



Eclipse Project CDT (C/C++) Plugin Tutorial ^{1, 2}

By Brian Lee

Department of Computer Science, University of Manitoba, Winnipeg, Manitoba, Canada

Last revised: February 20, 2004

Overview:

This tutorial describes the installation of the Eclipse Project CDT (C/C++ Development Tools) plugin on the Windows platform (running under Linux is mentioned at the end of this tutorial). It describes the entire installation process in detail and is intended for developers who are familiar with the Eclipse environment and would like to develop C/C++ applications in this environment. As a result, it has been assumed that Eclipse has already been installed and that the developer has a basic understanding of how to use Eclipse. To learn more about the Eclipse Project, please visit:

<http://www.eclipse.org>

In addition, this tutorial is especially useful for students who are uncomfortable writing C/C++ applications in a UNIX or Linux console environment. By using the Eclipse CDT plugin, students can develop C/C++ applications within the Eclipse IDE which provides an intuitive GUI for development. As well, applications developed this way require minimal effort in porting to a UNIX or Linux environment.

Requirements:

This tutorial has been written in a Windows XP Professional environment, satisfying the following requirements:

- Java 2 SDK v1.4.1
- Eclipse v2.1
- MinGW v3.0.0-1
- CDT v1.1.0 GA

¹ This work was funded by an IBM Eclipse Innovation Grant.

² © Brian Lee and David Scuse

For instructions on how to install Eclipse, please refer to the following document:

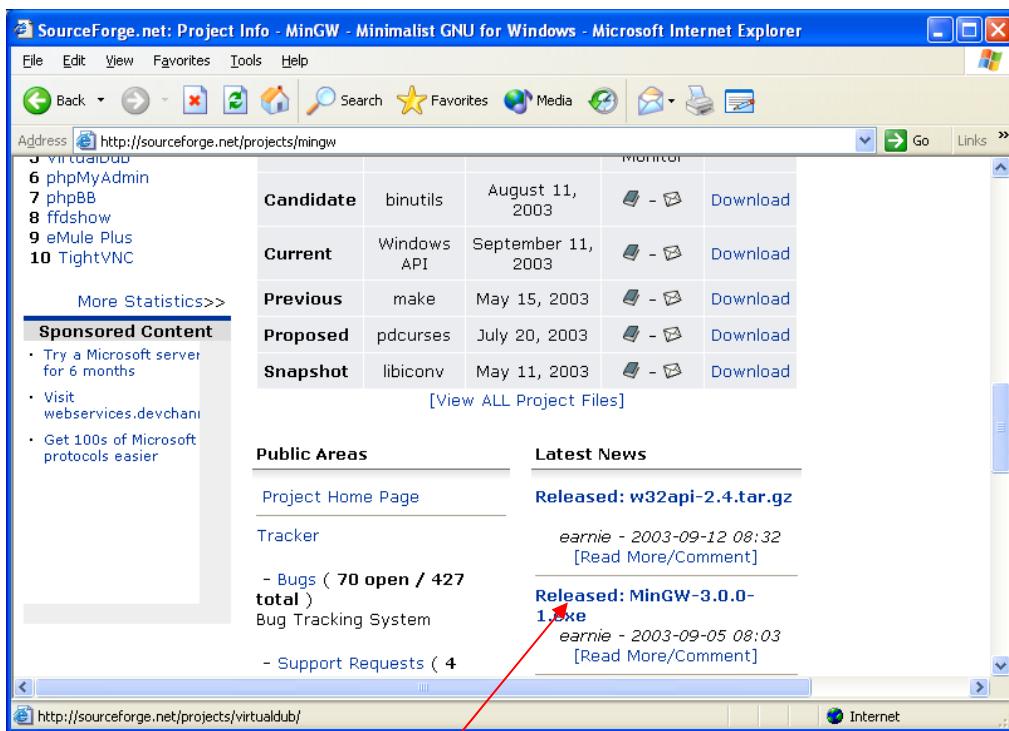
<http://www.cs.umanitoba.ca/~eclipse/1-Install.pdf>

Links to downloading MinGW and the Eclipse CDT plugin will be mentioned later in this document.

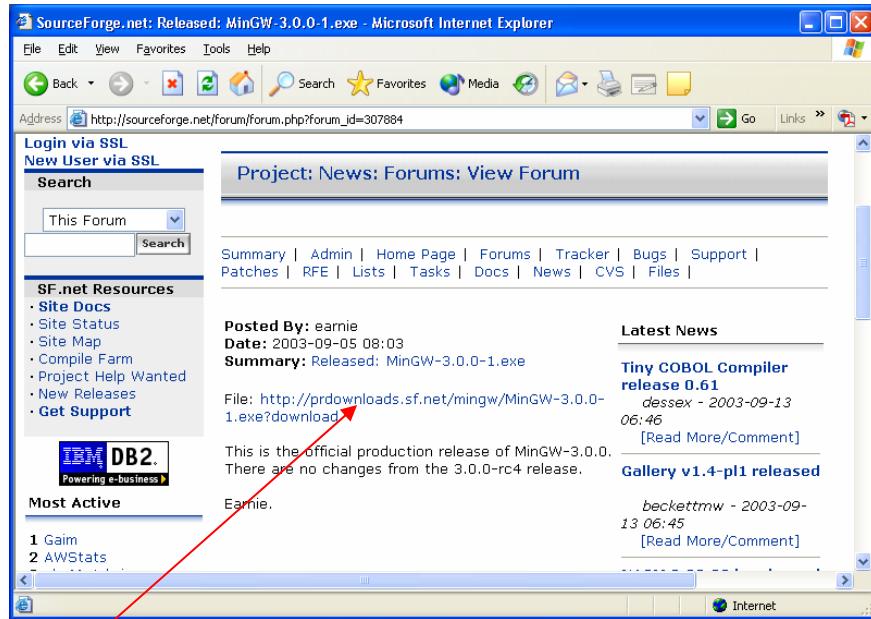
Installing and Setting Up MinGW:

In order to be able to compile C and C++ source code using Eclipse, it is necessary to install a C/C++ compiler for Windows. There are several C/C++ compilers available for Windows but MinGW has been selected for this tutorial due to its easy installation and configuration. MinGW is based on GNU toolsets and provides the header files and libraries needed for C/C++ development. At the time of writing this tutorial, the latest version of MinGW that has been tested to work properly with the Eclipse CDT plugin is version 3.0.0-1. To download MinGW, please visit the following site:

<http://sourceforge.net/projects/mingw/>



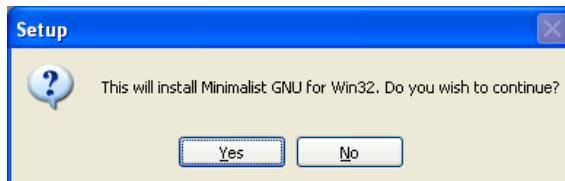
Click on the **Released: MinGW-3.0.0-1.exe** link.



Click on the link to download the file to your system.

Once the download is complete, locate the file on your system and double-click on the file to begin the installation of MinGW.

1.) Click **Yes** to continue with the installation of MinGW.

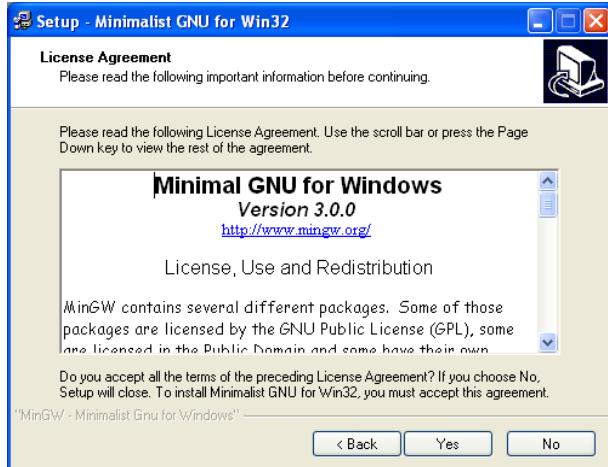


Click **Next** to continue.



2.) Click **Yes** to accept the License Agreement.

Note: Unlike other ports of GCC to Windows, the runtime libraries are not distributed using GNU's General Public License (GPL). You, therefore, do not have to distribute your source code with your programs unless, of course, you use a GPL library in your programs.³



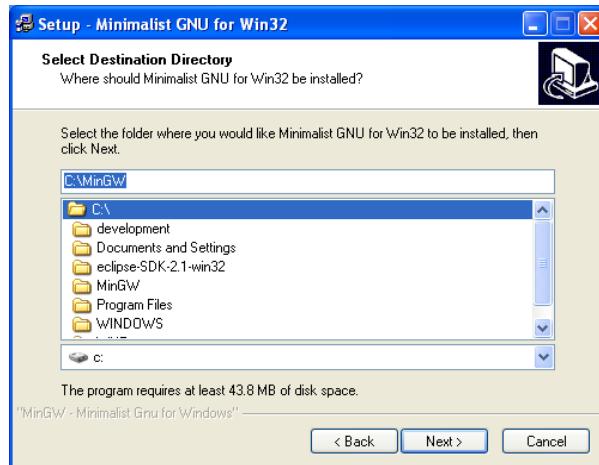
3.) Click **Next** to continue.



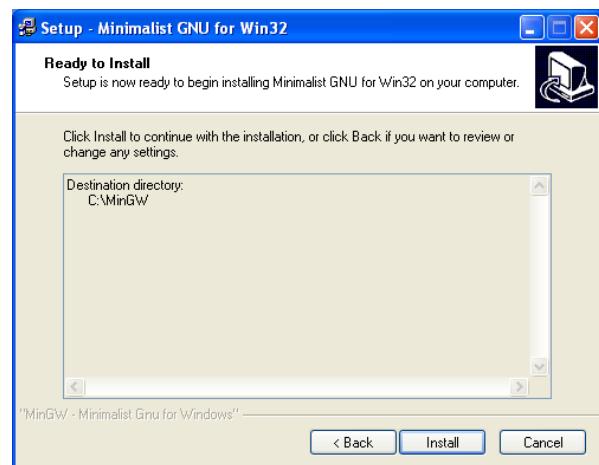
4.) Install MinGW to the following directory: **C:\MinGW**

Note: If you install to another directory, remember the directory you install to as it will be used later in this tutorial to complete the installation of the plugin.

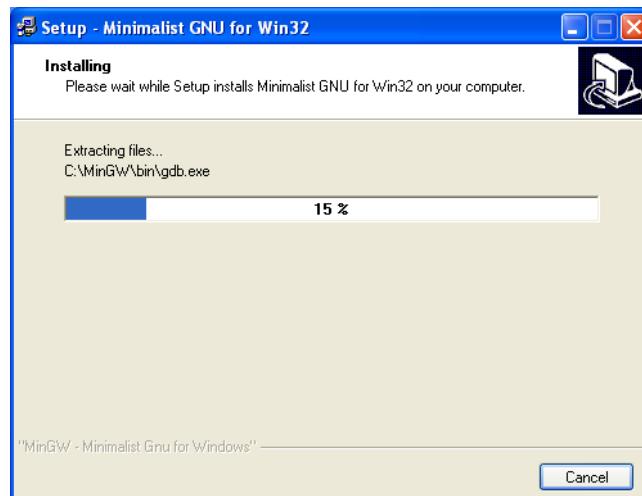
³ Taken from: <http://www.mingw.org/mingwfaq.shtml>



5.) Click **Install** to continue.



The following screen shows the progress of the installation. MinGW installs a number of GNU GCC and binutils projects, including GDB which is very useful in debugging your C/C++ applications.

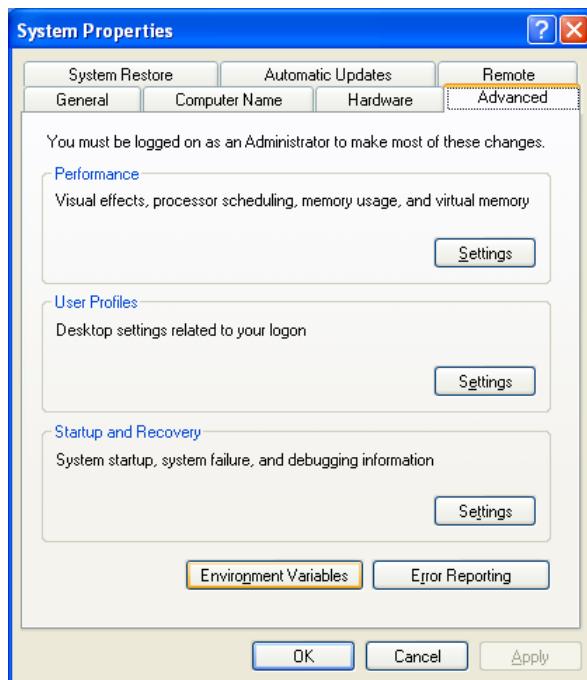


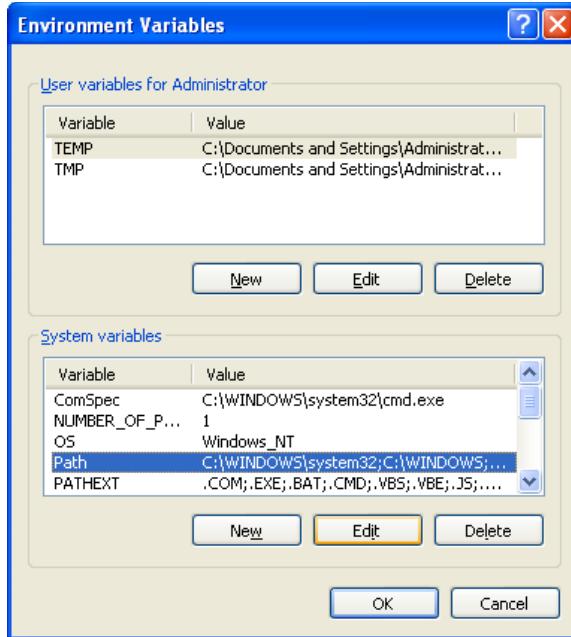
6.) Click **Finish** to exit the setup of MinGW.



