Projects in Code::Blocks

May 25, 2013

Contents

1 The Virtual PC lab 2

2 Trying Out the Compiler Suite 3

3 A More Elaborate Program 5

4 The Debugger 6

The activities listed here presume you have access to the GNU g++ compiler with the Code::Blocks IDE. You will find these installed in the CS Dept. PCs. You can also find instructions on installing these free programs on the CS 333 Library page.

1 The Virtual PC lab

I'm going to assume, for the moment that you are not in a CS Dept lab, but are working from home or sitting in one of the general University (OCCS) computing labs.

Open a browser and go to the CS Dept home page. Look for the link to the "Virtual Computing Lab". Follow the instructions given there under "Connecting" to open a *Remote Desktop Connection* to a virtual lab machine.

Log in with your account info as per the earlier Account Setup lab.

- If the login window comes up with something like "vclab-#*yourLoginName*", click on "Switch User" and then "Other User" so that you can log in as "csnet*yourLoginName*".

You should now be looking at a window with an image of a complete Windows Vista desktop. This is a "Virtual PC" with all of the software that the CS Dept normally provides on its physical PCs in its own labs. However, this virtual machine is designed for the kind of remote access that you are now enjoying.

- One thing you might consider doing almost immediately is clicking on the taskbar "start" button and selecting "Windows Security". One of the options given is to change your password. If you are running a newly created account with the initial default password, you want to change this *immediately* because it's quite easy for others to guess your default.

- The CS Dept's Windows network and Unix networks share a lot of information, but passwords are not among the shared info. So, changing your Windows network password does not affect your Unix password, and vice-versa. To change your Unix password, see changePasswordChanging Your Password in the CS252 website.

A few general cautions about working with virtual PCs.

- Don't try to store anything on the C: drive or on the desktop of this machine. If you lose your connection and then try to reconnect, you are likely to get a different virtual machine from the bank of machines available. Also, the drives on these virtual machines are reset periodically. Either way, you won't be able to recover that lost file.

  - Your "My Documents" area is safe, however, because it gets saved to a network store when you log out and reloaded when you log back in. However, if you have a lot of material stored there, it *significantly* slows down your login and logout.
  - You should have a Z: drive on the machine that actually maps to your account space on the Unix network. That's a safe place to keep your work.
  - When you used Remote Desktop Connection to connect to this virtual machine, there was probably an "Options" button. Under the "Local Resources" tab, you have the option of mapping the disk drives of your local PC (the one you are sitting at) onto the virtual machine. This makes for a very convenient (if somewhat slow) way to share files between the two.

- When you are done working, be sure to actually log out from the virtual PC. Don't simply disconnect. That ties up a virtual machine (for at least a while) and slows down the virtual machine server for everyone else.

Now if you *are* on campus and are in one of the CS Dept labs, you can, if you wish, log out of the virtual PC and complete the remainder of the assignment working directly on that PC instead of via a virtual machine. But even then, keep in mind that the virtual PCs are available to you from anywhere with an Internet connection.

2 Trying Out the Compiler Suite

The "official" compiler for this course is the Free Software Foundation's *g++* compiler. Without question, this is the most widely ported C++ compiler, available for almost all computing platforms. However, the compiler itself is just that - a compiler. Using it effectively generally requires some surrounding software, called an Integrated Development Environment (IDE), to provide code editors, project management support, and debugging support.

Luckily, there are several good-quality, free IDEs as well. In this course we will use *Code::Blocks*. The course Library page includes instructions on getting this, together with the g++ compiler, if you want to install it on your own PC. However, you will find it already in place on the CS Dept virtual PCs.

Run Code::Blocks (you should find this in the Start button menu on the CS lab PCs).[1]

[1] If you have installed it yourself, then the first time you run it, it may try to locate compilers it can work with. If so, select "GNU GCC Compiler". Click "Set as default", then "OK". It should find your new installation of g++.)

