

Netbeans for C on Linux

This provides some helpful information about using Netbeans for C on Linux. There is additional information available on the Netbeans website: [netbeans website](#)

Contents

1. Overview of Netbeans	2
1.1 What is an IDE?.....	2
1.2 What are projects?.....	2
1.3 Netbeans Window Areas	2
2. Creating a new project vs. using an existing project.....	4
2.1 Launching netbeans.....	4
2.2 Create a new project	5
2.3 Use an existing project	6
2.4 Creating a new C source code file	9
2.5 Creating a new C source code file from an existing file	10
2.6 Creating a new include file (Header file)	11
3. Compiling your C source code	12
3.1 Compile	12
3.2 Examining Compilation Errors	13
4. Running your program.....	14
4.1 Redirecting stdin and stdout	14
4.2 Setting command arguments	15
5. Debugging your program.....	16
5.1 Setting Breakpoints	16
5.2 Running the debugger	16
5.3 Examining the value of variables	17
5.4 Stepping through the code.....	18
5.5 Examining a complex structure or array.....	18
5.6 Examining an array which is a parameter.....	21
5.7 Removing a Breakpoint	22
6. Exiting Netbeans.....	22
7. Fixing Netbeans when it fails.....	22

1. Overview of Netbeans

Netbeans is an IDE which makes C code development, testing, and debuggin easier. It places the code for each program that you develop in a project.

1.1 What is an IDE?

An Integrated Development Environment (IDE) helps manage software by providing an integration of tools:

- source code editor
- compiler
- source compile/link builder (i.e., understands when code changes and compiles and/or links)
- debugger

This contrasts with software development using unrelated tools, such as vim, gcc, and make.

1.2 What are projects?

We can divide a large program into many source code files. Unfortunately, that makes it more difficult to manage manually. With an IDE, the multiple source code files belonging to one executable are managed as a project.

Prior to running Netbeans, you should probably create a folder for your course (e.g., cs2123) and within that folder create **netbeans** folder. That netbeans folder will contain your projects for the course.

```
$ mkdir ~/cs2123
$ cd ~/cs2123
$ mkdir netbeans
```

1.3 Netbeans Window Areas

Figure 1 shows the various areas of the Netbeans Window.

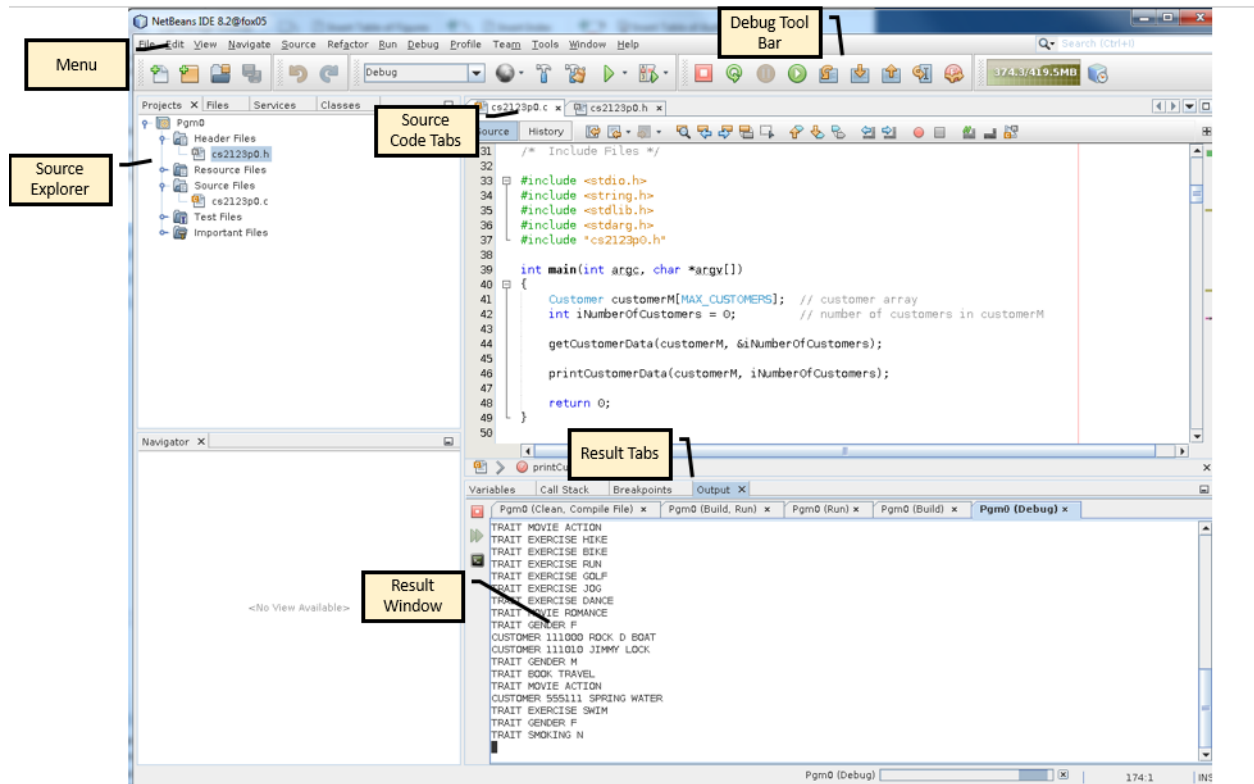






Figure 1: Netbeans Window Areas

The Netbeans window consists of the following areas:

Menu	This is a menu at the top of the window and includes <ul style="list-style-type: none">File Create new projects, open existing projects, save filesSource Special source editing capabilities like indenting blocks of codeRun Compile a file, build (compile/link) the entire project, run the executable, set the configuration (e.g., command arguments, stdin/stdout redirection)Debug Run the program in the debugger
Source Explorer	This provides an explorer view of the files in the project. This will include Header files (.h) and Source files (.c).
Source Code Tabs	Allows you to quickly look at different files.
Result Window	This has many uses which vary dependent on which Result Tab is selected: <ul style="list-style-type: none">Output Shows the compiler/linker results or program outputVariables (Only shows during debugging) Shows the values of variables.
Debug Tool Bar	(Only shows during debugging) This has buttons to advance through the code or terminate the code during debugging. <ul style="list-style-type: none"> finish execution (i.e., terminate execution) Continue execution (F5) Next (F8) – execute the code in the current statement, but give control to the user at the next statement. You will usually step through code using F8. Most debuggers (including Netbeans) call this Step Over. Step Into (F7) – continue stepping through execution by stopping at the first statement inside the current statement which is usually a function call

2. Creating a new project vs. using an existing project

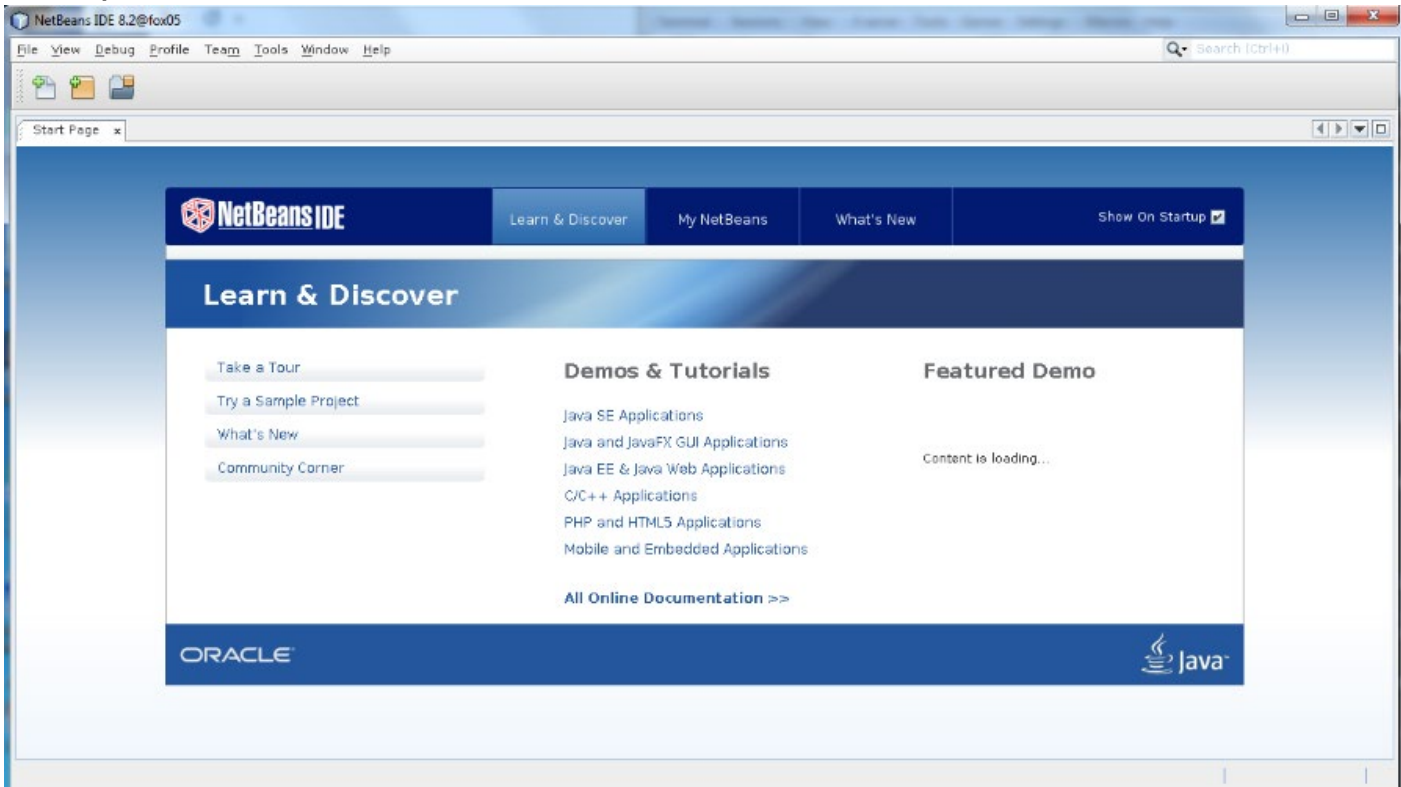
With Netbeans, you can either create a new project (section 2.2) or use an existing project (section 2.3). Section 2.1 shows how to launch netbeans.

2.1 Launching netbeans

From the terminal window on either a Linux workstation in the CS Main Lab or from a fox server:

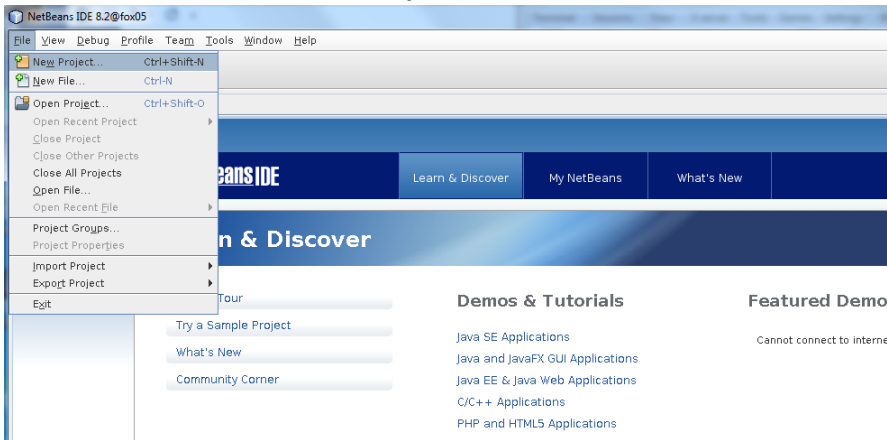
```
$ netbeans
```

Netbeans will launch in a **separate window**. You can disable the initial splash screen by unchecking the **Show On Startup** box.

