C++ Project Basics
Visual Studio

- Creating a new project
- Adding files to the project
- Building & Running the project
- Debugging with Breakpoints
- Debugging with Stack Trace
- Kludgey debugging with cout

Written by Rachel J. Morris, last updated 2013-06-14

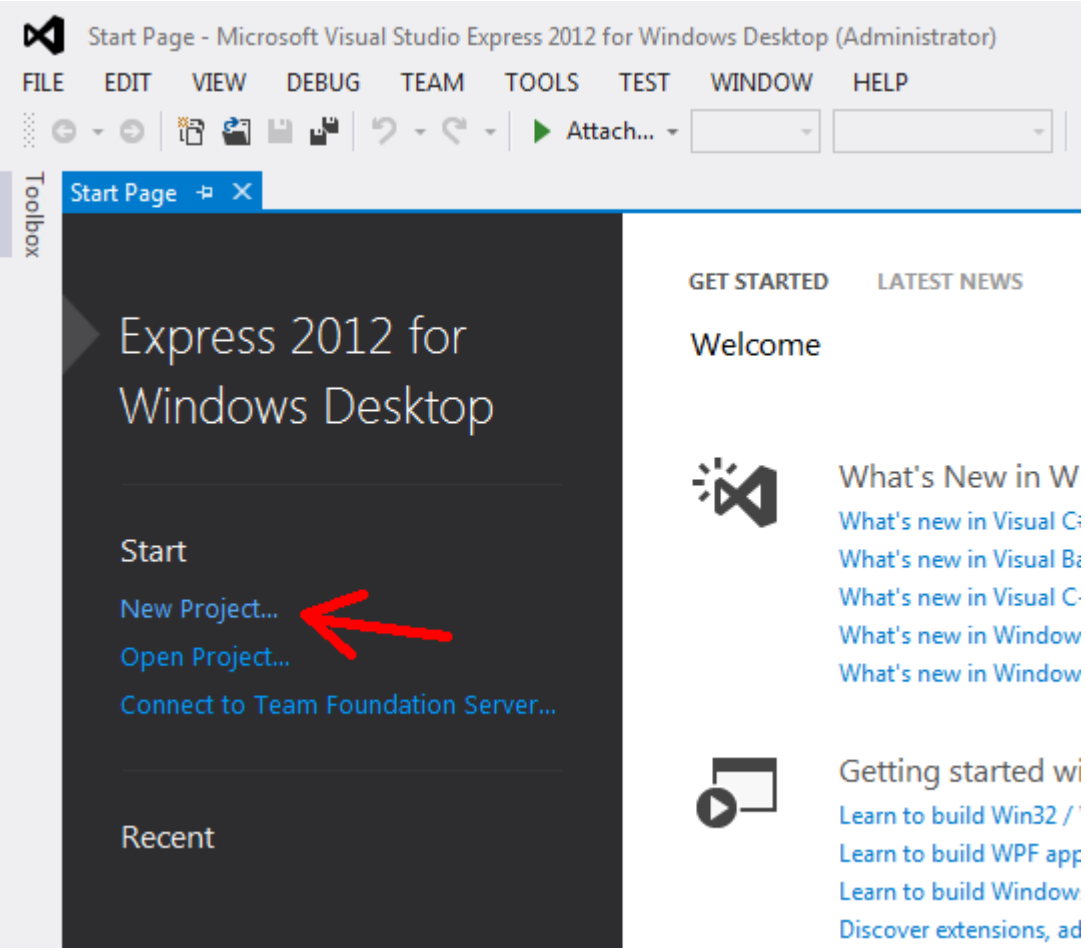
1. Getting Visual Studio

- As a student, you can get a free copy of Visual Studio. See the document on how to on the class webpage, under CS201L Lab documents, open the “Using Microsoft DreamSpark to Download Visual Studio.docx” file.
- Class webpage: https://github.com/Moosader/Problem-Solving-and-Programming-II

1. Getting Visual Studio

- You can also download Visual C++ Express for free at any time from Microsoft.

