.
3. Create a new subroutine called mylittlesubtractfunction. Place it below the mylittleaddfunction subroutine. Modify it so that it subtracts numbers. Call the subroutine like you called mylittleaddfunction so that you can add the numbers you got from the user, print the answer, then subtract the numbers and then print that answer. Make sure that you remember to define the function at the top of your code!

Event loop

The event loop allows the user to run the program, make choices, run the program again with new options, or exit. It loops continuously asking the user for input until he or she choose quit. It is one of the most important parts of an interactive program.

Here is a very simple event loop;

```
#include <cstdlib>
#include <iostream>

using namespace std;

int main(int argc, char *argv[])
{
    char prompt; //creates a character to get a "choice"

    do //does all the stuff between the brackets below in a loop
    {
        cout << "I am waiting for a decision Mr. Smartypants!" << endl;
        cout << "Press Q to quit" << endl;
        cin >> prompt; //brings in the choice
    }while (prompt != 'q'); //and keeps asking for it till you press q

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

Here is the output from the program. It looks like there is only one choice!!

```
I am waiting for a decision Mr. Smartypants!Press Q to quit
w
I am waiting for a decision Mr. Smartypants!Press Q to quit
y
I am waiting for a decision Mr. Smartypants!Press Q to quit
f
I am waiting for a decision Mr. Smartypants!Press Q to quit
d
I am waiting for a decision Mr. Smartypants!Press Q to quit
q
C:\training\VIDEPROJ\event>
```


Here is another example of an event loop that uses a separate function to get the input from the user. Study the code carefully, and compare it to the output. Notice that if you add an option in the switch statement, you will have to test for that character in the while () statement at the bottom of the getcommand () subroutine. Could you use this to make your calculator even better?

```
#include <cstdlib>
#include <iostream>
using namespace std;
char getcommand(void);
int a,b;
int main(int argc, char *argv[])
{
    char command;
    a = 9;
    b = 2;
    while((command = getcommand()) != 'q')
    {
        switch(command) //redirects based on the choice...
        {
            case 'a':
                cout << "You chose add!" << '\n';
                cout << a + b << endl;
                break;
            case 's':
                cout << "So, you want to deduct!" << '\n';
                cout << a - b << endl;
                break;
            //notice the lack of the optional default
        }
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}

char getcommand(void) //this subroutine prompts the user and gets the right
                     //choice, ignoring other characters other than q, a and s
{
    char command;
    do
    {
        cout << "Enter a to add, s to subtract, q to quit" << '\n';
        cin >> command;
        //while(getchar() != '\n');
    }while ((command != 'q') && (command != 'a') && (command != 's'));
    return command;
}
```

```
Enter a to add the numbers, s to subtract the numbers, q to quit
a
You chose add!
2
Enter a to add the numbers, s to subtract the numbers, q to quit
a
You chose add!
4
Enter a to add the numbers, s to subtract the numbers, q to quit
a
You chose add!
6
Enter a to add the numbers, s to subtract the numbers, q to quit
a
You chose add!
8
Enter a to add the numbers, s to subtract the numbers, q to quit
s
So, you want to deduct!
6
Enter a to add the numbers, s to subtract the numbers, q to quit
q
C:\training\VIDEPROJ\event2>
```

Exercise

1. Start a new console project, call it calculator. Save it in your mydocuments folder. Building on the code you wrote in the previous exercise, and what you learned in this last event loop code, modify the add subtract program so that you can enter both numbers, then choose if you want to add or subtract! It will be your first calculator.
2. Add a multiplication and division subroutine to your program.

C Operators for comparison, evaluation and processing

The following operators are supported. They are listed here in order of increasing precedence:

(=) Assignment Operator. Example $X = 5$

(|=) Replacement Operator. Example $A |= 5$; means $A = A | 5$;

(^=) Replacement Operator. Example $A ^= 5$; means $A = A ^ 5$;

(&=) Replacement Operator. Example $A \&= 5$; means $A = A \& 5$;

(<<=) Replacement Operator. Example $A <<= 5$; means $A = A << 5$;

(>>=) Replacement Operator. Example $A >>= 5$; means $A = A >> 5$;

(+=) Replacement Operator. Example $A += 5$; means $A = A + 5$;

(-=) Replacement Operator. Example $A -= 5$; means $A = A - 5$;

(*) Replacement Operator. Example $A *= 5$; means $A = A * 5$;

(/=) Replacement Operator. Example $A /= 5$; means $A = A / 5$;

(%=) Replacement Operator. Example $A \% = 5$; means $A = A \% 5$;

(?:) Ternary operator. $A ? B : C$ means if A then B else C

(||) Logical OR. Used in conditional statements. Example if $((x == 5) || (x == 7))$

(&&) Logical AND. Used in conditional statements. Example if $((x == 5) \&\& (x == 7))$

(|) Bitwise OR. It looks at the bits in two integers and returns true if either bit is true

(^) Bitwise exclusive OR.

(&) Bitwise AND.

(<<, >>) Bitwise left shift, and right shift.

(== !=) Comparative equality and inequality. Example if $(x == 5)$ or while $(prompt != 'x')$;

(<, >, <=, >=) Comparative less than, greater than, less than or equal greater than or equal.

(+, -) Addition and subtraction.

(*, /) Multiplication, division

(%) Modulus. Returns the remainder of a division operation.

(++, --) Increment and decrement. When appearing before a variable, the operation is performed before the variable is used in an expression; when appearing after it, the variable's value is used before the operation takes place.

(*) Pointer dereferencing. Allows you to access the data stored at that location.

(&) Address operator. Returns the numerical location of a variable in memory. Placed immediately prior to the variable name.

(-) Negative. Returns a negative value.

(!) Logical negation. Returns true if false, false if true.

(~) Bitwise complement operator. Flips the bits of an integer value.

(., ->) Structure member associators.

([]) Array indexing.

Bitwise Operations

C includes a few functions for working with bits of information. The functions provided in C allow you to toggle bit values based on conditions, or shift the bits to the left or right. Functions includes are the binary OR function (|) that will set a bit to true if either bits being compared is true. The binary AND (&) function will set a bit to true if both bits being examined are true. The XOR (^) will set a bit to true if one or the other but not both compared bits are true.

The other bit functions are meant to adjust the value of a single int, no comparison of two values needs to occur. These include the bit compliment operator (~), which merely inverts all the bits of an integer value. The Shift Left (<<) operator will move all the bits left, dropping the left most and feeding a 0 into the right side. The Shift Right (>>) operator does the opposite, dropping off the rightmost bit and feeding a 0 into the left side.

Structures

Structures allow you to group information into single organized elements. It is easier to access related information when it is grouped. Structures are easy to define; you declare them as a variable with elements in the format;

```
struct name
{
type name;
type name;
}structname;
```

The following code demonstrates defining, assigning and accessing structures.

```
#include <cstdlib>
#include <iostream>
using namespace std;
struct mystruct
{
    char name[50];           //members of the struct
    int age;
    char streetaddress[50];
    char city[30];
    char state[20];
    int zip;
    char email[50];
    char phone [16];
}userstruct;

int main(int argc, char *argv[])
{
    cout <<"Please enter your full name. --> ";
    gets(userstruct.name);    //the . operator accesses the member of the structure
    cout <<"Please enter your age. --> ";
    cin >> userstruct.age;
    while (getchar() != '\n');
    cout <<"Please enter your street address. --> ";
    gets(userstruct.streetaddress);
    cout <<"Please enter your city."<<endl;
    gets(userstruct.city);
    cout <<"Please enter your state."<<endl;
    gets(userstruct.state);
    cout <<"Please enter your zip code."<<endl;
    cin >> userstruct.zip;
    while (getchar() != '\n');
    cout <<"Please enter your email."<<endl;
    gets(userstruct.email);
    cout <<"Please enter your phone."<<endl;
    gets(userstruct.phone);
    cout <<"Thank you " << userstruct.name << endl;
    cout <<"Do you enjoy living in " << userstruct.city << "?\n";
    cout << "I think " << userstruct.state << " is a very pretty state. \n";
    cout << "Should I email you at " << userstruct.email <<
    " or mail you at " << userstruct.streetaddress <<"?\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Exercise

1. Create a new console app named structures, saving it in your programming folder. Type in, debug and run the code above.
2. Modify the code so that it gives you a different message depending how old you are and what State you are from...

for example

Arizona is a good state for folks over 60!

Colorado is a good State for young people because it has lots of snow boarding!

Or

You are much too young for a state like Iowa

Files

Many applications depend on the ability to read and write to files. C includes many functions to do this, some I will demonstrate shortly. First I will explain a little about the procedure.

To read or write to a file you need a file variable, declared as `FILE *myfile`. You can see that it is a pointer.

Once you have a variable you can use the `fopen ()` function to open the file. It can be opened with different modes or combinations of modes. This function is called like;

```
myfile = fopen ("myfilename", "mode");
```

The different modes have different uses, and are enclosed in quotes.

“r” reading, the file must already exist.

“w” writing, creates a new file, overwrites the existing file!

“a” open an existing file to append data, creating a new one if it does not exist.

“r+” opens for reading and writing, the file must exist.

“w+” opens empty file for reading and writing, overwrites existing files.

“a+” opens files for reading and appending data, creates the file if it does not exist.

You can use the access mode indicator to indicate binary or text mode access. Text mode assumes the end of file at the first Ctrl Z character, binary simply ends the file at the last byte. You combine the indicator with the mode like “rb” or “a+t”.

Once the file is open, you can read and write, then you must close the file. Study the following example.

Below is a project that both reads and writes to a file on your hard drive. It will create a file called mytestfile in your project folder.

I declared the file variable `f`, with the format `FILE *f`;. I then tested to see if the file already existed, by opening it with the “rb” mode. If it returns a null pointer I open it with the “w+” to create a new file for reading and writing.

```
// A minimal Win32 console app

#include <iostream.h>
void writeinfo(void);
void readinfo(void);
char getcommand(void);

FILE *f;

int main( void )
{
    char command;//declare a variable to hold user command

    if((f = fopen("mytestfile", "rb")) == NULL) //check to see if exists
        f = fopen("mytestfile", "w+");//if not create a new file
    fclose(f);//close it.

    while ((command = getcommand()) != 'q') //gets the command from user
    {
        switch (command) //makes a choice based on the funtion return
        {
            //value of getcommand()
            case 'w':
                writeinfo();
                break;
            case 'r':
                readinfo();
                break;
        }
    }
    return 0;
}
```

```
char getcommand(void) //this routine gets the choice
{
    char command;
    do
    {
        printf("Please enter w to write, r to read.\n");//prompt the user
        scanf("%c",&command);//scan it in
        while(getchar() != '\n');
    } while ((command != 'w') && (command != 'r') && (command != 'q'));
    return command; //send it back to the main loop
}
```

Here I get the user’s choice, then pass it back to main.

```

void readinfo()
{
char c;//make a var to hold the characters of the file
if ((f = fopen("mytestfile", "rb"))!= NULL)//open for reading in binary
{
    while (feof(f)== 0)//if not at the end of the file
    {
        c = fgetc(f); //get one character
        putchar(c);    //write it to screen
    }
fclose (f);//close the file when done
}
    else{printf("Failed to open file.\n");}
}

```

This routine reads info in from a file and writes it to screen. I open the file for read, in the binary access mode. Then I set up a while loop to read characters in with the fgetc () function until I reach the end of the file. I use putchar () to write each character to the screen.

```

void writeinfo()
{
char writebuffer[100];//make a variable to hold a line
char *line;           //and a pointer to it
printf("Please Enter the text that you want to write.\n");
if ((f = fopen("mytestfile", "a")) == NULL)
    printf("Failed to open file.\n");
else
    //the file is open, so proceed
    {
        do
        {
            line = writebuffer; //point to the first char of the buffer
            while((*line = getchar())!= '\n') //fill that character
            {
                line++; //move to the next char
            }
            *line = '\0'; //terminate the string
            fputs(writebuffer, f); // write it to file
            fputs("\n", f);       //write a carriage return
        }while(writebuffer[0] != 0); //if you did not enter a line of text exit
    }
fclose(f); //always close the file when done
}

```

This routine creates the write buffer, creates a pointer to it, and then executes a loop until the user hits return. It then writes that line to the file with fputs (). It will then get the next line from the user. When an empty line is encountered it will stop looking for data.

Now we have persistent data on the hard drive.

Just for fun

Here is a little card game for you!

```
#include <cstdlib>
#include <iostream>
#include <time.h>
using namespace std;
struct mystruct //here is a structure the will hold 5 cards
{
    int value;
    int suit;
}mycard[5]; //the structure is an array of 5!

void dealhand(void); //suroutine to deal a hand
void displayhand(void); //subroutine to display the hand
void taketwo(void); //sub to draw two
void reorganize(void); //or to reorganize your hand

int main(int argc, char *argv[])
{
    char c;
    srand(time(NULL)); //uses timer to fire up (seed)the random number generator
    cout << "Press a d to deal a hand, o to organize, t to take 2 --> ";
    while((c = getchar()) != 'q') // gets your choice;
    {
        switch (c) //redirects based on what you choose.
        {
            case 'd':
                dealhand();
                break;
            case 't':
                taketwo();
                break;
            case 'o':
                reorganize();
            default:
                cout << "Wrong choice, try again! \n";
                ;
        }
        displayhand(); //display after every operation
        cout << "Press a d to deal a hand, o to organize, t to take 2 --> ";
        while(getchar() != '\n'); //clear the buffer
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void dealhand(void)
{
    int i;
    for (i = 1; i <= 5;i++) //this loop fills up each of the 5 card structures
    {
        mycard[i].value = rand() % 13 + 1;
        mycard[i].suit = rand() % 4 + 1;
    }
}

void displayhand(void)
{
    int i;
    char buffer[3];
    for (i = 1; i <= 5;i++)
    {
        cout << "Card " << i << " is a ";
        switch (mycard[i].value)
        {
            case 1:
                printf("Ace of ");
                break;
            case 11:
                printf("Jack of ");
                break;
            case 12:
                printf("Queen of ");
                break;
            case 13:
                printf("King of ");
                break;
            default:
                itoa (mycard[i].value,buffer,10);
                printf("%s of ",buffer);
                ;
        }
    }
}
```

//Continued below.....

```
        switch (mycard[i].suit)
        {
            case 1:
                printf("clubs");
                break;
            case 2:
                printf("spades");
                break;
            case 3:
                printf("hearts");
                break;
            case 4:
                printf("diamonds");
                break;
            default:
                ;
        }
        cout << endl;
    }
}

void taketwo(void)
{
    int cardtodraw;
    cout << "Please enter the number of the first card you wish to replace -> ";
    cin >> cardtodraw;
    mycard[cardtodraw].value = rand() % 13 + 1;
    mycard[cardtodraw].suit = rand() % 4 + 1;

    cout << "Please enter the number of the second card to replace -> ";
    cin >> cardtodraw;
    mycard[cardtodraw].value = rand() % 13 + 1;
    mycard[cardtodraw].suit = rand() % 4 + 1;
}
```

//Continued below....

```
void reorganize(void)
{
    int cardtomove,tempvalue,tempsuit;
    cout << "Please enter the number of the first card to move down -> ";
    cin >> cardtomove;
    if (cardtomove < 5)
    {
        tempvalue = mycard[cardtomove+1].value;
        tempsuit = mycard[cardtomove+1].suit;
        mycard[cardtomove+1].value = mycard[cardtomove].suit;
        mycard[cardtomove+1].suit = mycard[cardtomove].suit;
        mycard[cardtomove].value = tempvalue;
        mycard[cardtomove].suit = tempsuit;
    }
    else
    {
        tempvalue = mycard[1].value;
        tempsuit = mycard[1].suit;
        mycard[1].value = mycard[cardtomove].value;
        mycard[1].suit = mycard[cardtomove].suit;
        mycard[cardtomove].value = tempvalue;
        mycard[cardtomove].suit = tempsuit;
    }
}
```

Have fun!!