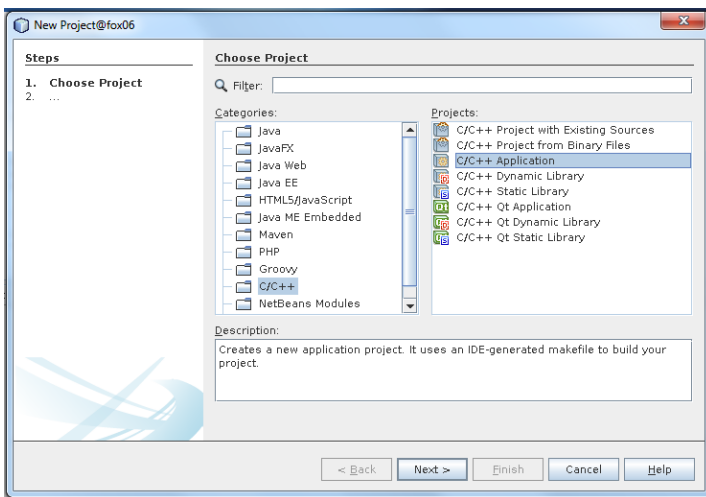


2.2 Create a new project

On the menu, click **File > New Project**.



It will show a screen like this:

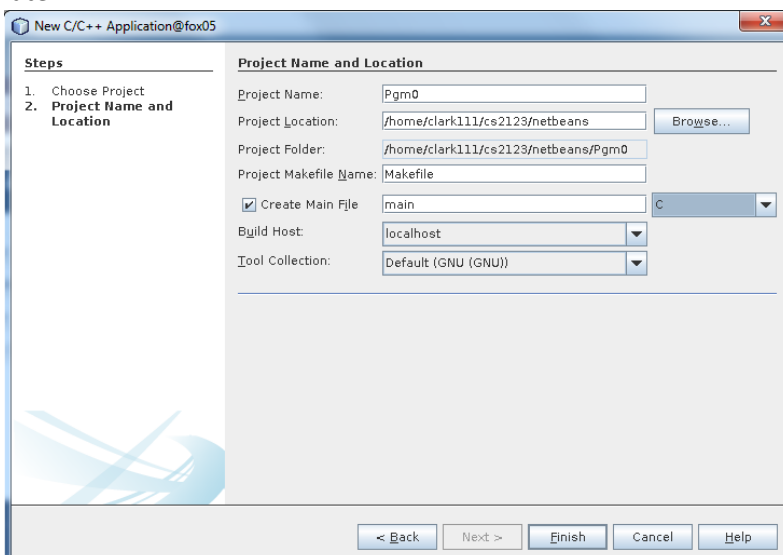


Select **C/C++** and **C/C++ Application**. Press the **Next** button.

Netbeans will show the screen that follows. Enter your project location as:

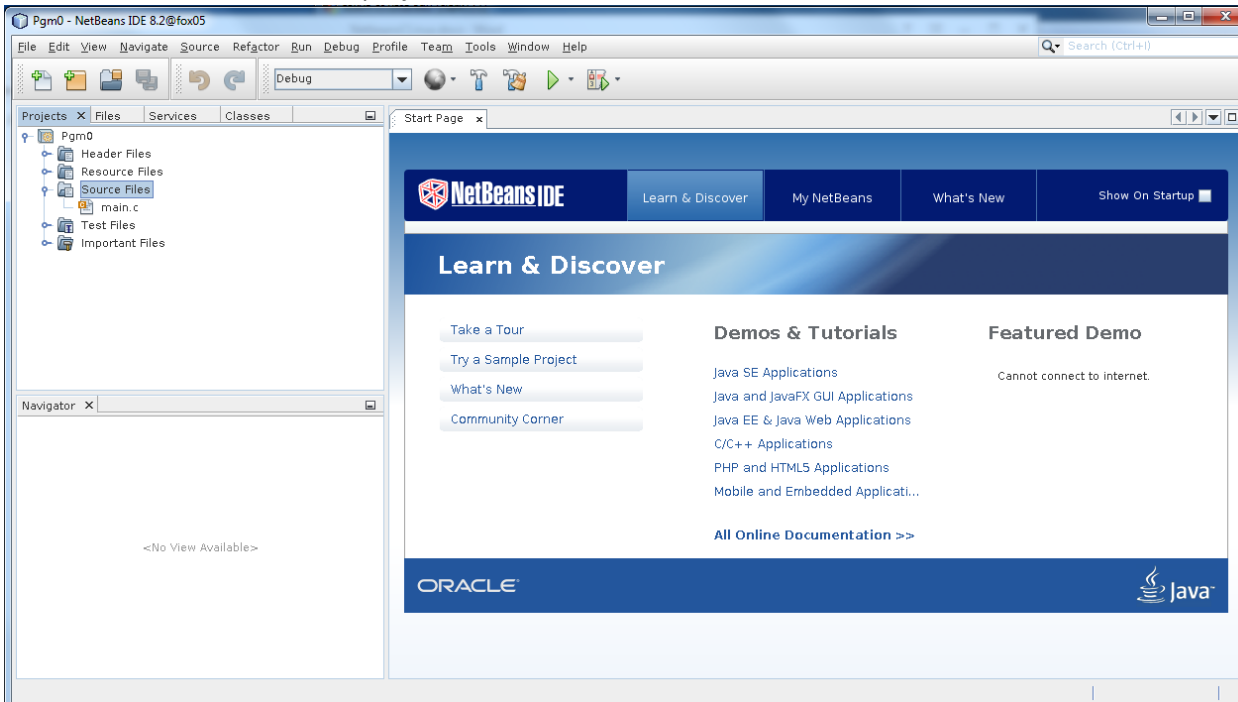
`/home/abc123/cs2123/netbeans`

Enter your project name (e.g., Pgm0, Pgm1). You can tell it to create a default main.c source file which can be removed later.



Press the **Finish** button.

Netbeans will show the initial project:

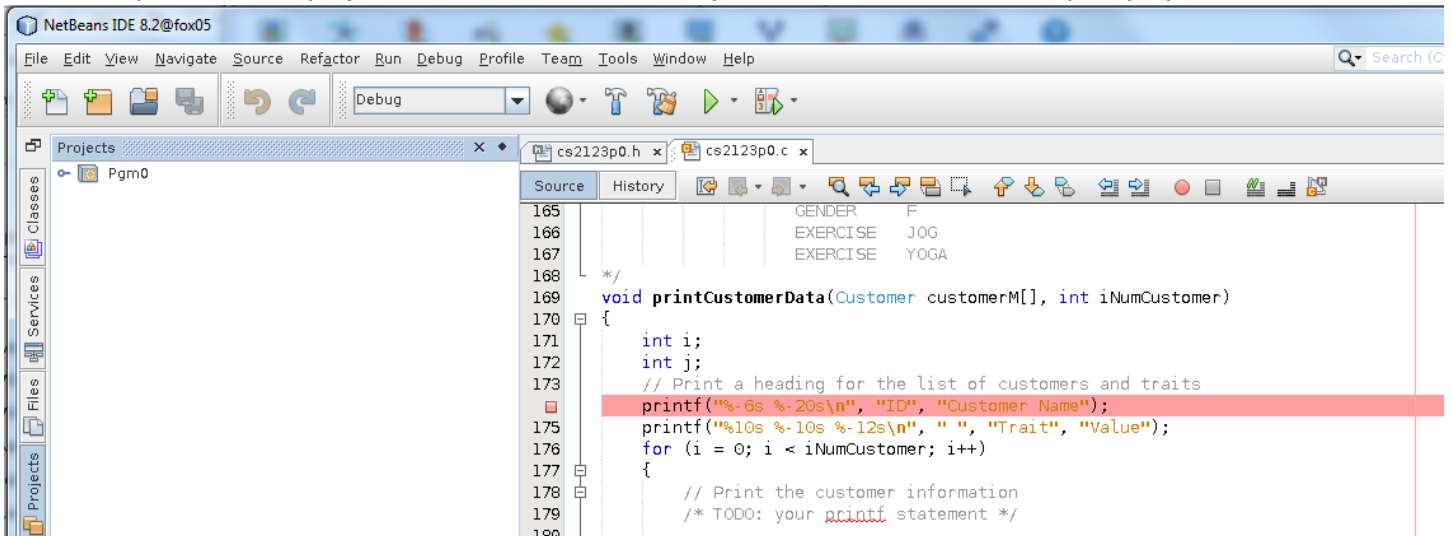


2.3 Use an existing project

When you launch netbeans by simply doing this:

```
$ netbeans
```


It will load your last used project. You can also click the Projects tab on the left to see all your projects.

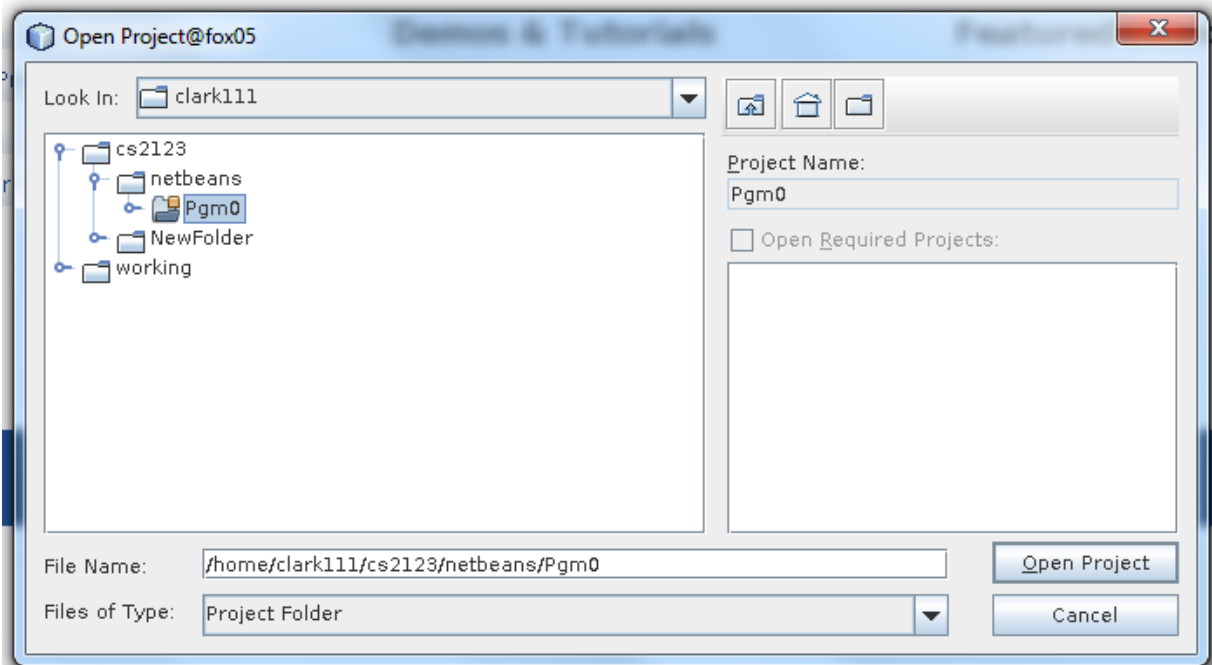


If you previously had the problem where netbeans wouldn't open and had to remove the ~/.netbeans directory (see section 6), netbeans won't show your previous projects. Go to your netbeans project directory and then execute the netbeans command:

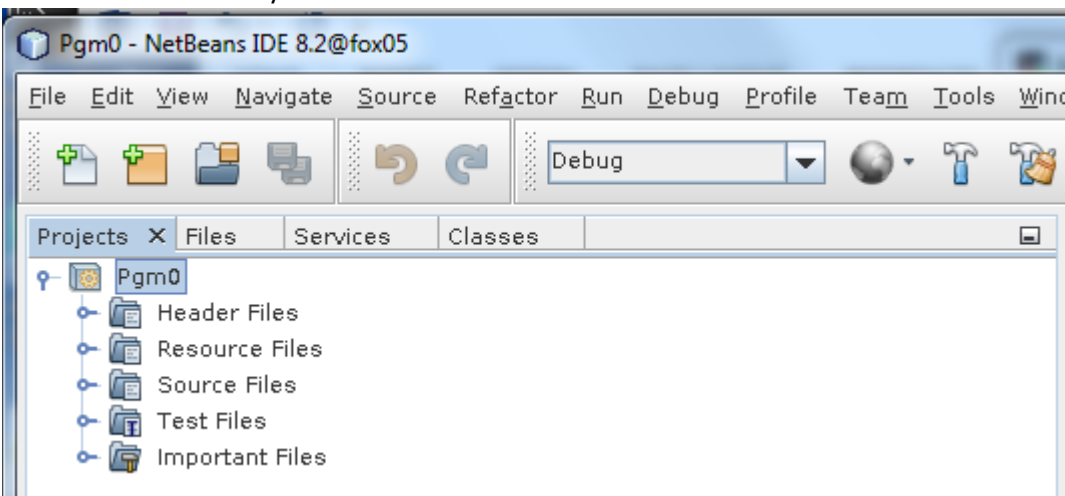
```
$ cd ~/cs2123
```

```
$ netbeans
```

