

Preprocessor

Topics

- Preprocessor
- Only C from now on
- Testing Functions
- In these slides, some topics are marked FYI
 - These are non-primary topics; skim through them

The Preprocessor

- Do a quick read of the book section on the Preprocessor. Skim through it for now.
- Preprocessing occurs before program compiles
 - Inclusion of external files
 - Definition of symbolic constants
 - Macros
 - Conditional compilation
 - Conditional execution
- All directives begin with #
 - Can only have whitespace before directives
- Directives are not C statements
 - Do not end with ;

The `#include` Directive

- **`#include`** directive
 - Puts copy of file in place of directive
 - Two forms
 - **`#include <filename>`**
 - For standard library header files
 - Searches pre-designated directories
 - **`#include "filename"`**
 - Searches in current directory where exe file is
 - Normally used for programmer-defined files

#include

- Now that you know, use
`#include <stdio.h>`

#define

- In the book this section starts with **Warning!**
- **#define**
 - Symbolic constants
 - Constants represented as symbols
 - When program compiled, all occurrences replaced
 - Format
 - **#define *identifier replacement-text***
 - **#define PI 3.14159**
 - Everything to right of identifier replaces text
 - **#define PI=3.14159**
 - Replaces **PI** with **"=3.14159"**
 - Probably an error!

#define

- Advantages
 - Takes no memory
- Disadvantages
 - Name not seen by debugger (only replacement text)
 - Do not have specific data type
- **const** variables preferred

Other Directives

- In the next few slides, an FYI marked slide means you can read it in passing.

Macros

- Macro
 - Operation specified in **#define**
 - Intended for legacy C programs
 - Macro without arguments
 - Treated like a symbolic constant
 - Macro with arguments
 - Arguments substituted for replacement text
 - Macro expanded
 - Performs a text substitution
 - No data type checking

#define : Macros – Quick Pass

- Example

```
#define CIRCLE_AREA( x ) ( PI * ( x ) * ( x ) )  
area = CIRCLE_AREA( 4 );
```

becomes

```
area = ( 3.14159 * ( 4 ) * ( 4 ) );
```

- Use parentheses

– Without them,

```
#define CIRCLE_AREA( x ) PI * x * x  
area = CIRCLE_AREA( c + 2 );
```

becomes

```
area = 3.14159 * c + 2 * c + 2;
```

which evaluates incorrectly

The **#error** and **#pragma** Preprocessor Directives – FYI

- **#error *tokens***
 - Prints implementation-dependent message
 - Tokens are groups of characters separated by spaces
 - **#error 1 - Out of range error** has 6 tokens
 - Compilation may stop (depends on compiler)
- **#pragma *tokens***
 - Actions depend on compiler
 - May use compiler-specific options
 - Unrecognized **#pragmas** are ignored

The # and ## Operators - FYI

- **# operator**
 - Replacement text token converted to string with quotes

```
#define HELLO( x ) cout << "Hello, " #x << endl;
```

 - **HELLO(JOHN)** becomes
 - `cout << "Hello, " "John" << endl;`
 - Same as `cout << "Hello, John" << endl;`
- **## operator**
 - Concatenates two tokens

```
#define TOKENCONCAT( x, y ) x ## y
```

 - **TOKENCONCAT(O, K)** becomes
 - `OK`

Line Numbers – FYI

- **#line**
 - Renumbers subsequent code lines, starting with integer
 - **#line 100**
 - File name can be included
 - **#line 100 "file1.cpp"**
 - Next source code line is numbered 100
 - For error purposes, file name is "**file1.cpp**"
 - Can make syntax errors more meaningful
 - Line numbers do not appear in source file

Predefined Symbolic Constants

- Though for now this FYI, in later units or future courses, come back to it.
- Useful predefined symbolic constants
 - `__FILE__` => The name of the current file, as a string literal
 - `__LINE__` => Current line of the source file, as a numeric literal
 - `__DATE__` => Current system date, as a string
 - `__TIME__` => Current system time, as a string
 - `__TIMESTAMP__` => Date and time (non-standard)
 - Cannot be used in **`#define`** or **`#undef`**