2. Creating a new Project



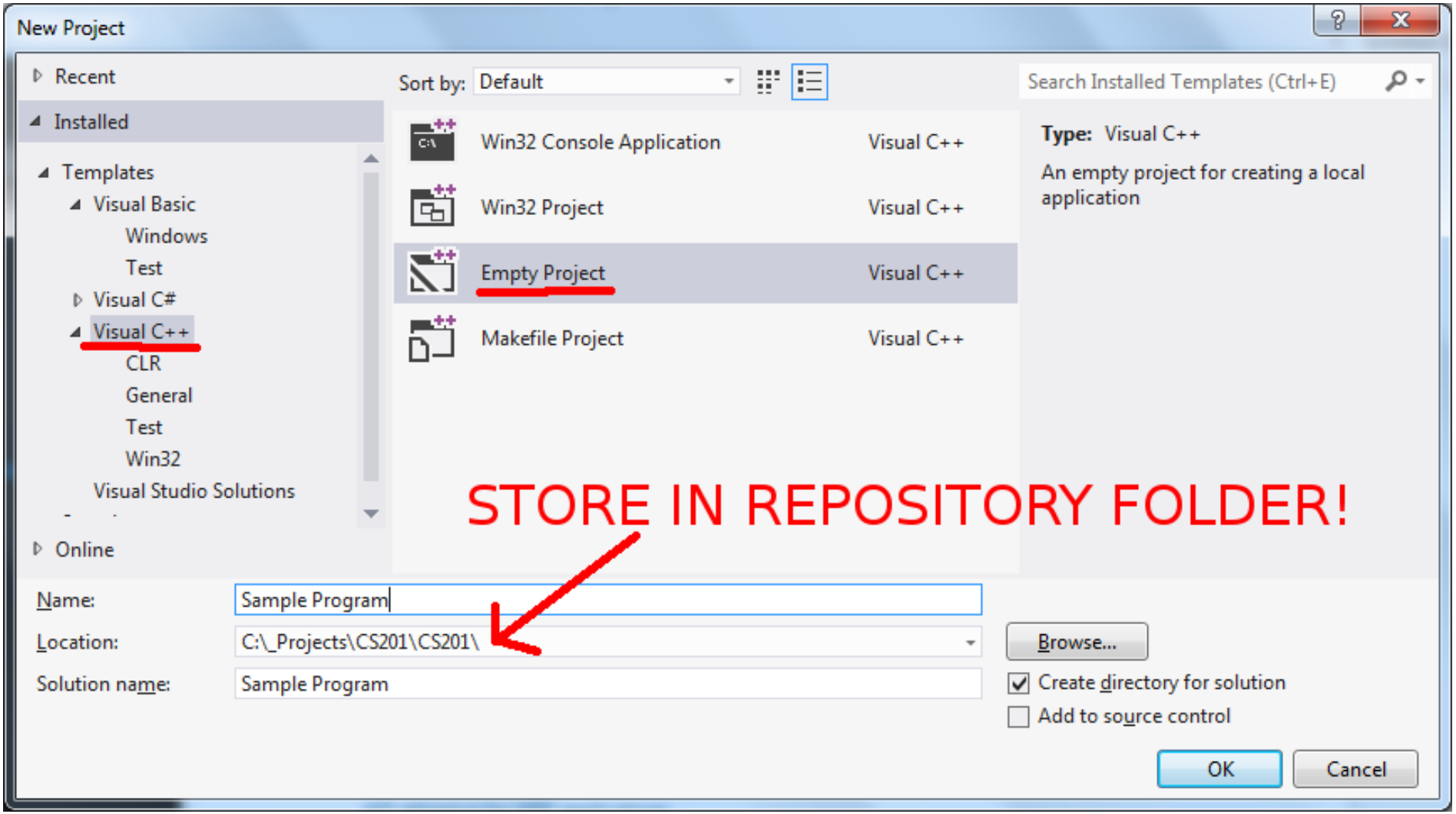
- Open up Visual Studio and click <u>New Project...</u>
- I am using Visual Studio Express 2012 here.

2. Creating a new Project

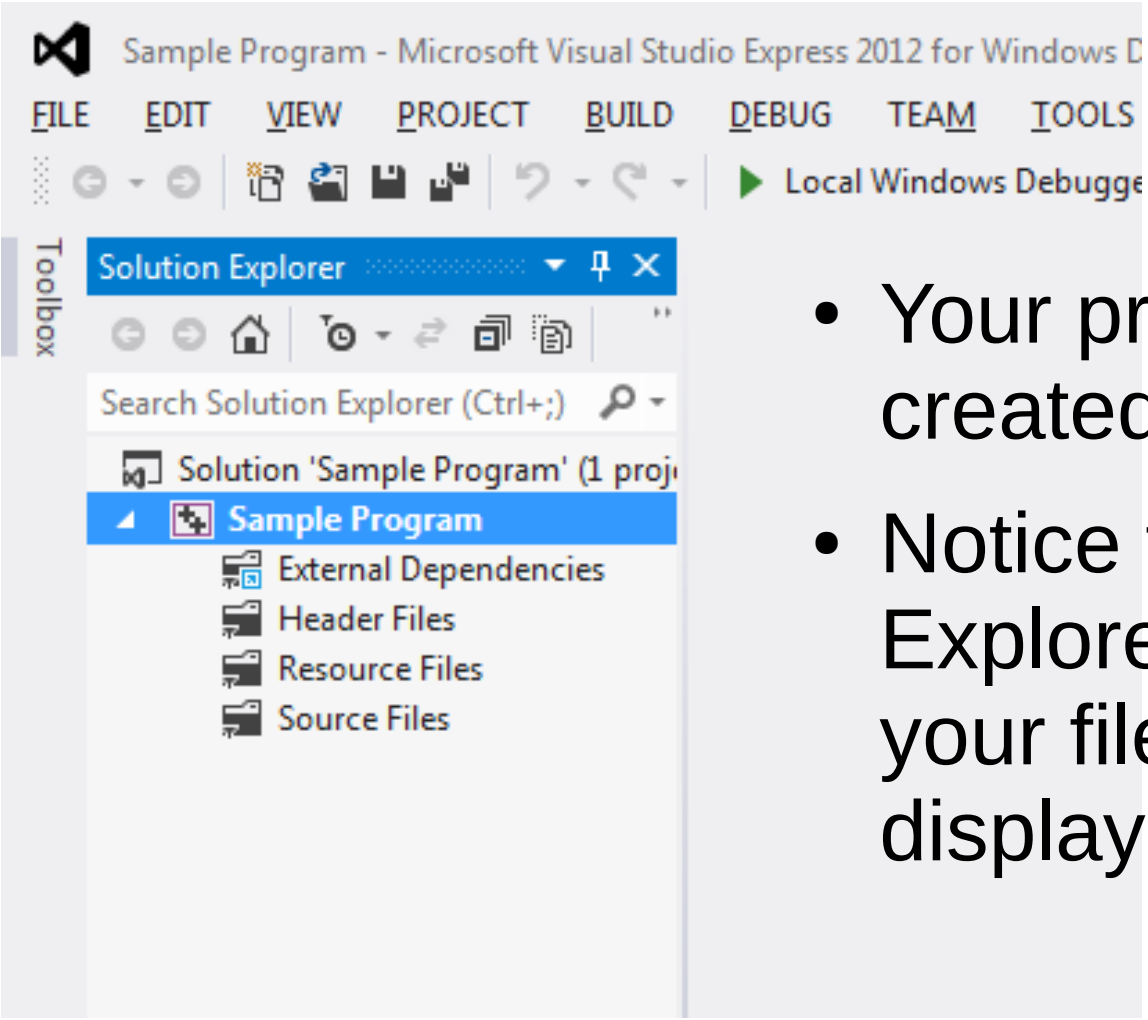
- Under Visual C++ templates, select Empty Project.
- Update the Location field -

THIS SHOULD BE INSIDE OF YOUR PROJECT REPOSITORY FOLDER!