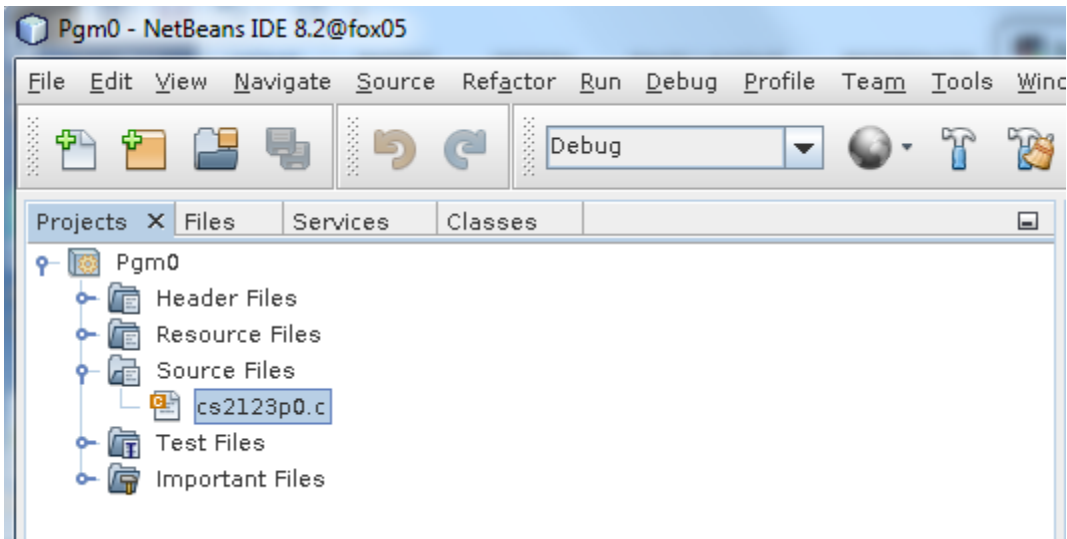
Netbeans will show a simple start up window. Select **File > Open Project**, causing netbeans to show a window to browse for your project. Use the  icon to open your folders until you get to your project folder. Double click on that folder and then press the **Open Project** button.



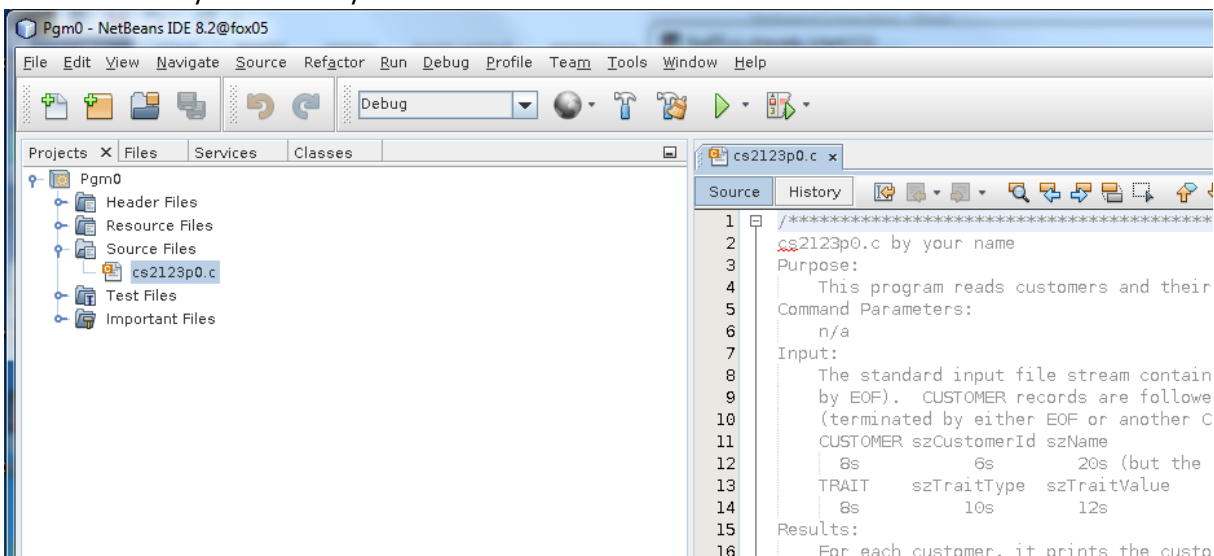
Netbeans won't show your files in the Source Code Tabs. Instead it will show this:



You can see your files by selecting **Source Files** and/or **Header Files**.

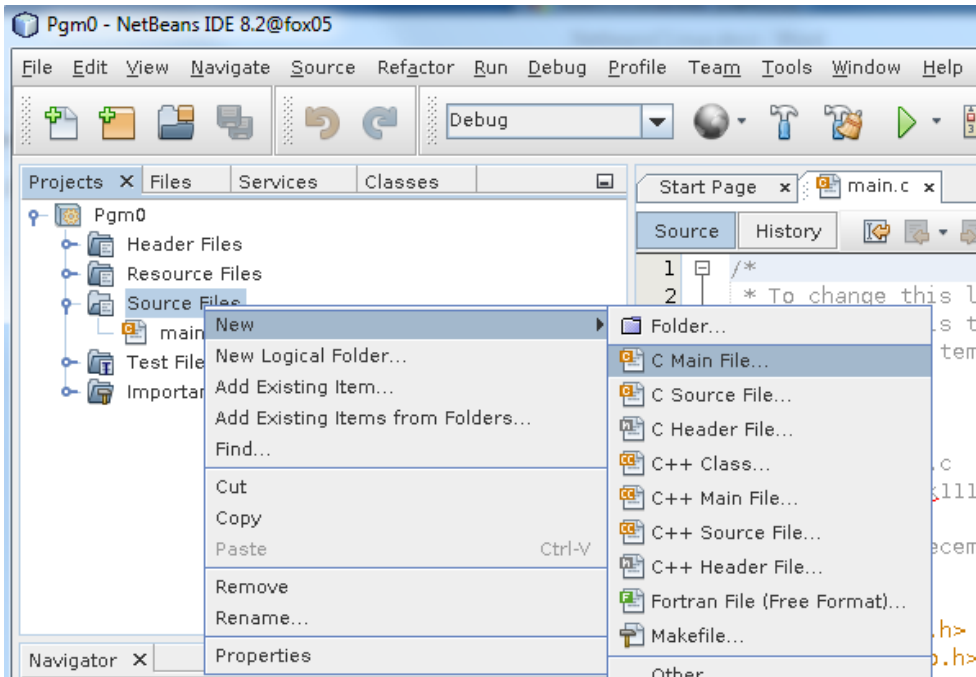


Double click on your file and you will see the source in a Source Tab:

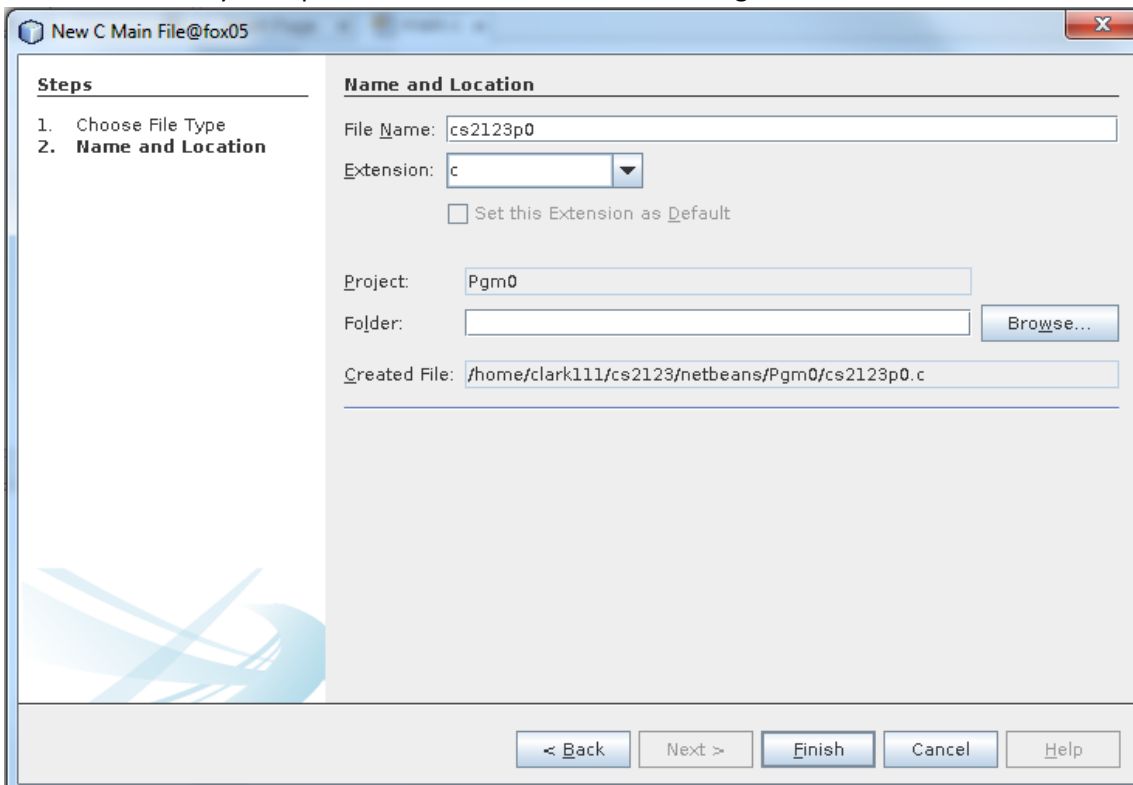


2.4 Creating a new C source code file

Within the Source Explorer, right click on **Source Files**. A submenu will appear as shown below. To add a new file containing main, select **New > C Main File**. If you wanted to add a new C source file that doesn't contain a main function, you would select **New > C Source File**.



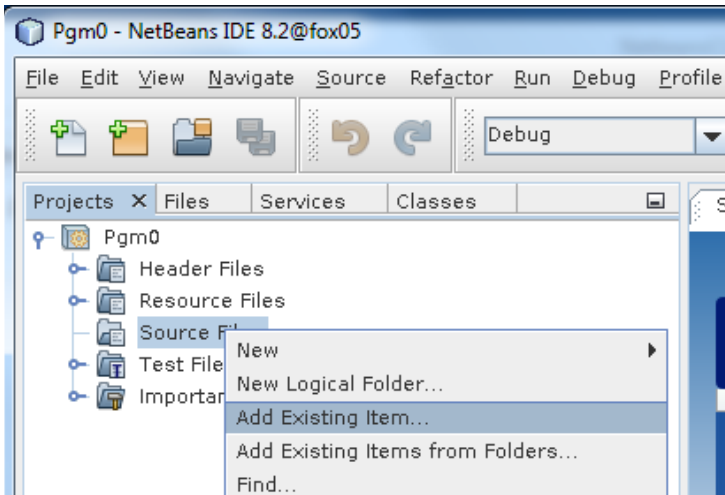
Netbeans will ask you to provide the file name in the following window:



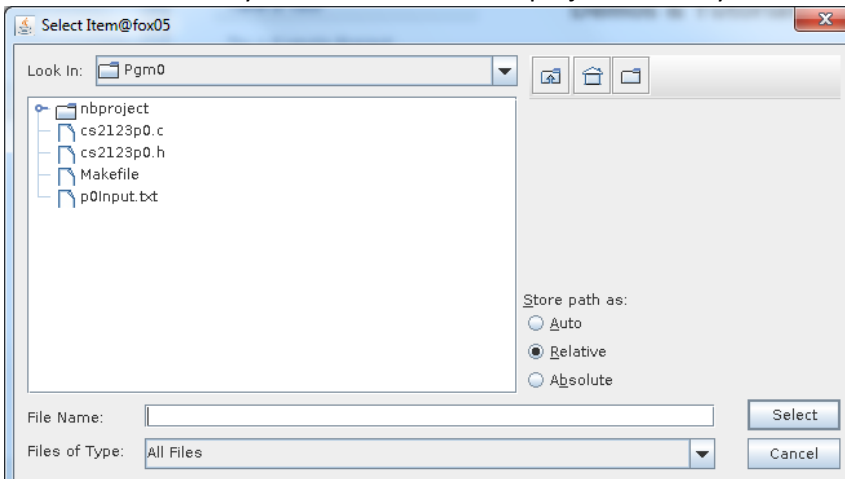
Simply provide the file name (without the .c) and press **Finish**.