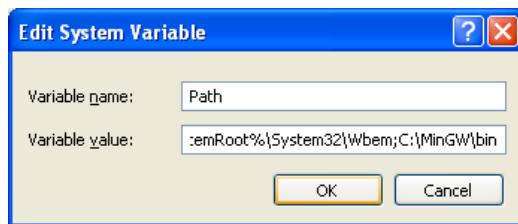
7.) In order for your system to recognize the MinGW executables from any directory without having to type the full path of the command, the PATH variable needs to be modified:

- a. Click on Start → Control Panel.
- b. Double-click on System.
- c. Click on the **Advanced** tab and then **Environment Variables**.





- d. Select **Path** under the System Variables section and click on **Edit**.



- e. Assuming that MinGW has been installed to C:\MinGW, **add** the following to the **end** of the Variable value:

;C:\MinGW\bin

Note: The semi-colon is used to separate different directories for the PATH variable and **must** precede the pathname C:\MinGW\bin. **Also, please make sure that there are no spaces between the semi-colon and the path name!**

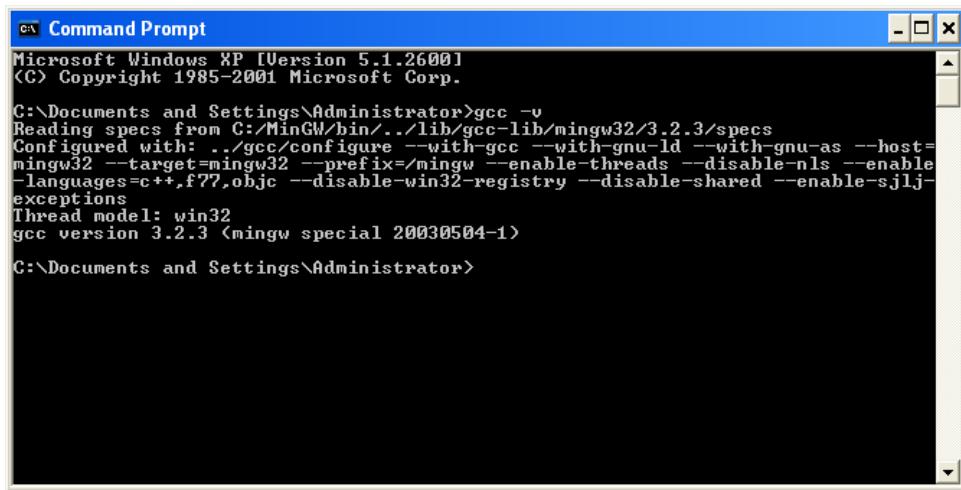
- f. Click OK, then OK, and OK again to apply the changes.

- 8.) To verify that the installation of MinGW is successful and that the PATH variable has been changed correctly, perform the following steps:

- Click on Start → Run ...
- Type **cmd** into the dialog box and press OK.
- Type the following into the command prompt and then press ENTER:

gcc -v

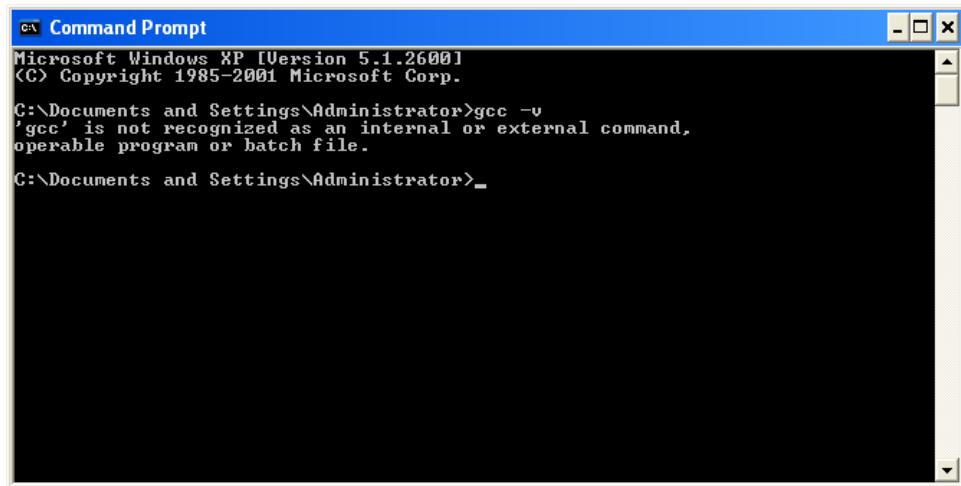
- d. If the output of the above command is similar to the screen shown below, then you have successfully installed MinGW and changed the PATH variable:



Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

```
C:\Documents and Settings\Administrator>gcc -v
Reading specs from C:/MinGW/bin/../lib/gcc-lib/mingw32/3.2.3/specs
Configured with: ../gcc/configure --with-gcc --with-gnu-ld --with-gnu-as --host=
mingw32 --target=mingw32 --prefix=/mingw --enable-threads --disable-nls --enable-
-languages=c++,f77,objc --disable-win32-registry --disable-shared --enable-sjlj-
exceptions
Thread model: win32
gcc version 3.2.3 (mingw special 20030504-1)
C:\Documents and Settings\Administrator>
```

- e. If the output of the above command is similar to the screen shown below, this means that the PATH variable has not been changed successfully. Follow the instructions carefully on the previous pages to change the PATH variable:



Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

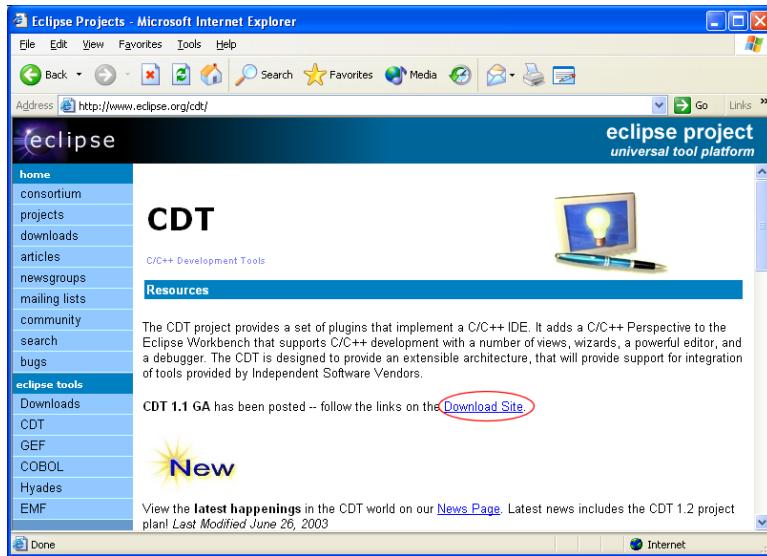
```
C:\Documents and Settings\Administrator>gcc -v
'gcc' is not recognized as an internal or external command,
operable program or batch file.
C:\Documents and Settings\Administrator>_
```

- 9.) After verifying that MinGW has been installed and that the PATH variable has been changed successfully, you are ready to install the Eclipse CDT plugin.

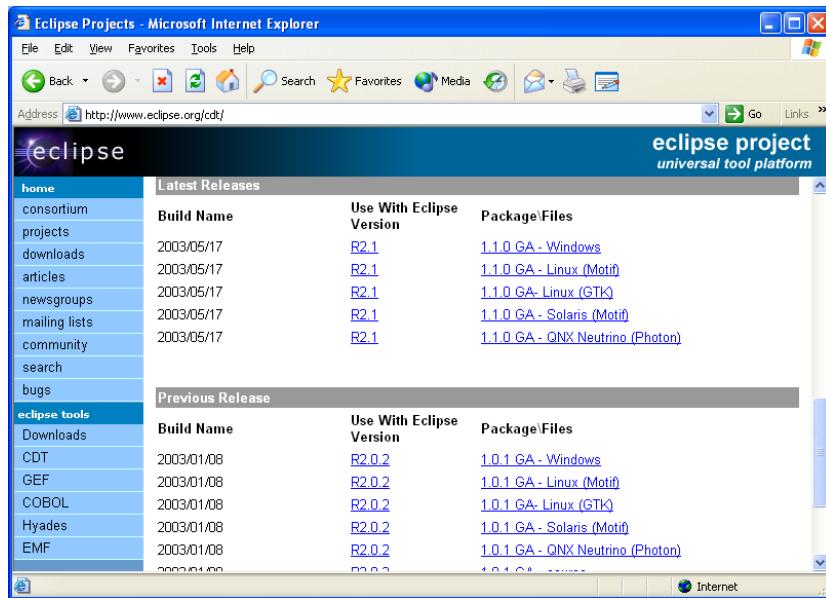
Installing the Eclipse CDT Plugin:

Now that your system is ready to compile C/C++ applications, download the Eclipse CDT plugin at the following site: <http://www.eclipse.org/cdt/>

1.) Click on the **Download Site** link:



2.) Scroll down and click on the **1.1.0 GA – Windows** link:



3.) Save the file to your system.

4.) Once the download is complete, unzip the contents of the file:

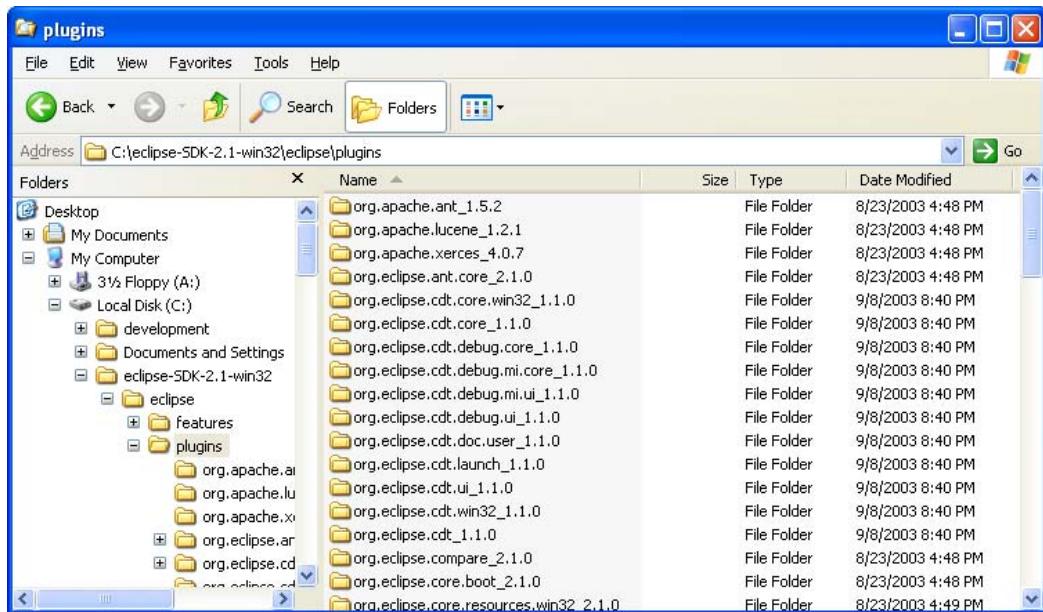
org.eclipse.cdt-win32_1.1.0.bin.dist.zip

to the directory where Eclipse.exe is located. For example:

C:\eclipse-SDK-2.1-win32\eclipse



Note: After unzipping the file into the Eclipse directory, the contents of the plugin directory should have the following **org.eclipse.cdt.*** folders:



- 5.) To verify that the Eclipse CDT plugin has been installed successfully, run Eclipse.exe from the install directory as you normally would. Then perform the following steps:
 - a. Click on Help → About Eclipse Platform.
 - b. Click on the Plug-In Details button.
 - c. If your screen looks similar to the following then the Eclipse CDT plugin has been successfully installed:

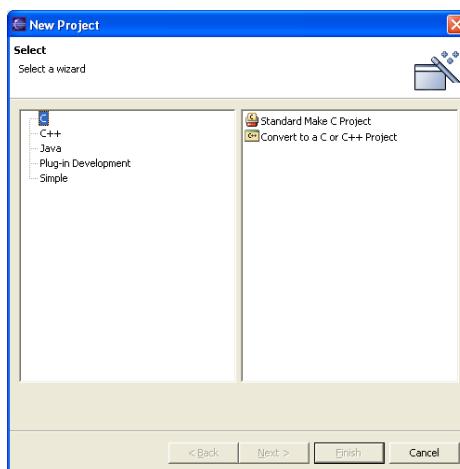
Provider	Plug-in Name	Version	Plug-in Id
Eclipse.org	Ant Build Tool Core	2.1.0	org.eclipse.a...
Eclipse.org	Apache Ant	1.5.2	org.apache.ant
Eclipse.org	Apache Lucene	1.2.1	org.apache.l...
Eclipse.org	C/C++ Development Tools Core	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools Debugger UI	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools Debug Model	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools GDB/MI CDI D...	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools GDB/MI CDI D...	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools Launching Sup...	1.1.0	org.eclipse.c...
Eclipse.org	C/C++ Development Tools UI	1.1.0	org.eclipse.c...
Eclipse.org	Compare Support	2.1.0	org.eclipse.c...
Eclipse.org	Core Boot	2.1.0	org.eclipse.c...
Eclipse.org	Core Resource Management	2.1.0	org.eclipse.c...
Eclipse.org	Core Runtime	2.1.0	org.eclipse.c...
Eclipse.org	CVS SSH Core	2.1.0	org.eclipse.t...
Eclipse.org	CVS Team Provider Core	2.1.0	org.eclipse.t...
Eclipse.org	CVS Team Provider UI	2.1.0	org.eclipse.t...
Eclipse.org	Debug Core	2.1.0	org.eclipse.d...
Eclipse.org	Debug UI	2.1.0	org.eclipse.d...
Eclipse.org	Default Text Editor	2.1.0	org.eclipse.ui...
Eclipse.org	Eclipse C/C++ Development Tools	1.1.0	org.eclipse.cdt
Eclipse.org	Eclipse C/C++ Development Tools (Windo...	1.1.0	org.eclipse.c...
Eclipse.org	Eclipse C/C++ Development User Guide	1.1.0	org.eclipse.c...