2. Creating a new Project



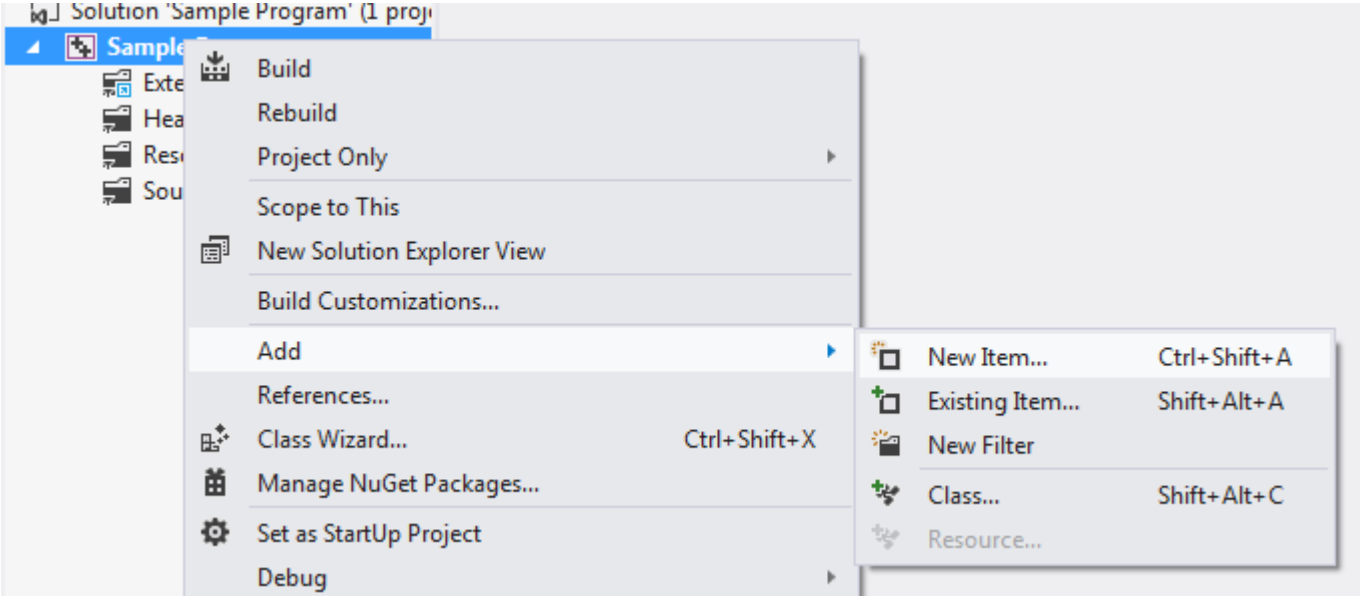
2. Creating a new Project



- Your project will be created without any files.
- Notice the Solution Explorer, this is where your files will be displayed.

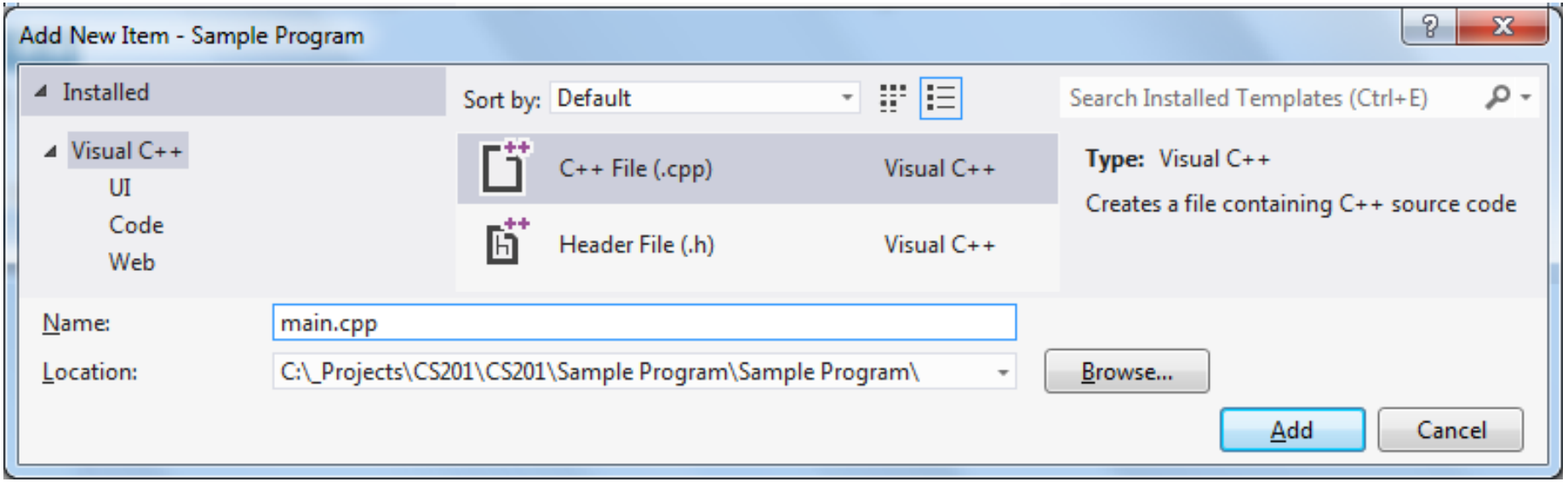
3. Adding Files

- To add a source file to the project, right-click the project, go to Add and select New Item...
- The keyboard shortcut is CTRL+SHIFT+A