2.5 Creating a new C source code file from an existing file

If the source code already exists in the project directory (this frequently happens when your professor provides source code) or you may want to externally copy the file to the netbeans project directory, right click on **Source Files**, and select **Add Existing Item**:



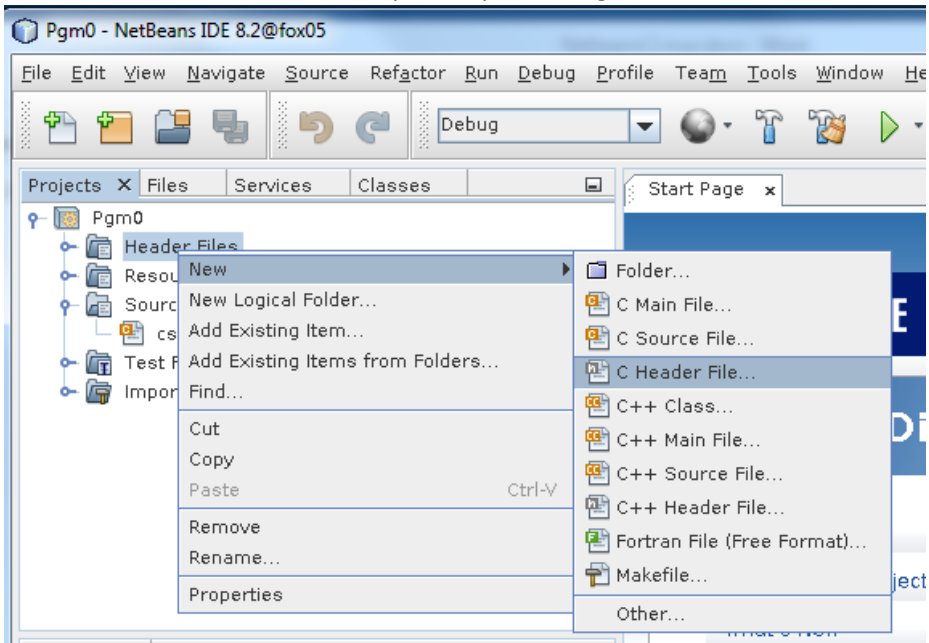
Netbeans will show you the contents of the project directory:



You can then select a file that already exists and add it internally into the project.

2.6 Creating a new include file (Header file)

This is similar to section 2.4 except that you will right click on **Header Files** in the Source Explorer.

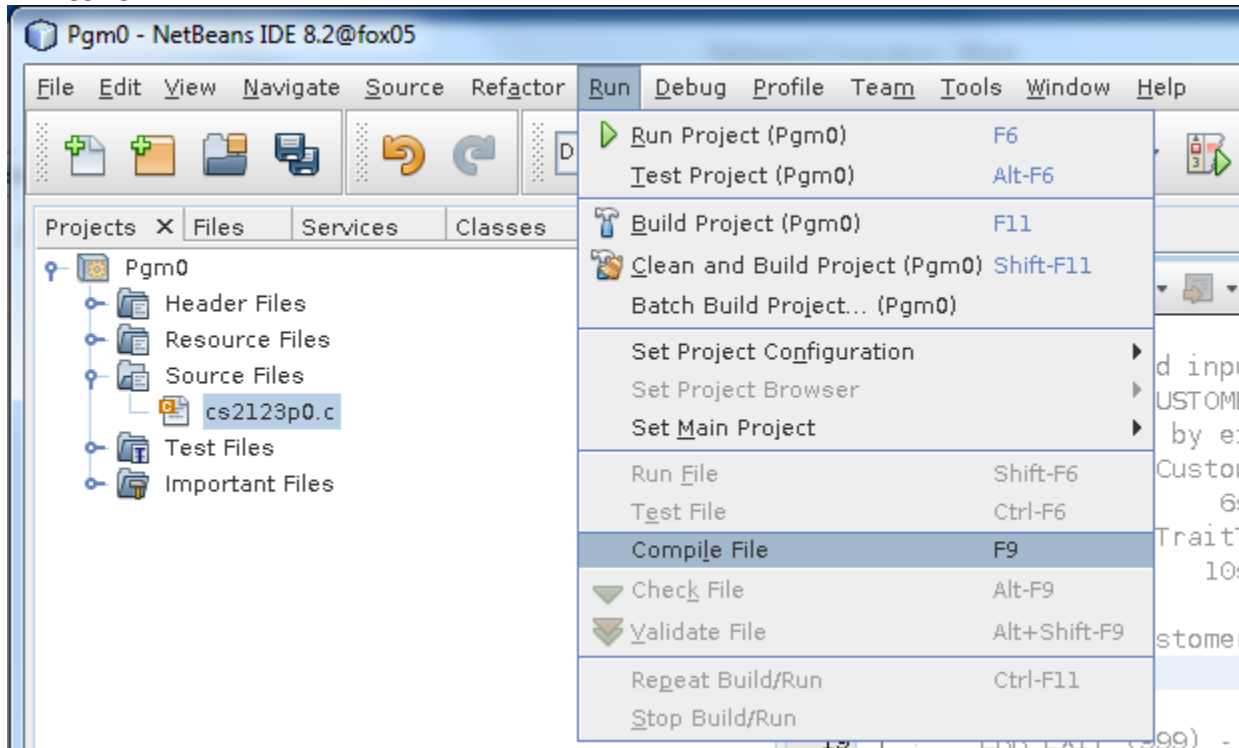


If you want to add an existing include file, instead of selecting **New**, select **Add Existing Item**. To then add that, refer to the documentation in section 2.5.

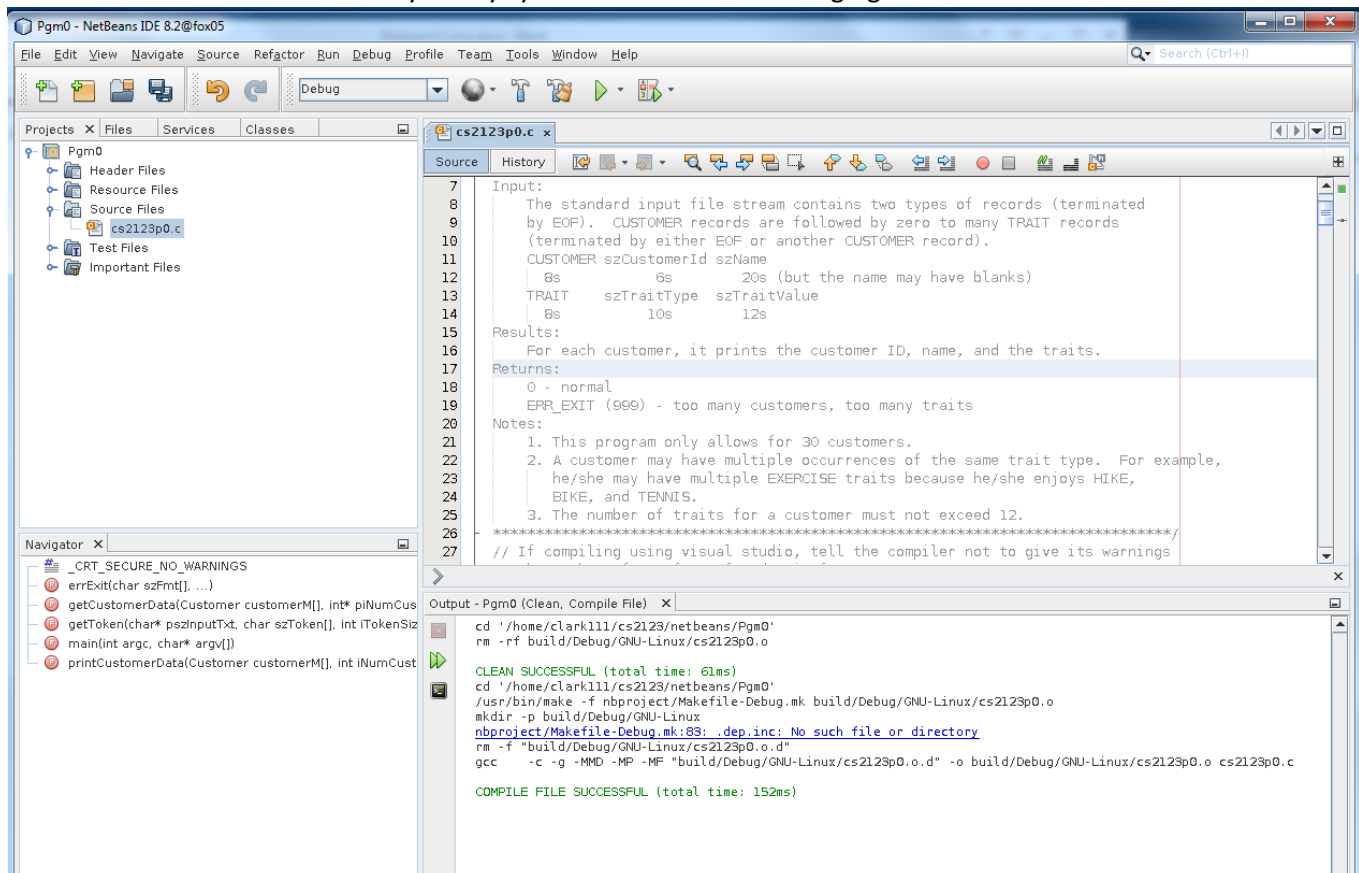
3. Compiling your C source code

3.1 Compile


There are several ways to compile your C source code. You can specify **Run > Compile File** or press **F9** (while not debugging).