Windows Programming

Up until now, you have been programming for the console. You have learned many aspects of the C++ language, and have developed some problem solving skills. Modern applications take advantage of the Graphical User Interface of the operating system. From this point on I will be teaching you how to program for, and interact with the elements of the Windows GUI. Topics covered will include:

- Creating a simple window
- Adding a menu
- Detecting mouse movements
- Detecting key presses
- Adding variables and specific functions
- Drawing to the window

You will learn many of the basics of Windows Programming with the next application, so pay attention, type the code in carefully, including the comments. Take the time to learn as you go, because short-cuts only make you short on skills when it comes to programming computers.

This application will be a very unique painting application. It uses random shapes as a paintbrush to create art. You can control the size and color of the brush to create abstract art. Let's get started!

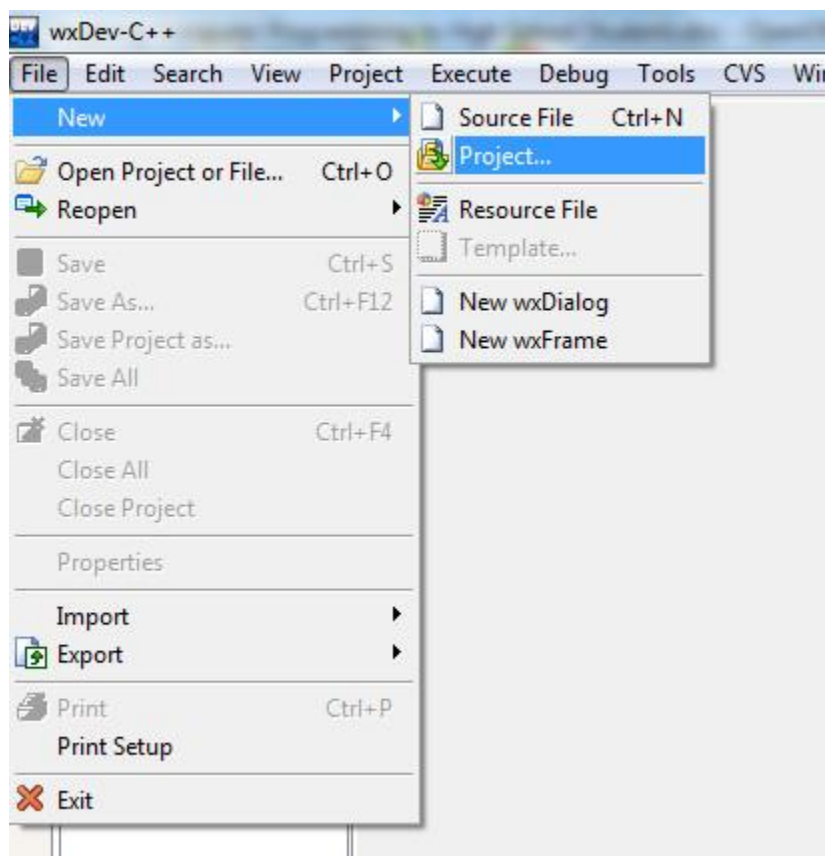
Creating a simple Window

When I refer to a window, it means the frame that most programs present themselves in. In simple form, they will consist of a border that may or may not be sizeable, and a blank, usually white area inside the border.

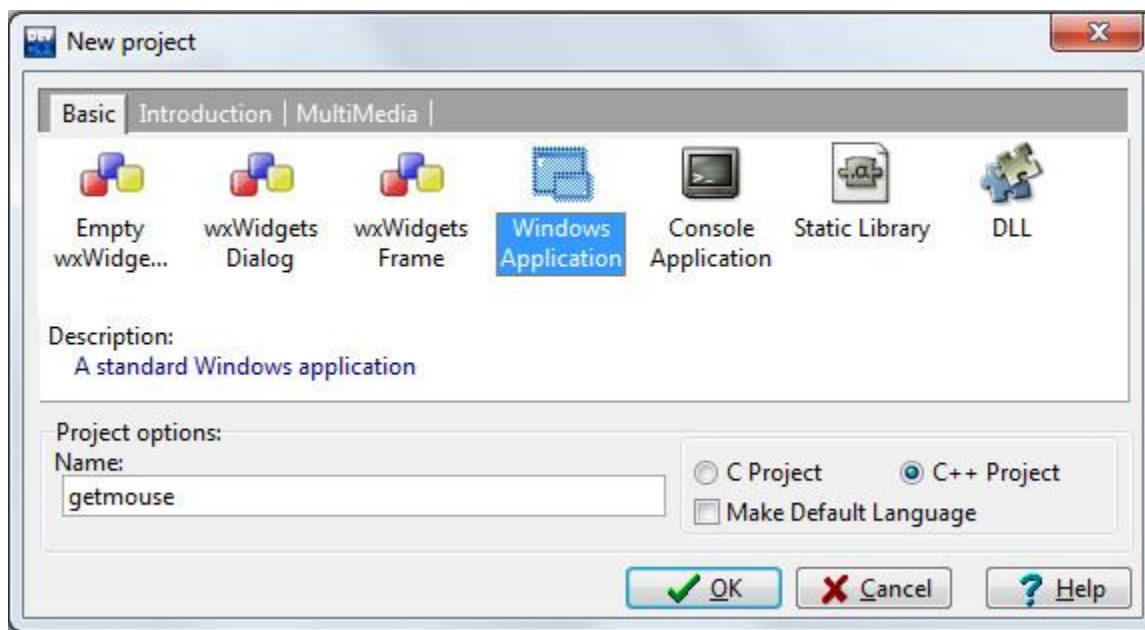
To create a Window, you first define a Windows class (variable), define some basic variables that control the options of the Window and then create the window and finally show the Window to the user. This is called constructing the Window.

After the Window is created, you can add functionality to the window with a procedure routine. This is called the Windows Procedure routine, and it can make your Window smart.

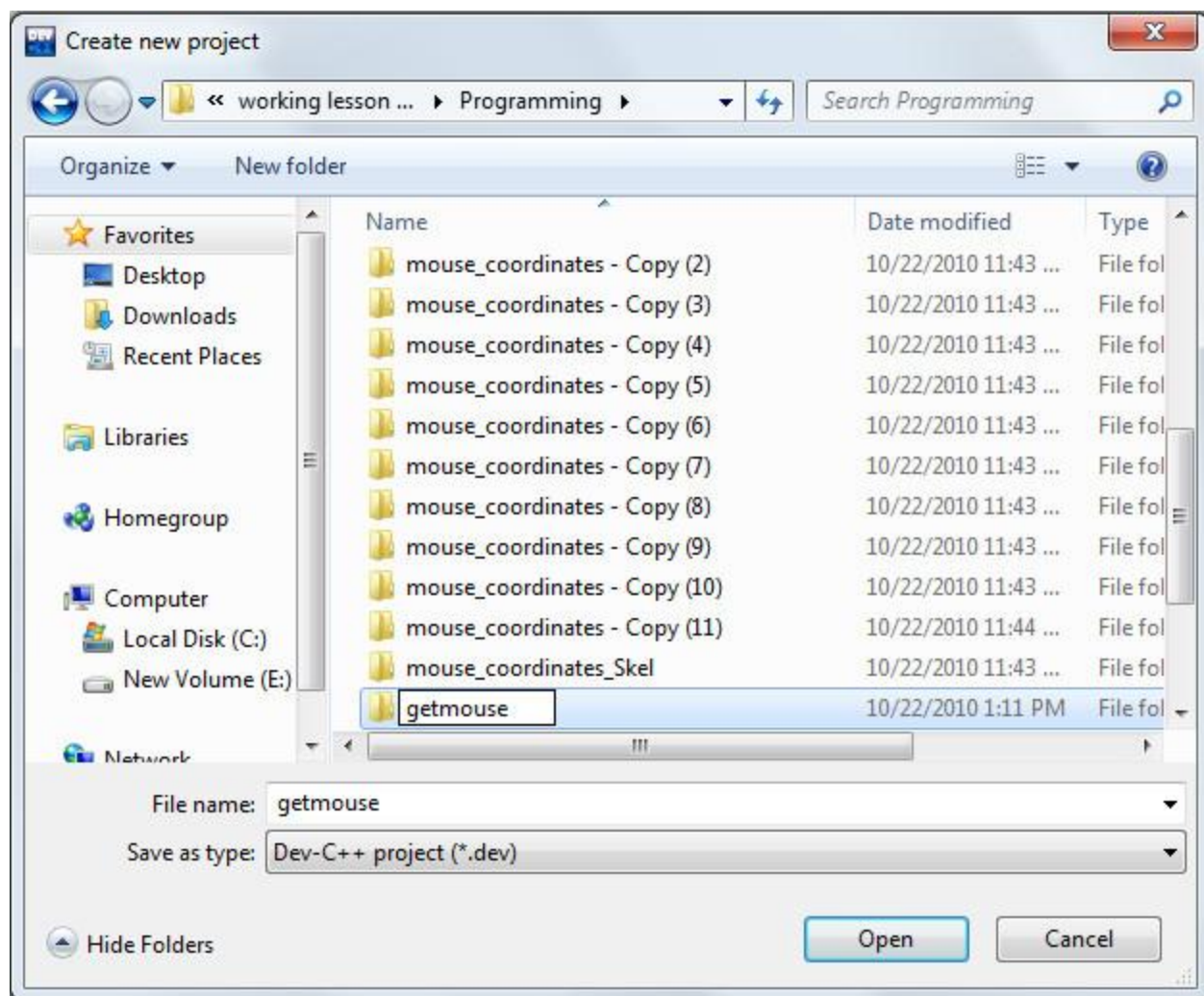
Open up DevC++ and select New Project from the File Menu.



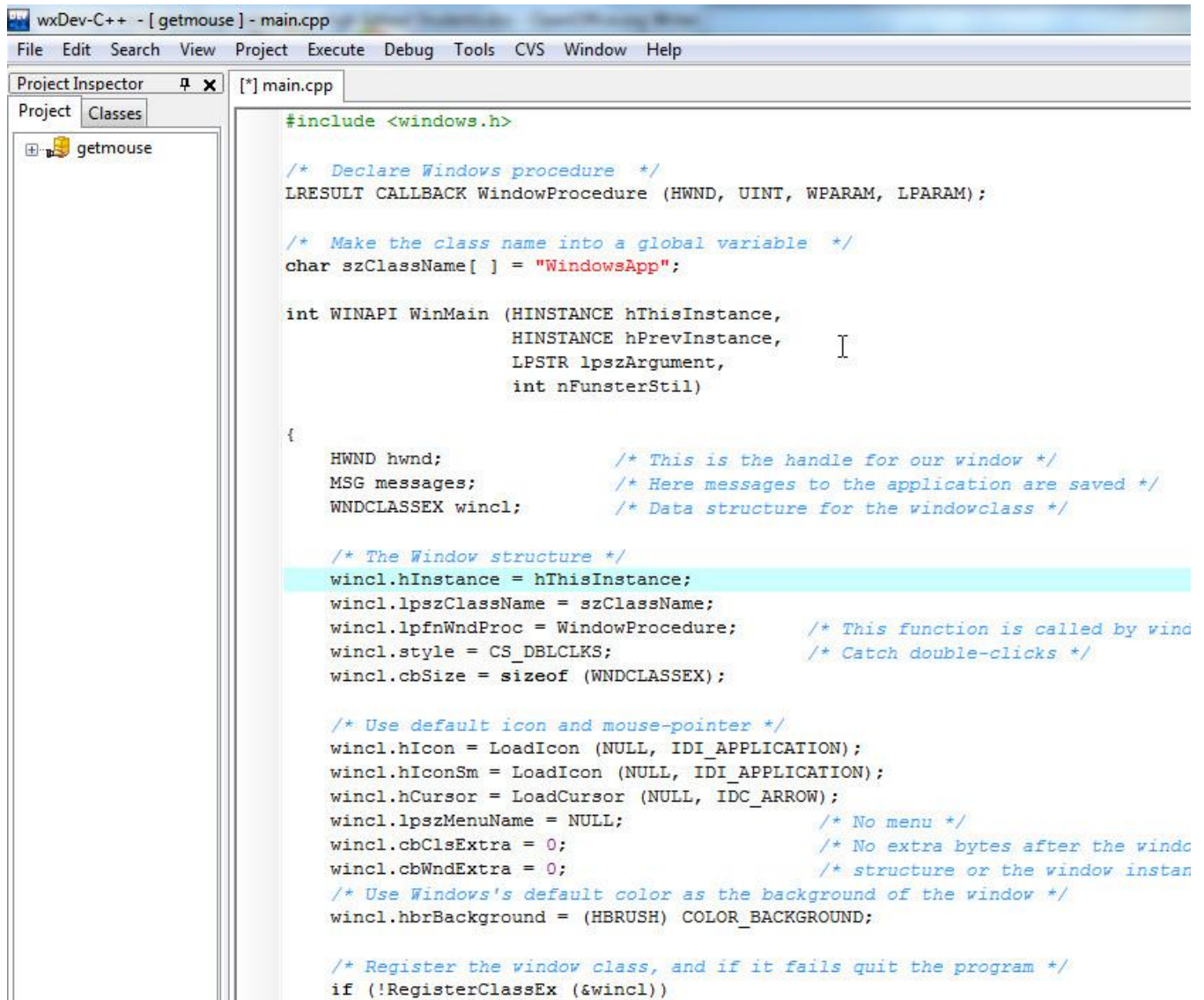
Select the Windows Application type, and name your project getmouse.



When you press OK, it will ask you where to store your project. Remember that every project requires a new folder, so create a new folder called getmouse, and save the new project inside that folder.