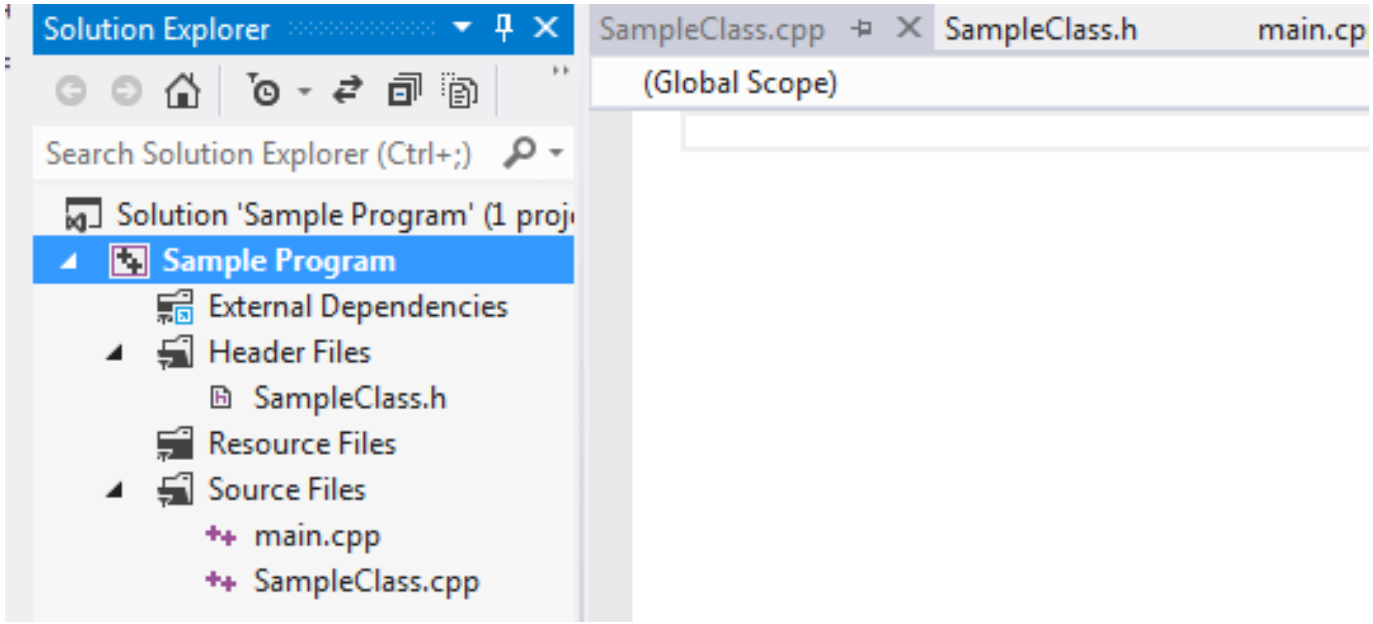
3. Adding Files

- Under Visual C++ on the left, select C++ File (.cpp).
- If we were creating a new class, you would use Header File (.h) as well.



3. Adding Files

- These files will be categorized based on their file type (source or header), though in Windows Explorer they will be in the same folder.



4. Creating a small program

- Open main.cpp and enter some code:

4. Creating a small program

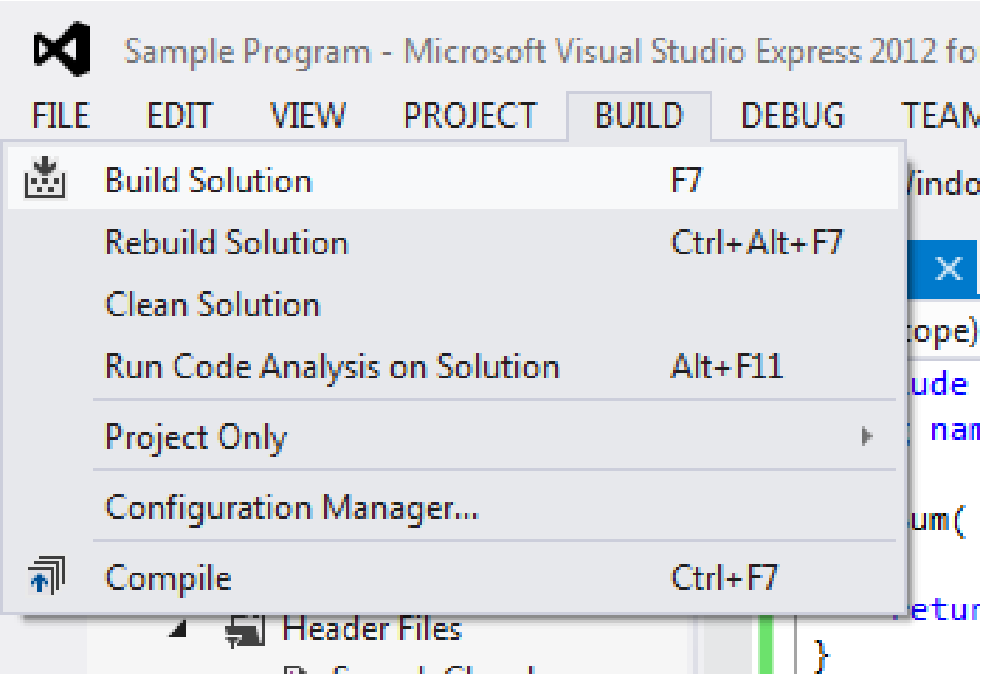
main.cpp

(Global Scope)

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

int Sum( int number1, int number2 )
{
    return number1 + number2;
}

int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
    cout << "Number 2? ";
    cin >> number2;

    cout << "Sum is " << Sum( number1, number2 ) << endl;

    return 0;
}
```

5. Building & running your program

- Build the program by going to BUILD and select Build Solution
- The keyboard shortcut is F7 in VS 2012



5. Building & running your program

- If there are no errors, the Output window at the bottom of the screen should say 1 succeeded

