Chapter 2 – Meet Mr. C++

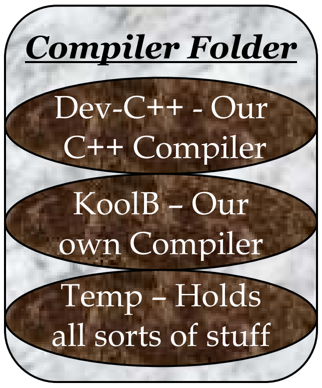
Setting Up Our C++ Compiler:

Welcome back! In this section we will be meeting mean old Mr. C++, setting up his home, and starting to program our compiler. With that in mind, we first need to choose a C++ compiler. Just as with BASIC, numerous C++ compilers exist. Choosing one isn't too difficult however because we can eliminate most because of either a hefty price tag (Microsoft Visual C++, for example), or because of a lack of a good user interface (I never liked typing in stuff at a command prompt). I finally settled on Dev-C++, a freeware compiler with a nice IDE (not RAD, though). Thankfully, we won't have to wait forever for it to download, as it size is only 7.5 MB; however, if your connection is slow or your ISP disconnects you often, you might want to use a download manager to pause and resume your downloads (FreshDownload is a good one without any ads, see http://www.freshdevices.com for more details). For our purposes, Dev-C++ 4.0 will work (5.0 is still beta and buggy). By now you are probably thinking, "Would he just hurry up and tell me where I can download it from?!?" Ok, ok, here are some download links:

http://prdownloads.sourceforge.net/dev-cpp/devcpp4.zip
http://www.simtel.net/pub/pd/17456.html
ftp://62.137.135.95/devcpp4.zip

If you find that these links don't work, try searching for devcpp4.zip on Google or Simtel.net.

While Dev-C++ downloads, lets create a home for our compiler project. First, we need a location for the project. I will put mine under C:\Compiler, and I strongly suggest you do as well. So create a folder under the C drive and name it 'Compiler.' Next, create three folders under the 'Compiler' directory: 'Dev-C++' (for the C++ compiler), 'Temp' (for temporary stuff), and…hey wait a minute. We haven't named our compiler yet! Lets do so now. Hmm, how about 'RBA' for Really Bad Acronym? No? Well, how about 'Kool-Bee?' OK, for lack of a better name (you can tell that naming things is one of my better strengths, right?), we'll just name our compiler Kool-Bee or KoolB for short. So, back on track, create another folder in 'Compiler' and name it 'KoolB.' Has Dev-C++ downloaded yet? Not quite yet? Well then, take a water break (coffee isn't that good for you) while the download finishes or look at the diagram to the right, which shows visually where everything goes.