DevC++ will load the default Windows code, that would create a very simple Window. It will look like this, with the source code file called main.cpp



```
#include <windows.h>

/* Declare Windows procedure */
LRESULT CALLBACK WindowProcedure (HWND, UINT, WPARAM, LPARAM);

/* Make the class name into a global variable */
char szClassName[ ] = "WindowsApp";

int WINAPI WinMain (HINSTANCE hThisInstance,
                   HINSTANCE hPrevInstance,
                   LPSTR lpszArgument,
                   int nFunsterStil)

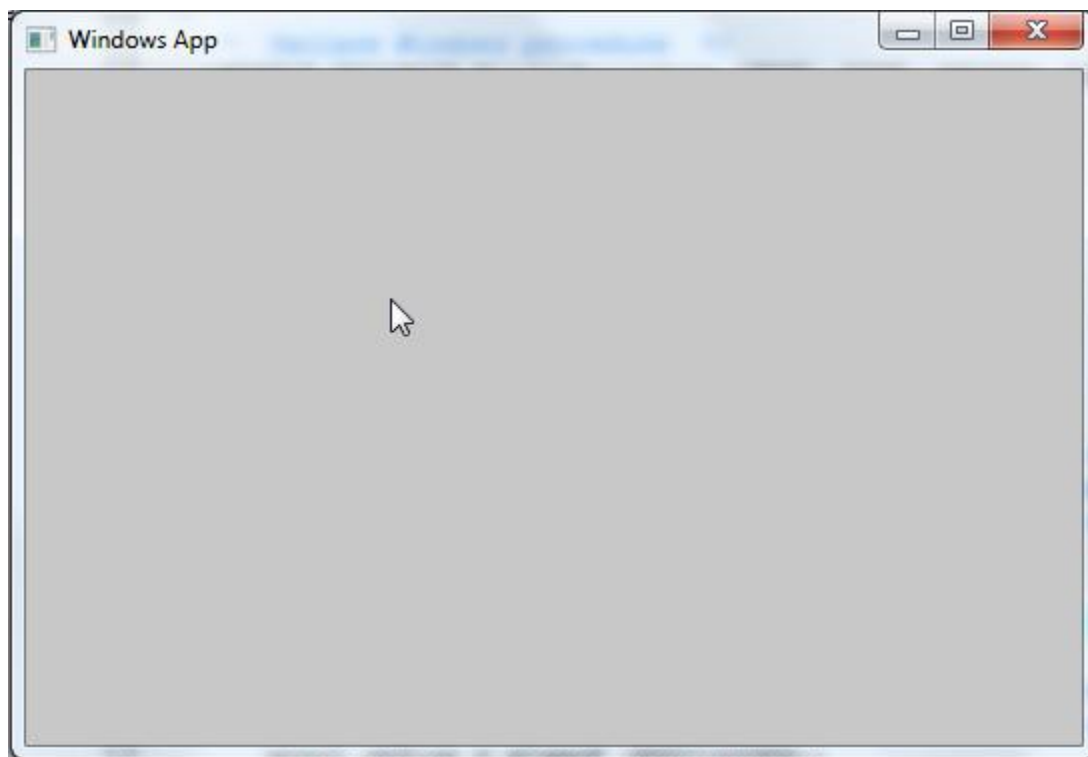
{
    HWND hwnd;           /* This is the handle for our window */
    MSG messages;         /* Here messages to the application are saved */
    WNDCLASSEX wincl;     /* Data structure for the windowclass */

    /* The Window structure */
    wincl.hInstance = hThisInstance;
    wincl.lpszClassName = szClassName;
    wincl.lpfnWndProc = WindowProcedure; /* This function is called by wind
    wincl.style = CS_DBLCLKS;             /* Catch double-clicks */
    wincl.cbSize = sizeof (WNDCLASSEX);

    /* Use default icon and mouse-pointer */
    wincl.hIcon = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hIconSm = LoadIcon (NULL, IDI_APPLICATION);
    wincl.hCursor = LoadCursor (NULL, IDC_ARROW);
    wincl.lpszMenuName = NULL;           /* No menu */
    wincl.cbClsExtra = 0;                /* No extra bytes after the windc
    wincl.cbWndExtra = 0;                /* structure or the window instan
    /* Use Windows's default color as the background of the window */
    wincl.hbrBackground = (HBRUSH) COLOR_BACKGROUND;

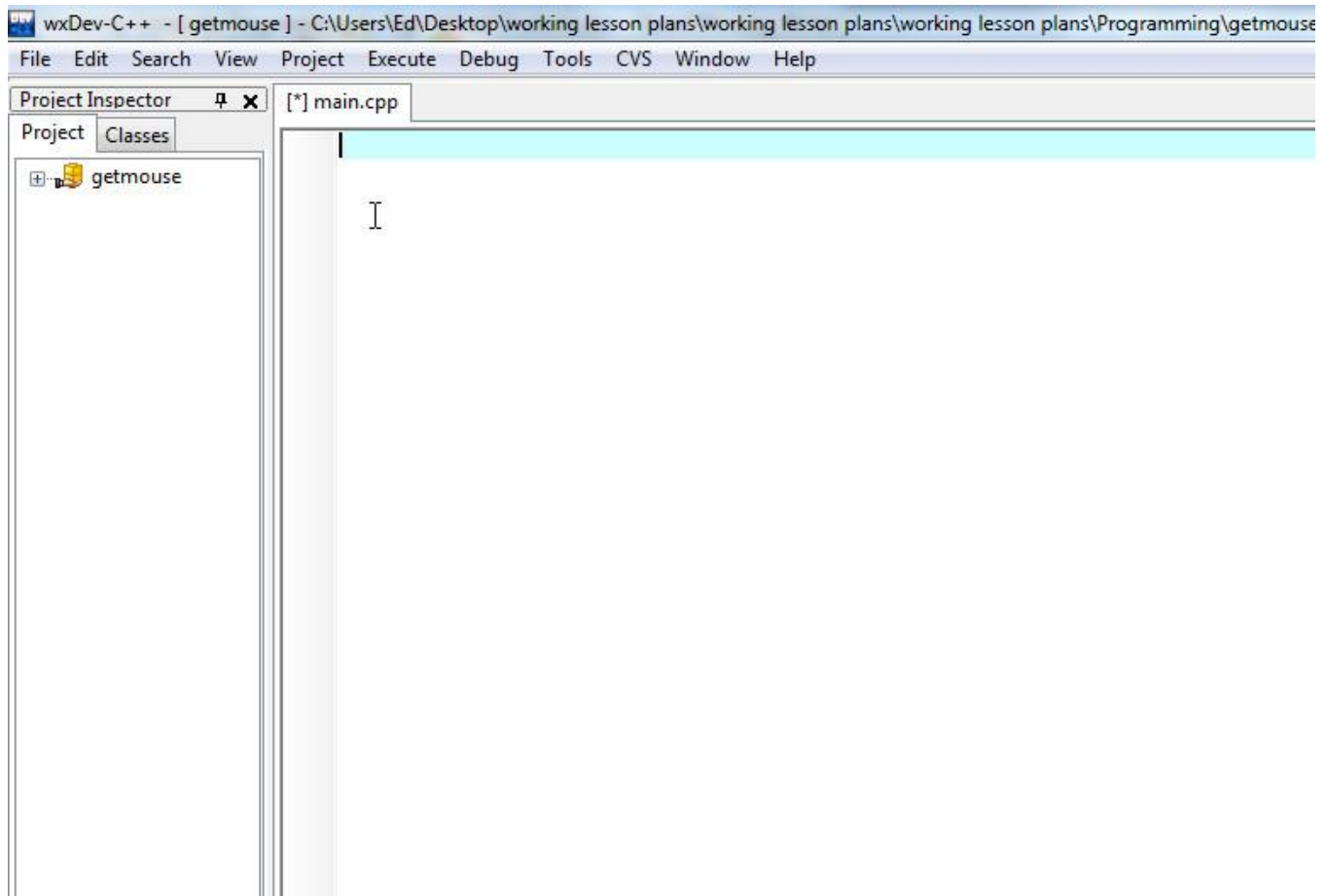
    /* Register the window class, and if it fails quit the program */
    if (!RegisterClassEx (&wincl))
```


If you press F9 now, it will ask you to save the source code as `main.cpp`, agree with that, and you should see the blank Window come up. You are a Windows Programmer!!



This little window does not allow you to do much, other than drag it around, re-size it, minimize it, maximize it and close it. We are going to change all of that though, so get ready.

The next step is to select all of the code in the main.cpp file and delete it so that it is blank. We will put new code in there, so do not worry. Your project will look like this. Now we will start adding our own window code, so that we can control what we want to .



The Windows Procedure

Type the following code into the main.cpp source code editor. Be very careful, and type in all of the comments, as they will help you know where to add code as we go along. Here we will create the Windows Class, and start typing in the procedure routine that will allow our window to have functionality. The windows procedure receives messages from the operating system and translates them into useable commands.

```
#include <windows.h>
//add declarations here

//For future reference, you need to know the difference between
//an l (Lower Case L) and a 1 (one)
//the difference is very subtle, look at the top, the one is slanted
//this is important when typing in this code

const char g_szClassName[] = "myWindowClass"; // we name the Windows Class

//we will define a function we can use to draw

//the Window Procedure subroutine

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    int i; //declare local variables
    HDC hdc; //device context (window canvas to "paint on")
    RECT clientrect; //a rectangle to hold window boundaries
    switch(msg) //this giant switch statement gets messages sent to the window by
    //the operating system, such as mouse clicks and movements, keypresses, timer
    //events, and other commands that you may create with menu items.
    {
        // place code for button down here

        // end button down code
        //place code for button up here

        // end button up code
        //place mouse movement code here

        //end mouse movement code here
        //begin keypress code here

        //end keypress code here
        case WM_CREATE:
            //place code for initialization here

            //end initialization
            break;
```

```
case WM_CLOSE:
DestroyWindow(hwnd);
break;
case WM_DESTROY:
PostQuitMessage(0);
break;
//place timer message code right here

//end timer message code
// begin command message code

//end command message code
default:
return DefWindowProc(hwnd, msg, wParam, lParam);
} //end of the giant message switch
return 0;
}
```

//Do not try to compile and run quite yet, there is still some more work to do!

The WinMain Routine

//OK, now we have finished the Windows Procedure routine, but your code is still not ready to run. Let's //get going on the constructor routine. This is the WinMain routine. It serves the same purpose as the main routine did in your console applications. Continue typing the following code.

```
//Here is the WinMain routine. This is the main routine of your Windows
//application, it serves the same purpose as the main routine of your console
//apps, but is a little more complicated.

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
    WNDCLASSEX wc; //wc is the Windows Class
    HWND hwnd;     //hwnd is a "handle" to the Window
    MSG Msg;       //Msg is the message that contains the commands

    //Here we define variables that control
    //certain aspects of the new windows class, such as style, icons, cursors
    //menus etc.
    wc.cbSize      = sizeof(WNDCLASSEX);
    wc.style       = 0;
    wc.lpfnWndProc  = WndProc;
    wc.cbClsExtra   = 0;
    wc.cbWndExtra   = 0;
    wc.hInstance   = hInstance;
    wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = NULL;
    wc.lpszMenuName = NULL;
    wc.lpszClassName = g_szClassName; //this is the class name we set at the top
    wc.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
    //add menu to the window here
    wc.lpszMenuName = NULL;

    // If we defined the variables of the new class correctly, we will register
    //the window with the operating system here. If this returns an error when
    //you try to compile, you likely need to look very closely at the code
    //above and correct any typing errors or missing code.

    if(!RegisterClassEx(&wc))
    {
        MessageBox(NULL, "Window Registration Failed!", "Error!",
            MB_ICONEXCLAMATION | MB_OK);
        return 0;
    }
}
```

```
//Creating the Window. If we were able to register the new windows class,
//then now we can use the CreateWindowEx function to create the Window in
//memory. We are setting certain styles here that we did not set in the
//variable definitions above, such as the size of the window, style and the
//title on the title bar.
hwnd = CreateWindowEx(
    WS_EX_CLIENTEDGE,
    g_szClassName,
    "Instant Art 9000",
    WS_OVERLAPPEDWINDOW,
    CW_USEDEFAULT, CW_USEDEFAULT, 640, 480,
    NULL, NULL, hInstance, NULL);

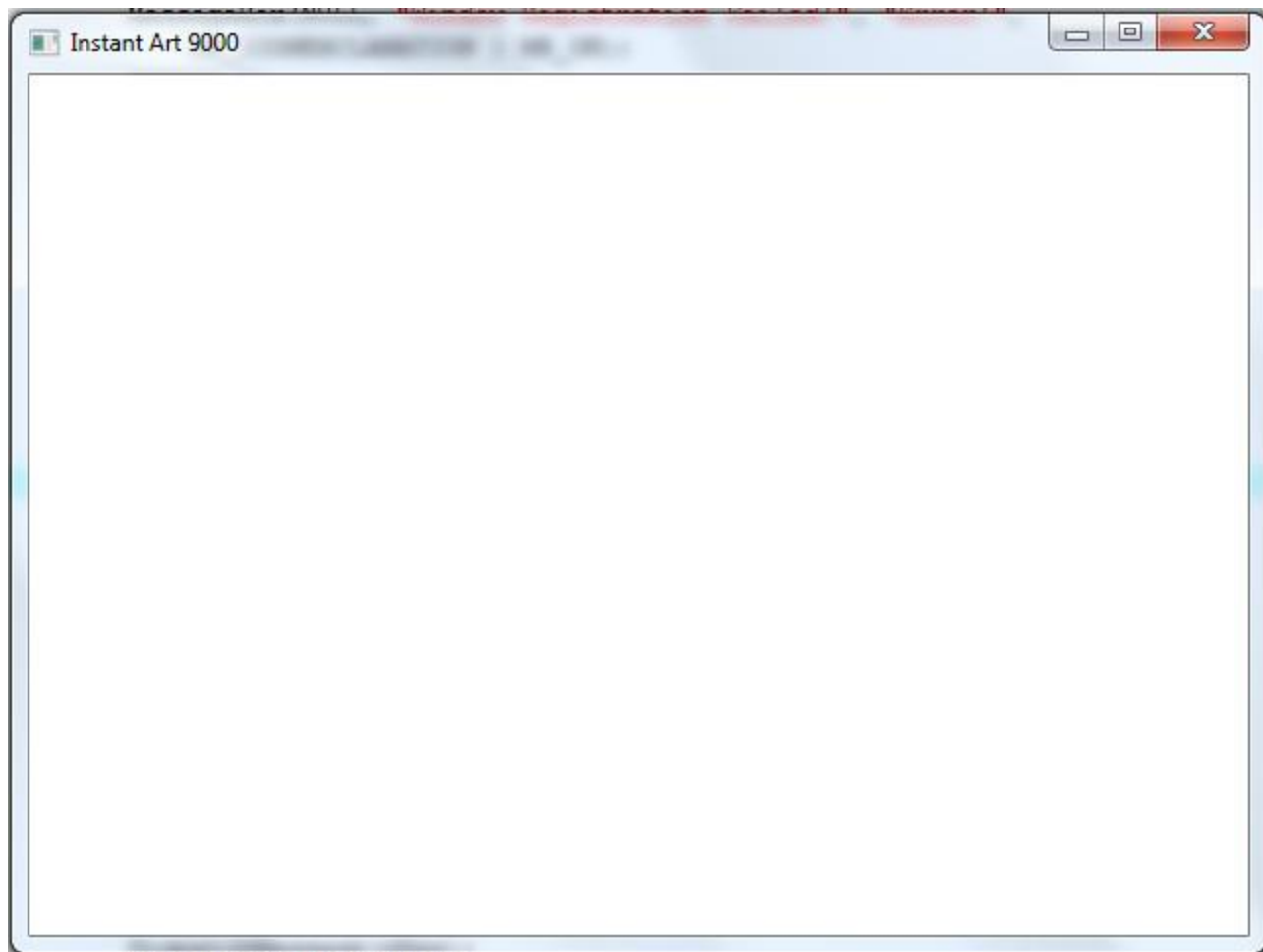
if(hwnd == NULL)
{
    MessageBox(NULL, "Window Creation Failed!", "Error!",
        MB_ICONEXCLAMATION | MB_OK);
    return 0;
}
//if all is well above, we can show the window on your desktop right here!
ShowWindow(hwnd, nCmdShow);
UpdateWindow(hwnd);

//The Message Loop. This is similar to the event loop that you had in your
//console application, the DispatchMessage routine sends the message to
//the windows procedure for processing.
while(GetMessage(&Msg, NULL, 0, 0) > 0)
{
    TranslateMessage(&Msg);
    DispatchMessage(&Msg);
}
return Msg.wParam;
}

//begin drawing function code

//end drawing function code
```