- d. If your screen is missing the **C/C++ Development Tools** Plug-In components, then verify that the file org.eclipse.cdt-win32_1.1.0.bin.dist.zip has been unzipped to the **eclipse\plugins** folder as noted on the previous page.

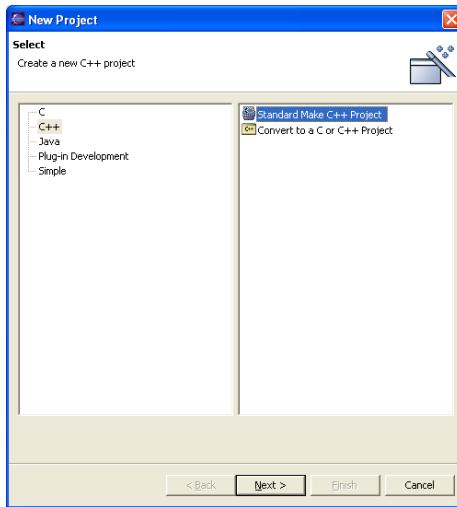
Creating a C++ Project using the Eclipse CDT Plugin:

This section describes how to create a C/C++ project in Eclipse using the CDT plugin. It has been assumed at this point, that the system is ready to compile C/C++ applications using tools from MinGW and that the Eclipse CDT plugin has been successfully installed.

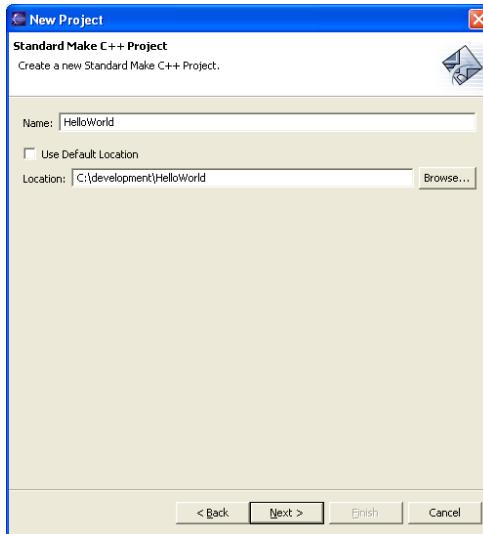
- 1.) Run Eclipse from the install directory as you normally would. Once Eclipse is open, create a new C/C++ project by going to File → New → Project.



- 2.) For this tutorial, you will be creating a C++ Project. When the dialog box pops up, click on **C++** and select **Standard Make C++ Project**. Click **Next** to continue.



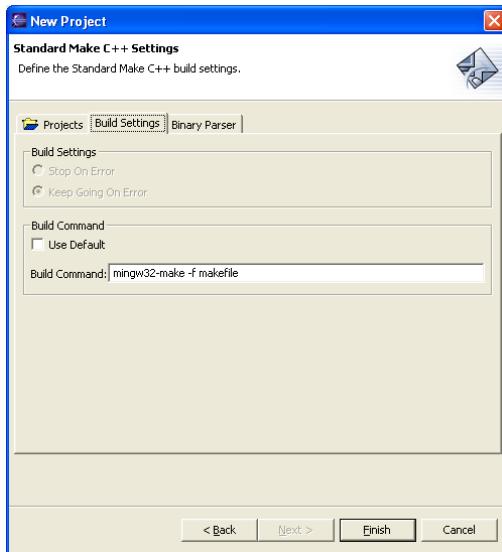
- 3.) Enter **HelloWorld** for the name of the project and either use the default workspace or select an alternate location. In the example below, the project is stored in **C:\development\HelloWorld**. Click **Next** to continue.



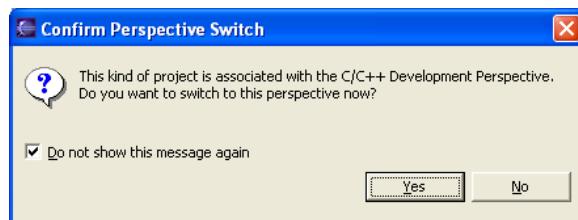
- 4.) Click on the **Build Settings** tab and under the **Build Command** section:
- Uncheck the Use Default checkbox.
 - Enter the following in the Build Command field (note that mingw32-make is one string, with no embedded blanks; also, the complete path to the mingw32-make command may be specified):

```
mingw32-make -f makefile
```

- Click **Finish** to continue.



5.) When the following dialog box pops up, click **Yes** to continue.



6.) You should now be in the C/C++ Development Perspective, where you can see the HelloWorld project you just created. Now that your project has been created:

- Click on File → New → File...
- Enter **main.cpp** as the name of the file.
- Click **Finish** to continue.

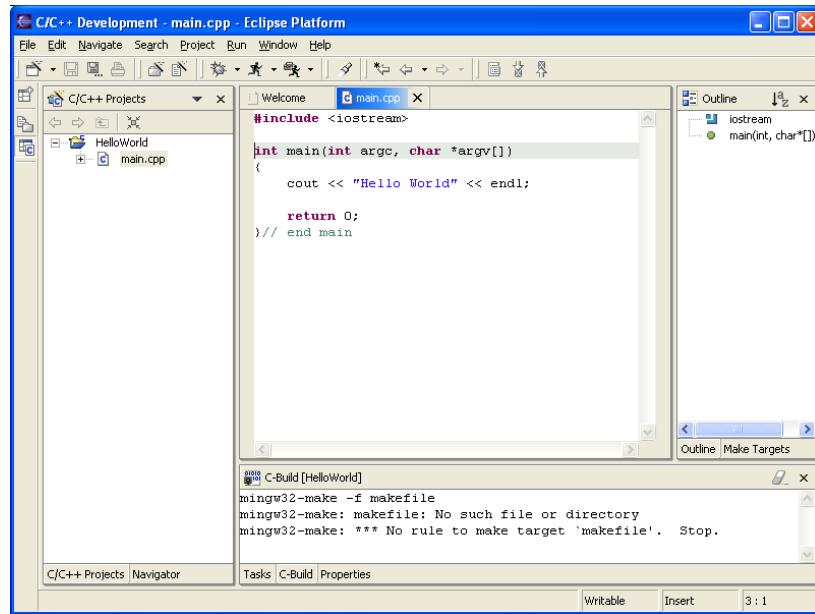


7.) Once the file is created, copy and paste the following code into **main.cpp**:

```
#include <iostream>

int main(int argc, char *argv[])
{
    cout << "Hello World" << endl;
    return 0;
}// end main
```

Press **Ctrl + S** to save the file. Note the following error messages in the C-Build window below; these error messages will eventually disappear:



8.) To create the makefile:

- Click on File → New → File...
- Enter **makefile** as the name of the file.
- Click **Finish** to continue.

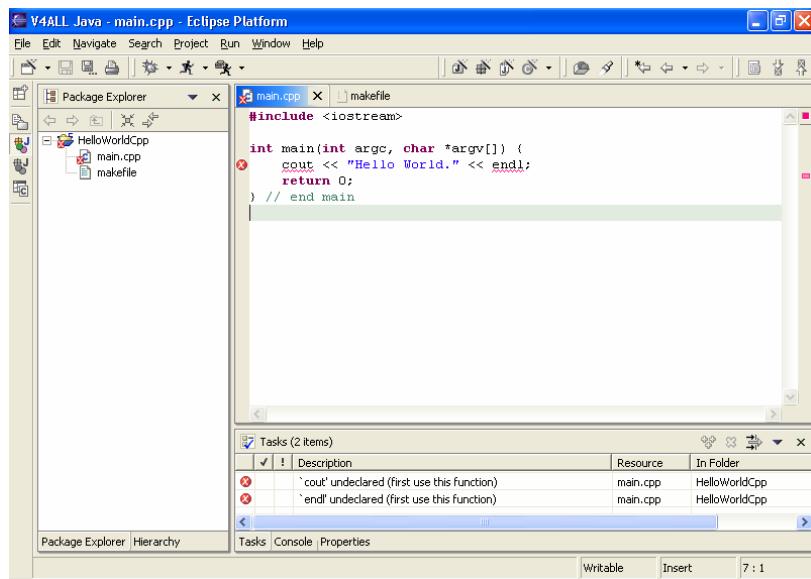


9.) Once the file is created, copy and paste the following into the **makefile**:

```
main : main.o  
        g++ -o main main.o -L C:/MinGW/lib/gcc-lib/mingw32/3.2.3/ -lstdc++  
  
main.o : main.cpp  
        g++ -ggdb -c main.cpp  
  
all :  
        ${MAKE} main  
  
clean :  
        -del main.o
```

The above makefile is simply a generic makefile that could easily be expanded to include a number of different source files. To learn more about the structure of makefiles and how to write your own, check out the many resources available on the Web.

After typing the above into the makefile, press **Ctrl + S** to save the file. Now, some different error messages are displayed in the **Tasks** window below:

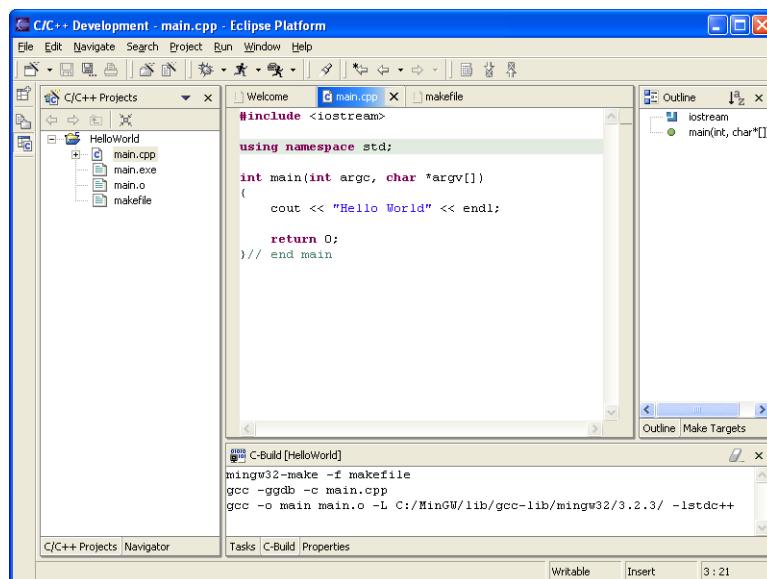


- 10.) If you look in the C/C++ Project window, you will notice a red 'X' beside main.cpp. The error messages indicate that 'cout' and 'endl' are both undeclared. But the source code in main.cpp looks fine! **This is a very common problem and developers tend to get very frustrated in their attempts to solve this problem.** The fix is actually quite simple:

- a. In main.cpp, add the following line right `#include <iostream>`:

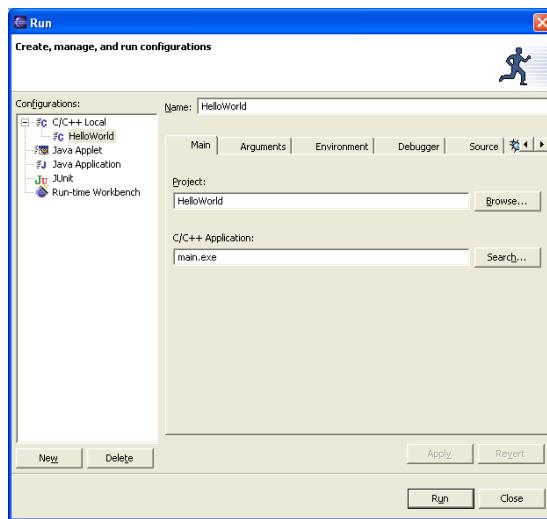
using namespace std;

- b. Press **Ctrl + S** and your application should compile successfully as shown in the screen below. Note that **main.exe** has now been created.

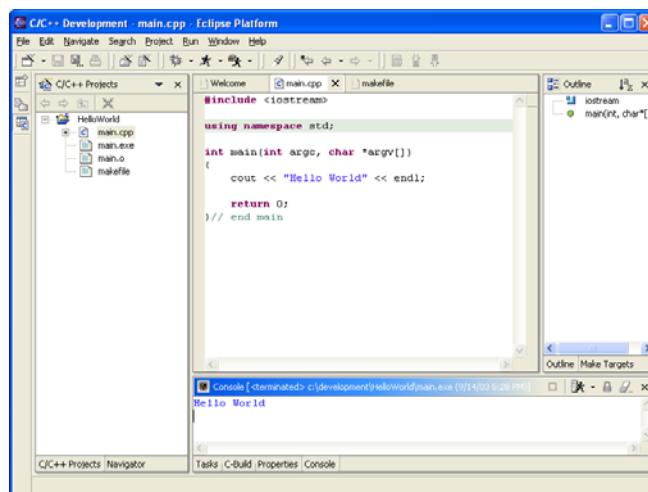


11.) We are almost done! Now to run your application:

- a. Click on **Run → Run...**
- b. In the Configurations window, click on **C/C++ Local**, and then the **New** button.
- c. In the Name field, type **HelloWorldCpp**.
- d. In the C/C++ Application field, type **main.exe**.
- e. The settings on the other tabs will be sufficient for the purpose of this tutorial.
- f. Then click on the **Apply** button. Your screen should look like the following:



- g. Now click the **Run** button to run your application. You should get the following output in the console window:



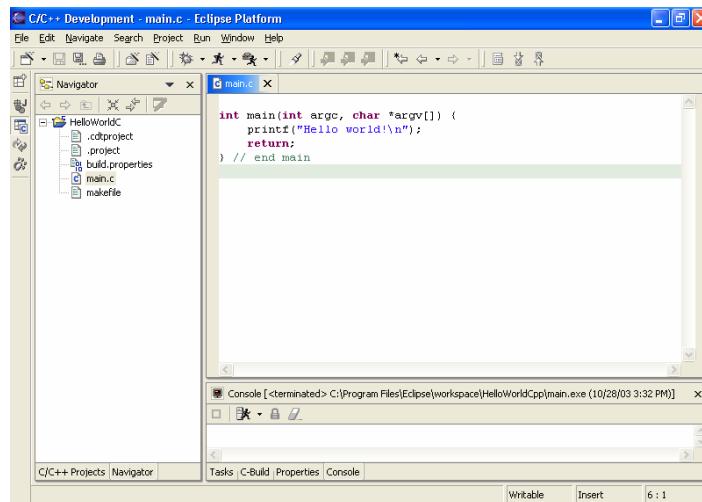
- h. Congratulations! You have just created your first C++ application in Eclipse using the Eclipse CDT plugin.