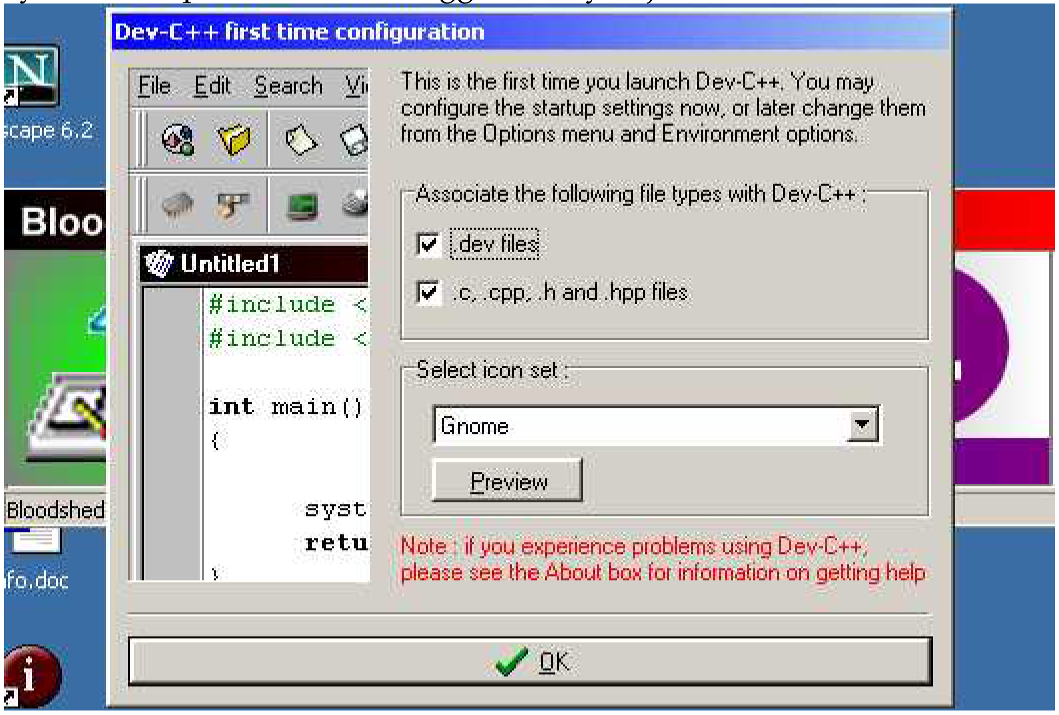


Now that Dev-C++ has downloaded, unzip it with WinZip or, my favorite, 7-zip (see http://www.7-zip.com) into the 'C:\Compiler\Temp' folder and run 'Setup.exe' (if your zipping software offers to automatically install it, go right ahead). Dev-C++

now presents you with the Setup Screen. Now begins the fun part: the installation. Hit the 'Yes' button to accept the license (having fun reading it?). The next screen gives you the option of a Typical, Compact, or Custom installation. If you have the room (about 25 MB) use the default Typical installation. We are now almost done; however, we need to change the destination directory, so click the 'Browse' button and select 'C:\Compiler\Dev-C++' folder (alternatively, just type it in). Click 'Next' and then kick back and relax while Dev-C++ installs itself.

Creating the Compiler Project:

After extracting the files and creating the icons, the installation process give you the opportunity to run the program, so check the option 'Yes, Launch the program file' and click 'Finish.' The installation process exits and Dev-C++ launches for the first time. It gives you some options, I would suggest that you just click 'OK.'


Dev-C++ first time configuration

This is the first time you launch Dev-C++. You may configure the startup settings now, or later change them from the Options menu and Environment options.

Associate the following file types with Dev-C++ :

✓ .dev files

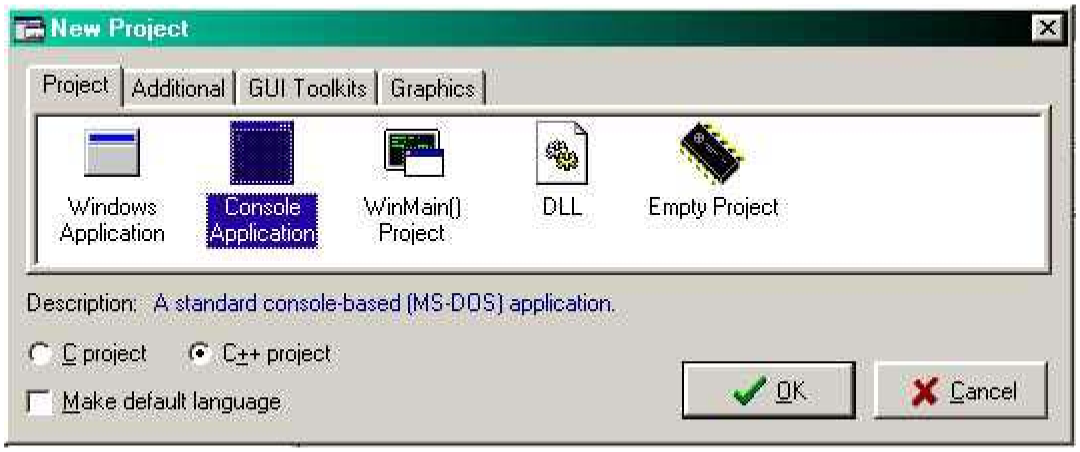
✓ .c, .cpp, .h and .hpp files

Select icon set : Gnome

Note : if you experience problems using Dev-C++, please see the About box for information on getting help

```
#include <
#include <

int main()
{

      syst
      retu
}
```

Finally, the main screen appears and leaves you there staring at it. Not very pretty, but it gets the job done. OK, enough staring at it; let's do something! How about we create a new project for our compiler? To do that, go File -> New Project. Dev-C++ displays a bunch of options, so lets logically think this through. We just want a compiler that translates BASIC to assembly language, so we don't need a fancy Windows interface; of course, later on we will need a nice IDE, but not now. That leaves the choice 'Console Application,' which will do, so click on the 'Console Application' option and make sure the 'C++ project' option is selected as you see below:

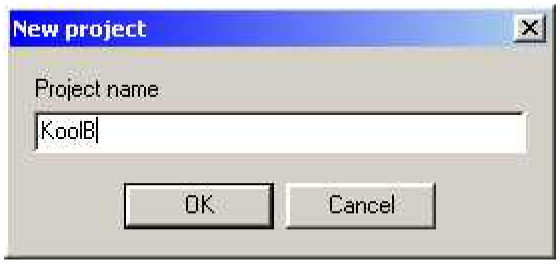


Description: A standard console-based (MS-DOS) application.