Assertions - FYI

- **assert** is a macro
 - Header **<cassert>**
 - Tests value of an expression
 - If 0 (**false**) prints error message, calls **abort**
 - Terminates program, prints line number and file
 - Good for checking for illegal values
 - If 1 (**true**), program continues as normal
 - **assert(x <= 10);**
- To remove **assert** statements
 - No need to delete them manually
 - **#define NDEBUG**
 - All subsequent **assert** statements ignored

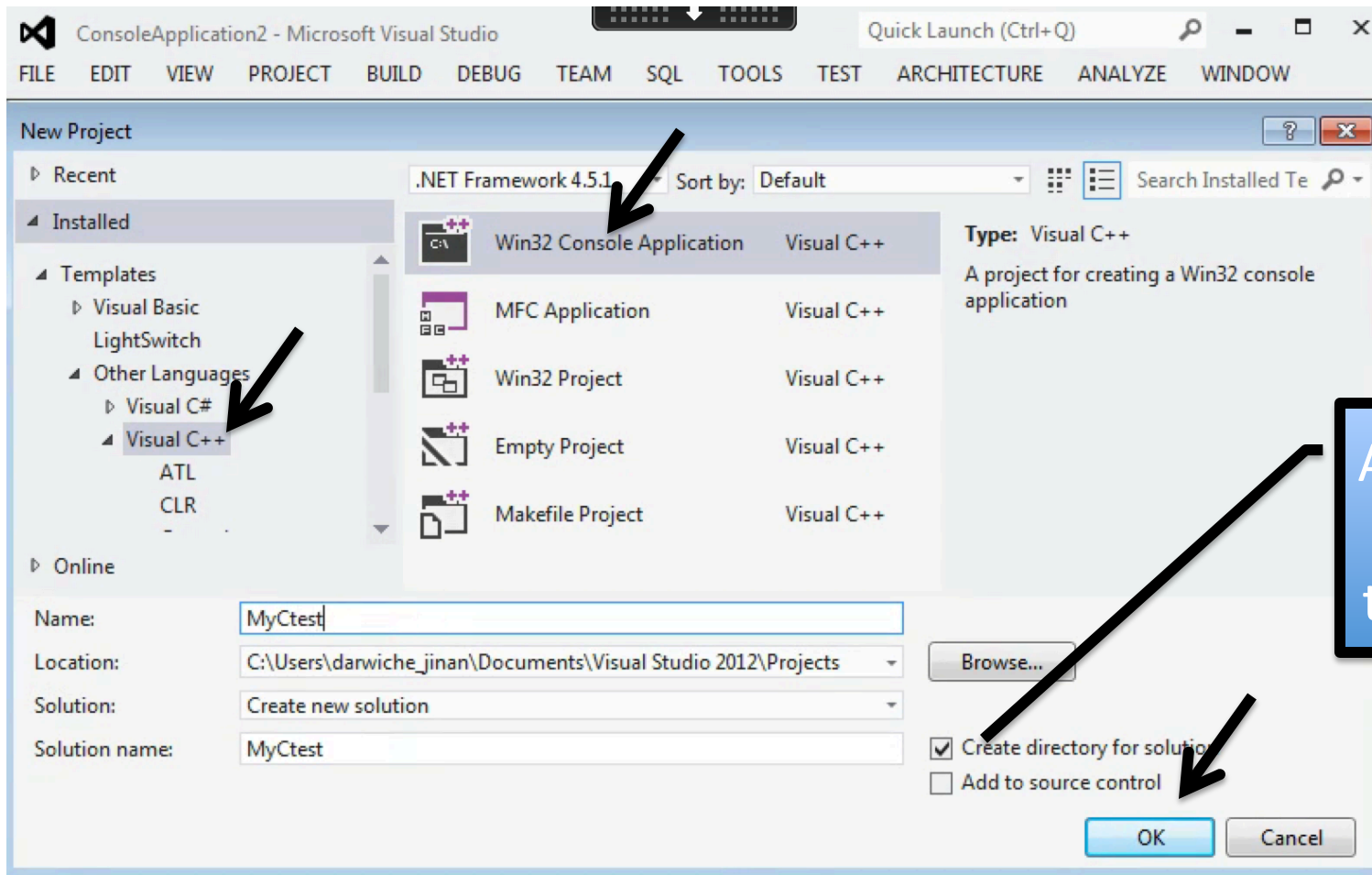
This Is Important:

Only C from now on

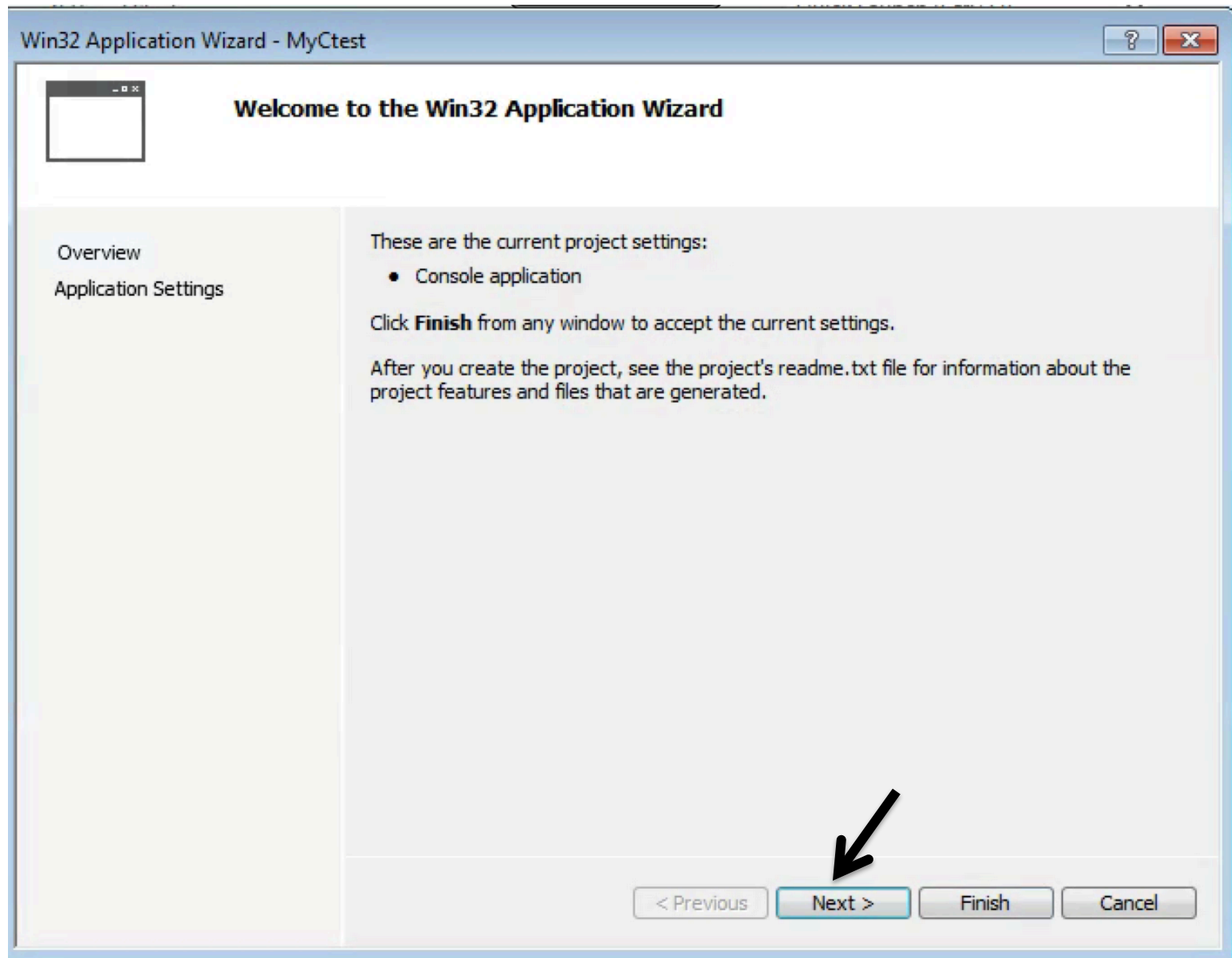
- Now that you are more comfortable with VS, and
- You know what the directives are
- Direct VS to accept only C code and not accept C++ code
- See next slide for details