```
        return 0;
    }
```

100 %

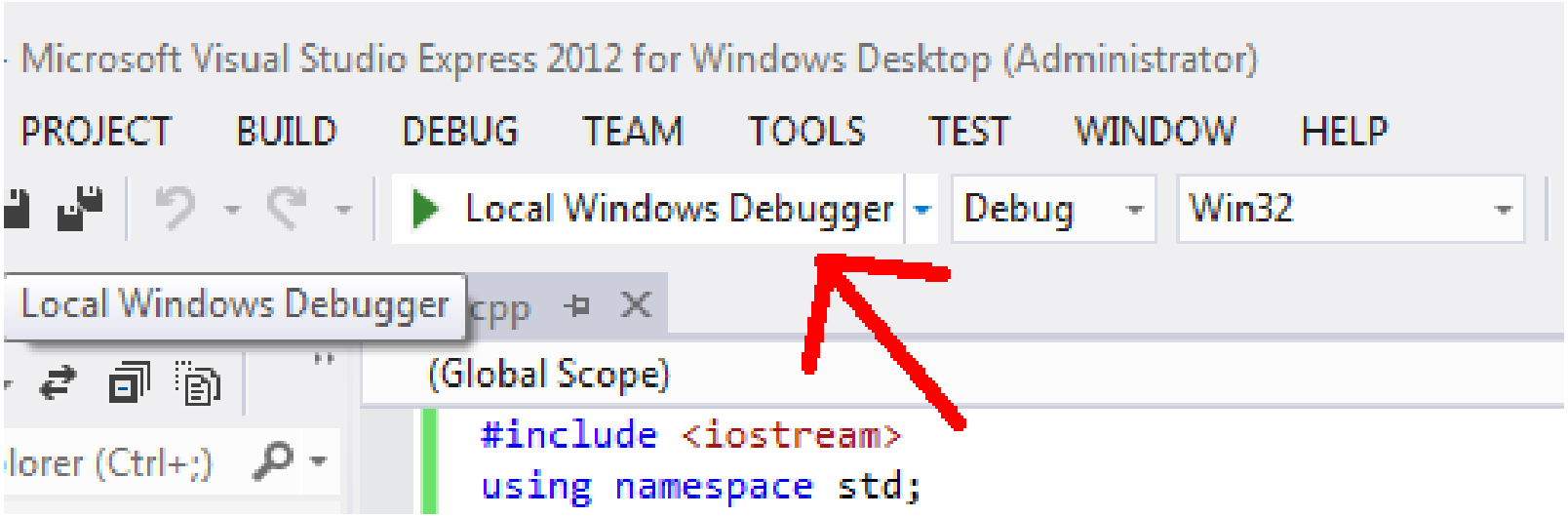
Output

Show output from: Build

```
1>------ Build started: Project: Sample Program, Configuration: Debug Win32 ------
1>  SampleClass.cpp
1>  main.cpp
1>  Generating Code...
1>  Sample Program.vcxproj -> C:\_Projects\CS201\CS201\Sample Program\Debug\Sample Program.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

5. Building & running your program

- You can use the play button to run the program.
- The program will quit automatically when it hits return 0;



6. Debugging with Breakpoints

- At some point, you might have a non-obvious bug in your program you need to debug.
- Visual Studio (and other IDEs) have debugging tools available to make it easier.

6. Debugging with Breakpoints

- You can activate a breakpoint by double-clicking in the margin next to your code.

```
{
    return number1 + number2
}

int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
```

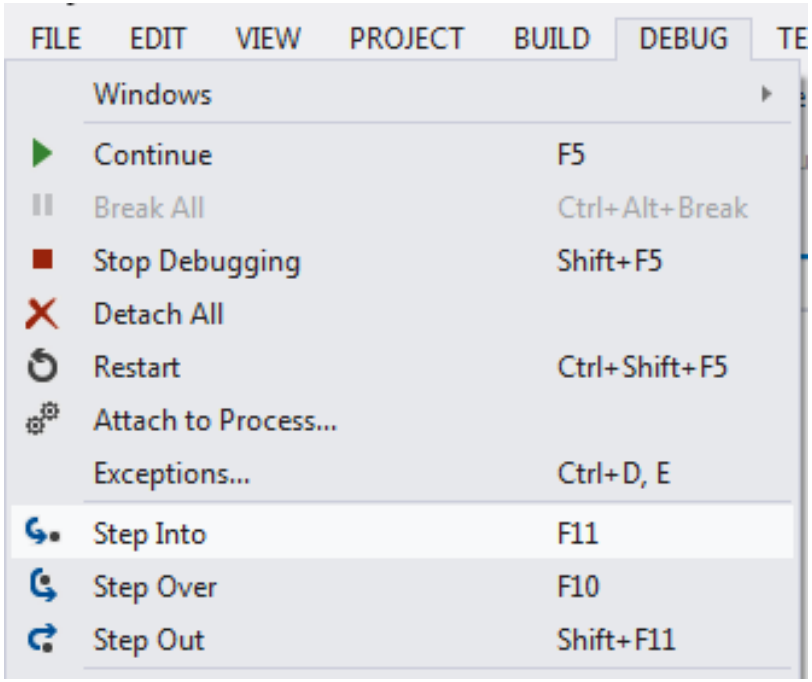
6. Debugging with Breakpoints

- When you run your program, it will start up as normal, but once it hits a breakpoint it will pause the program execution and bring you to the IDE.

```
int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
    cout << "Number 2? ";
    cin >> number2;

    cout << "Sum is " << Sum( numb
```

6. Debugging with Breakpoints

- You can continue stepping through the program execution line-by-line using the <u>DEBUG</u> menu.



- Step Into will take you into a function if called, or a block of code.
- Step Over will go over the line of code to the next line.
- Step Out will take you outside of a function or block.