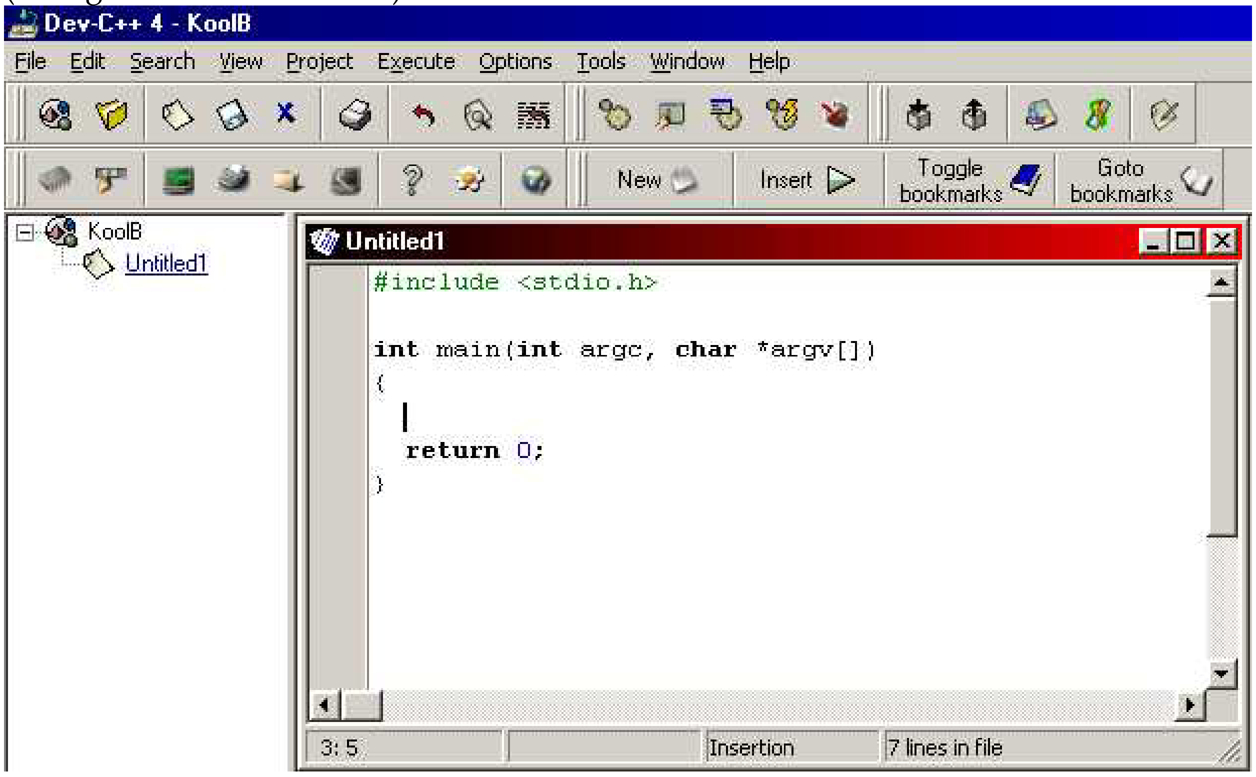
C project [✓ C++ project]

Make default language

Click 'OK' and type in 'KoolB' when it asks you for a project name. When it asks you where you want to save it, navigate to 'C:\Compiler\KoolB' and click 'Save.' Presto, you have just created KoolB project! You should now have a screen that looks like this (disregard the C++ for now):



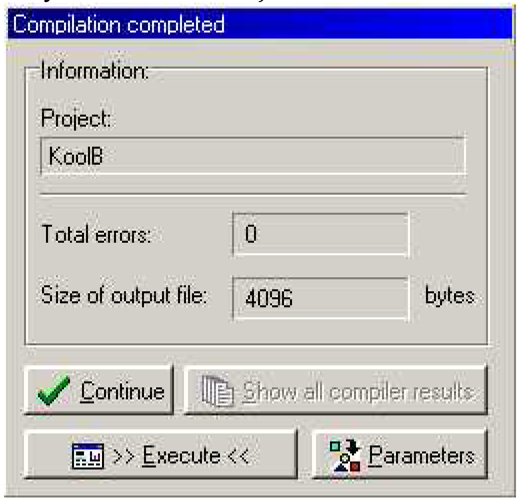
Project name: KoolB



```
#include <stdio.h>

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

  return 0;
}
```

Introducing Mr. C++:

First, lets make sure that everything works. Save everything by going, File -> Save All; Dev-C++ prompts you to save the 'Untitled1' file, so type in Main, since it is the main module of our compiler. Hit 'Save.' Before we meet Mr. C++, lets do a trial compile; to do so, go Execute -> Rebuild All. A dialog box pops up and informs you that Dev-C++ is busy compiling the C++ code; when it is finished compiling, it gives you the compilation results (as seen to the right). Here you see the Project name, the total number of errors, and the size of output file (in another words, the size of the program). The very first thing you want to look at is the 'Total Errors.' That tells us how many mistakes we have made. Right now it should be 0 since we haven't done anything. Later we will have to deal with errors. Take a look at the size: 4096 bytes. That is 4 KB! That is one of the reasons C++ makes a good choice for a compiler, it compiles into compact and fast programs. Once you have finished looking at the dialog box, click 'Continue.'


Compilation completed

Information:

Project: KoolB

Total errors: 0

Size of output file: 4096 bytes

Now the part you have been (hopefully) steeling yourself for: meeting Mr. C++. My approach in introducing you to Mr. C++ will be to show you how Mr. C++ wants you to program, explaining it to you, and then comparing it to Mrs. Rapid-Q. Ever heard the expression 'Mother knows best?' Well, it is true. Mrs. Rapid-Q has it all together and is just one bundle of fun. Mr. C++, on the other hand, can be moody, grumpy, and overall unpleasant. His motto is "My way or the high way." Nice guy, huh? Why do we want to use him, then? He has a lot more speed than Rapid-Q. Also, he requires less space and is more structured. This makes him a good choice for a compiler. My aim is to show you that once you understand how Mr. C++ relates to Mrs. Rapid-Q, you should be able to mentally translate between the two. After a while, you won't even need to translate it in your mind, you just will understand it.

First, (take a deep breath…it might be worse than meeting your in-laws!) let me introduce Mr. C++ (as generated in the Dev-C++ window):

```
#include <stdio.h>

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

  return 0;
}
```

Whew! Before you throw in the towel and say "This is too much for me. I'll stick to Mrs. Rapid-Q (which, by the way, isn't a half bad idea)," let me show you how Mr. C++'s methods compares to Mrs. Rapid-Q's line by line:

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

This includes the file “stdio.h”, which allows us to communicate with the outside world by printing to and from the console. Mr. C++’s #include statement is basically the same as Mrs. Rapid-Q’s $Include except you use ‘<’ and ‘>’ instead of quotes. Using Mrs. Rapid-Q, we would say:

```
$Include “Rapidq.inc”
```

The next line of C++ says:

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

Unlike Mrs. Rapid-Q’s system where the program’s code can be placed anywhere, Mr. C++ says, “Thou shalt put all code in a special function called main.” When the C++ program is run, this is the function that runs, so everything in it is the program and when this function exits, the program exits as well. The parameter argc refers to the number of parameters on the command line; Mrs. Rapid-Q’s equivalent is the internal variable CommandCount. The other parameter is argv, which contains the actual command line parameters, just like Mrs. Rapid-Q’s Command$() function. So overall, Mrs. Rapid-Q’s equivalent of this would be:

```
Function main (CommandCount As Integer, Command As String) As Integer
```

The next line of C++ is rather simple:

```
{
```

Here is another place where Mr. C++ is just plain weird. As my Algebra professor used to say, “Computers aren’t intelligent.” Well, in this situation, Mr. C++ is even less intelligent than Mrs. Rapid-Q: it doesn’t even know where the function begins! So you have to tell it by placing a { in front of the beginning of the function. Right now, the only thing in this function is:

```
   return 0;
```

Notice the semi-colon at the end of the line? That’s on of Mr. C++’s worst personality traits. He makes you, as in forces you, put a semi-colon at the end of each statement. Of all the contorted tortures he puts us C++ programmers through, that is the worst. To make it even worse, some statements in Mr. C++ vocabulary don’t need a semi-colon such as functions, if's, etc, but in general, Mr. C++ makes you put a semi-colon after each line. In Mr. C++'s vocabulary, return has a special meaning. With Mrs. Rapid-Q, you would always put the return value of a function in the variable Result (or the function name). Mr. C++ isn't always as swift in the brain as Mrs. Rapid-Q, you have tell him specificially what value you want to return by saying `'return'` and then the value you want to return. Mrs. Rapid-Q would only have you do this:

```
Result = 0
```

The final line is:

```
}
```

As I mentioned earlier, Mr. C++ isn't as bright as Mrs. Rapid-Q. Mr. C++ doesn't know when to end a function, so you have to tell it by using `'}'`. The Rapid-Q equivalent would be:

```
End Function
```

Now lets look a the two versions, both Mr. C++'s and Mrs. Rapid-Q's, side by side:

```
$Include "RapidQ.inc"

Function main (CommandCount As Integer,_
               Command As String) _
               As Integer

  Result = 0       'or Main = 0
End Function
```

```
#include <stdio.h>


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

  return 0;
}
```

OK, let's list our complaints against Mr. C++:

1. To include a file (or module), we must write #include <filename>, where the filename is the name of the file to include.
2. All code ***must*** be in the `main` function.