From now on: Only C

- To create new project, select Project New



Next screen choose Next





Application Settings

Overview

Application Settings

Application type:

- ☐ Windows application
- ☒ Console application
- ☐ DLL
- ☐ Static library

Add common header files for:

- ☐ ATL
- ☐ MFC

Additional options:

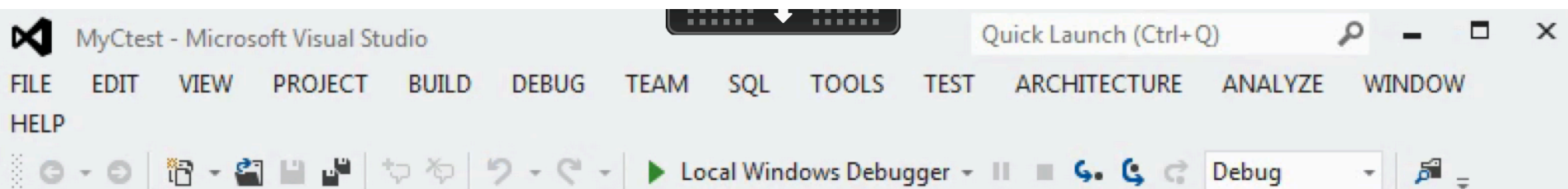
- ☒ Empy project
- ☐ Export symbols
- ☒ Precompiled header
- ☒ Security Development Lifecycle (SDL) checks