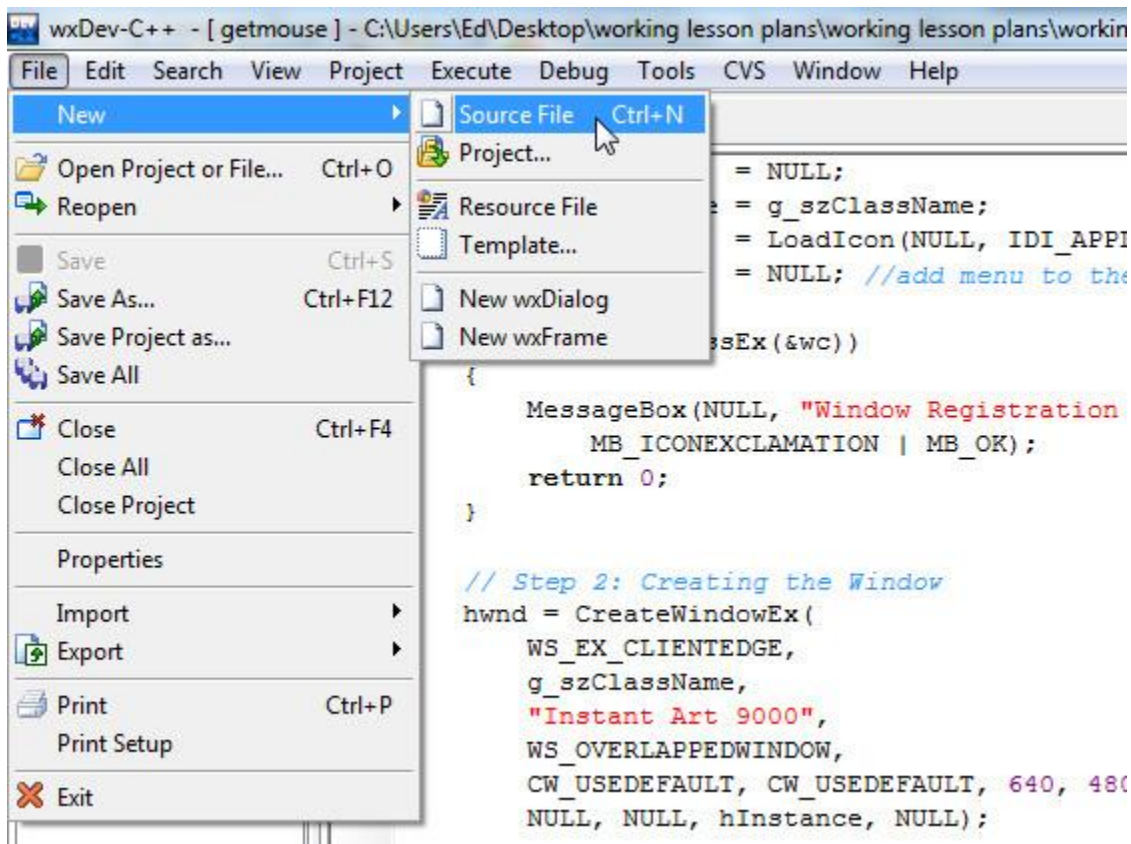
Your code is all typed in now, you are ready to press F9, and start debugging your code. It may take you a bit of time to get all the typo's squared away so that your program runs. Make sure your code is identical in every way. Your new window when it runs will be a little fancier, but not much. You can see that it has a title now, and it is not gray.



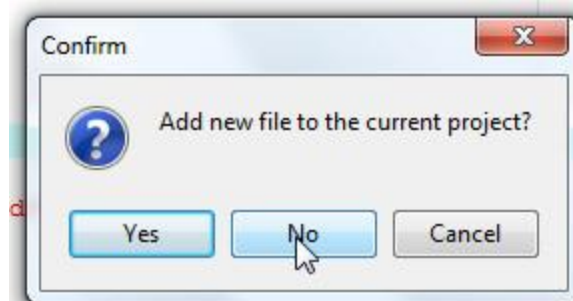
Adding a Menu

Now we will get ready to add some functionality. We are going to add a menu next to your application. The menu will not do much for now, but you will be able to access it, and select item on it. Later we will train the menu items to do something. First I want you to create two new files. The first file will be `getmouseheader.h`. This will be a file that we define our variables in.

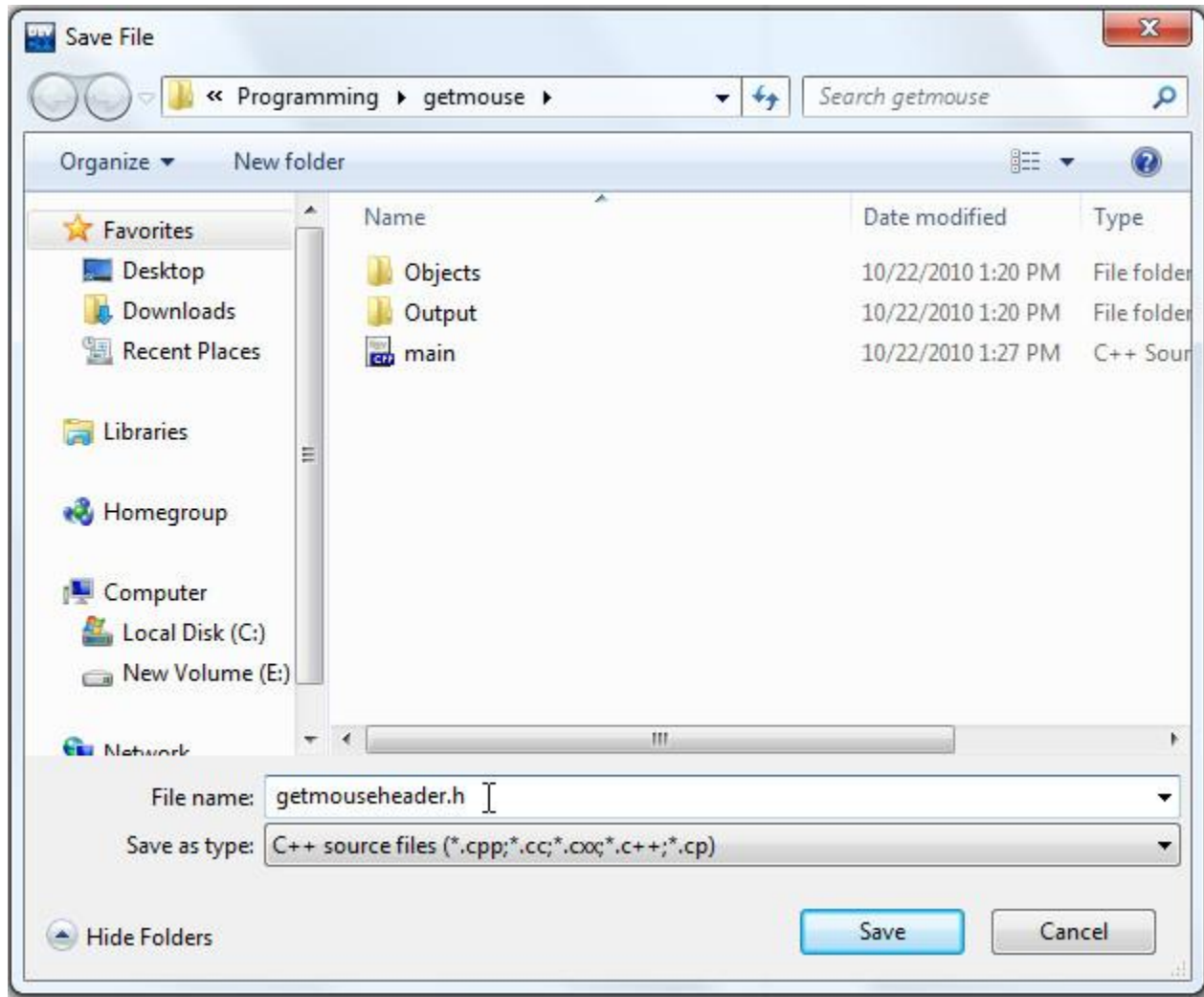
From the file menu, choose `new|source file`.



A dialog will come up, asking you if you would like to add it to your current project. Choose NO, as .h files do not need to be added to a project, we just reference them from within our source code as you will see later.

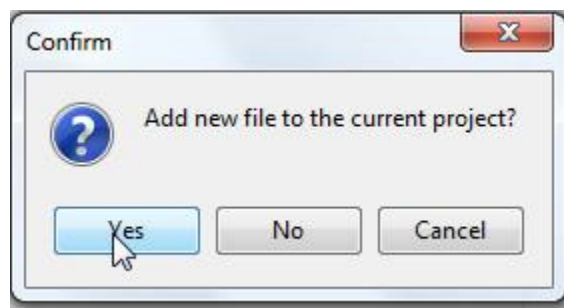


Now, from the file menu, choose save as and save it as `getmouseheader.h` in your folder that you created for your project.

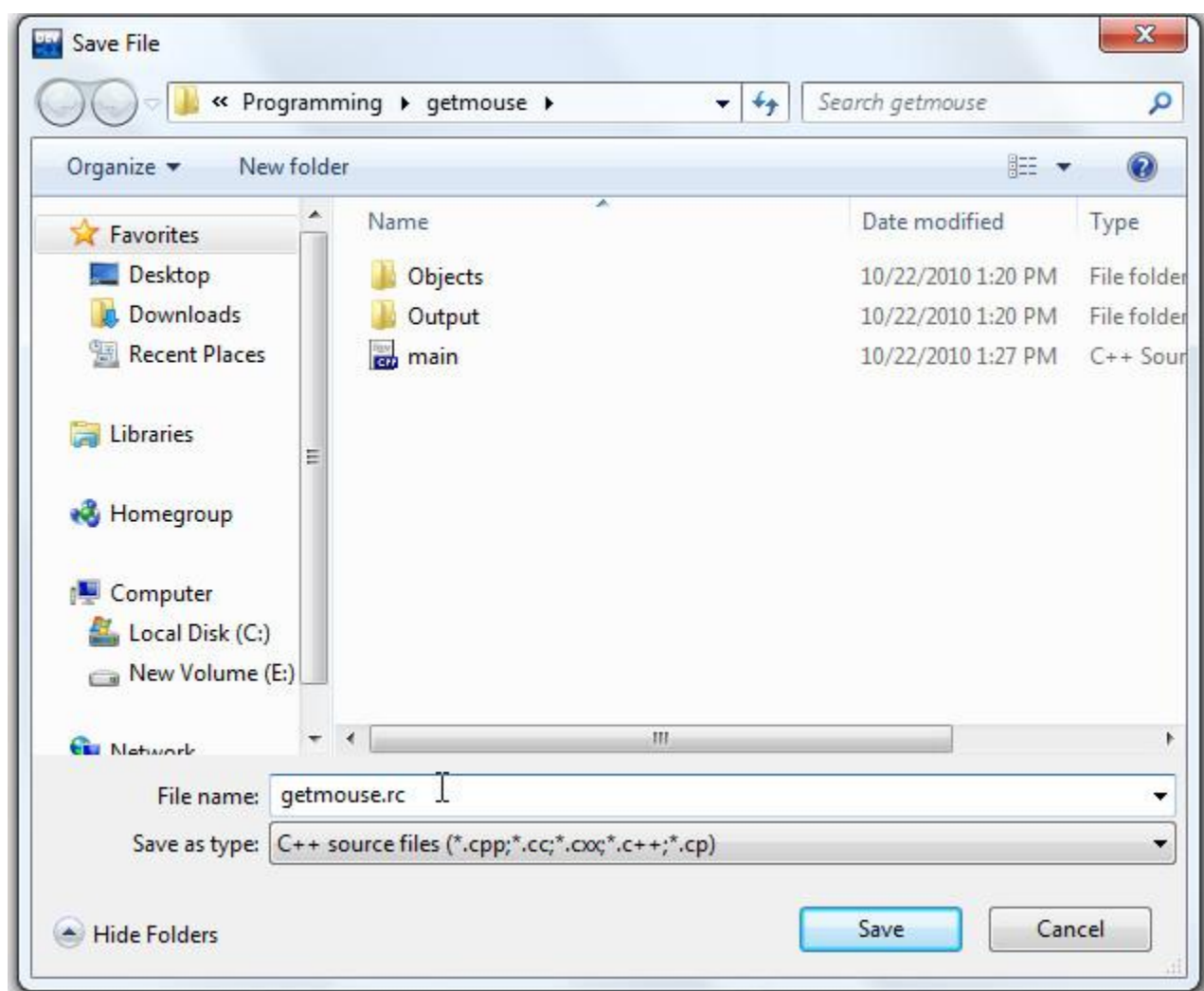


Now we will need to create another source code file. This will be a resource file. This file can hold references to things we want to place in our program, like menus, graphics, etc.