Creating a C Project:

Creating a C project is almost identical to creating a C++ project. First, create a new C project called HelloWorldC. Set the Build Command property to:

```
mingw32-make -f makefile (the same as for the C++ project).
```

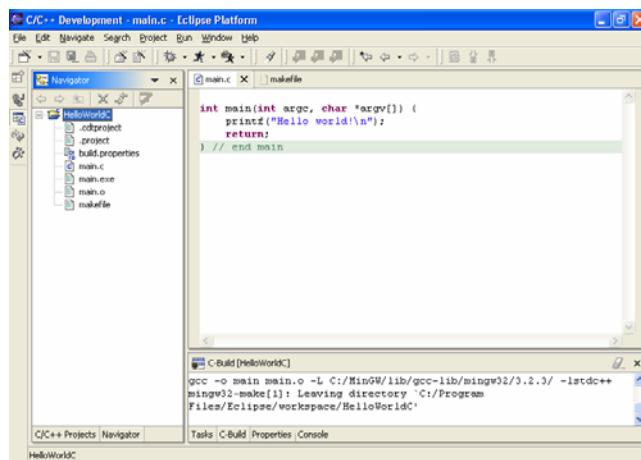
Create main.c with the following contents:



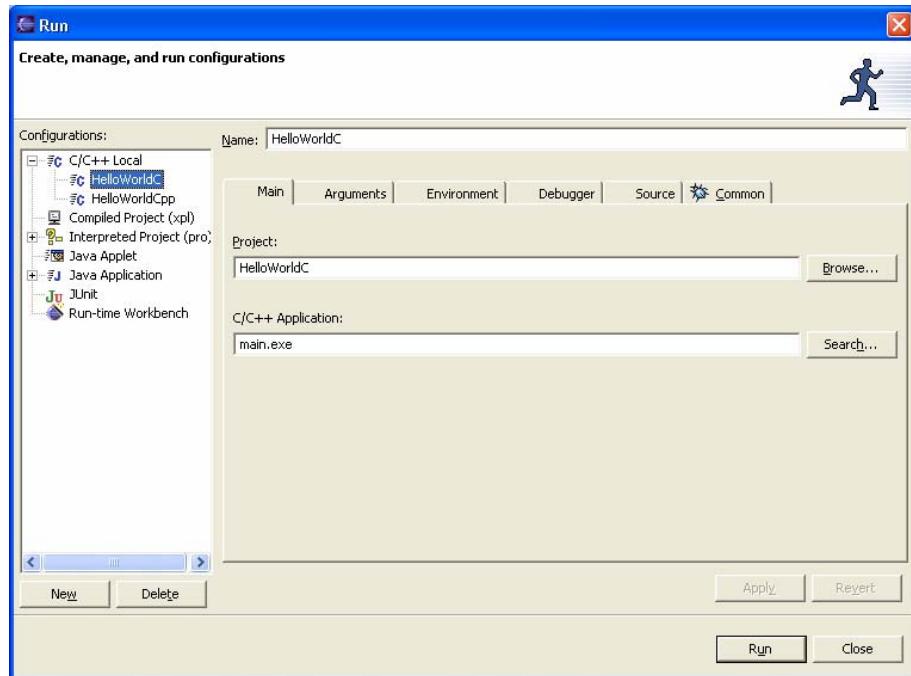
Create the makefile with the following contents:

```
main : main.o
    g++ -o main main.o -L C:/MinGW/lib/gcc-lib/mingw32/3.2.3/ -lstdc++
main.o : main.c
    g++ -ggdb -c main.c
all :
    ${MAKE} main
clean :
    -del main.o
```

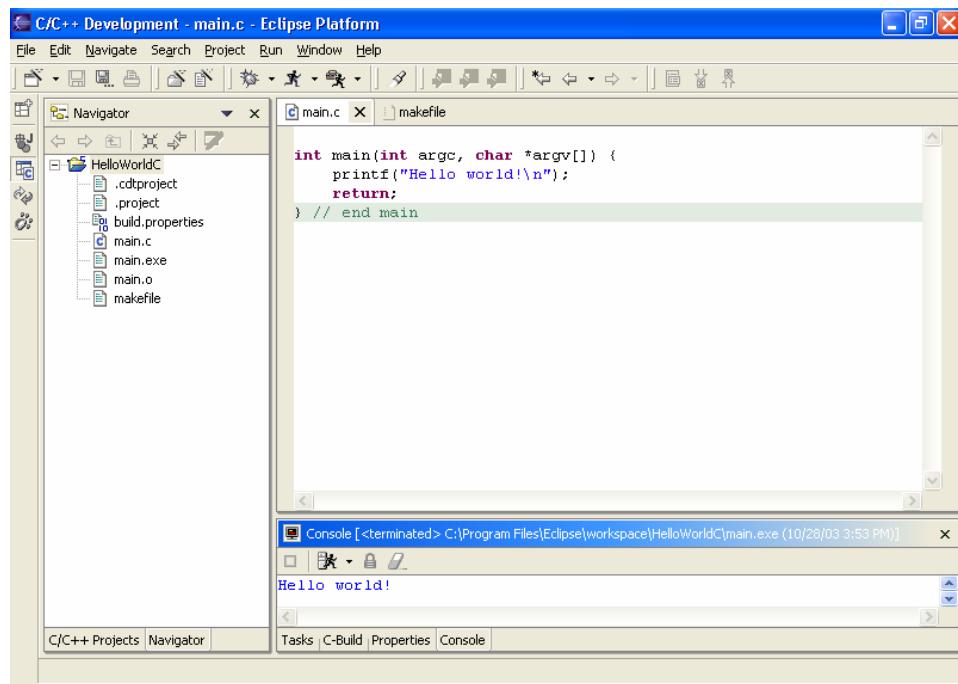
Compile the project causing main.exe to be generated.



Create the associated Run command.



and then Run the project:



Once again, Hello world should appear in the console.

Running CDT under Linux:

We did one simple test of CDT under Linux using the following configuration:

- Debian Linux with kernel 2.4
- Java 2 SDK v1.4.2
- Eclipse v2.1.1
- MinGW v3.3.2
- CDT v1.1.0

Once CDT was unzipped into the Eclipse directory, the default settings for “Standard Make C++ project” worked correctly since GCC was already installed. If you can compile a C++ program from a console, the default settings should be sufficient for compiling within Eclipse.

Basic Console Input/Output:

It is quite common that C/C++ programs require user input via the console. The following example shows how to enter input into the console of a program written using the CDT plugin:

- 1.) Create a new C++ project called **HelloWorld** which contains the following files:
 - main.cpp
 - makefile
- 2.) Copy and paste the following code into **main.cpp** and save the file:

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

using namespace std;

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

    cout << "Please enter your name: " << endl;
    cin >> name;

    cout << "Hello " + name << endl;

    return 0;
}// end main
```

3.) Copy and paste the following code into **makefile** and save the file:

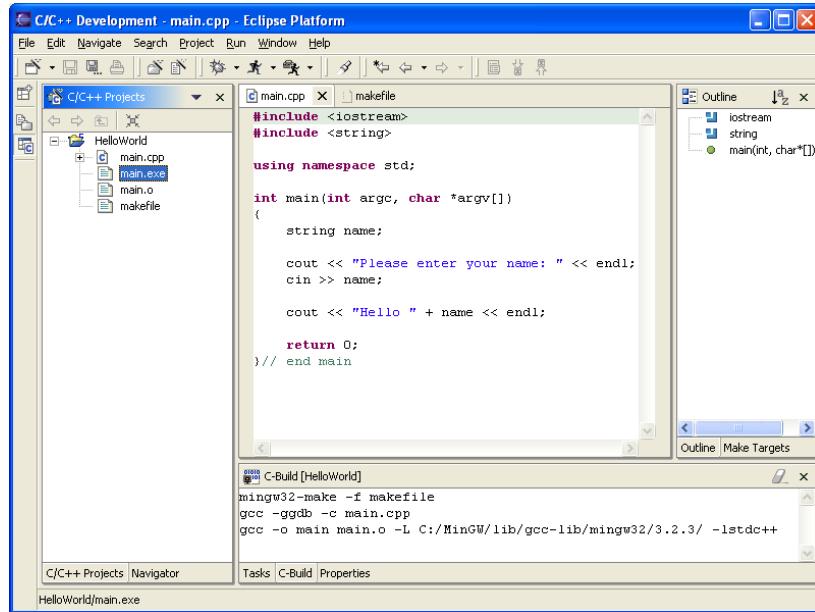
```
main : main.o
    g++ -o main main.o -L C:/MinGW/lib/gcc-lib/mingw32/3.2.3/ -
lstdc++

main.o : main.cpp
    g++ -ggdb -c main.cpp

all :
    ${MAKE} main

clean :
    -del main.o
```

4.) There should be no errors in the C-Build window and the project should compile itself successfully, generating an executable file called **main.exe**:



If you get a “**Build Error**” message in the C-Build window, verify that your build settings are setup correctly.

The screenshot shows the Eclipse C/C++ Development Platform interface. The main window displays a C/C++ project named "HelloWorld". In the code editor, the file "main.cpp" contains the following code:

```

#include <iostream>
#include <string>

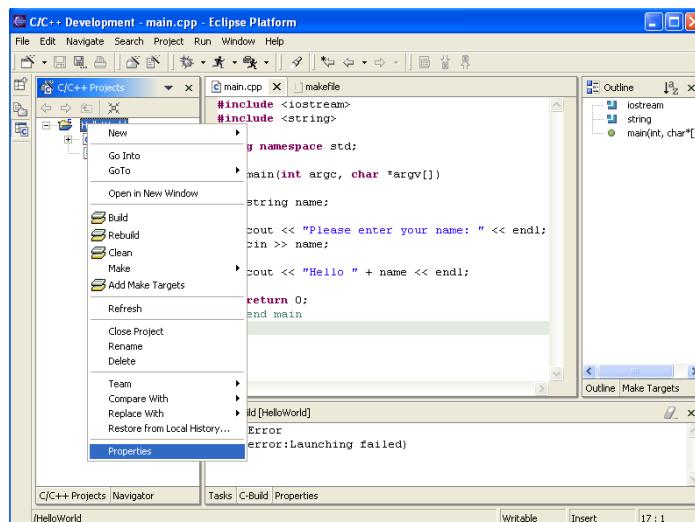
using namespace std;

int main(int argc, char *argv[])
{
    string name;
    cout << "Please enter your name: " << endl;
    cin >> name;
    cout << "Hello " + name << endl;
    return 0;
} // end main

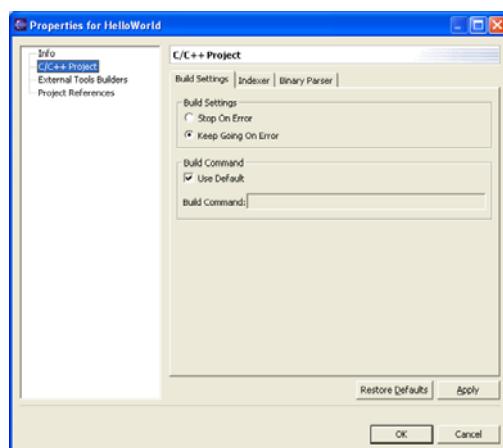
```

In the bottom right corner of the code editor, there is a small error icon. To the right of the code editor, the "Outline" view shows the symbols "iostream", "string", and "main(int, char*)". Below the code editor, the "Tasks" and "C-Build" tabs are visible. A status bar at the bottom indicates "Writable" and "Insert" modes, along with the character count "17 : 1".

- Right-click on the HelloWorld project and click properties.**

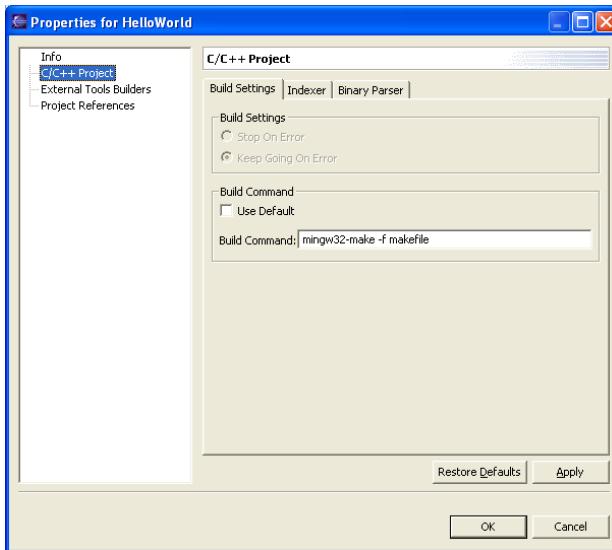


- Select C/C++ Project on the left hand side.**

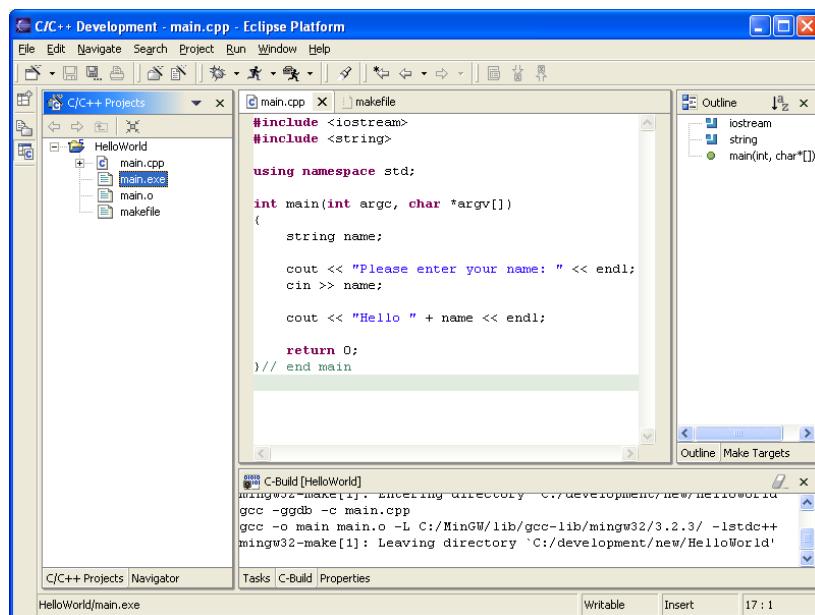


- c. Uncheck the **Use Default** checkbox under the Build Command section and type the following into the Build Command box:

```
mingw32-make -f makefile
```

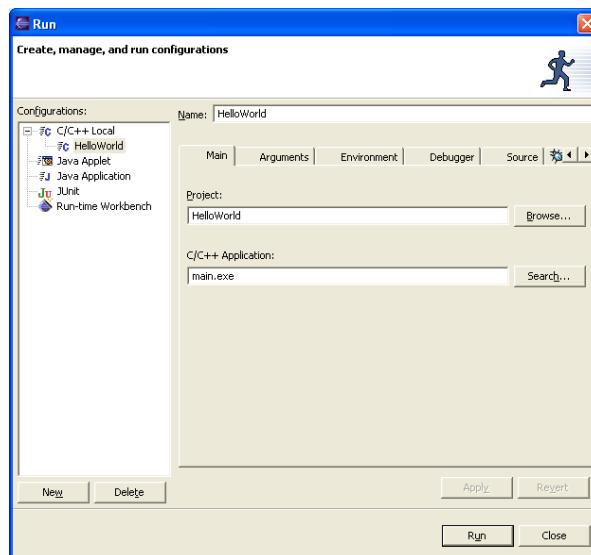


- d. Click **OK**.
- e. Right-click on the HelloWorld project and click on **Rebuild**. The project should compile itself successfully as shown in the screen below. Note that **main.exe** has now been created.

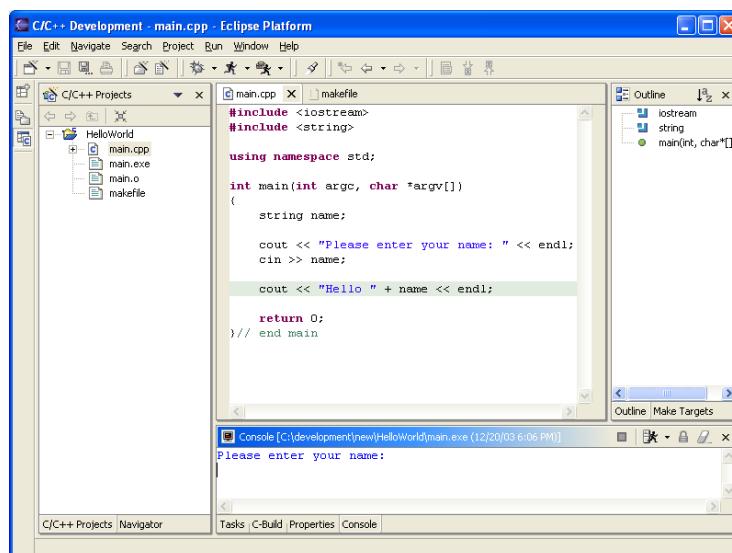


5.) To run the application:

- a. Click on **Run → Run...**
- b. In the Configurations window, click on **C/C++ Local**, and then the **New** button.
- c. In the Name field, type **HelloWorld**.
- d. In the C/C++ Application field, type **main.exe**.
- e. The settings on the other tabs will be sufficient for the purpose of this example.
- f. Then click on the **Apply** button. Your screen should look like the following:



- g. Now click the **Run** button to run your application. You should get the following output in the console window:



- h. Type your name and then press Enter. The next line should read “Hello [name]” and then the program terminates.

Note: If you are having problems entering text into the console, make sure that your cursor is blinking at the end of all text in the console. To position the cursor at the end, simply press **Ctrl + End** while the cursor is blinking in the console. After performing this step, you should be able to enter text into the console.

Basic File Input/Output and Command Line Arguments:

Occasionally, a question on an assignment may require you to create a program that reads in command line arguments, such as the names of input and output files. The following example shows how to provide command line arguments to applications at the beginning of execution:

- 1.) Create a new C++ project called **HelloWorld** which contains the following files:

- main.cpp
- names.txt
- makefile

(For detailed instructions on how to do this, please refer to Part I of this tutorial under the section “**Creating a Project using the Eclipse CDT Plugin**”.

- 2.) Copy and paste the following code into **main.cpp** and save the file:

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

using namespace std;

// Constants
#define BUFFER_SIZE      50
#define ARRAY_SIZE       20

// Global variables
int numElements = 0;

void sort(char *elements[]) {
    bool swap = true;
    char *temp = NULL;
    while(swap) {
        swap = false;
        for (int i = 0; i < (numElements - 1); i++) {
            if (strcmp(elements[i], elements[i + 1]) > 0) {
```

```

        temp = elements[i];
        elements[i] = elements[i + 1];
        elements[i + 1] = temp;
        swap = true;
    }
}
}

int main(int argc, char *argv[])
{
    char buffer[BUFFER_SIZE];
    char *listOfNames[ARRAY_SIZE];
    string inputFileName;
    string outputFileName;

    ifstream inputFile;
    ofstream outputFile;

    if(argc != 3) {
        cout << "Error: Please enter input and output files ";
        cout << "as command line arguments !" << endl;
    }
    else {
        inputFileName = argv[1];
        outputFileName = argv[2];

        inputFile.open(inputFileName.c_str());
        outputFile.open(outputFileName.c_str());

        // Read names from input file and store into array
        while(!inputFile.eof() && numElements < (ARRAY_SIZE - 1)) {
            inputFile.getline(buffer, BUFFER_SIZE);
            char *p = new char[strlen(buffer) + 1];
            strcpy(p, buffer);
            listOfNames[numElements] = p;
            numElements++;
        }
        // Sort elements in array
        sort(listOfNames);

        // Print elements in array to output file
        for(int i = 0; i < numElements; i++) {
            outputFile << listOfNames[i] << endl;
        }
        inputFile.close();
        outputFile.close();
    }
    return 0;
} // end main

```

The previous block of code simply reads in two command line arguments – the first argument is the name of the input file and the second argument is the name of the output file. The program will read from the input file, store the names into a vector,

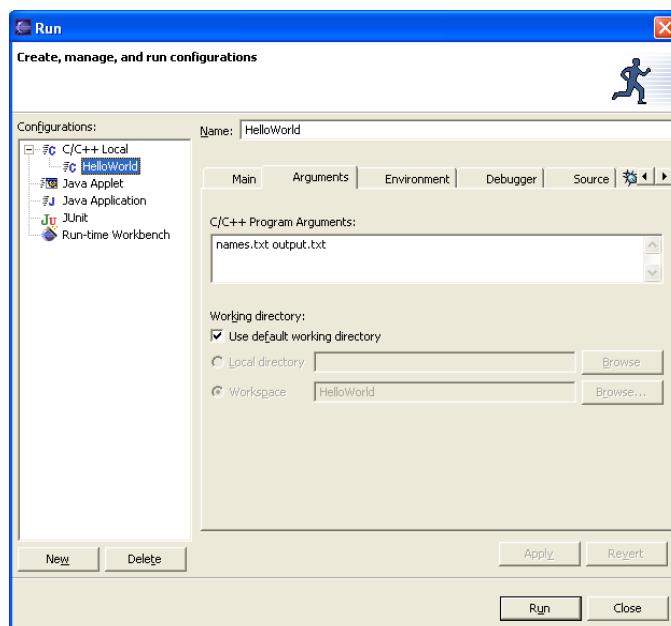
and sort these elements alphabetically. Lastly, it will write the ordered list of names to the output file.

- 3.) Copy and paste the following names into **names.txt** and save the file:

```
Brian  
Michelle  
Jeff  
Lori  
Ainsley  
Jimmy  
Bob  
Lorisa  
John
```

- 4.) Now, to specify command line arguments when running your application:

- a. Click on **Run → Run...**
- b. In the Configurations window, click on **C/C++ Local**, and then the **New** button.
- c. In the Name field, type **HelloWorld**.
- d. In the C/C++ Application field, type **main.exe**.
- e. Click the Arguments tab.
- f. In the C/C++ Program Arguments textbox, enter the following:
names.txt output.txt
- g. The settings on the other tabs will be sufficient for the purpose of this example.
- h. Then click on the **Apply** button. Your screen should look like the following:



- i. Now click the **Run** button to run your application. You should see a new file called **output.txt** in the directory view of your project:

The screenshot shows the Eclipse C/C++ Development Platform interface. The main window title is "C/C++ Development - main.cpp - Eclipse Platform". The left sidebar shows a project tree for "HelloWorld" containing files main.cpp, main.exe, main.o, makefile, names.txt, and output.txt. The central editor area displays the content of main.cpp, which includes headers iostream, fstream, and string, defines constants BUFFER_SIZE and ARRAY_SIZE, and implements a bubble sort function for characters. To the right of the editor is an "Outline" view showing symbols like iostream, fstream, string, and main(). Below the editor is a "C-Build [HelloWorld]" view showing the command: mingw32-make -f makefile and gcc -o main main.o -L C:/MinGW/lib/gcc-lib/mingw32/3.2.3/ -lstdc++. At the bottom, tabs for C/C++ Proj..., Navigator, Tasks, C-Build, Properties, and Console are visible, along with status bars for Writable, Insert, and 1:1.

- j. Double-click on **output.txt** to open the file and you should see the list of names that was in **input.txt**, sorted alphabetically in ascending order:

The screenshot shows the Eclipse C/C++ Development Platform interface again. The main window title is "C/C++ Development - output.txt - Eclipse Platform". The left sidebar shows the same project tree as before. The central editor area now displays the contents of output.txt, which contains a sorted list of names: Ainsley, Bob, Brian, Jimmy, John, Lori, Lorisa, Michelle, and Ryan. The "Outline" view to the right shows the message "An outline is not available." Below the editor is the same "C-Build [HelloWorld]" view as in the previous screenshot. The bottom tabs and status bars are identical to the first screenshot.

Debugging Code:

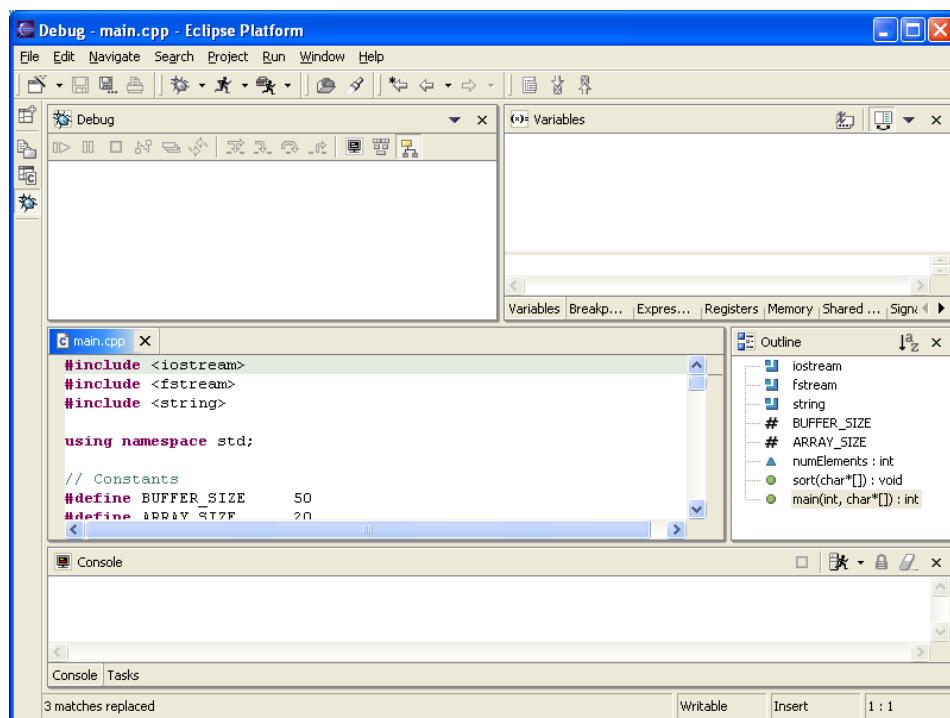
Debugging is an essential skill that every programmer needs to master, whether to fix a problem in an application or even to step through the application slowly to see how it works. Fortunately debugging C/C++ applications on the Eclipse platform using the CDT plugin is very straightforward. Continuing from the example described in the previous section, let's take a look at how to set breakpoints and debug the application:

1.) Switch to the Debug perspective:

- Click on **Window → Open Perspective → Other...**
- Select the **Debug** perspective and click OK.