3. To return a value from a function, we must use the statement `return` and then the value to return. (For the main function, this value is typically zero.)
4. Returning from the `main` function means the program has finished and that the program has nothing more to do.
5. The function body needs a stop and a finish, to start a function use {, to stop a function, use }.
6. Most statements need a semi-colon at the end, it is similar to pressing ENTER in Mrs. Rapid-Q.

Programming the Main Module:

Now that you understand all the intimate personality traits of Mr. C++…lets do some programming. Well, OK, maybe you don't still don't know Mr. C++ that well, but lets do some hand-on exercises to help cement what you just learned. If you get lost, don't worry, I will put the entire code at the end of this section. Swap back to the Dev-C++ window (If you closed it, just go File -> Open and navigate to KoolB.dev) and type the following line in between the { and before the `return 0;` statements:

```
printf("Welcome to the KoolB Compiler programmed by Brian C. Becker.\r\n");
```

Before I explain what this does, replace "Brian C. Becker" with your own name (after all, you are the one doing all the hard work!). Now let me explain what this does. Mr. C++ likes to confuse us by renaming Mrs. Rapid-Q's print statement to `printf`; he also makes us use parentheses and a semi-colon at the end. The \r\n at the end means to print a newline. Here is Mrs. Rapid-Q's form (much easier if I do say so):

```
print "Welcome to the KoolB Compiler by Brian C. Becker."
```

To compile that, go Execute -> Rebuild All. Hopefully you won't get any errors, if you do, check to make sure that all your quotes are there and you have a semi-colon at the end. Once it compiles successfully, click 'Continue.' We will run it in just a couple of minutes, but first lets add some more stuff. I am a big fanatic of lots of functions and objects, so let's add some functions right before the `main` function but below the `#include` statement:

```
void Start(void);
void Compile(void);
void Stop(void);
```

Now I know you are thinking "I thought he said he was going to add some functions. Those don't look like functions!" But yes, these are Mr. C++ style functions.

His way of declaring functions is radically different from Mrs. Rapid-Q's. Mr. C++'s general form to declare a function is:

```
DataType FunctionName (Parameters);
```

Let's compare that with Mrs. Rapid-Q's way of declaring a function:

```
Declare Function FunctionName (Parameters) As DataType
```

In this example, DataType is the type of data that the function returns, FunctionName is the name of the Function, and the Parameters are of course the parameters. When we want to skip something, we cannot just leave it blank as Mrs. Rapid-Q allows us to do, mean Mr. C++ makes us use a special word: void. Void just means nothing. "Hey then," I hear you say, "if void means nothing, that means that the functions Start, Compile, and Stop return nothing, which makes them subroutines, not functions!" Well, doggone it, you are so totally right, but try telling that to Mr. C++; he just doesn't understand. To his warped mind, everything is a function. So let me restate these quote functions the correct way in Mrs. Rapid-Q's style:

```
Declare Sub Start
Declare Sub Compile
Declare Sub Stop
```

Nothing too hard about that, is there?

Now that we have declared these functions, lets fill them out. Usually we like to do that at the bottom of the file, so move to the last line below the main function and copy & paste this:

```
void Start(void){

}

void Compile(void){

}

void Stop(void){

}
```

So here we have the three functions with space to put the code inside. Mrs. Rapid-Q way is:

```
Sub Start

End Sub

Sub Compile

End Sub

Sub Stop

End Sub
```

As you can see, Mrs. Rapid-Q doesn't need to be told that there are not any parameters (with the void keyword) while Mrs. C++ does, Mrs. Rapid-Q doesn't need to be told where the sub begins, while Mr. C++ does, and Mr. Rapid-Q doesn't need to be told that there aren't any return values, while Mr. C++ does. I am not sure about you, but this leads me to the conclusion that Mr. C++ is sort of stupid! Technically, in today's world you aren't supposed to call anybody stupid; the 'politically correct' term is 'slow learner.' Whatever: the fact still remains, C++ is dim-witted.