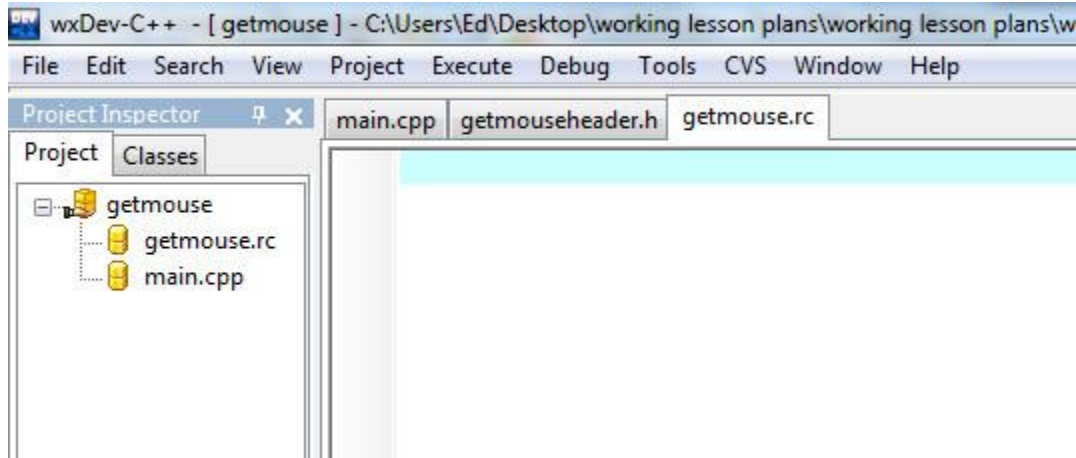
From the file menu, choose new|Source File, this time, choose YES when asked if you want to add it to the current project.



Now, from the file menu, choose save as, and name the file getmouse.rc



Now that you have added both of those files, your project IDE will look like this.



The getmouse.rc file is open in the editor, which is good, as we are going to add the code to create the menu to this file.

Type the following code into the getmouse.rc file.

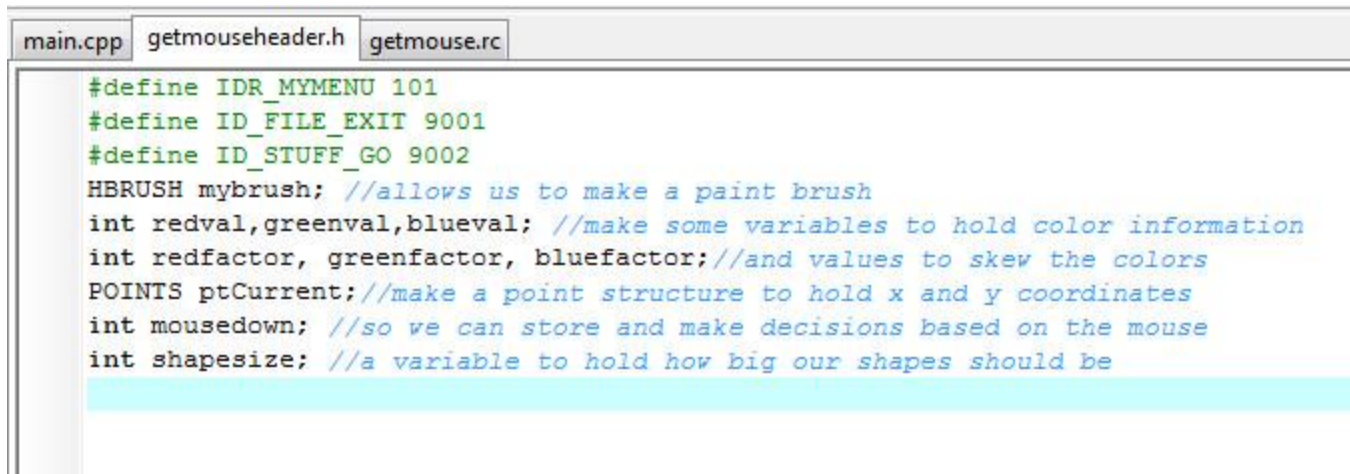
```
#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END

    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Do Stuff!", ID_STUFF_GO
    END
END
```

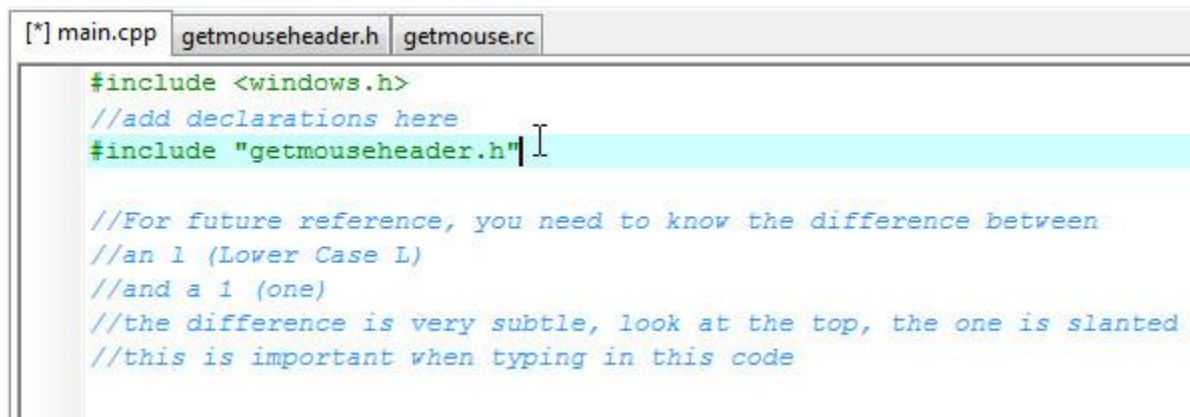
In order to access the variables that we are using in the .rc file, we need to define them in the getmouseheader.h file. Type the following code into the getmouseheader.h file. We are going to add some other variable while we are at it for color, size and position information. Make sure you do a save to save the work that you have done so far!

Look at how we are defining the menu items, IDR_MYMENU, ID_FILE_EXIT and ID_STUFF_GO. This will allow us to connect to them from within our windows procedure to make them do things! The number after the definition must be unique, never duplicate the number on other resources, but you can number them how you want.



```
#define IDR_MYMENU 101
#define ID_FILE_EXIT 9001
#define ID_STUFF_GO 9002
HBRUSH mybrush; //allows us to make a paint brush
int redval,greenval,blueval; //make some variables to hold color information
int redfactor, greenfactor, bluefactor; //and values to skew the colors
POINTS ptCurrent; //make a point structure to hold x and y coordinates
int mousedown; //so we can store and make decisions based on the mouse
int shapsize; //a variable to hold how big our shapes should be
```

Now we need to connect our main.cpp file to our getmouseheader.h file. Select the main.cpp file in the editor, and at the top of the code, just after the line `// add declarations here`, add the line shown below.



```
#include <windows.h>
//add declarations here
#include "getmouseheader.h"

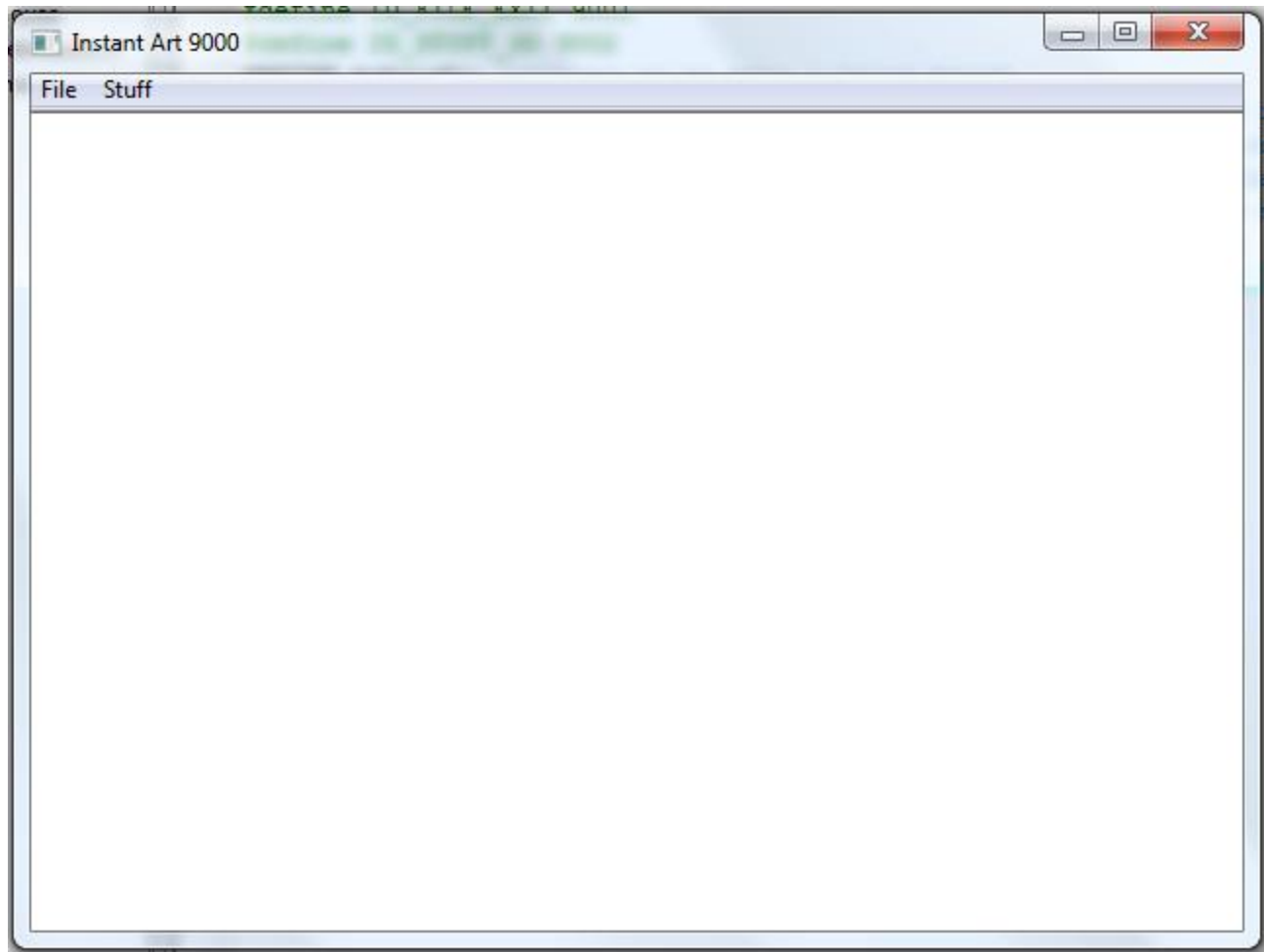
//For future reference, you need to know the difference between
//an l (Lower Case L)
//and a 1 (one)
//the difference is very subtle, look at the top, the one is slanted
//this is important when typing in this code
```

This will point your main.cpp source code file to all of the variables in the getmouseheader.h file so that you can use them.

It is time to press F9 to compile your program. When you are done debugging, you will see the same window, with no changes. This is normal, we have not told the window in the constructor routine to use this new menu, but we will now. In the WinMain routine, where it says registering the Window Class, you will see a list of variables. At the bottom, where it says *//add menu to the window here*, modify that line to look like the code below. Here we USE the menu we created in the getmouse.rc file with the MAKEINTRESOURCE function.

```
//Here we define variables that control  
//certain aspects of the new windows class, such as style, icons, cursors  
//menus etc.  
wc.cbSize      = sizeof(WNDCLASSEX);  
wc.style       = 0;  
wc.lpfnWndProc = WndProc;  
wc.cbClsExtra  = 0;  
wc.cbWndExtra  = 0;  
wc.hInstance   = hInstance;  
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);  
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);  
wc.hbrBackground = NULL;  
wc.lpszMenuName = NULL;  
wc.lpszClassName = g_szClassName; //this is the class name we set at the top  
wc.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);  
//add menu to the window here  
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MYMENU);
```

Run your program now with F9, and you will see your menu if you typed everything correctly.



Controlling Window Background Color

Right now we have not really assigned a specific background color to our window. We can do this when we set the windows registration variables easily.

Find the code that you just modified to add the menu, it looks like this.

```
//Here we define variables that control
//certain aspects of the new windows class, such as style, icons, cursors
//menus etc.
wc.cbSize      = sizeof(WNDCLASSEX);
wc.style       = 0;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra  = 0;
wc.cbWndExtra  = 0;
wc.hInstance   = hInstance;
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = NULL;
wc.lpszMenuName = NULL;
wc.lpszClassName = g_szClassName; //this is the class name we set at the top
wc.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
//add menu to the window here
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MYMENU);
```

Add a line at the very top of the variables. This new line allows us to create a paintbrush using values from red, green, and blue. Look at the modified code below, the added line is highlighted for you. The first value is red, then green, then blue, and 255 is the maximum amount for the values to create white.

```
//Here we define variables that control
//certain aspects of the new windows class, such as style, icons, cursors
//menus etc.
mybrush = CreateSolidBrush(RGB(255,255,255));
wc.cbSize      = sizeof(WNDCLASSEX);
wc.style       = 0;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra  = 0;
wc.cbWndExtra  = 0;
wc.hInstance   = hInstance;
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = NULL;
wc.lpszMenuName = NULL;
wc.lpszClassName = g_szClassName; //this is the class name we set at the top
wc.hIconSm     = LoadIcon(NULL, IDI_APPLICATION);
//add menu to the window here
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MYMENU);
```

Now we will set the windows background variable to the value of the brush. Look at the modified line highlighted in the code below. Change your code to replace the NULL with mybrush.

```
//Here we define variables that control
//certain aspects of the new windows class, such as style, icons, cursors
//menus etc.
mybrush =CreateSolidBrush( RGB(255,255,255) );
wc.cbSize      = sizeof(WNDCLASSEX);
wc.style       = 0;
wc.lpfnWndProc = WndProc;
wc.cbClsExtra  = 0;
wc.cbWndExtra  = 0;
wc.hInstance   = hInstance;
wc.hIcon       = LoadIcon(NULL, IDI_APPLICATION);
wc.hCursor     = LoadCursor(NULL, IDC_ARROW);
wc.hbrBackground = mybrush;
wc.lpszMenuName = NULL;
wc.lpszClassName = g_szClassName; //this is the class name we set at the top
wc.hIconSm      = LoadIcon(NULL, IDI_APPLICATION);
//add menu to the window here
wc.lpszMenuName = MAKEINTRESOURCE(IDR_MYMENU);
```

You can press F9 to compile and run, and now your window will always have a nice white background. You can change the values in the CreateSolidBrush() function if you like to adjust the color of the window, but the values must always be between 0 and 255.

Try 255,0, 255 and recompile with F9! Pretty slick huh? What we did was remove all the green from the white to produce a pink. Add a little green back with 255,127,255 and see what happens. You can do a web search for RGB color charts if you want, or just mix your own colors!