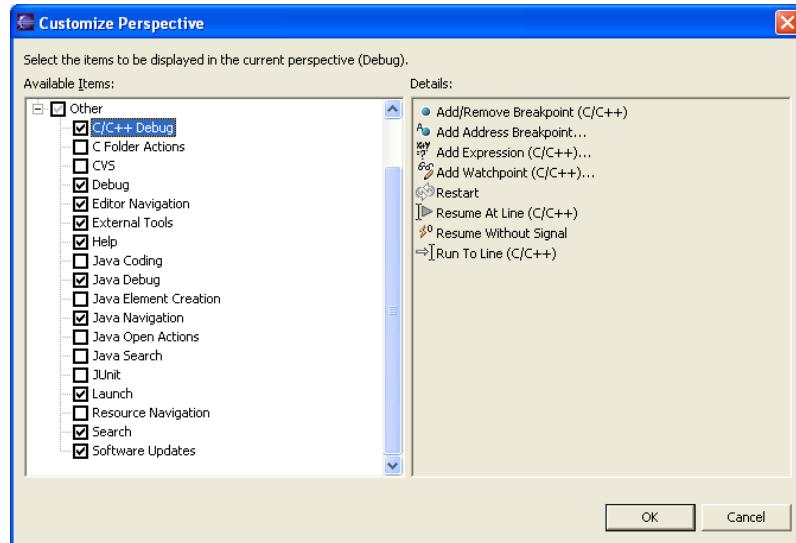


c. Your perspective should resemble something like the following:



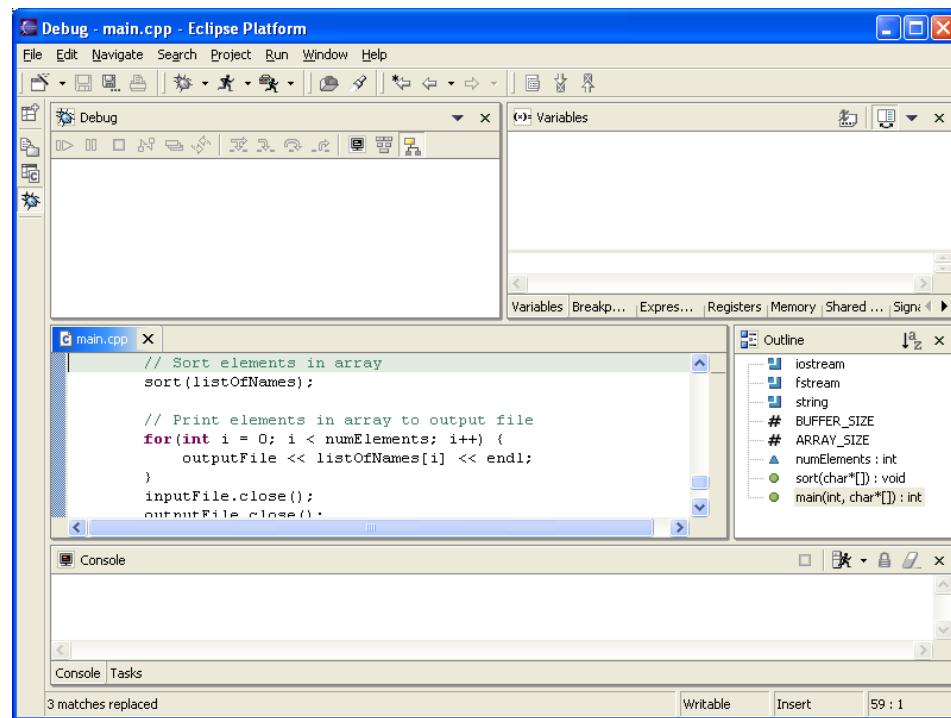
2.) In order to have access to the basic C/C++ debugging functionality, perform the following steps:

- Click on **Window** → **Customize Perspective...**
- Expand the **Other** branch, check the **C/C++ Debug** box, and click **OK**.

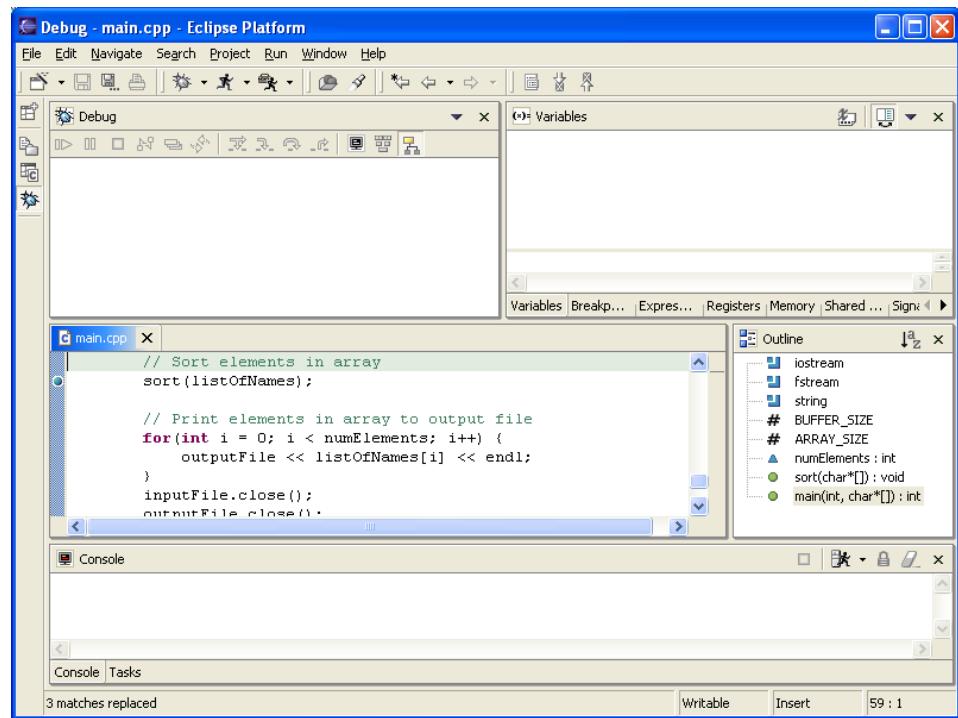
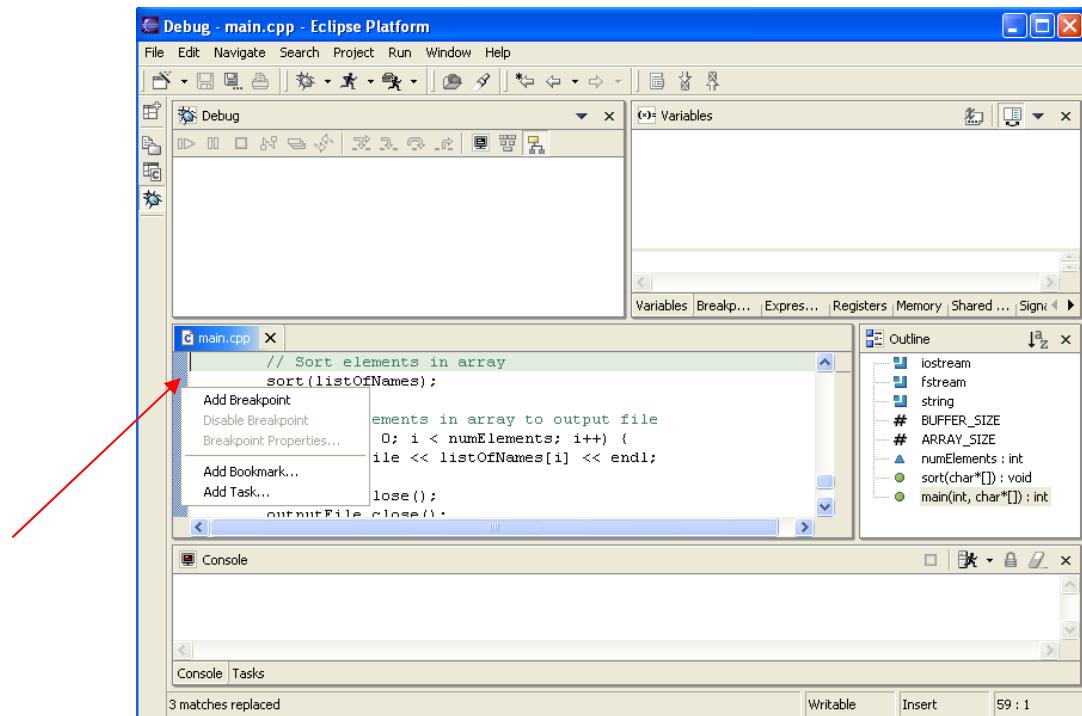


3.) Let's set our first breakpoint:

- Open **main.cpp** and scroll down to just before the vector of names is sorted:

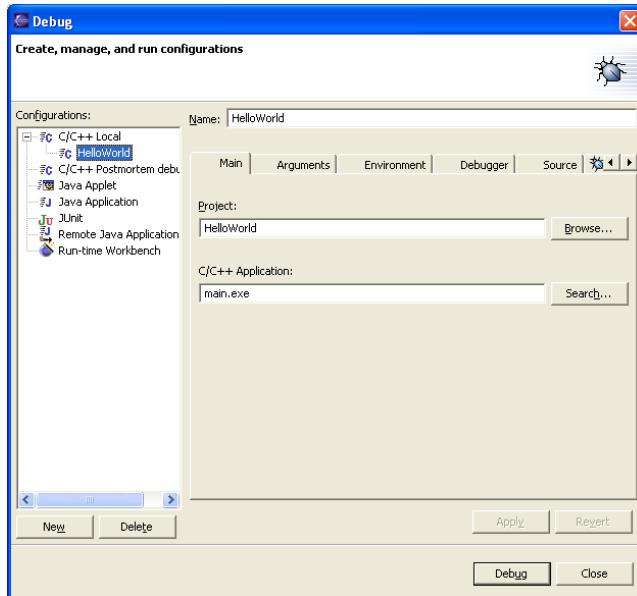


b. Right-click in the margin and then click **Add Breakpoint**.

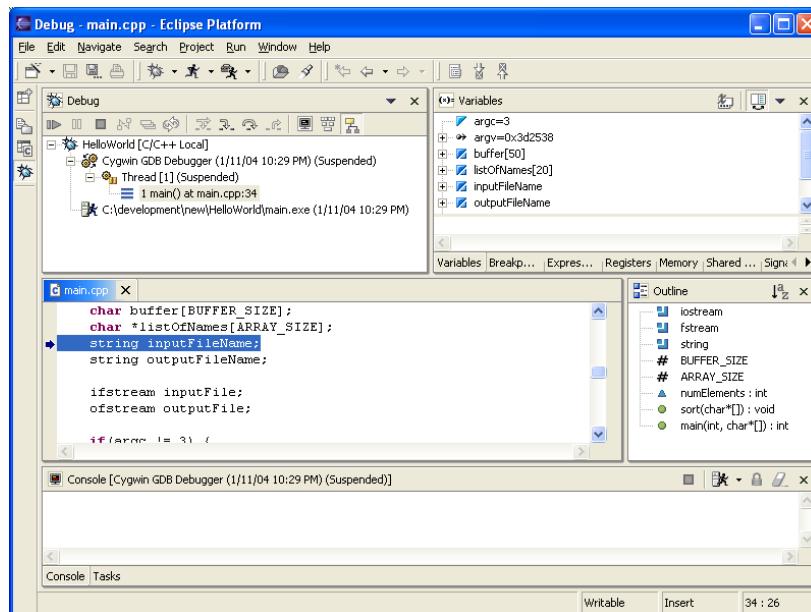


4.) Now that we have a breakpoint at that spot, we can begin to debug our application:

- a. Click on **Run → Debug...**
- b. In the Configurations window, click on **HelloWorld** (this configuration should already exist after completing the example from the previous section).

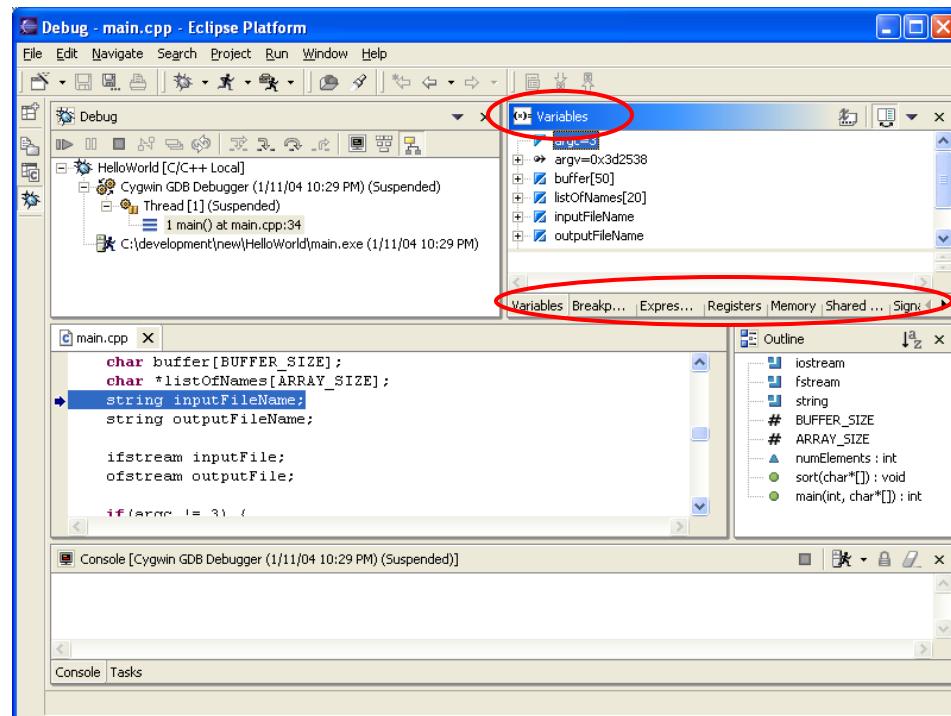


- c. Click the **Debug** button. Just minimize the Windows console window that pops up and your screen should look something like the following:

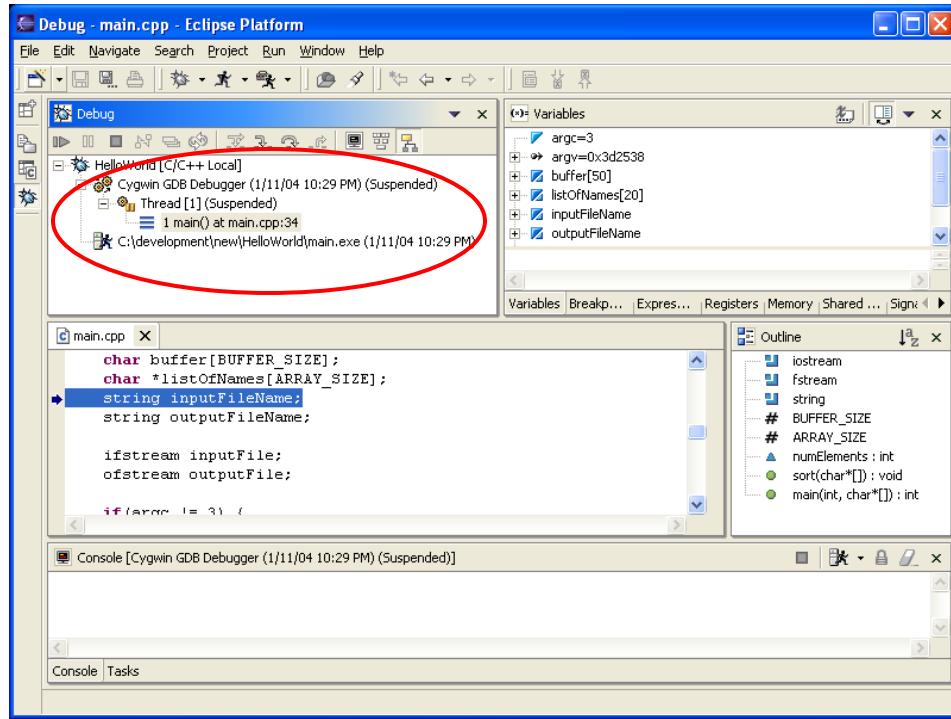


5.) Let's further analyze the previous screen before proceeding with the debugging of our application:

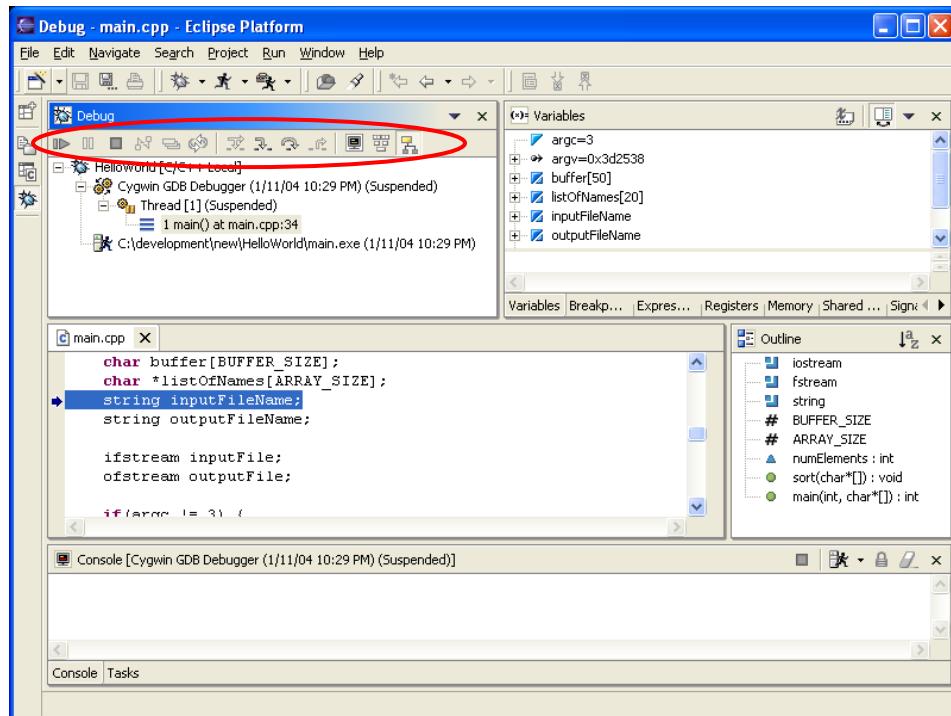
- a. The **Variables** window displays all the local variables currently within the scope of the current block of code. Also note the tabs at the bottom of the Variables window such as **Breakpoints**, **Expressions**, etc.. We will take a look at some of these windows in future tutorials.



- b. The **Debug** window displays the current state of the stack frame of your application. This will prove to be very helpful when debugging programs with recursive functions.



- c. The buttons circled below provide you with the basic debugging functionality.



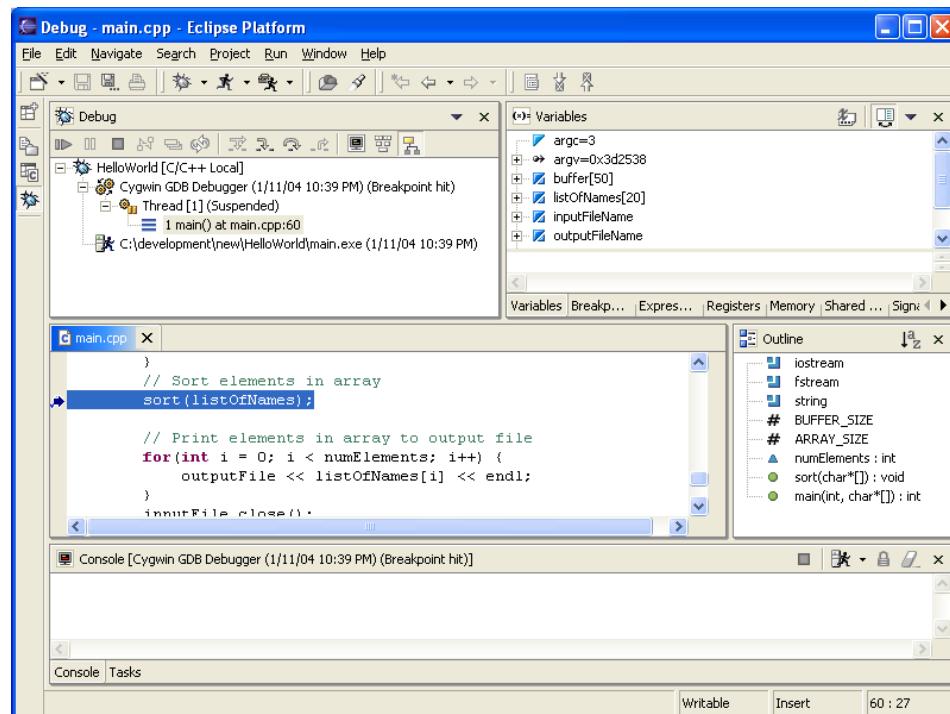
Familiarize yourself with the shortcut keys, as they will greatly speed up the debugging process!

Command	Name	Shortcut Key	Description
	Resume	F8	Resumes a suspended thread.
	Terminate	-	Terminates the selected debug target.
	Step Into	F5	Steps into the highlighted statement.
	Step Over	F6	Steps over the highlighted statement. Execution will continue at the next line either in the same method or (if you are at the end of a method) it will continue in the method from which the current method was called.
	Step Out	F7	Steps out of the current method.

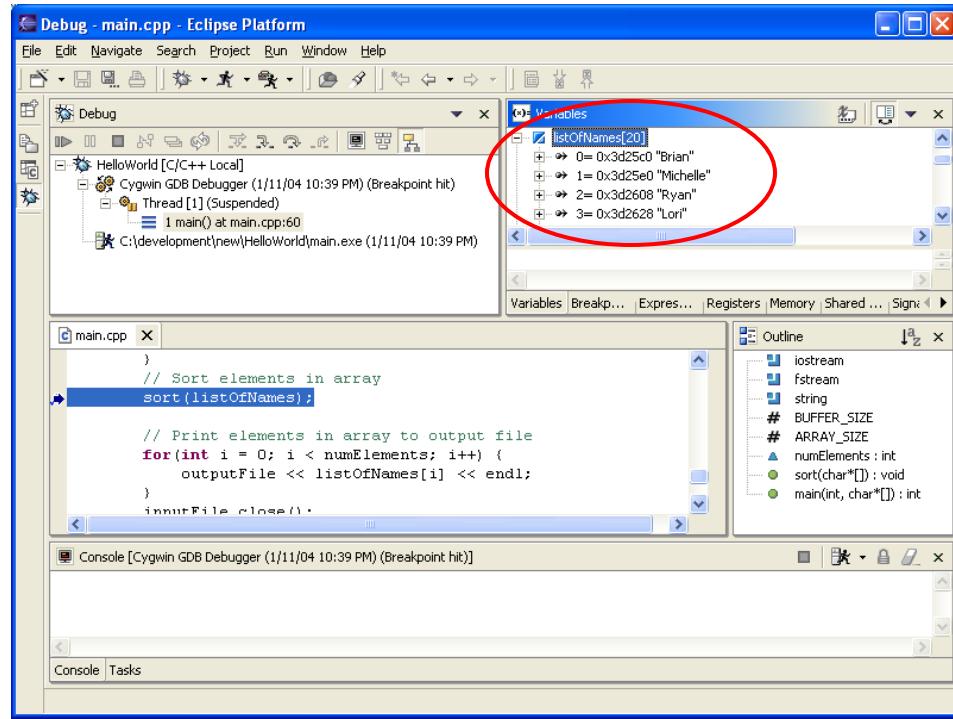
*Note: Above table is a modified version of the material from Eclipse's Help file.

6.) Continue on with the debugging process:

- Click on the button or press F8 to allow the application to resume the suspended thread and hit the breakpoint we set earlier.

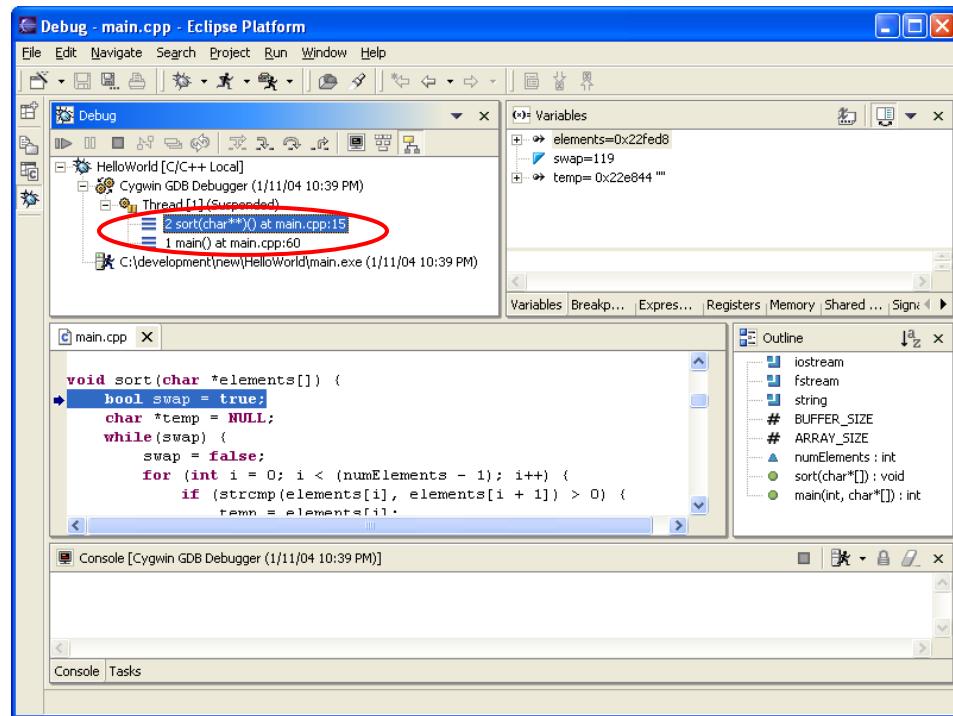


- Before we step into the sort method, let's take a look at our variables. If we expand the **listOfNames** array, you will notice that each element in the array corresponds to a name read in from the unsorted input file.



- c. Now click on the button or press F5 to step into the sort method. Once inside the sort method, if you will look in the **Debug window**, you will notice an extra frame on top of the stack called:

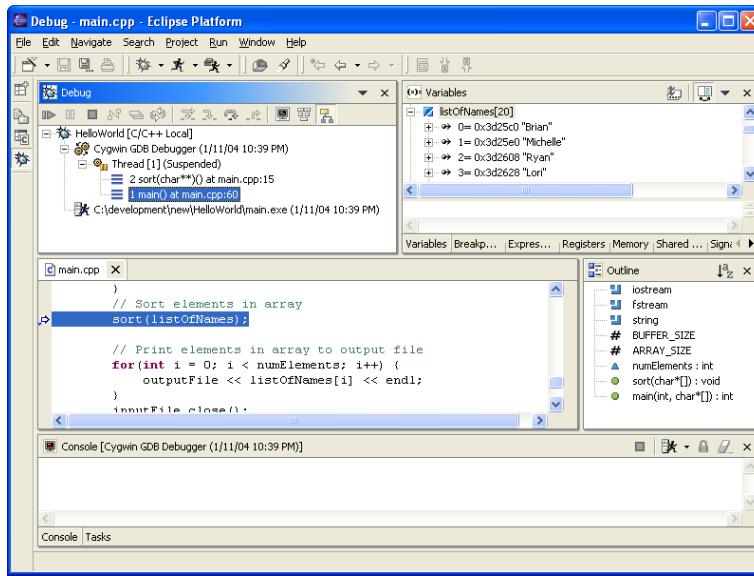
2 sort(char**)() at main.cpp:15



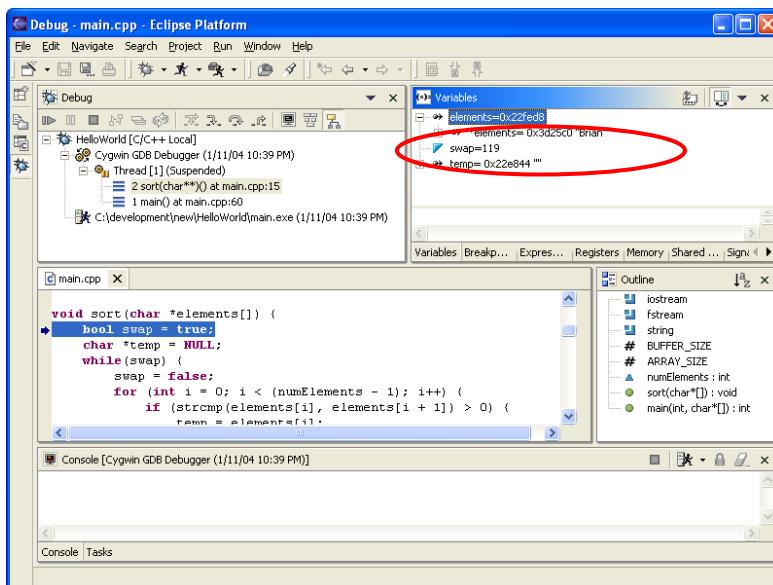
- d. If you click on the frame below called:

1 main() at main.cpp:60

You will notice that you can get view the variables from the main method in the Variables window. Sometimes during debugging, you may need to refer to variables from the calling method and you could easily do this by clicking on the corresponding stack frame. As well, this proves quite handy when debugging through recursive functions.