6. Debugging with Breakpoints

- The Locals window will show you variables that are ***local to the current function***.

```
int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
    cout << "Number 2? ";
    cin >> number2;

    cout << "Sum is " << Sum( number1, number2 ) << endl;
```

100 %

Locals

Name	Value	Type
number2	-858993460	int
number1	-858993460	int

6. Debugging with Breakpoints

- At this point in the program, number1 has been declared but the cin >> number1; command has not executed.
- Both variables have “garbage” for their values.

```
int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
    cout << "Number 2? ";
    cin >> number2;

    cout << "Sum is " << Sum( number1, number2 ) << endl;
```

100 %

Locals

Name	Value	Type
number2	-858993460	int
number1	-858993460	int

Garbage

6. Debugging with Breakpoints

- If we use “Step Over” (then go back to the program and enter a number), the Locals window will be updated with the new value.

```
int main()
{
    int number1;
    cout << "Number 1? ";
    cin >> number1;

    int number2;
    cout << "Number 2? ";
    cin >> number2;

    cout << "Sum is " << Sum( number1, number2 ) << endl;
```

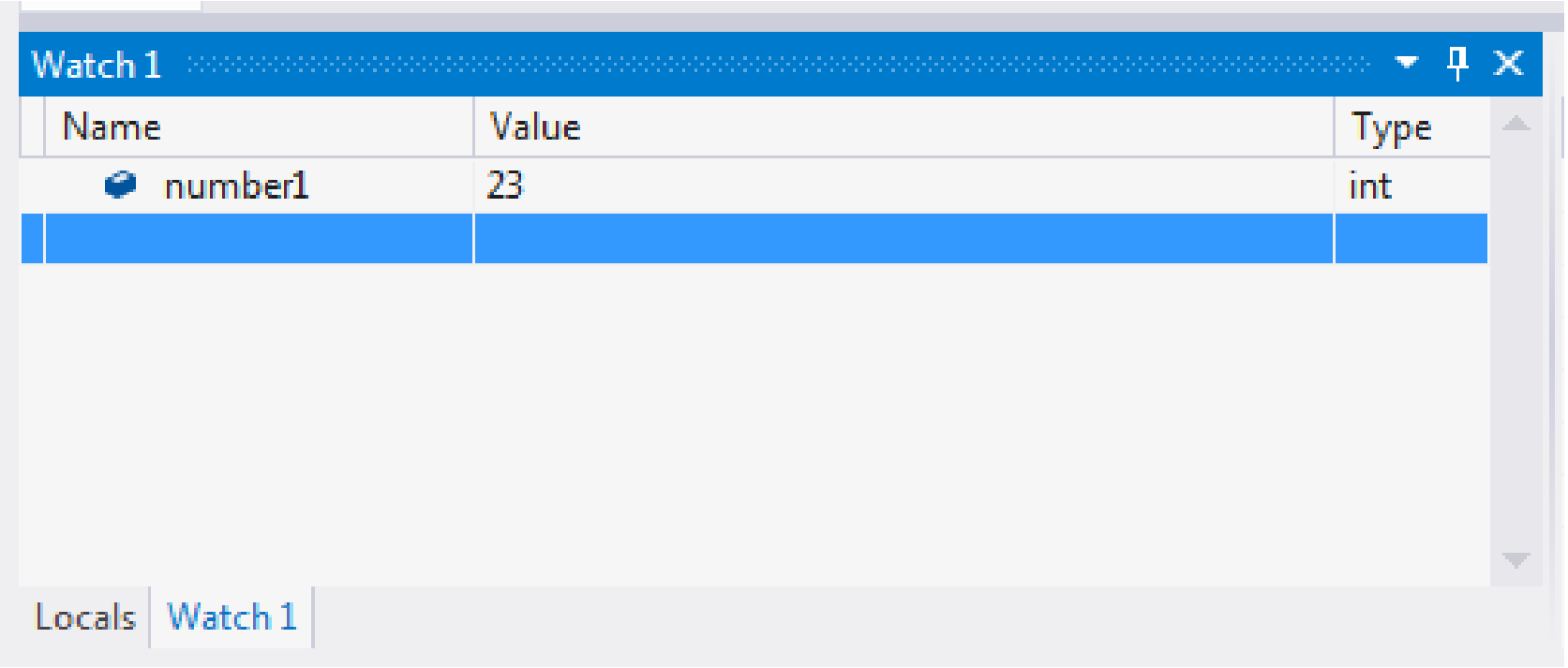
100 %

Locals

Name	Value	Type
number2	-858993460	int
number1	23	int

6. Debugging with Breakpoints

- In the same pane as the Locals window is the Watch tab. Here, you can type in a variable's name and keep track of its value as the program runs.