Handling Messages and Commands

In the last lesson, we added a menu to our window. The menu gave some options that we could click on, but we have not “trained” them yet. For a menu command to be useable in a program, we have to create the menu item and give it an ID in the .rc file, give that ID a value in the .h file, and finally in the .cpp file, we look for that ID in the Windows Procedure subroutine. Let's look at the .rc file again.

```
#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END
    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Do Stuff!", ID_STUFF_GO
    END
END
```

You can see that we create a MENU called IDR_MYMENU, and then we BEGIN to define how the menu looks. We start with a POPUP menu (a menu that opens up and shows you other items to click on) and we give that popup menu the label &File. The & symbol causes the F to be a shortcut key in your application.

We give the File popup menu an item labeled E&xit (the x is the shortcut key) and we give that item the identifier ID_FILE_EXIT. Look in the .h file now and look at what we have done for the ID_FILE_EXIT identifier.

main.cpp	getmouseheader.h	getmouse.rc
----------	------------------	-------------

```
#define IDR_MYMENU 101
#define ID_FILE_EXIT 9001
#define ID_STUFF_GO 9002
HBRUSH mybrush; //allows us to make a paint brush
int redval,greenval,blueval; //make some variables to hold color information
int redfactor, greenfactor, bluefactor; //and values to skew the colors
POINTS ptCurrent; //make a point structure to hold x and y coordinates
int mousedown; //so we can store and make decisions based on the mouse
int shapessize; //a variable to hold how big our shapes should be
```

We defined it so that our program can use it, and we gave it a value of 9001. We can give it any number value we choose, but it is important that we do not give more than one identifier the same value. Now we can add a message handler within the Windows Procedure subroutine to handle the message that is sent to the window when someone clicks Exit from the File Menu.

In the WndProc subroutine, look for the comments that tell you where the command message code should go as shown below. You will put new code between these two comments that I have highlighted.

```
//end timer message code
// begin command message code

//end command message code
default:
return DefWindowProc(hwnd, msg, wParam, lParam);
```

They indicate where you will place additional message handling code. If you have neglected to type in the comments, go back to pages 67 and 68 where I provided the WinMain code and type the comments into the code in the proper place, because you will need them as I direct you to add additional lines of code.

Between the two comments, type the following code.

```
// begin command message code
case WM_COMMAND:
    switch(LOWORD(wParam)) //gets the parameter of the WM_COMMAND
    {
        case ID_FILE_EXIT: //if it the exit menu
            PostMessage(hwnd, WM_CLOSE, 0, 0);
            break;
        case ID_STUFF_GO: //if it is the do stuff menu item
            MessageBox(hwnd, "clearing your masterpiece", "My Message Box", 0);
            GetClientRect(hwnd, &clientrect); //gets window rectangle
            InvalidateRect(hwnd, &clientrect, TRUE); //clears it blank
            break;
    }
    break;
//end command message code
```

Press F9 to compile and run your code. Debug and correct any errors that you may find. Now when you click on the menu items, they do something. The File Exit item will close your program, and the Stuff Do Stuff menu item will present to you a message box, and in the future provide you with a way to erase anything that you may put on the window.

The Windows Create message.

There are utility messages that your window can respond to that give you additional functionality. One of these is the WM_CREATE message. Anything that you want to do the instant that the window is created you can do with this message. There may be variables that you need to set, or other functions that you need to run so that you are prepared to work with the window. In the WndProc subroutine, look for the comments that I have highlighted below.

```
case WM_CREATE:
//place code for initialization here

//end initialization
break;
```

We will place some utility functions in this message handler, between the comments.

Place the following code in between the initialization comments in the WM_CREATE message handler.

```
//place code for initialization here
srand(time(NULL)); // 'seed' the random number generator with a value from timer
SetTimer(hwnd, 1, 100, NULL); // create a timer to go off every 1/10 of a sec
shapsize = 50; // set an initial size for the shapes
break;
//end initialization
```

We are going to use our program to generate some random values later, so we need call the strand() function (see the code above) to give the windows random number generator a place to start. Here we use a value from the system time. Since "time" never happens twice it is a good value to use to kick start the random number generator.

Also, we are going to use a timer to perform actions according to a very regular interval. If you look at the SetTimer function above, you will a value of 100 on the parameters. This means that the timer for this window will "click" every 100 milliseconds, or 1/10th of a second.

Finally, we are giving the shapsize variable a value of 50. We will be creating shapes later, and initially, the size will be based on a value of 50.

We are not ready to compile and run yet, as there is a little more housekeeping to do. Because we used functions related to "time", we need to include the Windows time header file at the top of our program. Look at the very top of your main.cpp file for the following comment;

```
#include <windows.h>
//add declarations here
#include "getmouseheader.h"
```

Right where it says add declarations here, modify your code so that it looks like this.

```
#include <windows.h>
//add declarations here
#include <time.h> //include windows time functions here
#include "getmouseheader.h"
```

We added one line of code, highlighted above.

Now would be a good time to press F9 to compile and run your program. You will not see anything different, but if it runs you know that you have typed in your code correctly, and the proper things are being initialized for you to use later.

Mouse Commands

Now for the fun part! Windows Programming would just not be complete without the mouse. Next we are going to learn how to detect mouse button clicks, and use those clicks to determine when you are dragging the mouse.

We will use the Windows Messages to determine when the mouse is moving. A mouse move message is generated when the cursor moves across your window. I have put comments in your WndProc subroutine to show you where to place the message handler. Look for the comments that I have highlighted below.

```
//place mouse movement code here

//end mouse movement code here
```

Modify your code to look like the following.

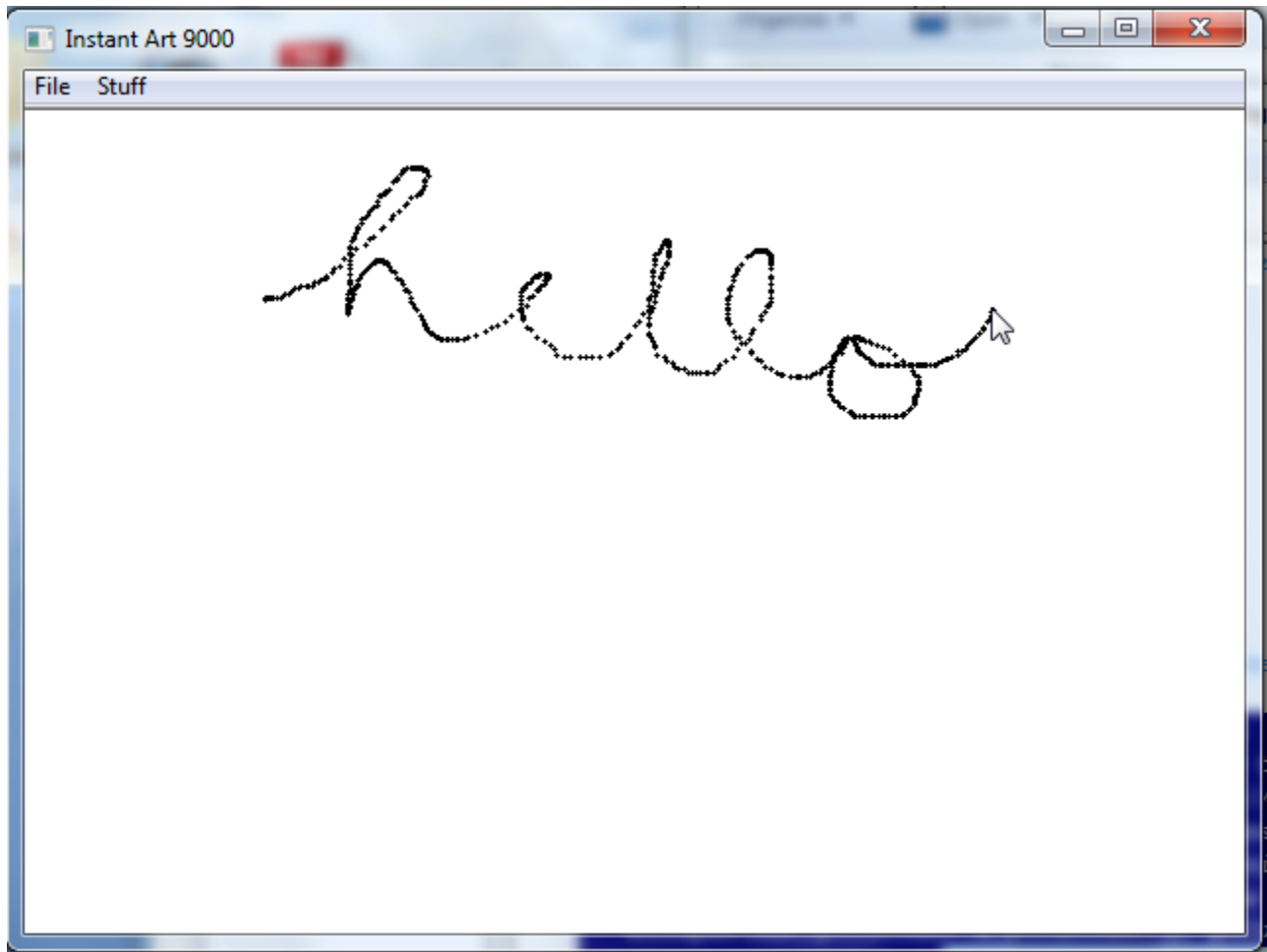
```
//place mouse movement code here
case WM_MOUSEMOVE:
    ptCurrent = MAKEPOINTS(lParam); //get the coordinates of the mouse while
    // moving
break;
//end mouse movement code here
```

We added a WM_MOUSEMOVE message handler, and used the MAKEPOINTS() function to get the X and Y of the mouse cursor, and place those values in a variable we named ptCurrent. This variable will hold the X and Y values in ptCurrent.x and ptCurrent.y.

Now we are going to add some temporary code so that you can see the WM_MOUSEMOVE handler in action. In the WM_MOUSEMOVE handler that you just created, modify it to add the temporary code that you see below. In this temporary code we get the device context of the window with the GetDC function, create a black brush with the CreateSolidBrush() function that we used before, we select the brush and use it to draw a tiny ellipse wherever the mouse cursor is.

```
//place mouse movement code here
case WM_MOUSEMOVE:
    ptCurrent = MAKEPOINTS(lParam); //get the coordinates of the mouse while
    // moving
    //temporary code, delete when instructed
    hdc = GetDC(hwnd); //gets the handle for the window "canvas"
    mybrush =CreateSolidBrush(RGB(0,0,0)); //creates a brush
    //to paint with out of the three values created above (BLACK).
    SelectObject(hdc,mybrush); //tells the window to select the brush
    //and now we draw a tiny circle with the Ellipse function
    Ellipse(hdc,ptCurrent.x,ptCurrent.y, ptCurrent.x+3,ptCurrent.y+3);
    //end temporary code
break;
//end mouse movement code here
```

Press F9 to compile and run. When debugged, you should be able to draw all over that window of yours.



OK, That was interesting, but it would be nice to learn a little more about drawing to a window. Next we are going to modify our program so that we can choose colors with a dialog, draw with lines and shapes, and fill objects with color.

We will also have to modify our menu a little bit, to give us a couple more options. For all of this to be possible, we need to start by adding some variables to your program. This will be done in the getmouseheader.h file. Open it up, and make the following changes. We will add two new control definitions at the top, a new brush, and several new variables at the bottom that allow us to control different features of our program. Make sure you read and type in the comments!

```
#define IDR_MYMENU 101
#define ID_FILE_EXIT 9001
#define ID_STUFF_GO 9002
#define ID_STUFF_CHOOSEFILL 9003 //a new control to choose a fill color
#define ID_STUFF_DRAWPATTERN 9004 //a new control for changing what we draw

HBRUSH mybrush, myfillbrush; //allows us to make a paint brush |
int redval,greenval,blueval; //make some variables to hold color information
int redfactor, greenfactor, bluefactor; //and values to skew the colors
POINTS ptCurrent,oldpoints; //make a point structure to hold x and y coordinates
int mousedown; //so we can store and make decisions based on the mouse
int shapesize; //a variable to hold how big our shapes should be

HGDIOBJ mypen; //allows us to make a pen for drawing lines
int startdraw,brushsize,DrawPattern; //new variables to control drawing
CHOOSECOLOR cc; // common dialog box structure
static COLORREF acrCustClr[16]; // array of custom colors
static DWORD rgbCurrent; // initial color selection
COLORREF BorderColor,LineColor; //new variable to indicate a border for our fills
HMENU hMenu; // a variable that allows us to monkey with our menu
```

Now in the getmouse.rc file, make changes to your menu to look exactly like this. We will change the label of one menu item and add two new ones to the "Stuff" menu.

```
#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END

    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Erase Canvas!", ID_STUFF_GO

        MENUITEM "&Choose Fill Color!", ID_STUFF_CHOOSEFILL

        MENUITEM "Draw with &Patterns", ID_STUFF_DRAWPATTERN
    END
END
```

OK, next we are going to define a couple of new subroutines before we add them. This is done at the very top of the program. Add the two highlighted lines to your program just like this. We are adding a drawshapes and a drawlines subroutine.

```
#include <windows.h>
//add declarations here
#include <time.h> //we include windows time functions here
#include "getmouseheader.h"
//For future reference, you need to know the difference between
//an l (Lower Case L) and a 1 (one)
//the difference is very subtle, look at the top, the one is slanted
//this is important when typing in this code
const char g_szClassName[] = "myWindowClass"; // we name the Windows Class
void drawshapes(HDC drawhdc, HWND drawHwnd); //a new subroutine definition
void drawlines(HDC drawhdc,HWND drawHwnd); //another new subroutine
//we will define a function we can use to draw

//the Window Procedure subroutine

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
```


Now in our WM_CREATE message handler, make the following highlighted changes, so that we can initialize some more startup variables.

```
case WM_CREATE:
{
//place code for initialization here
srand(time(NULL)); // 'seed' the random number generator with a value from timer
SetTimer(hwnd, 1, 100, NULL); // create a timer to go off every 1/200 of a sec
shapessize = 50; // set an initial size for the shapes
startdraw = -1; // this sets startdraw to -1 (negative).
redfactor = 255; // a variable to hold how red we want our colors to be
greenfactor = 255; // same for green
bluefactor = 255; // and blue
DrawPattern = 0; // when we start we are not drawing a pattern
}
break;
//end initialization
```

What we are trying to accomplish right now, is for the drawing of lines to start when we click out left mouse button within the window. Then we want the drawing to stop when we click the left button again. That is why I start off with startdraw equal to -1. That way, if I multiply it by -1 it becomes a +1, and then back to -1 if I click it again. This is a neat way to toggle a variable between negative and positive.