Now let's flesh out these 'functions.' I was thinking that it would be nice to perhaps show the time that it took to compile a program, sort of like Rapid-Q. We could do this by checking the time in the Start function and then subtracting that from the time in the Stop function. How about it? Sound good to you? OK, lets plan out what we need:

1. We need a global variable to keep track of the starting time.
2. We need a way of getting the time.
3. We need a way of displaying the time.

With that in mind, lets will start. First, we need a global variable to keep track of the time. Sounds simple enough, we will just name it 'StartTime.' Unfortunately, we again encounter another one of Mr. C++'s quirks. He wants us to declare the variable differently from how we normally do it with Mrs. Rapid-Q. Ah, well, I guess we will just have to humor him. Since this is a global variable, lets put it at the top of the program right after the functions:

```
DWORD StartTime;
```

A bit strange, huh? Mr. C++ wants us to first tell him what data type it is and then what the name of the data is. Here, we want a DWORD type of data; don't worry; DWORD is just another name for a positive integer. Mrs. Rapid-Q would have us do it like this:

```
Dim StartTime As Long
```

Secondly, we need a way of getting the time. To do so, we need the Windows API. The Windows API is a set of functions enclosed in Dynamic Link Libraries that are provided by Microsoft to help Windows programmers program using Windows newest features. The Windows API can be even stranger than Mr. C++, so we will go slowly when we dig further into it; however, for now, we just need a function to get the current time. Digging through the API time functions, I noticed one that might work: GetTickCount. It gets the number of milliseconds (1/1000 of a second) since the computer was turned on. That should work, but hey, wait a second, if GetTickCount is a Windows function, don't we need to somehow import it from the DLL? With Mrs. Rapid-Q, we would do this:

```
Declare Function GetTickCount Lib "kernel32.dll" alias "GetTickCount"_
() As Long
```

Now please don't faint or anything, but for once, Mr. C++'s way is easier. Yes, you heard me; Windows API functions are actually easier in C++! Do I hear a hearty 'Hurrah' back there? Do you know why Dev-C++ is so much larger than Rapid-Q? That is because Dev-C++ contains 10 MB worth of other peoples work! Somebody else has already done the hard stuff for you, all you have to do is include his or her work (which happens to be named, what else, windows.h). So without further delay, put the following line at the very top of the file:

```
#include <windows.h>
```

That allows us to use most of the Windows API functions, including GetTickCount. Thirdly, we needed a way to print the time; that's easy, we will just use the printf function that we used earlier. Here are the functions all fleshed out :

```
void Start(void){
  StartTime = GetTickCount();
  return;
}

void Compile(void){

  return;
}

void Stop(void){
  printf("Compile Time:\t%f", (GetTickCount()-StartTime)/1000.0);
  return;
}
```