< Previous

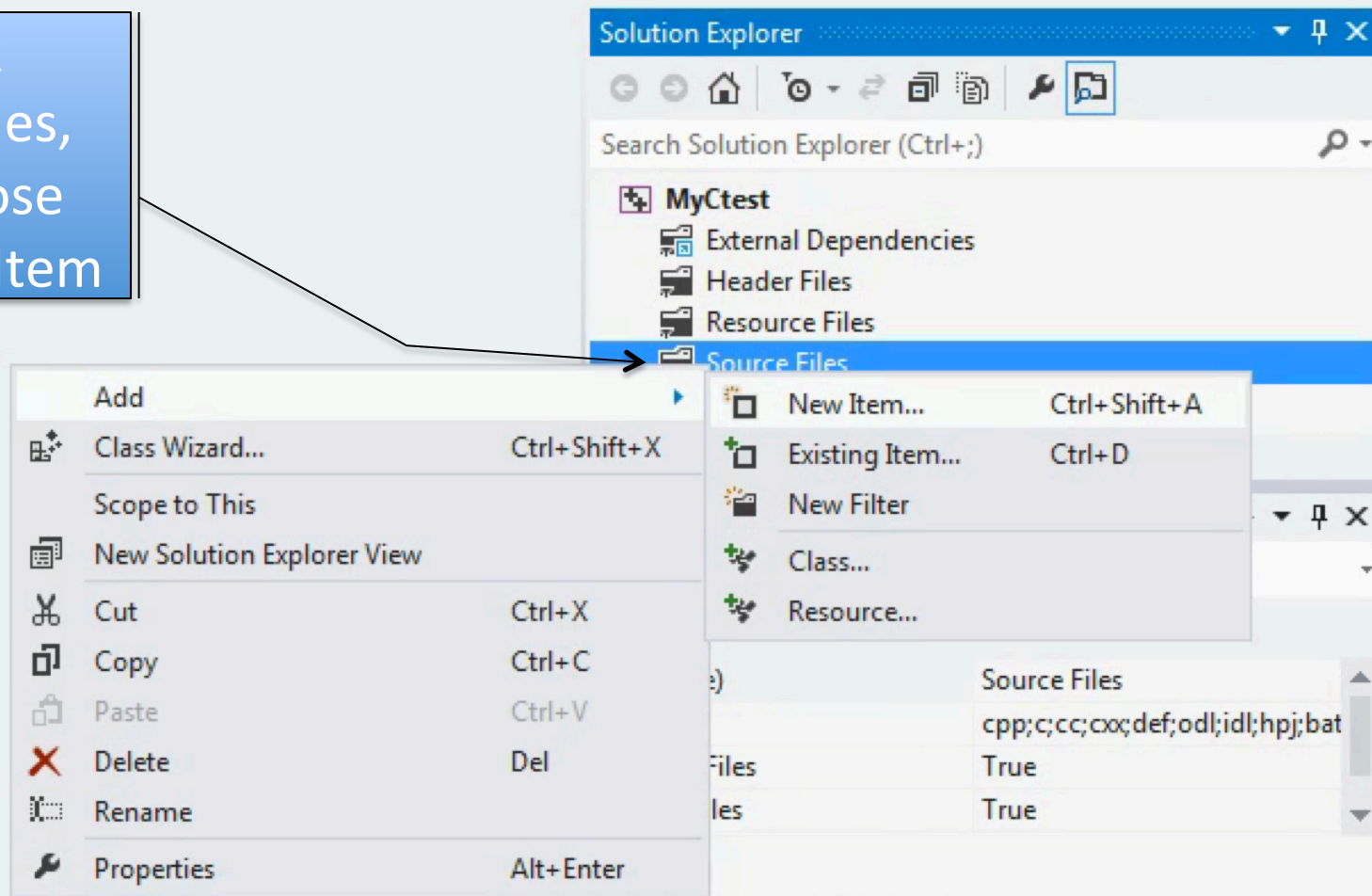
Next >

Finish

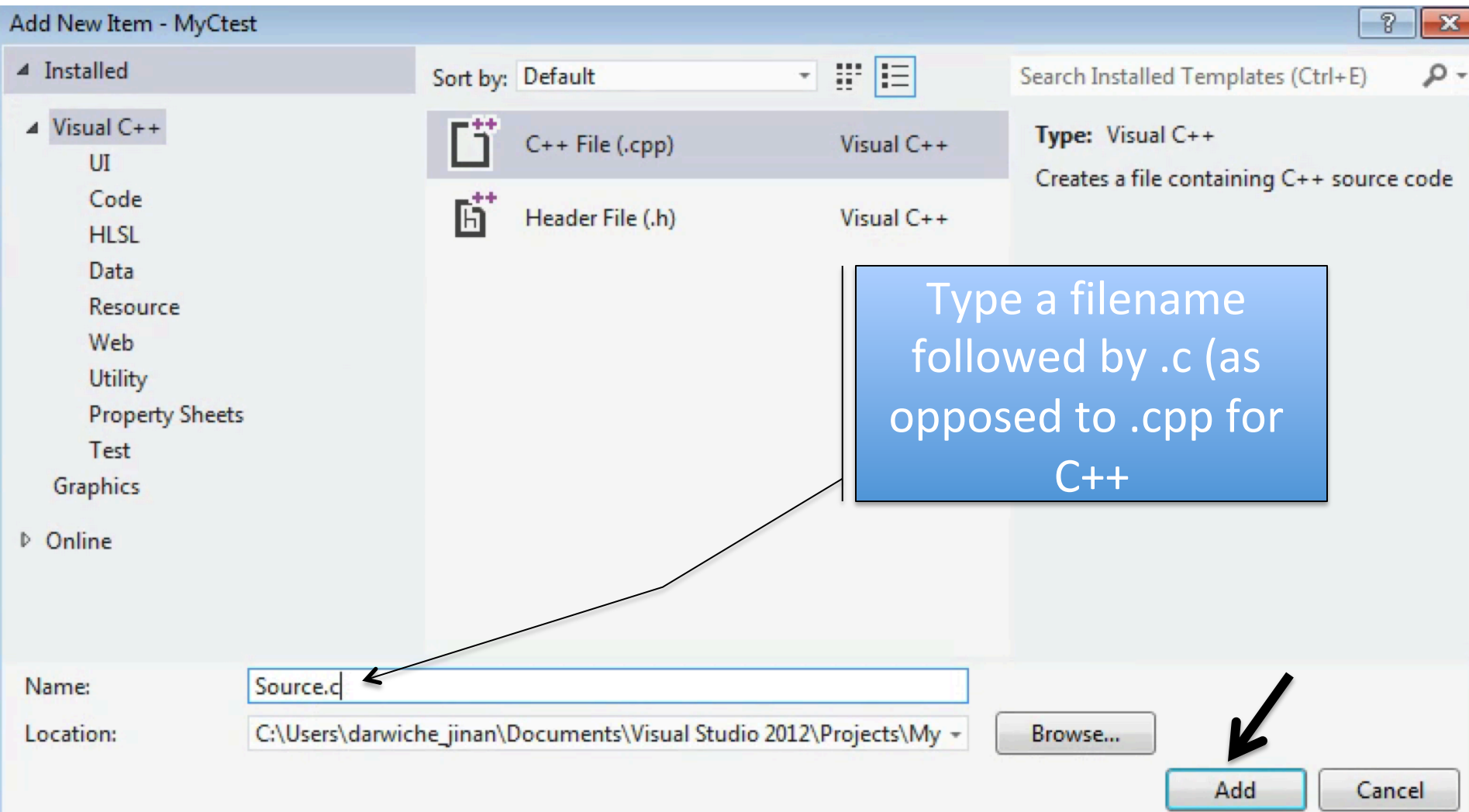