We will add a message handler now for the left click of the mouse. Find the place in your code where it says:

```
//place code for button up here

// end button up code
```

Now add code between the comments so that it looks like this:

```
//place code for button up here
case WM_LBUTTONDOWN:
{
startdraw *= -1; // allows us to start our line with the first click
if (startdraw == 1) // if we have started drawing find the coordinates
{
ptCurrent = MAKEPOINTS(lParam); // create the mouse coordinates
oldpoints.x = ptCurrent.x; // and record the position as
oldpoints.y = ptCurrent.y; // oldpoints x and y
}
}
break;
// end button up code
```

This will allow our program to set the variable that we will use to start the drawing process and find the position of that "First Click" so that we can start the line from that point. Any line must have a "from" and a "to" location. That is what we use oldpoints for. It is our "from" location when we start drawing.

Now would be a great time to press F9 to compile and debug. We have made quite a few changes to your program and we do not want to get behind. About the only thing you will notice now is that the "Stuff" menu will look different with different options available to you.

Ok, once we have your program debugged, we are going to give you a better way to draw lines. In the WM_MOUSEMOVE message handler, delete the temporary code that I had you type in the last lesson. Your routine will look like this when you have deleted the temporary code:

```
//place mouse movement code here
case WM_MOUSEMOVE:
    ptCurrent = MAKEPOINTS(lParam); //get the coordinates of the mouse while
    // moving
    //temporary code, delete when instructed
    |
    //end temporary code
break;
//end mouse movement code here
```

Now add the following code so that the routine looks like this:

```
//place mouse movement code here
case WM_MOUSEMOVE:
    if (startdraw == 1) //if we are drawing
    {
        ptCurrent = MAKEPOINTS(lParam); //get the mouse coordinates while moving
        if (DrawPattern == 0) //if we do not want to draw a pattern...
        {
            drawlines(hdc, hwnd); //draw lines
        } else
        {
            drawshapes(hdc, hwnd); //otherwise draw a pattern of shapes
        }
    }
break;
//end mouse movement code here
```

Basically what this says is if we are trying to draw lines we will call the drawlines routine, or for patterns, we will call the drawshapes routine. But we will only do this IF we have activated drawing with the initial click of the mouse. Your program will not compile yet, as this new code is calling the drawlines and drawshapes subroutines, and we have not created them yet. At the very bottom of your main.cpp file, where it says:

```
//begin drawing function code  
  
//end drawing function code
```

Add these two routines. Be very careful as they are complicated. First is the drawshapes routine.

```
//begin drawing function code  
void drawshapes(HDC drawhdc, HWND drawHwnd)  
{  
    int i; //creates a local variable to use in the loop below  
  
    for (i = 0; i < 10;i++)//creates a loop that executes 10 times to draw 10 shapes  
    {  
        redval = rand() % redfactor; //creates a red value  
        greenval = rand() % greenfactor; //creates a green value  
        blueval = rand() % bluefactor; //creates a blue value  
        mybrush =CreateSolidBrush( RGB(redval,greenval,blueval)); //creates a brush  
        //to paint with out of the three values created above.  
        SelectObject(drawhdc,mybrush); //assigns the brush to the HDC of the window  
  
        //now we will create a rectangle of random size, using the brush to color it  
        Rectangle(drawhdc,ptCurrent.x + (rand() % shapesize) - shapesize,ptCurrent.y  
            + (rand() % shapesize) - shapesize,  
            ptCurrent.x+rand()% shapesize ,ptCurrent.y+rand() % shapesize);  
        DeleteObject(mybrush);//deletes the brush from memory  
  
        // now create another random colored brush for the ellipse we want to draw  
        redval = rand() % redfactor; //creates a red value  
        greenval = rand() % greenfactor; //creates a green value  
        blueval = rand() % bluefactor; //creates a blue value  
        mybrush =CreateSolidBrush( RGB(redval,greenval,blueval)); //creates a brush  
        // to paint with out of the three values created above.  
        SelectObject(drawhdc,mybrush); //assigns the brush to the HDC of the window  
  
        //now we will create an ellipse of random size, using the new brush  
        Ellipse(drawhdc,ptCurrent.x + (rand() % shapesize) - shapesize,ptCurrent.y  
            + (rand() % shapesize) - shapesize,  
            ptCurrent.x+rand()% shapesize ,ptCurrent.y+rand() % shapesize);  
  
        DeleteObject(mybrush);//deletes the brush from memory  
    }  
}
```

Now add the drawlines routine immediately below the code you just added.

```
void drawlines(HDC drawhdc, HWND drawhwnd)
{
    drawhdc = GetDC(drawhwnd);
    mypen = CreatePen(PS_SOLID,1,RGB(0,0,0));
    SelectObject(drawhdc, mypen);
    MoveToEx(drawhdc,oldpoints.x,oldpoints.y,NULL);
    LineTo(drawhdc,ptCurrent.x,ptCurrent.y);
    oldpoints.x = ptCurrent.x;
    oldpoints.y = ptCurrent.y;
    DeleteObject(mypen);//deletes the brush from memory
    ReleaseDC(drawhwnd,drawhdc);
}
//end drawing function code
```

Compile, Debug and run your program. You will notice now that you can start drawing a line with a click, move your mouse, and click to stop drawing.

Another thing that you will notice, is that you should be able to use the menu item "Erase Canvas!" that is in the Stuff menu to blank out your canvas when you want to erase it all.

Dialogs

We can draw lines now, so what about filling or coloring in shapes? This is what we will be working on next. To fill in a shape with a color, you must first select a color. The Windows operating system gives us a tool to do this which is pre-made. It is called the Color Selection Dialog. This lesson will show you how to activate that dialog with a menu click, and then use the color that you select in the dialog within your program.

Earlier in your program, I had you define some new variables in the getmouseheader.h file. These variables give the color selection dialog a place to store the selections that you will make, and also control how the dialog comes up. Here is a review for you; the variables I am talking about are highlighted below.

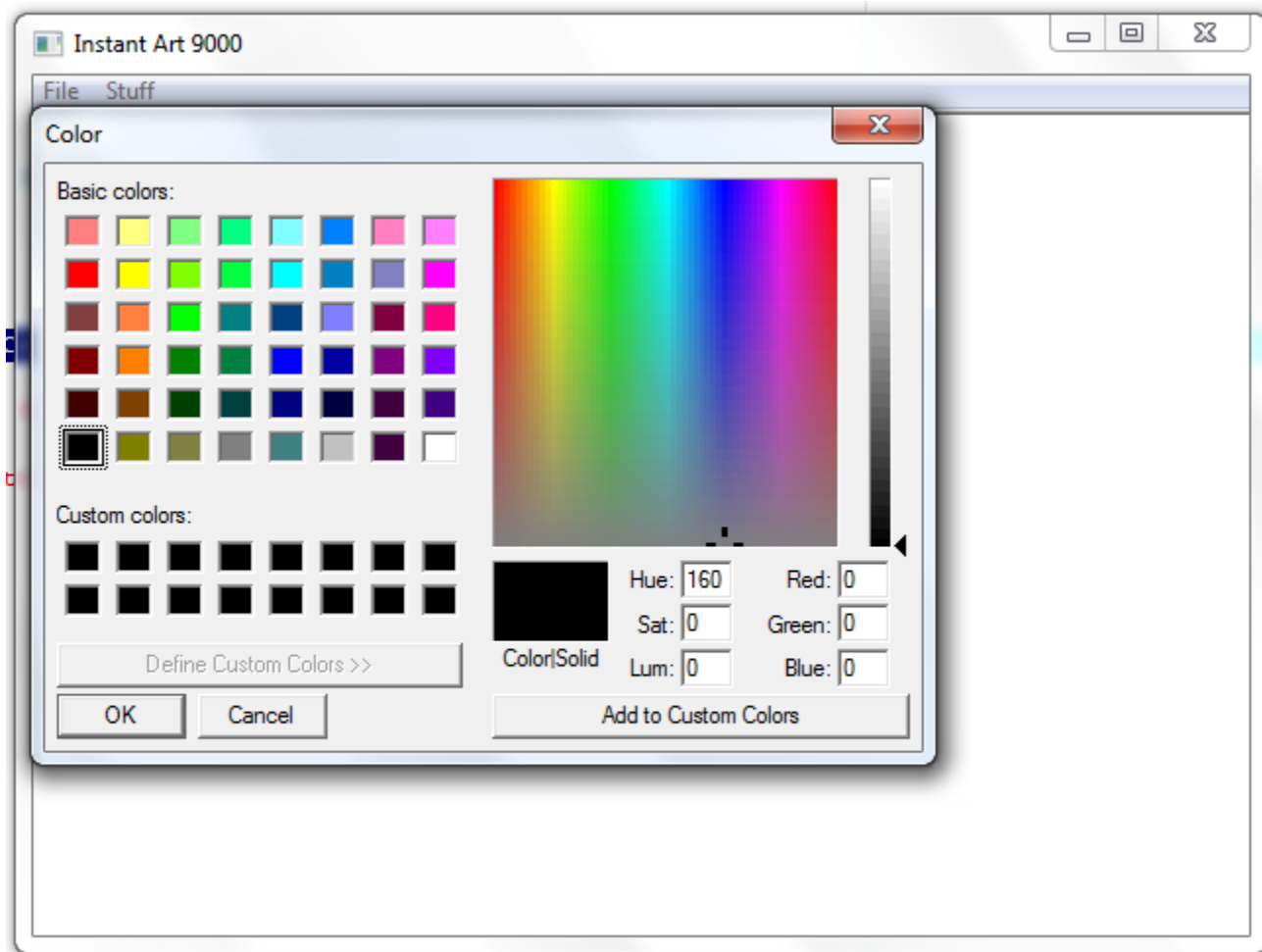
```
HGDIOBJ mypen; //allows us to make a pen for drawing lines
int startdraw,brushsize,DrawPattern; //new variables to control drawing
CHOOSECOLOR cc; // common dialog box structure
static COLORREF acrCustClr[16]; // array of custom colors
static DWORD rgbCurrent; // initial color selection
COLORREF BorderColor,LineColor; //new variable to indicate a border for our fills
HMENU hMenu; // a variable that allows us to monkey with our menu
```

We are ready to bring up our color selection dialog now, with the menu item that we created for it. In the Stuff menu, we created an item labeled "Choose Fill Color" To use that to bring up the dialog, we will have to create a message handler for it, and place the code to initialize and activate the dialog within that handler. There is a section in your code that we used to create message handlers for our other menu items. It is the case WM_COMMAND switch. Directly beneath the ID_STUFF_GO handler place the highlighted code. This code will activate when you press the "Choose Fill Color" menu item, and the color dialog will come up.

```
case WM_COMMAND:
    switch(LOWORD(wParam)) //gets the parameter of the WM_COMMAND
    {
        case ID_FILE_EXIT: //if it the exit menu
            PostMessage(hwnd, WM_CLOSE, 0, 0);
            break;
        case ID_STUFF_GO: //if it is the do stuff menu item
            MessageBox(hwnd, "clearing your masterpiece", "My Message Box", 0);
            GetClientRect(hwnd, &clientrect); //gets window rectangle
            InvalidateRect(hwnd, &clientrect, TRUE); //clears it blank
            break;

        case ID_STUFF_CHOOSEFILL:
            // Initialize CHOOSECOLOR
            ZeroMemory(&cc, sizeof(cc)); //Initialize the memory
            cc.lStructSize = sizeof(cc); // for the cc structure
            cc.hwndOwner = hwnd; //your program will own the dialog
            cc.lpCustColors = (LPDWORD) acrCustClr; //holds Custom Colors
            cc.rgbResult = rgbCurrent; //sets the dialog to the current
            //selected color. The first time you use the dialog it will
            //be black. Move the slider on the right up and down to
            //lighten or darken your colors.
            cc.Flags = CC_FULLOPEN | CC_RGBINIT; //configures the dialog
            //to open with all options, and initialized
            if (ChooseColor(&cc) == TRUE) //opens the dialog
            {
                //this next line creates a brush from the color you
                //picked, called myfillbrush
                myfillbrush = CreateSolidBrush(cc.rgbResult);
                //this next line stores the selected color for the
                //time you open the dialog
                rgbCurrent = cc.rgbResult;
            }
            break;
```

Press F9 to compile your code, debug as required. Test out the menu item now, to see the new dialog box!



When you are selecting colors, you **MUST** move the right slider up and down, or the dialog will return black. Your new color will be presented in the Color|Solid box on the dialog and you play with the dialog.

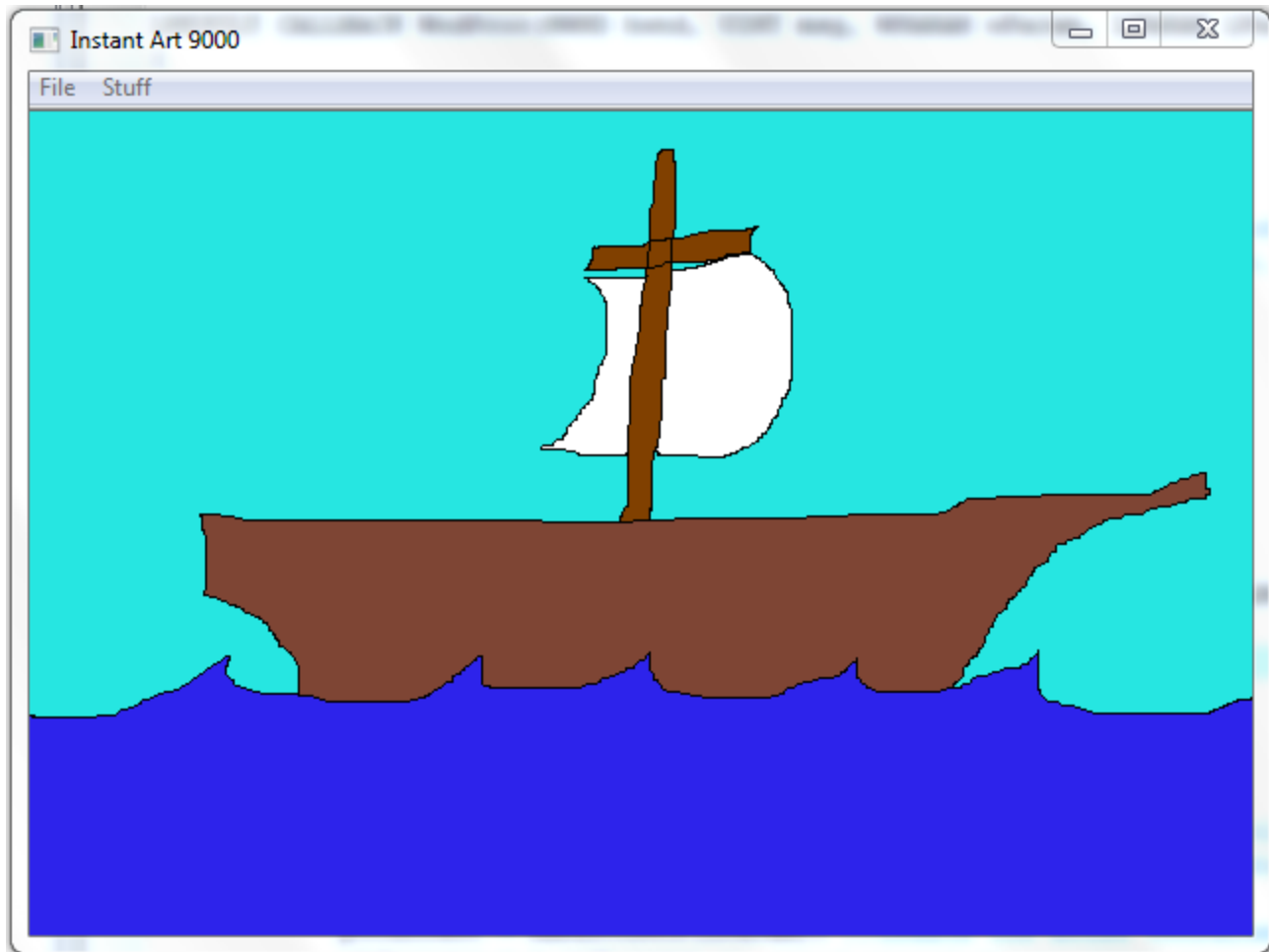
The next step will be to do something with this new color. We can draw with the left mouse button by clicking it, dragging to draw, and clicking it again to stop drawing. We can select a new color now as well. Now we will give the program the ability to fill the shaped we draw with a right mouse button click. At the top of your windows procedure, in the message switch, you will see the two comments that I had you type in, that tell you where to place the button down code. Place the highlighted code between the comments.