7. Debugging with the Stack Frame

main.cpp

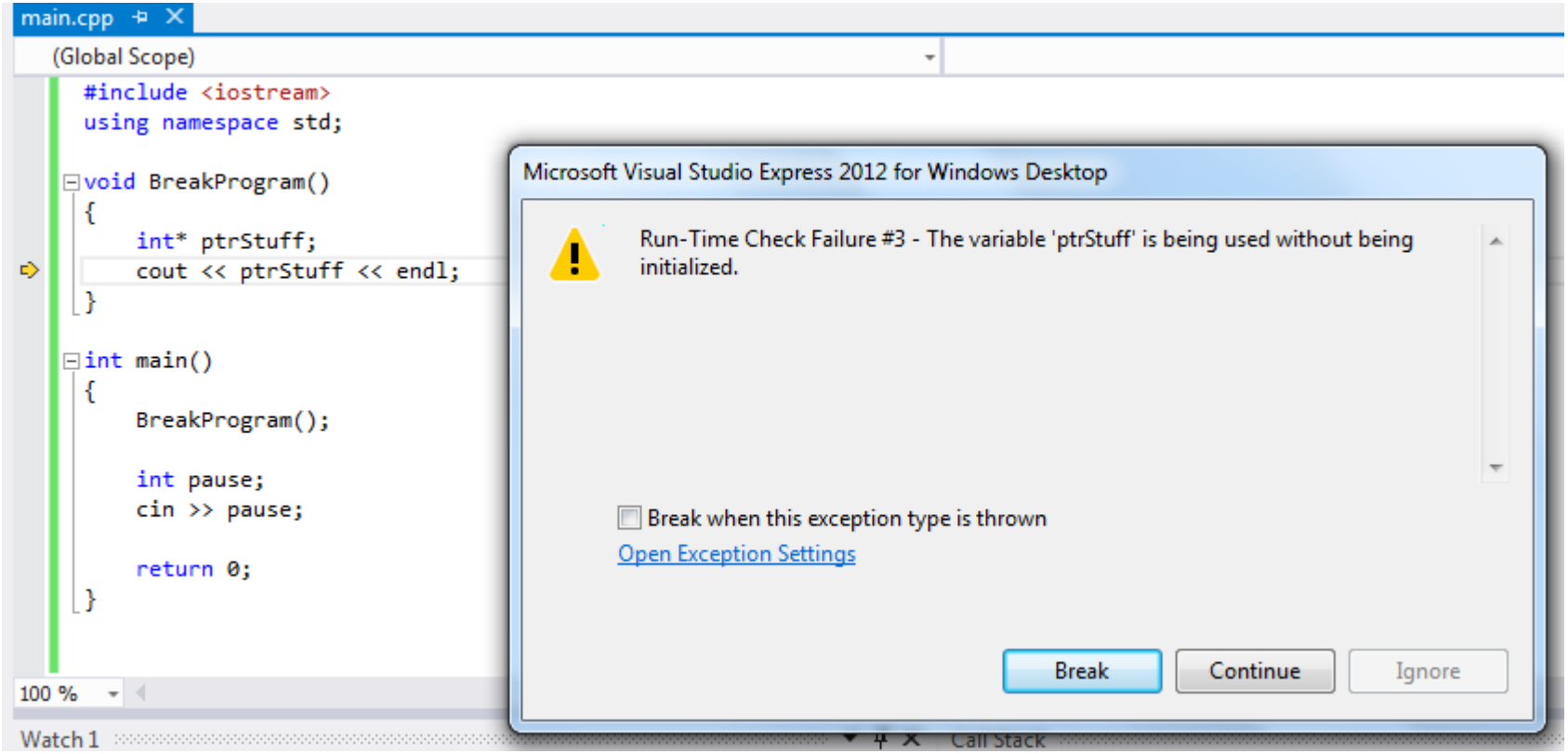
(Global Scope)

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

void BreakProgram()
{
    int* ptrStuff;
    cout << ptrStuff << endl;
}

int main()
{
    BreakProgram();

    int pause;
    cin >> pause;