Cancel



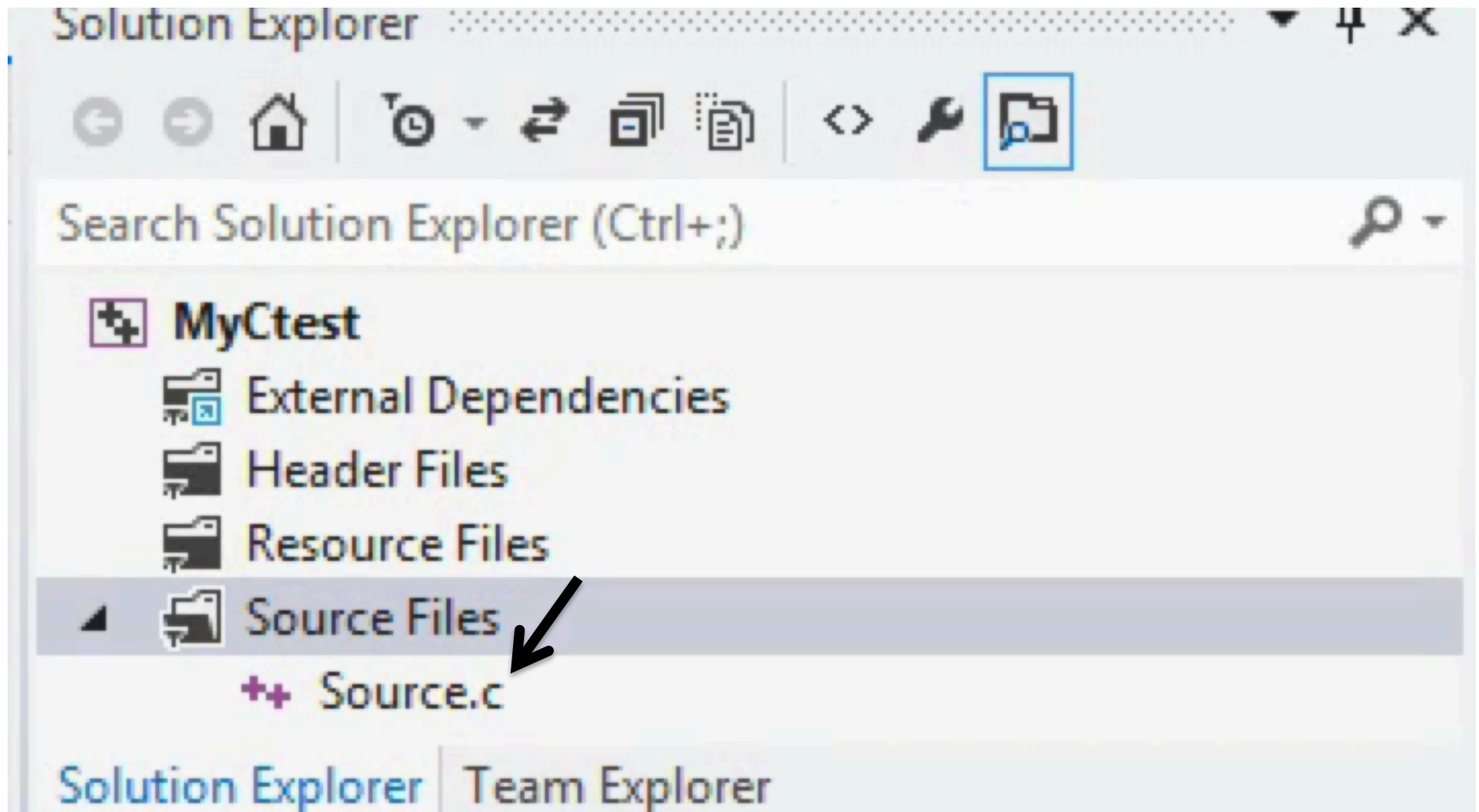
R-Click
Source Files,
then choose
Add, New Item