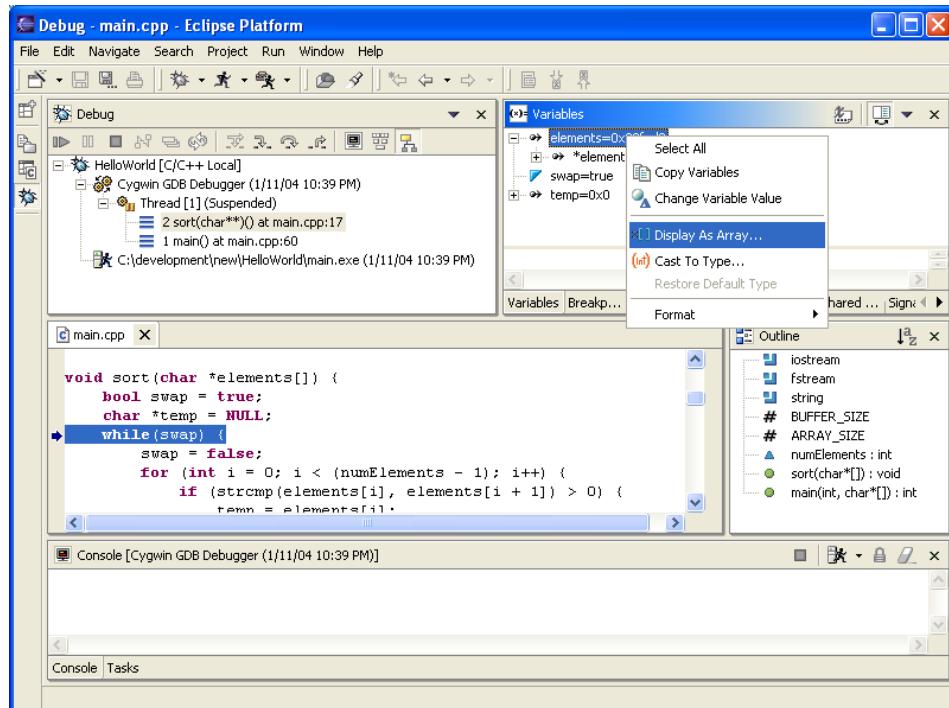


- e. Click on the **2 sort(char**()) at main.cpp:15** frame to get back to the sort method and expand the **elements** variable in the Variables window. It appears that there's only one element in the array!



What happened to all the other elements?? Since this method takes in a pointer to a char array, and we know that there are 20 elements in the array, we can easily display this variable as an array instead of a pointer.

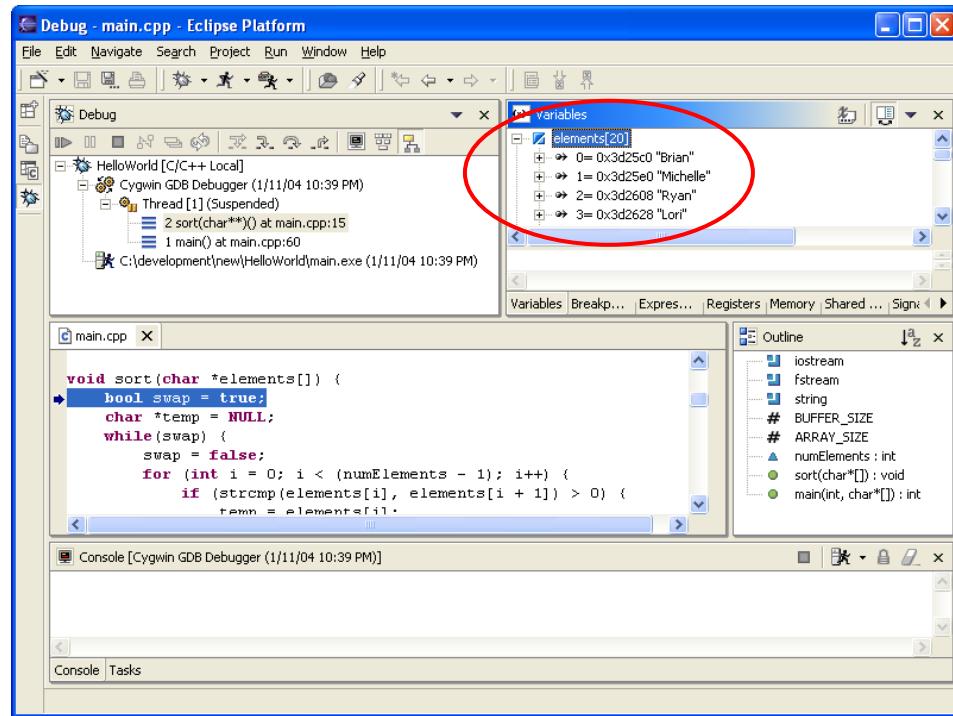
- f. Right-click on the **elements** variable and select **Display as Array...**



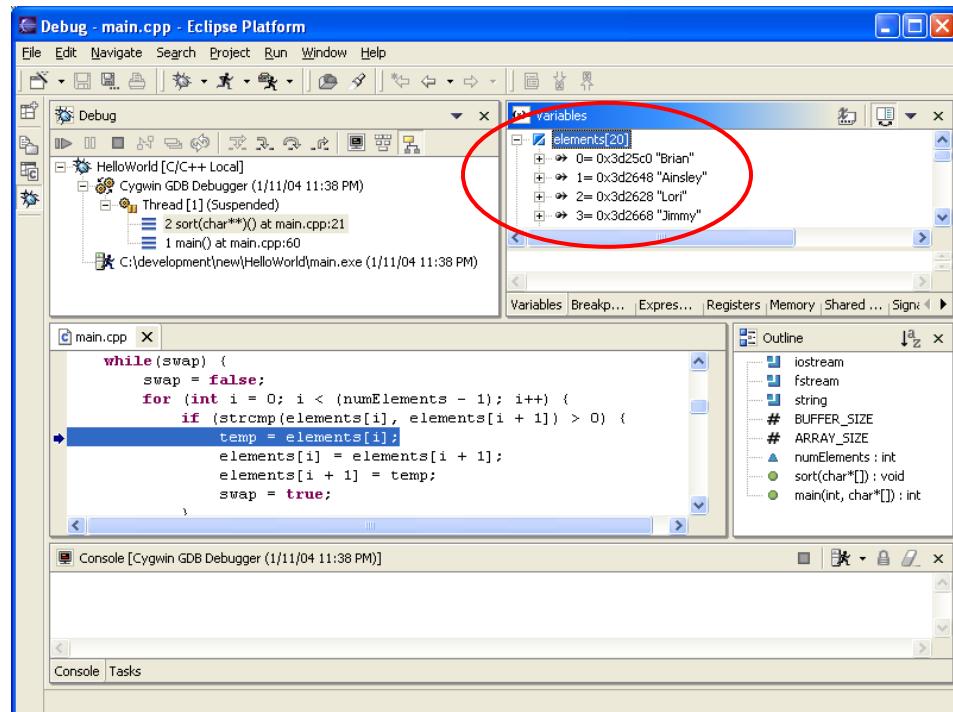
- g. Change the Length to 20 and click OK.



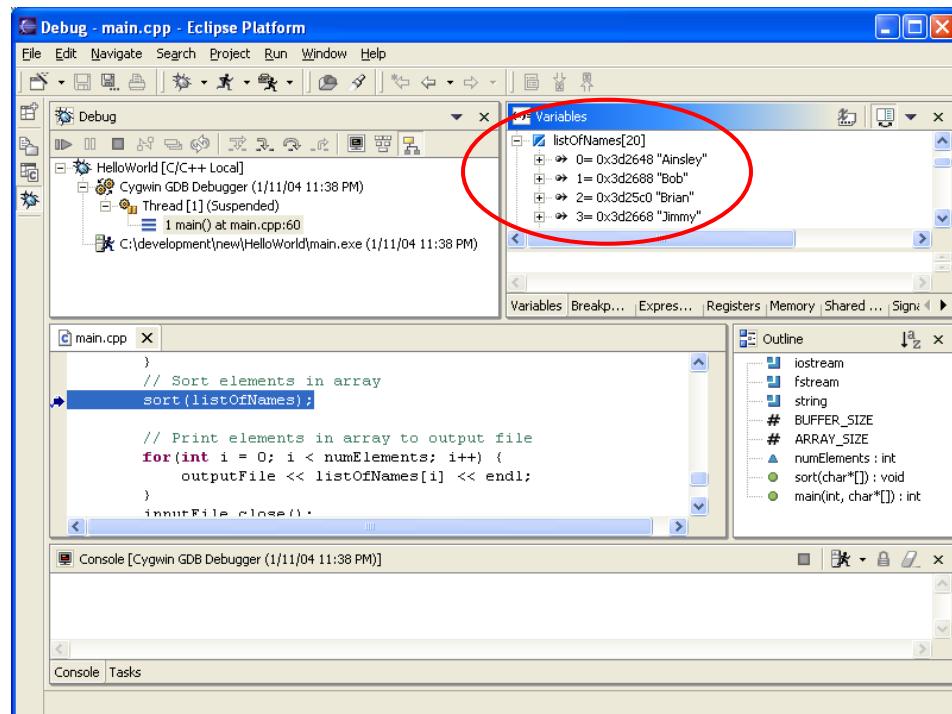
h. You should now be able to see all 20 elements in the array.



i. Now click on the button or press F6 to step through the rest of the sort method. You should notice that the array is slowly sorted into ascending order.



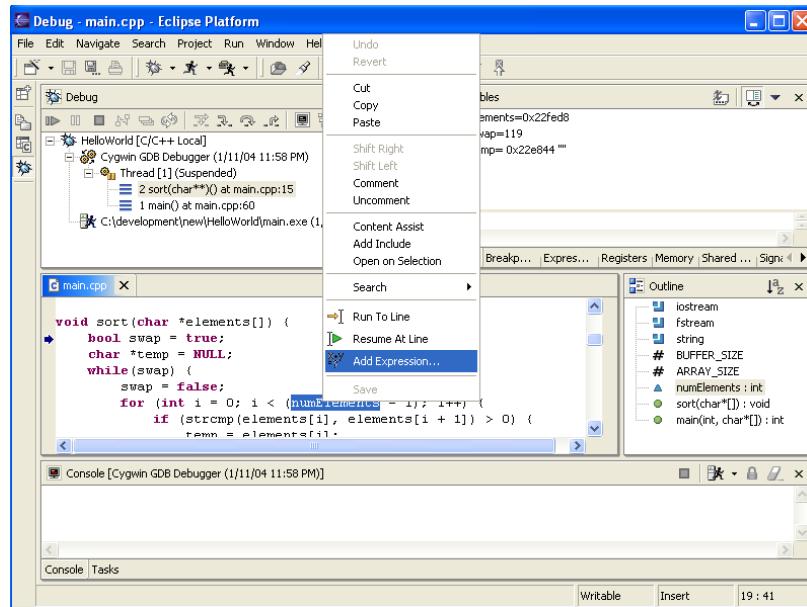
- j. Once you get tired of stepping through the while/for loops and you understand how the sort works, click on the button or press F7 to step out of this sort method and to return back to the main method. Once in the main method, expand the **listOfNames** variable to verify that the array has been sorted.



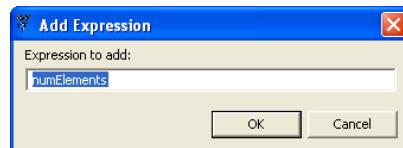
- k. Click on the button or press F8 to resume the rest of the application. After the application terminates, you can switch back to C/C++ Development Perspective and open **output.txt** to verify that the input file has been sorted correctly.

7.) One of the “quirks” of the C/C++ debugger is that if you have **global variables** in your program (variables defined outside your functions and main method), you will not see them appear in the Variables window. So what if you wanted to inspect those variables during debugging? The solution is actually quite simple:

- Start up the application in debug mode by clicking on the bug in the toolbar.
- Click on the button or press F8 to hit the first breakpoint and then click on the button or press F5 to step into the sort method.
- Now that we’re in the sort method, to determine the value of the global variable **numElements**, highlight the variable, right-click and click on **Add Expression...**



- d. Verify that the Expression to add is **numElements** and then click OK.



- e. You should now see that the global variable numElements has been added to the **Expressions** window. Repeat the above steps for any other global variable you would like to verify during debugging.