    return 0;
}
```

- Here is the new program. It will crash.

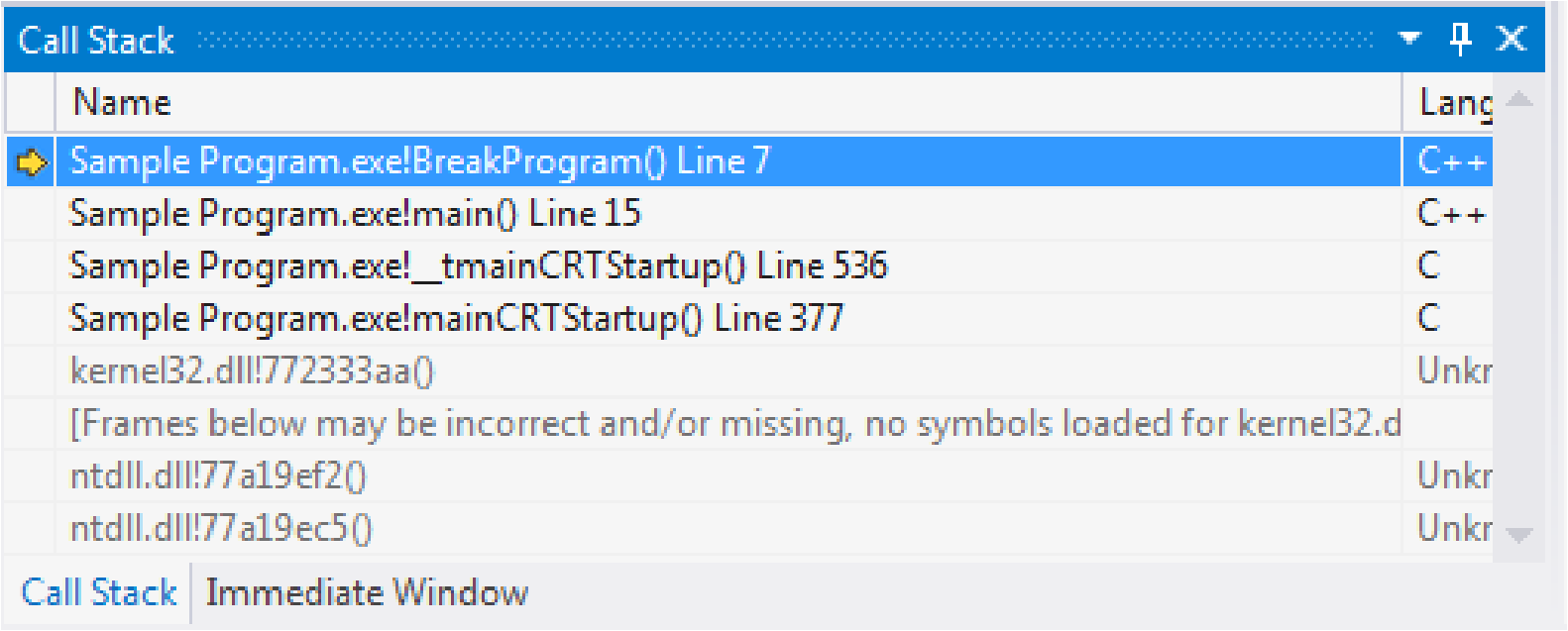
7. Debugging with the Stack Frame

- If we run the program, it will begin, crash, and then bring us to the IDE.



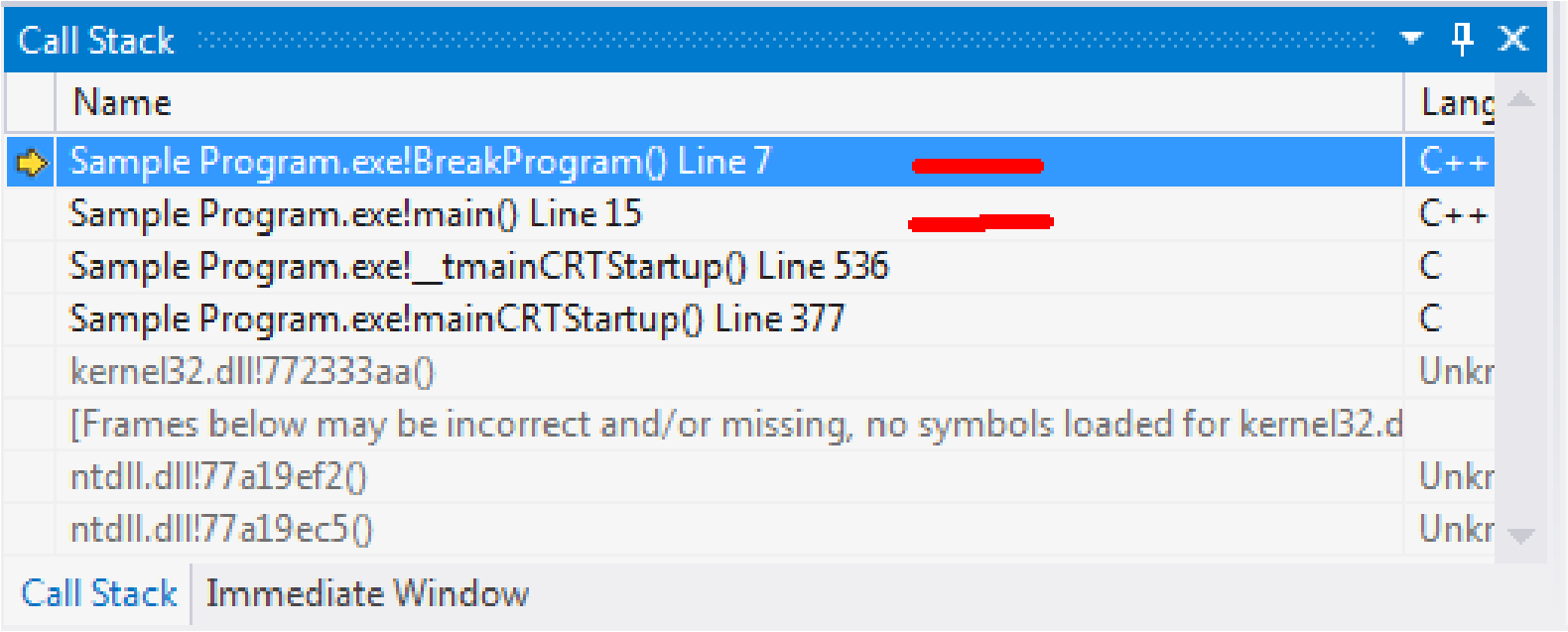
7. Debugging with the Stack Frame

- If you click Break, you can view the Call Stack in Visual Studio. This shows a list of functions called in the program thus far.



7. Debugging with the Stack Frame

- We can see that BreakProgram() was the most recent function called
- Under that, we were in main()



7. Debugging with the Stack Frame

- The Stack Trace is a handy way to figure out the path the program had taken prior to crashing, or prior to your breakpoint pausing the program.

8. Debugging with cout

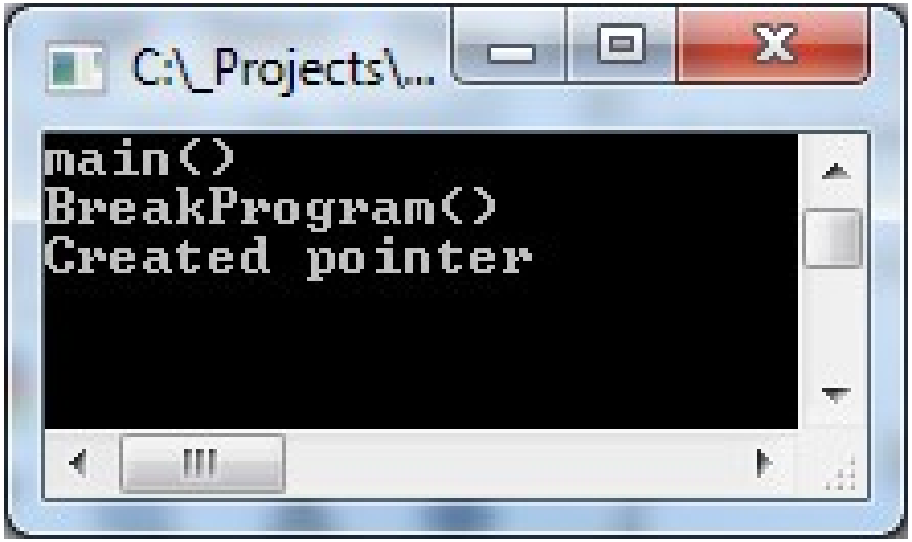
- If you want to be lazy, you can try to pinpoint where your program broke just by outputting information to the console at certain points to follow the program flow.

8. Debugging with cout

main.cpp

(Global Scope)

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

void BreakProgram()
{
    cout << "BreakProgram()" << endl;
    int* ptrStuff;
    cout << "Created pointer" << endl;
    cout << ptrStuff << endl;
    cout << "Output pointer" << endl;
}

int main()
{
    cout << "main()" << endl;
    BreakProgram();

    return 0;
}
```



We can tell that the “Output pointer” text was never printed to the screen, so the crash happened between “Created pointer” and “Output pointer”.