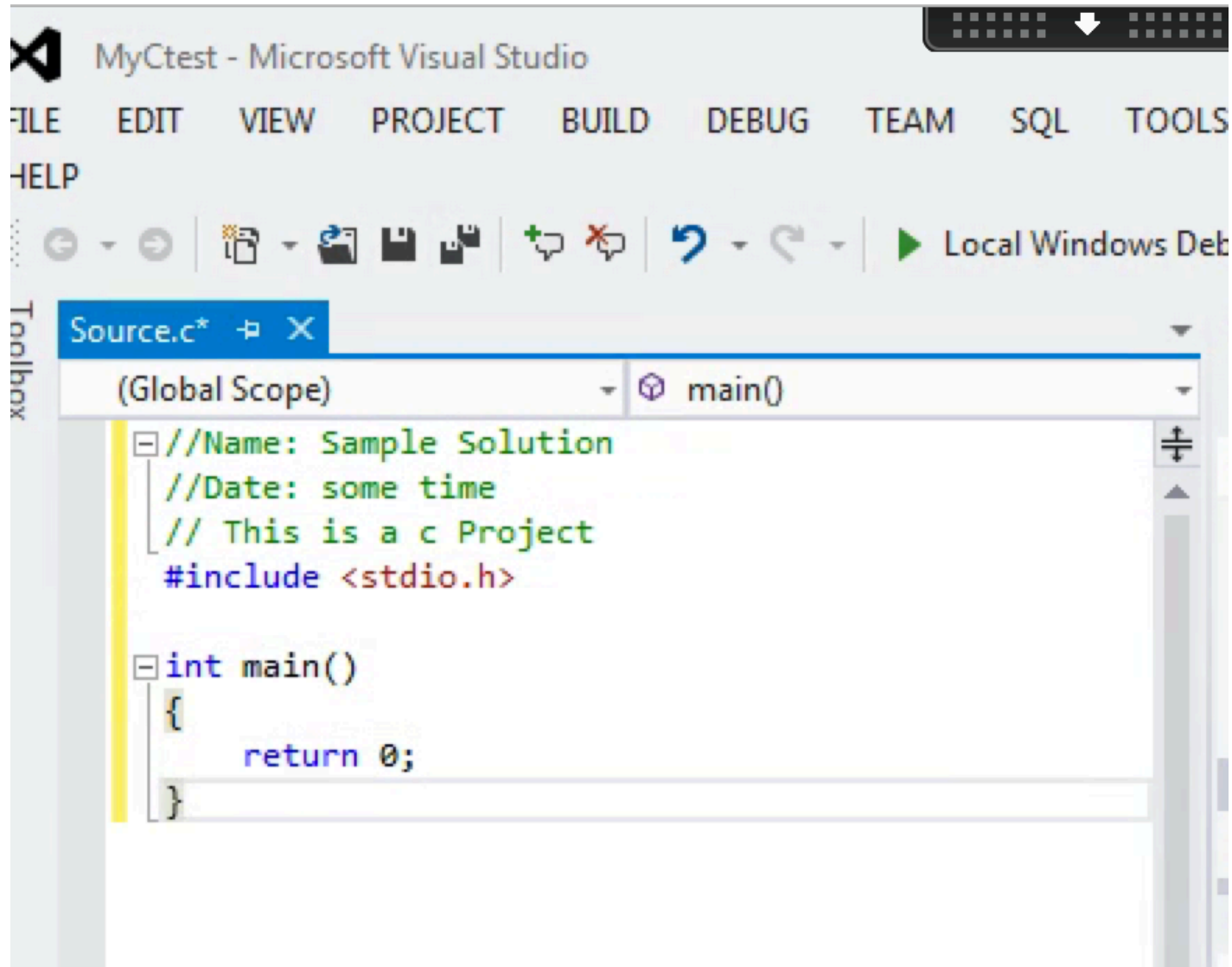
Specifies the name of the filter.