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    int i;    //declare local variables
    HDC hdc;  //device context (window canvas to "paint on")
    RECT clientrect; //a rectangle to hold window boundaries

    switch(msg) //this giant switch statement gets messages sent to the window by
    //the operating system, such as mouse clicks and movements, keypresses, timer
    //events, and other commands that you may create with menu items.
    {
        // place code for button down here
        case WM_RBUTTONDOWN:
        {
            hdc = GetDC(hwnd);
            ptCurrent = MAKEPOINTS(lParam);
            SelectObject(hdc, myfillbrush);
            BorderColor = RGB(0,0,0);
            ExtFloodFill(hdc, ptCurrent.x, ptCurrent.y, BorderColor, FLOODFILLBORDER);
        }
        break;
        // end button down code
        //place code for button up here
    }
}
```

This code uses the ExtFloodFill function to fill in all pixels (points on your screen) with the color you selected in the dialog, by clicking within a closed shape with the right mouse button. Try it out. What can you draw? If you click outside of a closed shape, it will fill in the entire window, or until it hits a black line that has no gaps.

Press F9 now, and draw something! When you have it debugged, you will be able to create something like this! Don't laugh too hard; just try to draw something better. It is a bit tricky actually! You can fill and refill in shapes to correct mistakes, you can erase your entire canvas with the menu now, and if a color floods outside of your shapes, look for gaps in your lines and draw them in with your left mouse button. HAVE FUN!



Modifying Menu Items

Sometimes a menu item is used to set an option for a program. This is the case with this little program. We are going to use a menu item to set the program to either draw lines or patterns. We have already added the menu item, but we need to program the handler for it now. This handler will be for the Draw with Patterns menu item. We gave it the ID of `ID_STUFF_DRAWPATTERN`.

Directly beneath the handler for `ID_STUFF_CHOOSEFILL`, place the highlighted code that you see below. This code will set the `drawpattern` variable either on or off when you click it, and either check or uncheck the menu item in the menu.

```

case ID_STUFF_CHOOSEFILL:
    // Initialize CHOOSECOLOR
    ZeroMemory(&cc, sizeof(cc)); //Initialize the memory
    cc.lStructSize = sizeof(cc); // for the cc structure
    cc.hwndOwner = hwnd; //your program will own the dialog
    cc.lpCustColors = (LPDWORD) acrCustClr; //holds Custom Colors
    cc.rgbResult = rgbCurrent; //sets the dialog to the current
    //selected color. The first time you use the dialog it will
    //be black. Move the slider on the right up and down to
    //lighten or darken your colors.
    cc.Flags = CC_FULLOPEN | CC_RGBINIT; //configures the dialog
    //to open with all options, and initialized
    if (ChooseColor(&cc)==TRUE) //opens the dialog
    {
        //this next line creates a brush from the color you
        //picked, called myfillbrush
        myfillbrush = CreateSolidBrush(cc.rgbResult);
        //this next line stores the selected color for the
        //time you open the dialog
        rgbCurrent = cc.rgbResult;
    }
break;

case ID_STUFF_DRAWPATTERN:
    if (DrawPattern == 0) //if we are not already drawing patterns
    {
        DrawPattern = 1; //turn pattern drawing on
        hMenu= GetMenu(hwnd); // here we are going to get
        //the handle of the window menu so we can change the
        //menu item to indicate that it is "Checked". The
        //ModifyMenu command below sets it to checked.
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_CHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    } else
    {
        //if we were drawing with patterns before we clicked the
        //menu item, we turn pattern drawing off, then use the
        //ModifyMenu function to "Uncheck" the menu.
        DrawPattern = 0;
        hMenu= GetMenu(hwnd);
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_UNCHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    }
break;

```

Now we need to make a small adjustment to the WM_MOUSEMOVE message handler. If we are starting to draw, we want to get the handle to the device context so that we can send it to the drawshapes subroutine. This way, drawshapes will know what to draw on. Find the WM_MOUSEMOVE message handler and insert the highlighted code you see below.

```
//place mouse movement code here
case WM_MOUSEMOVE:
    if (startdraw == 1) //if we are drawing
    {
        //put this new line of code here, it gets the device context for the
        //drawing operations
        hdc = GetDC(hwnd);
        ptCurrent = MAKEPOINTS(lParam); //get the mouse coordinates while moving
        if (DrawPattern == 0) //if we do not want to draw a pattern...
        {
            drawlines(hdc, hwnd); //draw lines
        } else
        {
            drawshapes(hdc, hwnd); //otherwise draw a pattern of shapes
        }
    }
    break;
```

Press F9 to Compile and Run your program, debug as required. Try out your Draw with Patterns option now and see what happens on your screen.



Exercise and review

Add two new menu items, one to set the program to draw a small pattern, and one to set the program to draw a large pattern. Have the new menu items control the `shapessize` variable, by making it either 20 for the small pattern, or 50 for the large pattern. You can name the menu item anything you like, I would use something like `ID_STUFF_SMALLPATTERN` and `ID_STUFF_LARGE_PATTERN`.

Good Luck!

Solution

How would we solve the above problem? The first thing I would do is to break it down into individual problems, solve each one in order. I would then look in my code for examples of how similar problems were solved. If I break down the exercise, I see two puzzles.

1. Add the `ID_STUFF_SMALLPATTERN` and `ID_STUFF_LARGE_PATTERN` menu items.
2. Handle each menu item click so that it controls the `shapesize` variable.

Solving the first puzzle.

We added menu items to our program earlier. We learned that to add a menu item you have to;

Create the menu items in the `.RC` file

Define the menu item IDs in the `.h` file

Create a handler for them in the `.cpp` (source code) file.

So let's look at the `.RC` file first. Currently it looks like this;

```
#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END

    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Erase Canvas!", ID_STUFF_GO

        MENUITEM "&Choose Fill Color!", ID_STUFF_CHOOSEFILL

        MENUITEM "Draw with &Patterns", ID_STUFF_DRAWPATTERN
    END
END
```

We could add the menu items to the Stuff popup, using the same format that we did when we added the Fill Color and Draw with Patterns items.

The resulting file would look like this, with the new code highlighted.

```

#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END

    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Erase Canvas!", ID_STUFF_GO

        MENUITEM "&Choose Fill Color!", ID_STUFF_CHOOSEFILL

        MENUITEM "Draw with &Patterns", ID_STUFF_DRAWPATTERN

        MENUITEM "Use &Large Pattern", ID_STUFF_LARGE_PATTERN
        MENUITEM "Use &Small Pattern", ID_STUFF_SMALL_PATTERN
    END
END

```

OR if we want to be really fancy we can add another popup for options. The result would look like this with the new code highlighted.

```

#include "getmouseheader.h"

IDR_MYMENU MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit", ID_FILE_EXIT
    END

    POPUP "&Stuff"
    BEGIN
        MENUITEM "&Erase Canvas!", ID_STUFF_GO

        MENUITEM "&Choose Fill Color!", ID_STUFF_CHOOSEFILL

        MENUITEM "Draw with &Patterns", ID_STUFF_DRAWPATTERN
    END

    POPUP "&Options"
    BEGIN
        MENUITEM "Use &Large Pattern", ID_STUFF_LARGE_PATTERN

        MENUITEM "Use &Small Pattern", ID_STUFF_SMALL_PATTERN
    END
END

```

Choose one way or the other. I actually like the second way best myself. Regardless of how you do this, you must now define the new IDs in the .h file. Let's look at the two definitions that you need to add, highlighted here.

```
#define IDR_MYMENU 101
#define ID_FILE_EXIT 9001
#define ID_STUFF_GO 9002
#define ID_STUFF_CHOOSEFILL 9003 //a new control to choose a fill color
#define ID_STUFF_DRAWPATTERN 9004 //a new control for changing what we draw
#define ID_STUFF_SMALLPATTERN 9005 //a new control to draw small patterns
#define ID_STUFF_LARGE PATTERN 9006 //a new control to draw large patterns

HBRUSH mybrush, myfillbrush; //allows us to make a paint brush
int redval, greenval, blueval; //make some variables to hold color information
int redfactor, greenfactor, bluefactor; //and values to skew the colors
POINTS ptCurrent, oldpoints; //make a point structure to hold x and y coordinates
int mousedown; //so we can store and make decisions based on the mouse
int shapsize; //a variable to hold how big our shapes should be

HGDIOBJ mypen; //allows us to make a pen for drawing lines
int startdraw, brushsize, DrawPattern; //new variables to control drawing
CHOOSECOLOR cc; // common dialog box structure
static COLORREF acrCustClr[16]; // array of custom colors
static DWORD rgbCurrent; // initial color selection
COLORREF BorderColor, LineColor; //new variable to indicate a border for our fills
HMENU hMenu; // a variable that allows us to monkey with our menu
```

Notice how we just continued the numbering from 9004 to 9005 and 9006. Every ID must have a unique number assigned to it.

You should be able to press F9 to compile and run your program. Of course you may have to debug a little. So puzzle 1 is complete!

Solving the second puzzle

Now we need to give our menu items the ability to change the value of the shapsize variable. For any command to have "smarts", we have to give it a handler. We just finished this when we added the Color Selection option and the Draw Patterns Option. Look at the following code, where we added the DrawPattern handler.

```
case ID_STUFF_DRAWPATTERN:
    if (DrawPattern == 0) //if we are not already drawing patterns
    {
        DrawPattern = 1; //turn pattern drawing on
        hMenu= GetMenu(hwnd); // here we are going to get
        //the handle of the window menu so we can change the
        //menu item to indicate that it is "Checked". The
        //ModifyMenu command below sets it to checked.
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_CHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    } else
    {
        //if we were drawing with patterns before we clicked the
        //menu item, we turn pattern drawing off, then use the
        //ModifyMenu function to "Uncheck" the menu.
        DrawPattern = 0;
        hMenu= GetMenu(hwnd);
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_UNCHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    }
    break;
```

Immediately after the break, we will add the two handlers that we need. Edit your code so that it looks like this.


```
case ID_STUFF_DRAWPATTERN:
    if (DrawPattern == 0) //if we are not already drawing patterns
    {
        DrawPattern = 1; //turn pattern drawing on
        hMenu= GetMenu(hwnd); // here we are going to get
        //the handle of the window menu so we can change the
        //menu item to indicate that it is "Checked". The
        //ModifyMenu command below sets it to checked.
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_CHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    } else
    {
        //if we were drawing with patterns before we clicked the
        //menu item, we turn pattern drawing off, then use the
        //ModifyMenu function to "Uncheck" the menu.
        DrawPattern = 0;
        hMenu= GetMenu(hwnd);
        ModifyMenu(hMenu, ID_STUFF_DRAWPATTERN, MF_BYCOMMAND |
        MF_UNCHECKED, ID_STUFF_DRAWPATTERN, "Draw with &Patterns");
    }
    break;
case ID_STUFF_LARGE_PATTERN:
    shapsize = 50;
    break;
case ID_STUFF_SMALL_PATTERN:
    shapsize = 20;
    break;
```

Now are menu items are smart. We can borrow some additional code from the ID_STUFF_DRAWPATTERN handler to check these items when selected. I am going to take it another step, in that when I check one item, I am going to uncheck the other. Look at the changes very carefully.

```
case ID_STUFF_LARGE_PATTERN:
    shapsize = 50;
    hMenu= GetMenu(hwnd); // here we are going to get
    //the handle of the window menu so we can change the
    //menu item to indicate that it is "Checked". The
    //ModifyMenu command below sets it to checked.
    ModifyMenu(hMenu, ID_STUFF_LARGE_PATTERN, MF_BYCOMMAND |
    MF_CHECKED, ID_STUFF_LARGE_PATTERN, "Use &Large Pattern");
    ModifyMenu(hMenu, ID_STUFF_SMALL_PATTERN, MF_BYCOMMAND |
    MF_UNCHECKED, ID_STUFF_SMALL_PATTERN, "Use &Small Pattern");
break;
case ID_STUFF_SMALL_PATTERN:
    shapsize = 20;
    hMenu= GetMenu(hwnd); // here we are going to get
    //the handle of the window menu so we can change the
    //menu item to indicate that it is "Checked". The
    //ModifyMenu command below sets it to checked.
    ModifyMenu(hMenu, ID_STUFF_LARGE_PATTERN, MF_BYCOMMAND |
    MF_UNCHECKED, ID_STUFF_LARGE_PATTERN, "Use &Large Pattern");
    ModifyMenu(hMenu, ID_STUFF_SMALL_PATTERN, MF_BYCOMMAND |
    MF_CHECKED, ID_STUFF_SMALL_PATTERN, "Use &Small Pattern");
break;
```

Press F9 and Debug, Compile and Run your program. You can now use the menu to select a large or small size to draw the patterns on your window.

Conclusion and resources

<http://cplus.about.com> contains very well thought out C and C++ tutorials complete with lessons and practice exercises, along with solutions.

<http://cplusplus.com> A very well organized C++ reference with tutorials, references, etc. One of my favorites!

<http://www.cppreference.com/> contains a complete C and C++ language reference. Use this with VIDE as a language reference.

<http://winprog.org/tutorial/> is the home of a very good introductory Win32 tutorial for those of you wishing to move beyond the console.

Good Luck!!!