The **Result Window** should show you any syntax errors. The following figure shows a result area without errors:

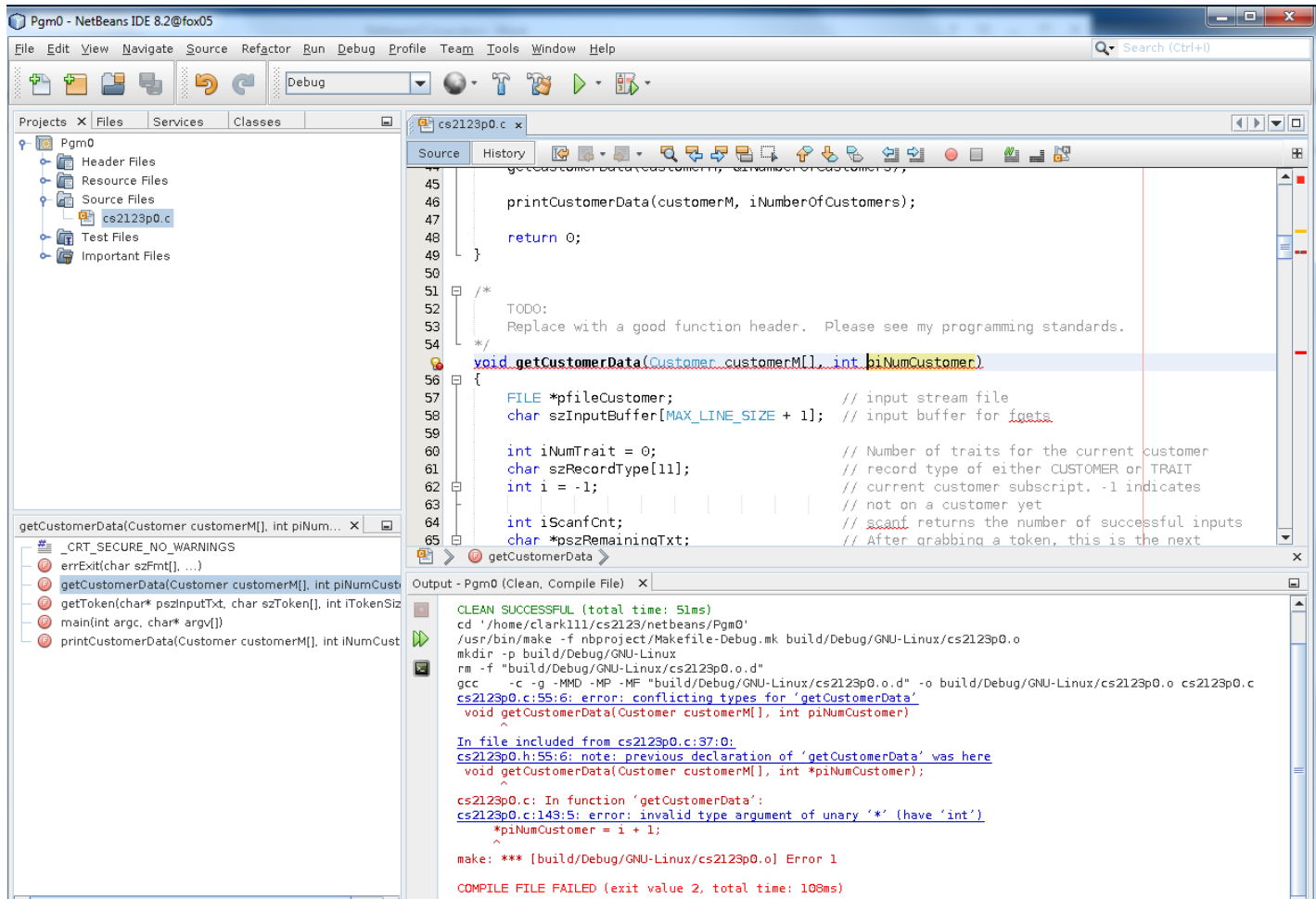




You can also compile/link all your code by selecting **Run > Build Project** or by pressing the  button. This will compile each source file that hasn't been compiled since you changed them and then link the resulting object files.

3.2 Examining Compilation Errors

If the compiler detected errors, the **Result Window** will show those errors. This example shows that line 55 has a syntax problem (a parameter has a conflicting type). It also shows that the include file (cs2123p0.h) had a different prototype declaration.



The screenshot shows the NetBeans IDE interface. The main editor displays the source code for `cs2123p0.c`. The code includes a function `getCustomerData` with the following signature: `void getCustomerData(Customer customerM[], int piNumCustomer)`. The function body contains several variables and a loop. The error message in the Output window is as follows:

```
Output - Pgm0 (Clean, Compile File) X
CLEAN SUCCESSFUL (total time: 51ms)
cd '/home/clark111/cs2123/netbeans/Pgm0'
/usr/bin/make -f nbproject/Makefile-Debug.mk build/Debug/GNU-Linux/cs2123p0.o
mkdir -p build/Debug/GNU-Linux
rm -f "build/Debug/GNU-Linux/cs2123p0.o.d"
gcc -c -g -MMD -MP -MF "build/Debug/GNU-Linux/cs2123p0.o.d" -o build/Debug/GNU-Linux/cs2123p0.o cs2123p0.c
cs2123p0.c:55:6: error: conflicting types for 'getCustomerData'
void getCustomerData(Customer customerM[], int piNumCustomer)
^
In file included from cs2123p0.c:37:0:
cs2123p0.h:55:6: note: previous declaration of 'getCustomerData' was here
void getCustomerData(Customer customerM[], int *piNumCustomer);
^
cs2123p0.c: In function 'getCustomerData':
cs2123p0.c:143:5: error: invalid type argument of unary '*' (have 'int')
*piNumCustomer = i + 1;
^
make: *** [build/Debug/GNU-Linux/cs2123p0.o] Error 1
COMPILE FILE FAILED (exit value 2, total time: 108ms)
```

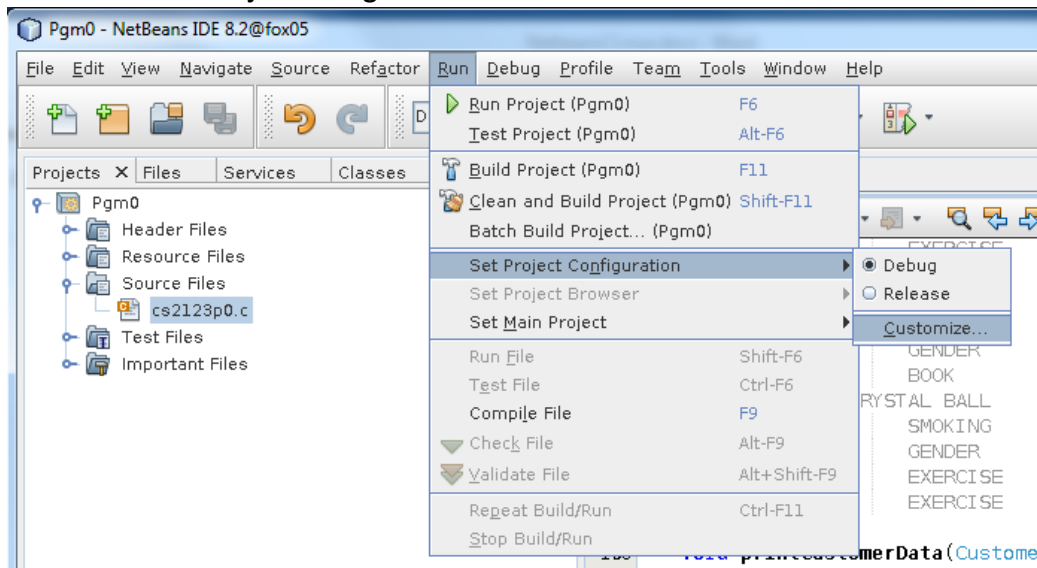
4. Running your program

If your source files have no compilation errors, you can execute your program by specifying **Run > Run Project**; however, you might need to redirect stdin (section 4.1) or specify command arguments (section 4.2) before executing it.

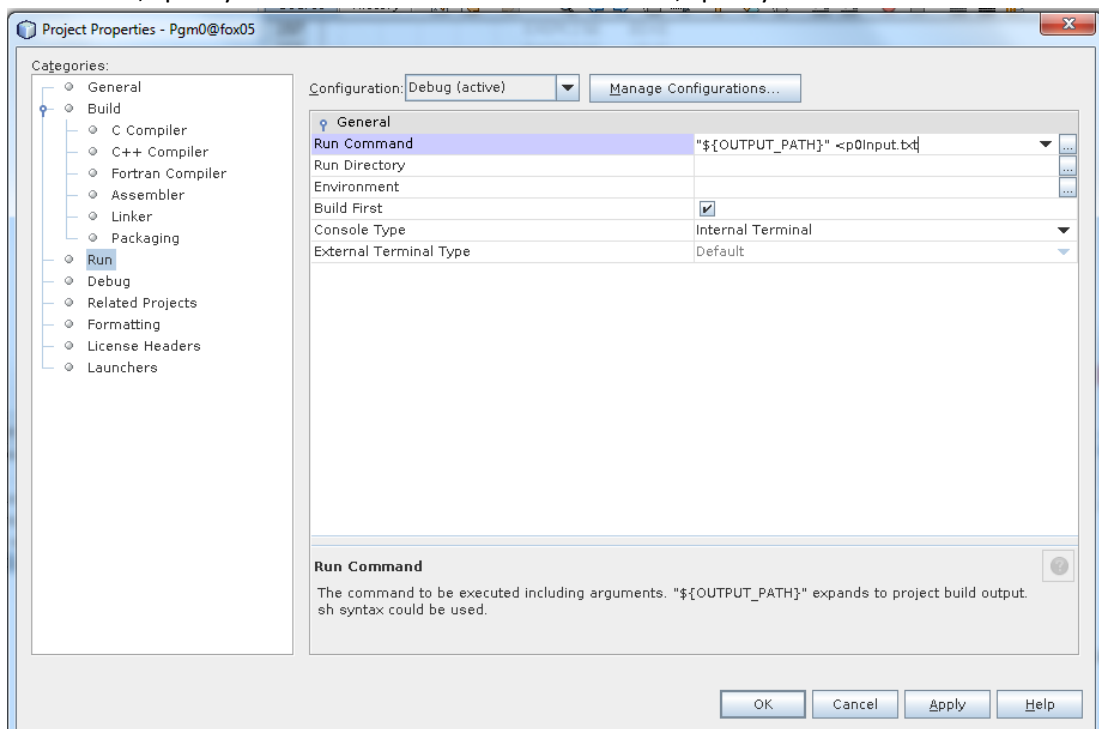
4.1 Redirecting stdin and stdout

One of the conveniences of Linux is that you can use stdin from a command line. You can even pipe the result of one program to the stdin of another program. (Piping is discussed in CS3423.) If your program needs to redirect input from a file using stdin or if you want to redirect the output to a file, this section shows you how that is done.

Select **Run > Set Project Configuration > Customize**.



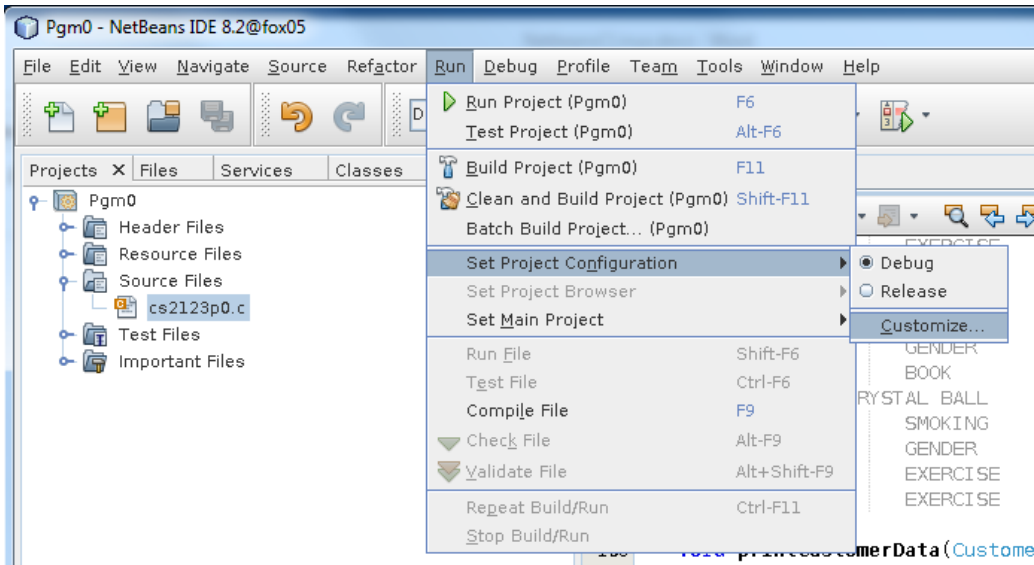
The Project Properties window is shown below. Select the **Run** category within the left window area. For stdin redirection, specify `< inFilename`. For stdout redirection, specify `> outFilename`. You can also specify both.