Let's start out with a very simple project. Select "File->New->Project... Select "Console Application" and click "Go". Move through the Wizard, choosing C++ as the language. Give the project the title "FirstProject", click on the "..." button to choose a place in which to save your work.

- Keep the earlier discussion about safe places to store your work in mind as you choose this.

Accept the rest of the defaults until the wizard closes.

On the left you can now see your project structure. If you open the Sources folder, you can see that it has already created a basic main.cpp file for you. Double-click this to open it in the editor.

Let's make some changes before compiling this. Change the program to look like this:

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

using namespace std;

int main()
{
  string greeting = "Hello world!";
  cout <<
  return 0;
}
```

Save this and click the Build button (the blue gear) to attempt compilation. You should see some compilation errors due to the incomplete statement.[2]

Now put the cursor at the end of the "cout" line and type "greet" (without the quotes). Pause a moment and notice that a box pops up suggesting a possible completion for this variable name. Hit Tab to accept this suggestion. Now finish the program like this:

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

[2] If you see an error message that says you are using "an invalid compiler", then Code::Blocks did not actually locate the compiler properly. From the "Settings" menu, select "Compiler and Debugger", On the tab "Toolchain excutables", Use the "..." button for the "Compiler's Installation Directory" to point it to the directory where you installed MinGW.

```
using namespace std;

int main()
{
  string greeting = "Hello world!";
  cout << greeting << endl;
  return 0;
}
```

Save and again click the Build button. This time things should succeed.
Click the Run button (the blue triangle) to execute your program.
When you are satisfied, close that project.

3 A More Elaborate Program

Create a new project (still a Console Application) named "tempConvert". Edit the `main.cpp` file to look like this:

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

using namespace std;

// Farenheit to Centigrade converter
int main()
{
  double f, c;
  cout << "Enter a temperature measured in degrees Farenheit: "
       << flush;
  cin >> f; // reads f from the keyboard
  c = (5.0 / 9.0) * (f - 32.0);
  cout << "In Centigrade, that would be " << c << " degrees."
       << endl;
```


```
  return 0;
}
```

Try compiling and running this. If you have made any typos, you may get error messages when you compile. That's OK, just make the fixes, save, and re-compile. (In fact, if you *didn't* get any error messages, why not make a few deliberate mistakes just to get familiar with how the IDE behaves?)

4 The Debugger

Now, let's suppose that this behavior was not what we wanted, but we weren't sure why. We might want to run the program in a debugger. First, let's set a breakpoint. In your temperature conversion project, find the line that starts: `cin >>`... Click just to the right of the line number. The small red dot indicates that a breakpoint has been set at this line.

From the "Debug" menu, select "Start". The program will start running, but will pause at the line where you set the breakpoint. You can now look at the state of the running program in some detail. For example, in the "Debug" menu, select "Debugging Windows", then "Watches". The window that pops up allows you to look at the value of variable. Click on the plus sign to open up the Local variables list.[3] You can use the various debugger buttons (the ones with the dark red curved arrows) to step through the code, a line at a time.

Try using the "Next" button, a step at a time. By the way, don't be surprised if `c` and/or `f` appear to contain garbage at first. That's typical for an uninitialized variables. In fact, had I not wanted to improve the odds of your seeing that, I would have written the code like this:

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

using namespace std;

// Farenheit to Centigrade converter
int main()
{
  cout << "Enter a temperature measured in degrees Farenheit: "
```

[3] Unfortunately, I find that the current version has trouble displaying strings.

```
        << flush;
    double f;
    cin >> f; // reads f from the keyboard
    double c = (5.0 / 9.0) * (f - 32.0);
    cout << "In Centigrade, that would be " << c << " degrees."
        << endl;
    return 0;
}
```

to minimize the time during which either variable is left uninitialized.

Try making those changes, run the debugger again, and single-step through. Do you see any difference in how the variables behave?