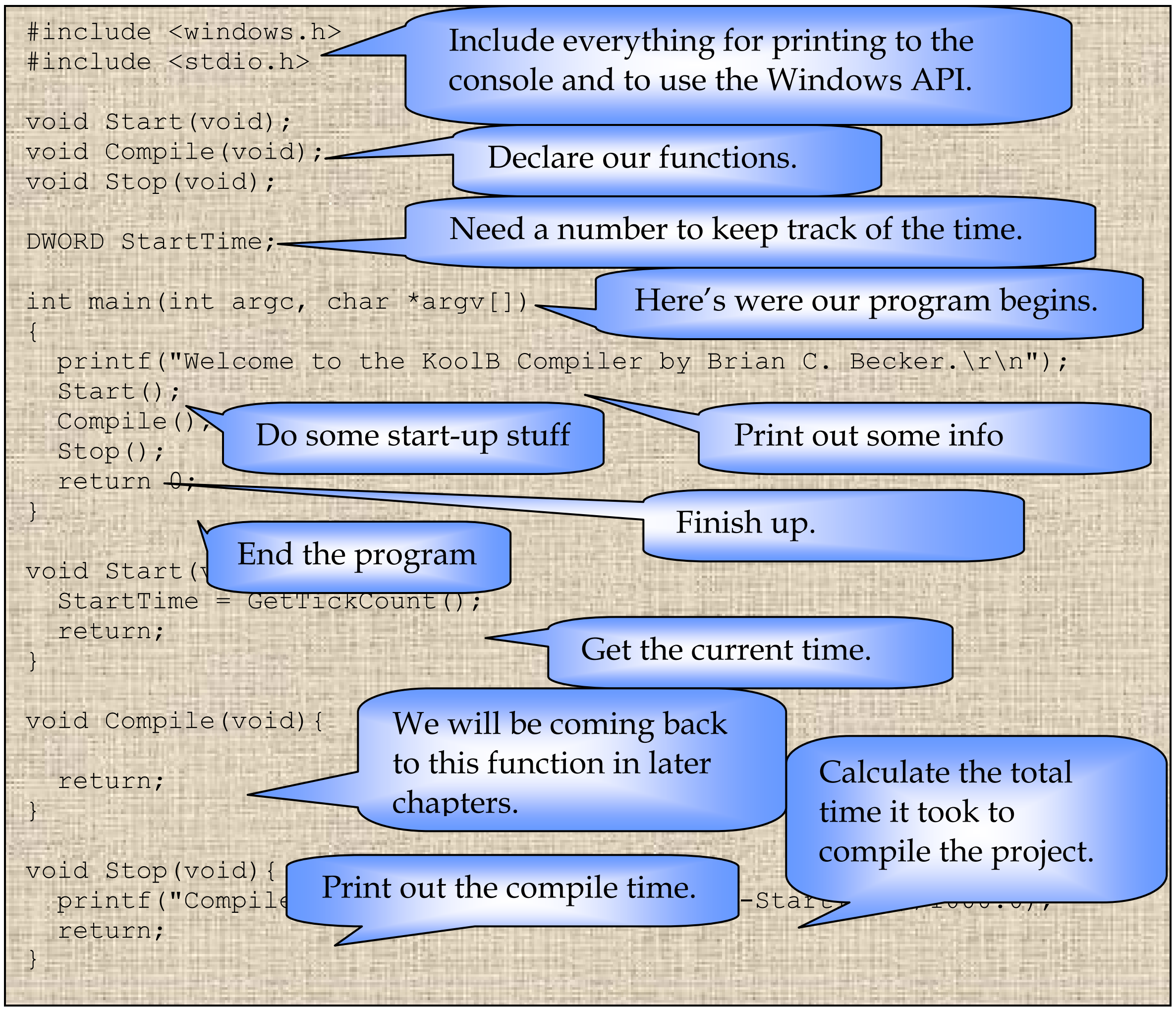
The function Start gets the time when the program started, then Compile will do its hard work (we'll get to that in a later chapter), and then Stop will figure out how long the compile process took. If you look carefully, you will notice that the printf function has a different number of parameters. And look! What is that weird symbol %f doing right after the Compile Time? Mr. C++ has made the printf function very useful to us, to embed a number into a string and then display it, we just type %f were we want to put the number and then add the number at the end of the parameters. Mrs. Rapid-Q's style would be:

```
Sub Start
  StartTime = GetTickCount()
End Sub

Sub Compile

End Sub

Sub Stop
  Print "Compile Time:\t"; (GetTickCount()-StartTime)/1000.0
End Sub
```

There's only one last thing we have to do. Have you noticed it? KoolB doesn't call the functions Start, Compile, and Stop from its main function! We need to correct that, so right after the first printf statement and before the return 0; in the main function insert:

```
Start();
Compile();
Stop();
```

Here is the entire code with some comments on the side:



```
#include <windows.h>
#include <stdio.h>

void Start(void);
void Compile(void);
void Stop(void);

DWORD StartTime;

int main(int argc, char *argv[])
{
   printf("Welcome to the KoolB Compiler by Brian C. Becker.\r\n");
   Start();
   Compile();
   Stop();
   return 0;
}

void Start(v
   StartTime = GetTickCount();
   return;
}

void Compile(void){

   return;
}

void Stop(void){
   printf("Compile                                -Start
   return;
}
```

Include everything for printing to the console and to use the Windows API.
Declare our functions.
Need a number to keep track of the time.
Here’s were our program begins.
Do some start-up stuff
Print out some info
Finish up.
End the program
Get the current time.
We will be coming back to this function in later chapters.
Calculate the total time it took to compile the project.
Print out the compile time.