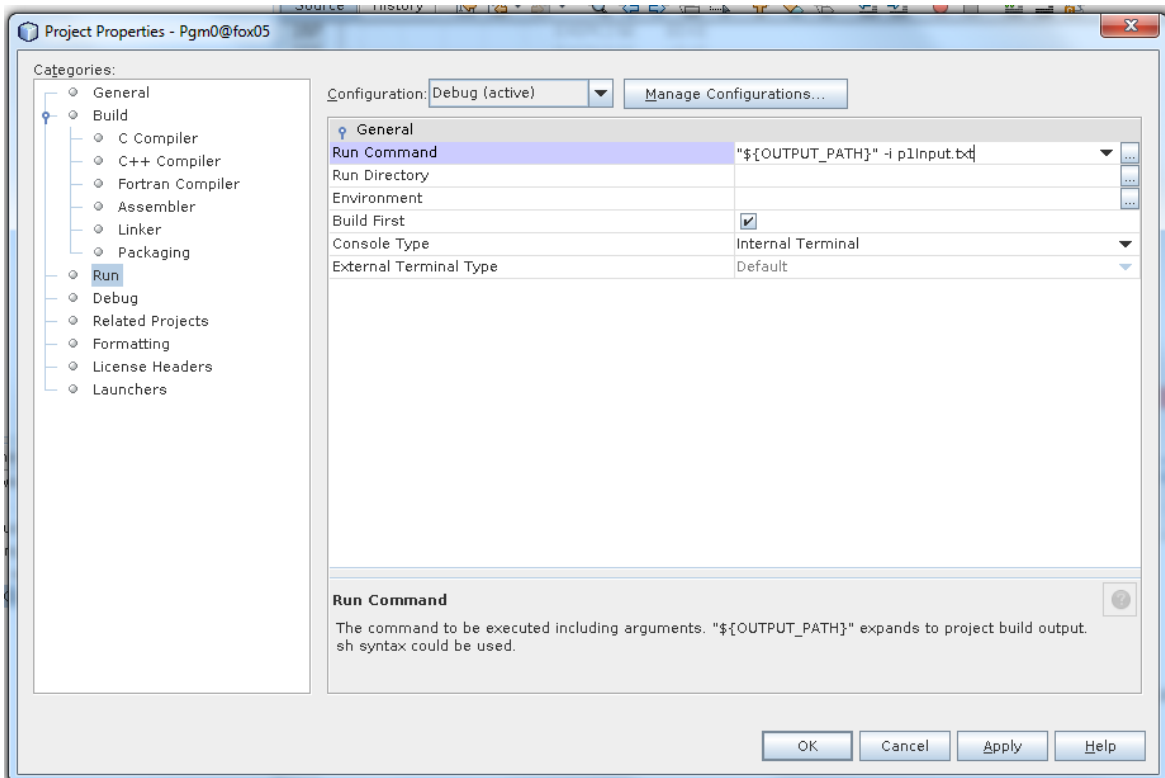
By default, Netbeans will look for your input files in the project directory.

4.2 Setting command arguments

Linux provides the ability to pass command line arguments into a program so that your C argv array contains them. NetBeans provides a mechanism to pass command arguments. Just like in section 4.1, we select **Run > Set Project Configuration > Customize**.



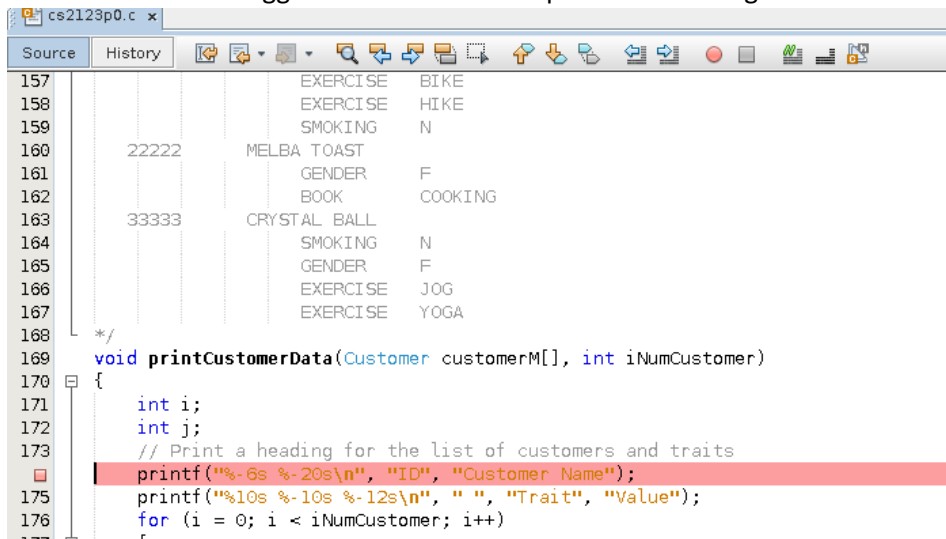
As shown in the diagram, include your example command line arguments. You can also redirect files as shown in section 4.1.



5. Debugging your program

5.1 Setting Breakpoints

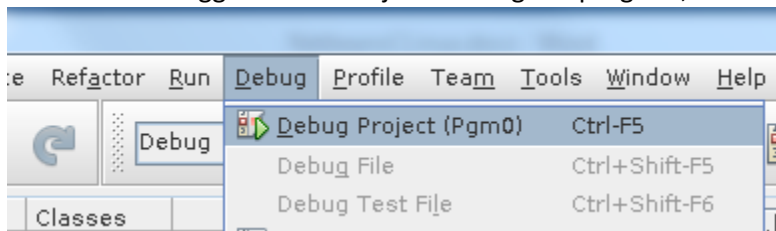
Breakpoints are one of the most important features of a debugger. Code execution will halt when a breakpoint is encountered. This allows you to examine the value of variables. To set a Breakpoint, click on the line number of the statement. The debugger will halt execution prior to executing that statement.



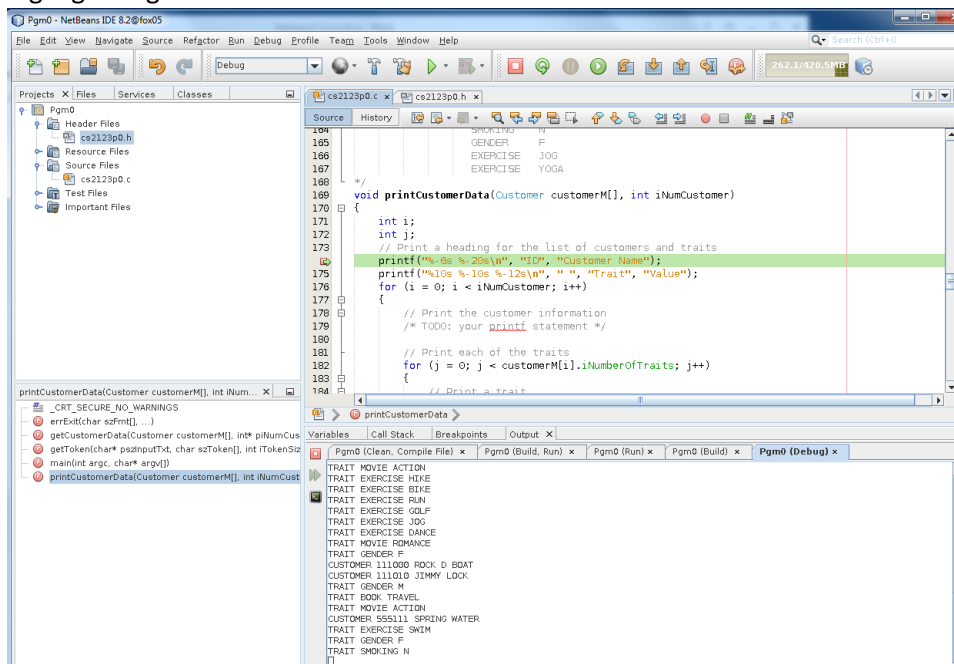
```
cs2123p0.c x
Source History
157      EXERCISE BIKE
158      EXERCISE HIKE
159      SMOKING N
160      22222 MELBA TOAST
161      GENDER F
162      BOOK COOKING
163      33333 CRYSTAL BALL
164      SMOKING N
165      GENDER F
166      EXERCISE JOG
167      EXERCISE YOGA
168      */
169      void printCustomerData(Customer customerM[], int iNumCustomer)
170      {
171          int i;
172          int j;
173          // Print a heading for the list of customers and traits
174          printf("%-6s %-20s\n", "ID", "Customer Name");
175          printf("%10s %-10s %-12s\n", " ", "Trait", "Value");
176          for (i = 0; i < iNumCustomer; i++)
```

5.2 Running the debugger

To run the debugger instead of just running the program, select **Debug > Debug Project**.

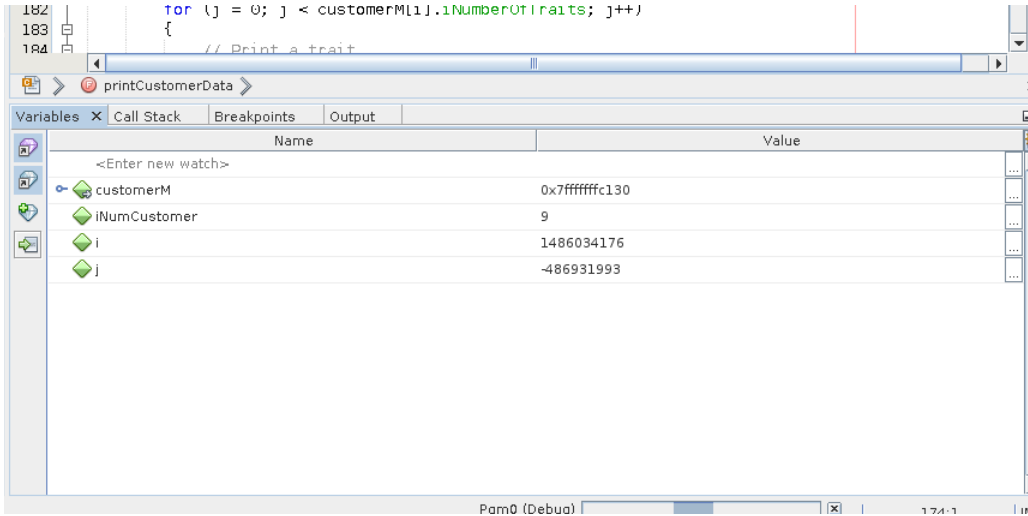


If the code reaches a breakpoint, it will halt there to allow you to examine variables or continue execution. It will highlight in green the next statement to execute.

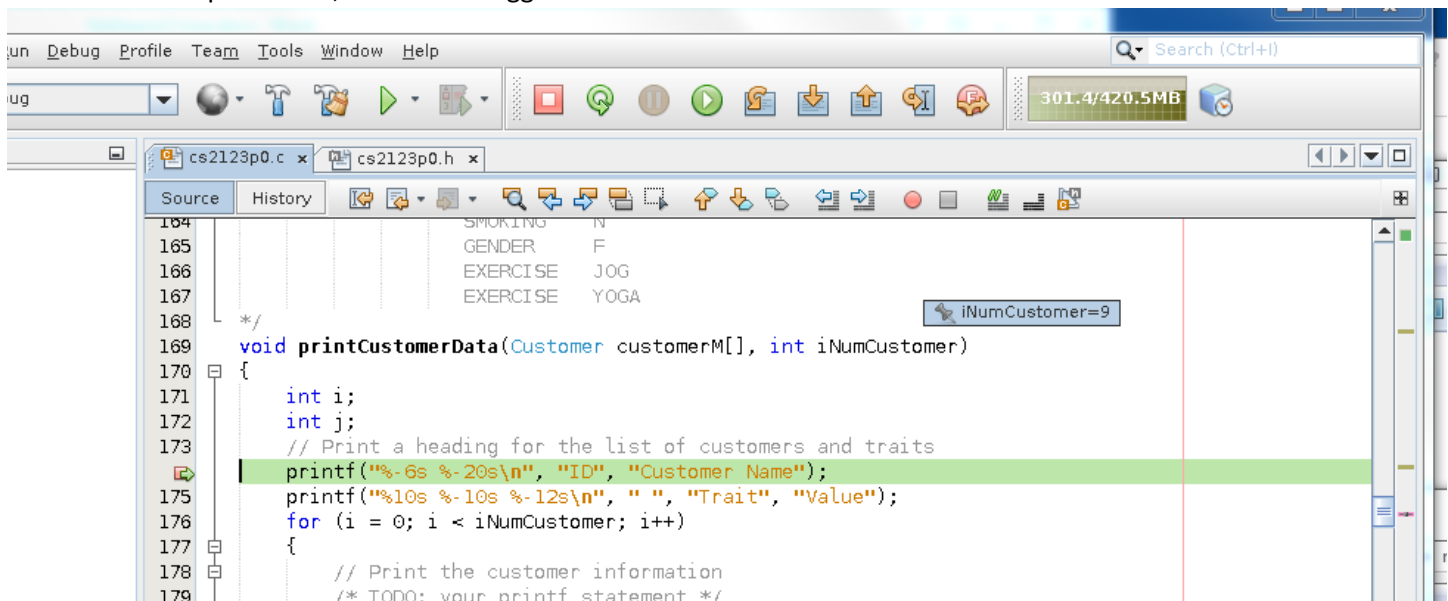


5.3 Examining the value of variables


Within the **Result Window**, the **Variables** Result Tab is used to see the contents of your variables. Notice that the variable `iNumCustomer` is 9, but the variables `i` and `j` have garbage values since we haven't yet executed the for statements that give them values.