Note the Solution Explorer



Type C Code – Enjoy!



The screenshot shows the Microsoft Visual Studio IDE with the title bar "MyCtest - Microsoft Visual Studio". The menu bar includes FILE, EDIT, VIEW, PROJECT, BUILD, DEBUG, TEAM, SQL, and TOOLS. The toolbar contains icons for navigation, file operations, and a "Local Windows Deb" button. The active window is "Source.c*", which displays the following C code:

```
//Name: Sample Solution
//Date: some time
// This is a c Project
#include <stdio.h>

int main()
{
    return 0;
}
```

Important:

Functions: Testing Strategy

- How do you test functions?
 - Test one function at a time
 - Display intermediate results
 - You may need to create test data to use via “driver programs”
 - If the function being tested calls other functions, create “stubs”
 - Try varying one thing at A time
 - If something goes wrong, you know what changed

Testing Strategy

- Drivers
 - allows you to test a function without the rest of a program
 - just to execute the function and show its results
 - often, provides a loop to retest the function on different arguments

Testing Strategy

- Stubs
 - simplified version of a function not written or tested yet
 - often used when testing another function
 - does not necessarily deliver correct values
 - works best when stubs are replaced by actual functions, one at a time

You Are Two

- Programmers think end users don't know what they're doing
- End users think programmers are disconnected from reality
- When you write code you are the programmer
- When you test code, you are the end user
- You need to please both of you; it's not easy, but good things don't come easy

Driver Demo!

`/* In this scenario, we are developing a hard-working }
function that determines if a certain year is a leap
year. It returns, following the C convention, an int.`

`Returning 0 will mean false. Returning 1 will be
true.`

`*/`

`int leapYear(int year);`

`int main() {`

`/* Driver code will want to call the function
many times, with lots of different data to validate that
it is`

`working correctly... */`

`printf("leapYear(2000) = %d\n", leapYear(2000));
printf("leapYear(1900) = %d\n", leapYear(1900));
printf("leapYear(1950) = %d\n", leapYear(1950));
printf("leapYear(1999) = %d\n", leapYear(1999));
printf("leapYear(2001) = %d\n", leapYear(2001));
printf("leapYear(2004) = %d\n", leapYear(2004));
return(0);`

`/* Here is the function. You might notice some
bugs.... */`

`int leapYear(int year) {`

`int isTrue = 1;`

`int isFalse = 0;`

`int result = isFalse;`

`if (year % 4 == 0) {
 result = isTrue;`

`}`

`return(result);`

Summarizing Driver Demo!

- Drivers Are Throwaway Code Meant To Exercise Other Code
- Stubs Are Fake StandIns For Code That Will Be Fleshed Out Later
- Make sure to clean up before shipping (submitting) your assigned work

Summary

- Focus on:
Preprocessor,
C only,
Testing Strategy