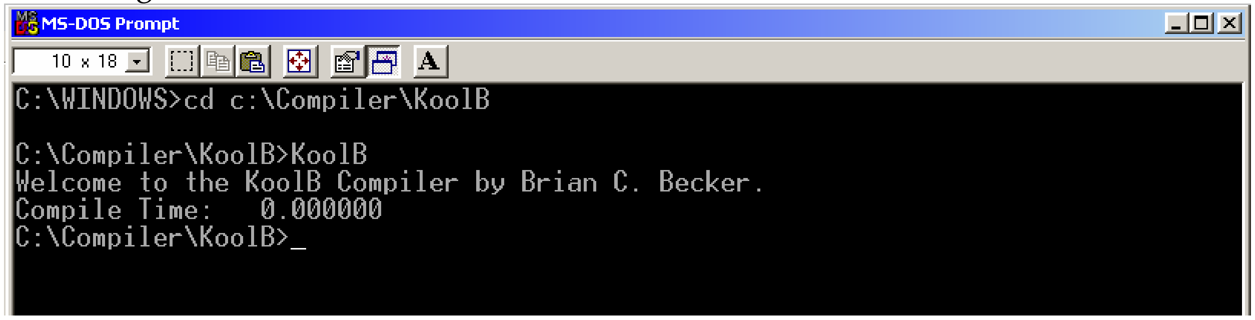
Click Execute -> Rebuild All to compile it and there you have it – the start of KoolB, our BASIC compiler. Granted it is pretty limited, but hey, we will get there eventually. For those of you who need to see how Mrs. Rapid-Q would do it, here it is:

```
$Include "Rapidq.inc"

Declare Function GetTickCount Lib "kernel32.dll" alias "GetTickCount"_
 () As Long

Declare Sub Start
Declare Sub Compile
Declare Sub Stop
Declare Function main (CommandCount As Integer, Command As String) _
                 As Integer

Dim StartTime As Long

CALL main(CommandCount, Command$(1))

Function main (CommandCount As Integer, Command As String) As Integer
  print "Welcome to the KoolB Compiler by Brian C. Becker."
  Start
  Compile
  Stop
  Result = 0      'or Main = 0
End Function

Sub Start
  StartTime = GetTickCount()
End Sub

Sub Compile

End Sub

Sub Stop
  $EscapeChars ON  'to translate \t to a tab
  Print "Compile Time:\t"; (GetTickCount()-StartTime)/1000.0
End Sub
```

If you compare these two programs together, you will see some startling differences. Obviously, Mrs. Rapid-Q is a lot easier than Mr. C++. However, look at the file sizes: Mrs. Rapid-Q is 163 KB and Mr. C++ is 5 KB. Wow! That is a really large difference, and to top that, all Mrs. Rapid-Q's programs are compressed, meaning there is an even bigger difference. In this example, there is no speed comparison since the example doesn't do much of anything that can be timed (i.e. no heavy work), but if there was, Mrs. Rapid-Q would lose. For example, just looping a million times takes .857 seconds with Mrs. Rapid-Q and only .005 seconds with Mr. C++, which means that Mr. C++ is over 150 times faster in this circumstance! But simplicity does have its advantages for new programmers and experienced programmers who just want a simple program.

Testing Our Work:

Since we haven't done much except play around with Mr. C++, there isn't that much to test, but how about we at least run it to see how it works? To do so, click on the Start Button and the navigate to Programs. We are looking for a program named 'MS-DOS Prompt' or 'Command Prompt.' Microsoft has a habit of re-organizing things, so you might find it under the sub-folder 'Accessories.' Anyhow, if you cannot find it, let me know along with your version of Windows and I will try to help you. Open it up and you get something that looks entirely too much like DOS (Urr, I don't like DOS). Type in 'cd c:\Compiler\KoolB' and hit ENTER. That changes the current directory to our compiler's folder. Now type in 'KoolB' and press ENTER.' You should get something like this:



```
C:\WINDOWS>cd c:\Compiler\KoolB

C:\Compiler\KoolB>KoolB
Welcome to the KoolB Compiler by Brian C. Becker.
Compile Time:    0.000000
C:\Compiler\KoolB>_
```

Gee, that was a fast compile! I wish other compilers did it that fast. Type 'Exit' to exit the console and then save all the work by going File -> Save All in Dev-C++. Then exit (File -> Exit).

Conclusion:

Well, that concludes Chapter 2 – Meet Mr. C++ (by the way, I didn't mean half the derogatory things I said about C++; I really do like the language, although I wish it was easier to learn). You should now have an idea of what C++ is and how it works. I would suggest for those that might still be sort of sketchy on C++ that you check out some of these tutorials:

1. http://www.rit.edu/~jpw9607/ctut/: This tutorial requires some programming skills from any other language, especially BASIC or Pascal,

since this will only explain working with the language C/C++, not programming in general.

2. http://www.intap.net/~drw/cpp/index.htm: The purpose of this tutorial is to give a good understanding of the programming language C++ to any person that wants it.
3. http://www.cprogramming.com/tutorial.html: This tutorial is for everyone; if you've never programmed before, or if you have extensive experience programming in other languages and want to expand into C++. It is for everyone who wants the feeling of accomplishment from a working program.

We also started to create our compiler, KoolB and added the some small features to it like the credits and timing. In the next chapter, we will be adding some modules to our compiler to handle different tasks like reading the BASIC input file. We won't be writing the whole modules; we will just start to build the framework for KoolB.