You can also examine the contents by placing the mouse on a variable. In this example the mouse is on the `iNumCustomer` parameter, and the debugger shows its value which is 9.




5.4 Stepping through the code

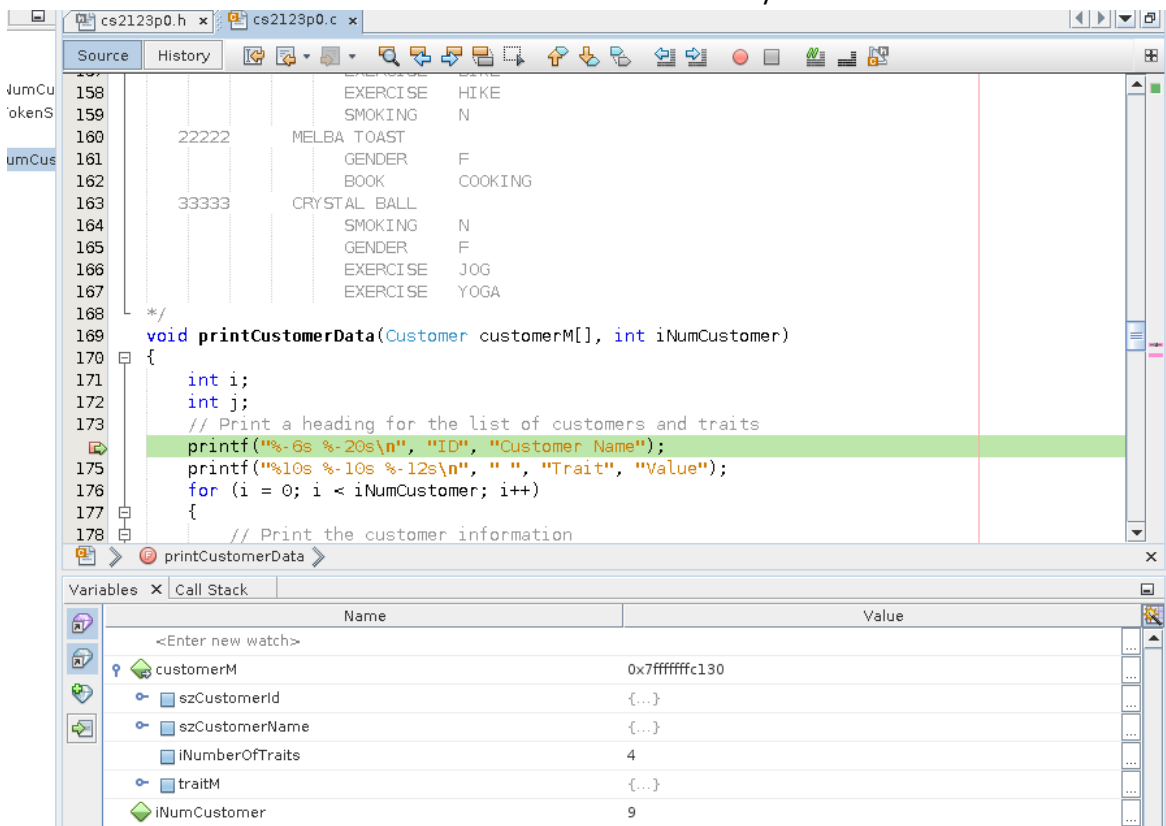
The Debug Tool Bar at the top of the Netbeans window has the tool buttons for stepping through your code during execution. Most of the time you will hit the  button to execute the current line of code. Alternatively, you can press **F8**.

If you want to step through the code of a function that is encountered, press the **Step Into** button  or **F7**.

If you would rather continue execution until the next breakpoint is encountered or program exit, press the **Continue** button  or **F5**.

5.5 Examining a complex structure or array


Within the Variable Result Tab, we can expand `customerM` so that we can examine its contents by clicking the  icon. This will then show us the contents of the first item of that array:

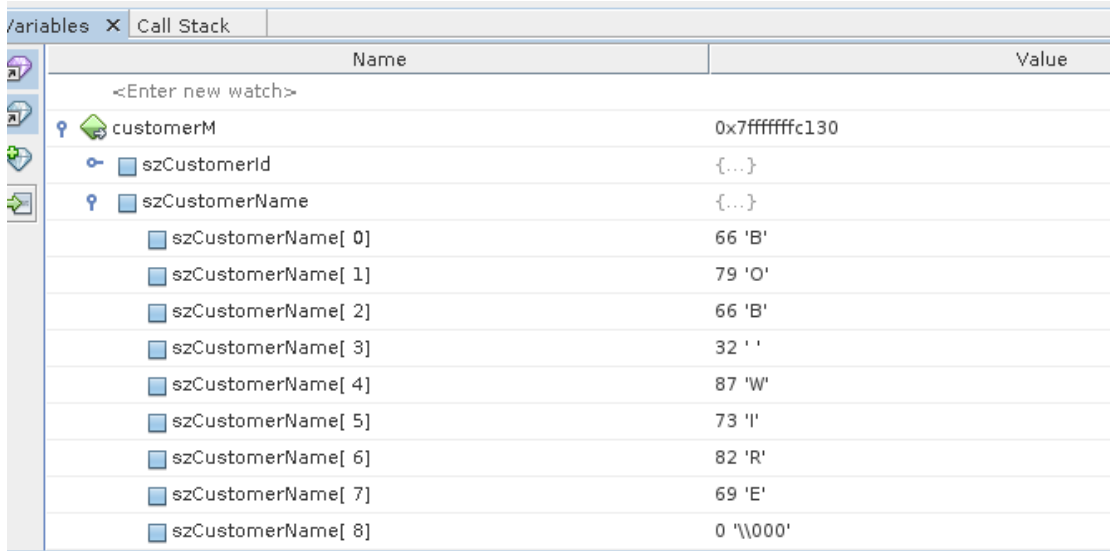


The screenshot shows the NetBeans IDE with a C program open. The source code is displayed in the main window, and the Variable Result Tab is open at the bottom. The code defines a `Customer` structure and a `printCustomerData` function. The function prints the ID and name of each customer, followed by their traits. The Variable Result Tab shows the state of the `customerM` array, which contains 9 elements. The first element is expanded to show its fields: `szCustomerId`, `szCustomerName`, `iNumberOfTraits`, and `traitM`.

```
158         EXERCISE HIKE
159         SMOKING  N
160     22222     MELBA TOAST
161         GENDER  F
162         BOOK    COOKING
163     33333     CRYSTAL BALL
164         SMOKING  N
165         GENDER  F
166         EXERCISE JOG
167         EXERCISE YOGA
168     */
169     void printCustomerData(Customer customerM[], int iNumCustomer)
170     {
171         int i;
172         int j;
173         // Print a heading for the list of customers and traits
174         printf("%-6s %-20s\n", "ID", "Customer Name");
175         printf("%10s %-10s %-12s\n", " ", "Trait", "Value");
176         for (i = 0; i < iNumCustomer; i++)
177         {
178             // Print the customer information
```

Name	Value
<Enter new watch>	
customerM	0x7ffffffc130
szCustomerId	{...}
szCustomerName	{...}
iNumberOfTraits	4
traitM	{...}
iNumCustomer	9

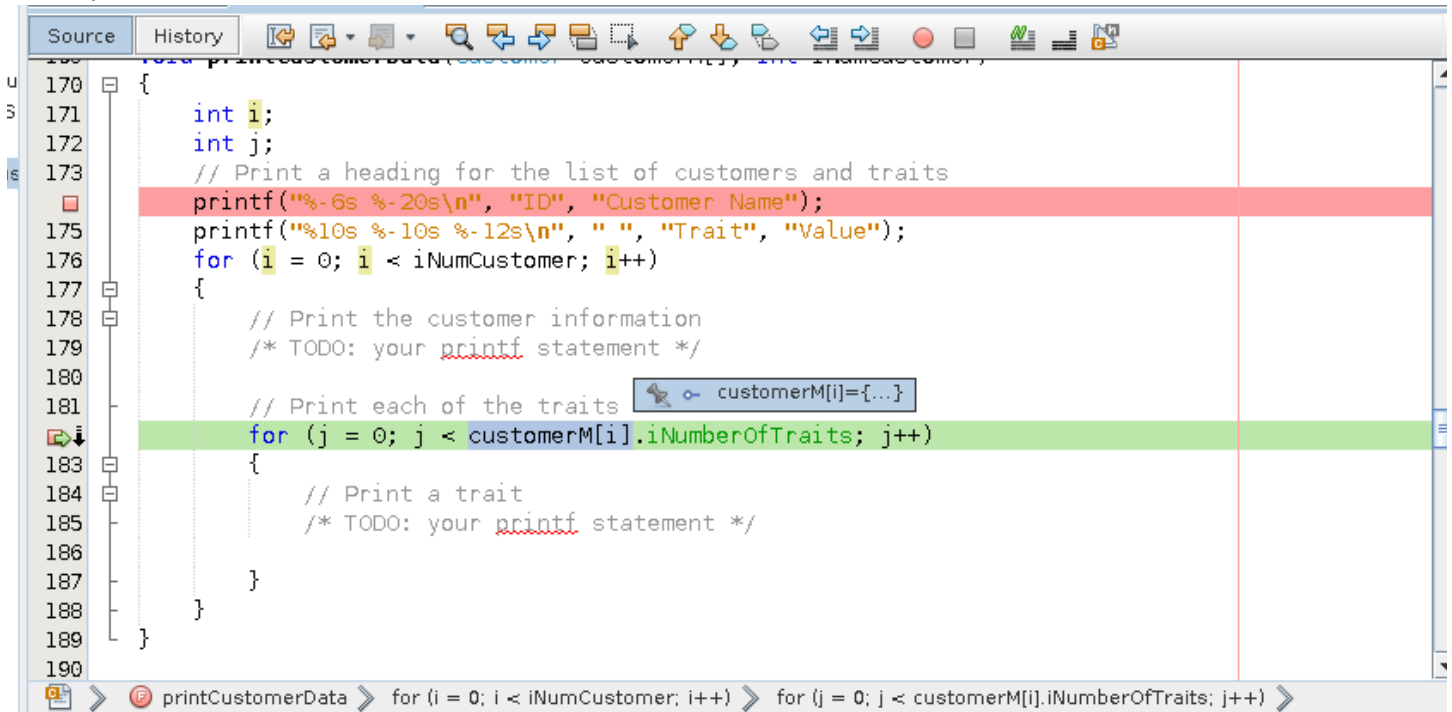
We can then examine the contents of szCustomerName by clicking its  icon. We notice that the value is "BOB WIRE".



The screenshot shows a debugger's variables window with the following data:

Name	Value
<Enter new watch>	
customerM	0x7fffffff130
szCustomerId	{...}
szCustomerName	{...}
szCustomerName[0]	66 'B'
szCustomerName[1]	79 'O'
szCustomerName[2]	66 'B'
szCustomerName[3]	32 ' '
szCustomerName[4]	87 'W'
szCustomerName[5]	73 'I'
szCustomerName[6]	82 'R'
szCustomerName[7]	69 'E'
szCustomerName[8]	0 '\000'

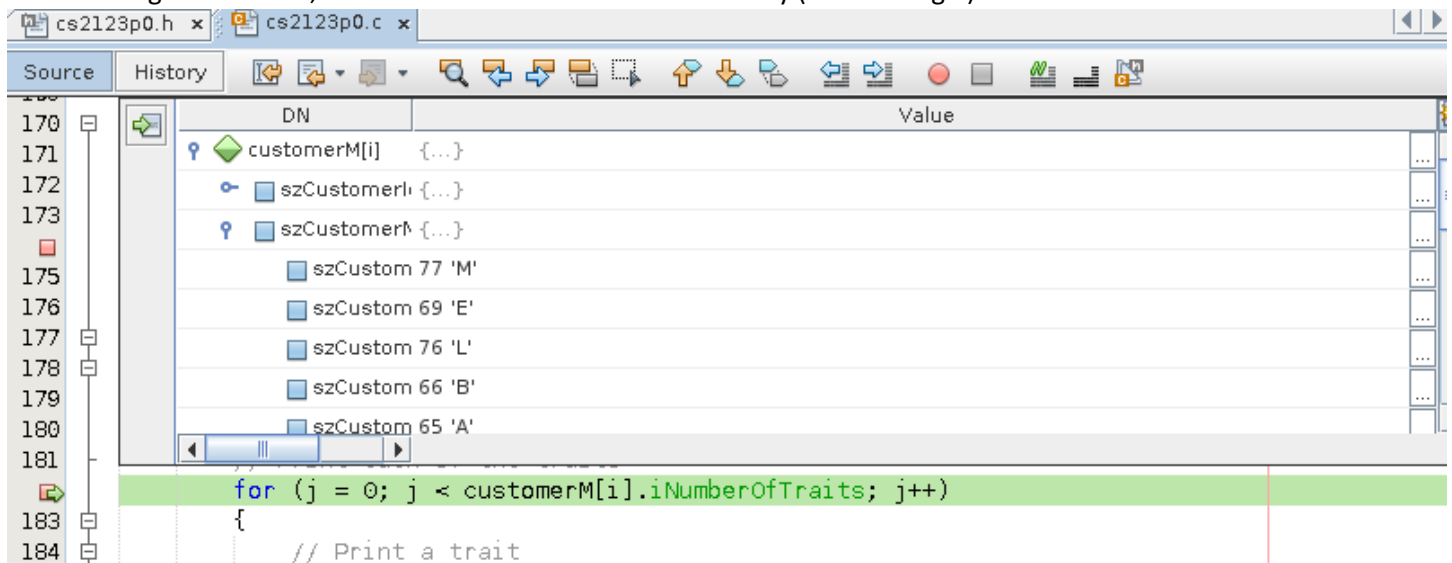
We can also show the array while executing the for statement. This allows us to see particular elements based on the subscript.



```
170 {
171     int i;
172     int j;
173     // Print a heading for the list of customers and traits
174     printf("%-6s %-20s\n", "ID", "Customer Name");
175     printf("%10s %-10s %-12s\n", " ", "Trait", "Value");
176     for (i = 0; i < iNumCustomer; i++)
177     {
178         // Print the customer information
179         /* TODO: your printf statement */
180
181         // Print each of the traits
182         for (j = 0; j < customerM[i].iNumberOfTraits; j++)
183         {
184             // Print a trait
185             /* TODO: your printf statement */
186
187         }
188     }
189 }
190
```

printCustomerData > for (i = 0; i < iNumCustomer; i++) > for (j = 0; j < customerM[i].iNumberOfTraits; j++) >

After clicking the  icon, we see a different element of the array (with i being 1):



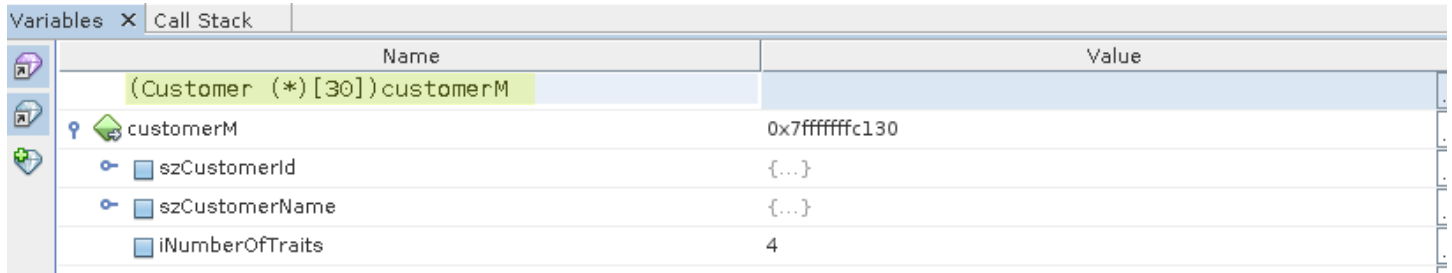
```
170 {
171     int i;
172     int j;
173     // Print a heading for the list of customers and traits
174     printf("%-6s %-20s\n", "ID", "Customer Name");
175     printf("%10s %-10s %-12s\n", " ", "Trait", "Value");
176     for (i = 0; i < iNumCustomer; i++)
177     {
178         // Print the customer information
179         /* TODO: your printf statement */
180
181         // Print each of the traits
182         for (j = 0; j < customerM[i].iNumberOfTraits; j++)
183         {
184             // Print a trait
185             /* TODO: your printf statement */
186
187         }
188     }
189 }
190
```

DN	Value
customerM[i] {...}	
szCustomerI {...}	
szCustomerI {...}	
szCustom 77 'M'	
szCustom 69 'E'	
szCustom 76 'L'	
szCustom 66 'B'	
szCustom 65 'A'	

for (j = 0; j < customerM[i].iNumberOfTraits; j++)
{
 // Print a trait

5.6 Examining an array which is a parameter

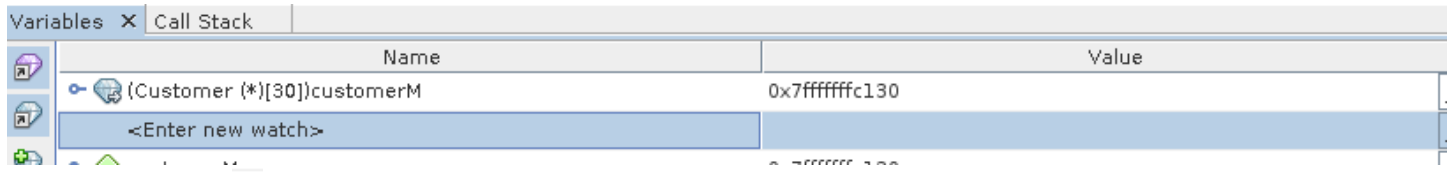
Some IDEs (e.g., Microsoft Visual Studio) have better debugging capability for showing the contents of arrays which are passed as parameters. In the example of 5.5, we only saw only element of the array at a time. We can show the entire array by using a capability of the underlying debugger, gdb. We can enter a **Watch** as shown



The screenshot shows the 'Variables' window in Visual Studio. The 'Call Stack' tab is active. A watch expression `{Customer (*)[30]}customerM` is entered in the 'Name' column. The 'Value' column is empty. Below the watch expression, the variable `customerM` is expanded to show its members: `szCustomerId`, `szCustomerName`, and `iNumberOfTraits`.


Name	Value
<code>{Customer (*)[30]}customerM</code>	
<code>customerM</code>	<code>0x7fffffff130</code>
<code>szCustomerId</code>	<code>{...}</code>
<code>szCustomerName</code>	<code>{...}</code>
<code>iNumberOfTraits</code>	<code>4</code>

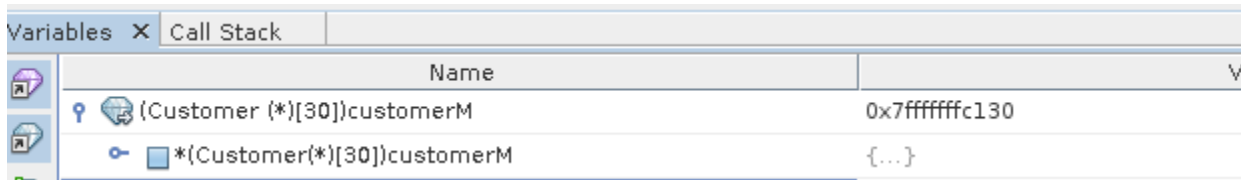
That provided the datatype of the array, a pointer since arrays are passed as addresses, the number of elements to show, and the name of the variable. When we hit enter on that highlighted line, the following appears:



The screenshot shows the 'Variables' window in Visual Studio. The 'Call Stack' tab is active. A new watch expression `<Enter new watch>` is entered in the 'Name' column. The 'Value' column is empty.


Name	Value
<code>{Customer (*)[30]}customerM</code>	<code>0x7fffffff130</code>
<code><Enter new watch></code>	

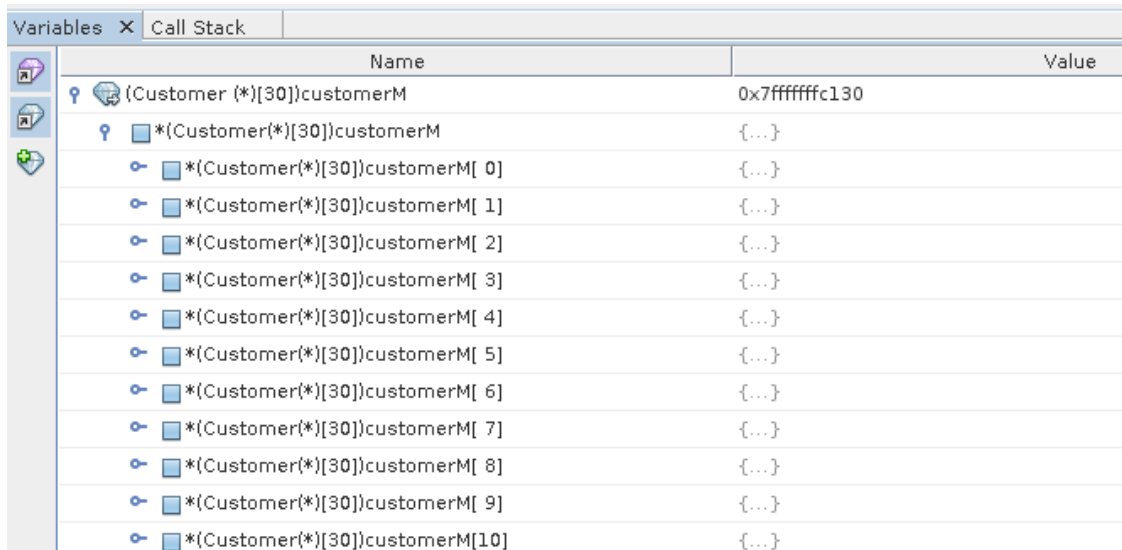
When we click the  icon, we see the array listed, but not expanded.



The screenshot shows the 'Variables' window in Visual Studio. The 'Call Stack' tab is active. The watch expression `{Customer (*)[30]}customerM` is expanded to show the array `*(Customer(*)[30]}customerM`.

Name	Value
<code>{Customer (*)[30]}customerM</code>	<code>0x7fffffff130</code>
<code>*(Customer(*)[30]}customerM</code>	<code>{...}</code>

When we click the  icon, we see the entire array listed.



The screenshot shows the 'Variables' window in Visual Studio. The 'Call Stack' tab is active. The watch expression `{Customer (*)[30]}customerM` is expanded to show the entire array `*(Customer(*)[30]}customerM`. The array is expanded to show 11 elements, from `*(Customer(*)[30]}customerM[0]` to `*(Customer(*)[30]}customerM[10]`.

Name	Value
<code>{Customer (*)[30]}customerM</code>	<code>0x7fffffff130</code>
<code>*(Customer(*)[30]}customerM</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[0]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[1]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[2]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[3]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[4]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[5]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[6]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[7]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[8]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[9]</code>	<code>{...}</code>
<code>*(Customer(*)[30]}customerM[10]</code>	<code>{...}</code>

5.7 Removing a Breakpoint

Click on the line number of the Breakpoint.

6. Exiting Netbeans

After saving your files, you can exit Netbeans, by pressing the **X** in the upper right corner or use the menu **File > Exit**.

7. Fixing Netbeans when it fails

If you incorrectly shut down Netbeans, it could affect your next execution. You can clean up an old execution of netbeans by removing its temporary directory:

```
$ rm -r ~/.netbeans
```

©2019 Larry W. Clark, UTSA CS students may make copies for their personal use