24-780B
Engineering Computation

Mon/Wed 4:40-6:40pm, Fall 2022

Lecture 1

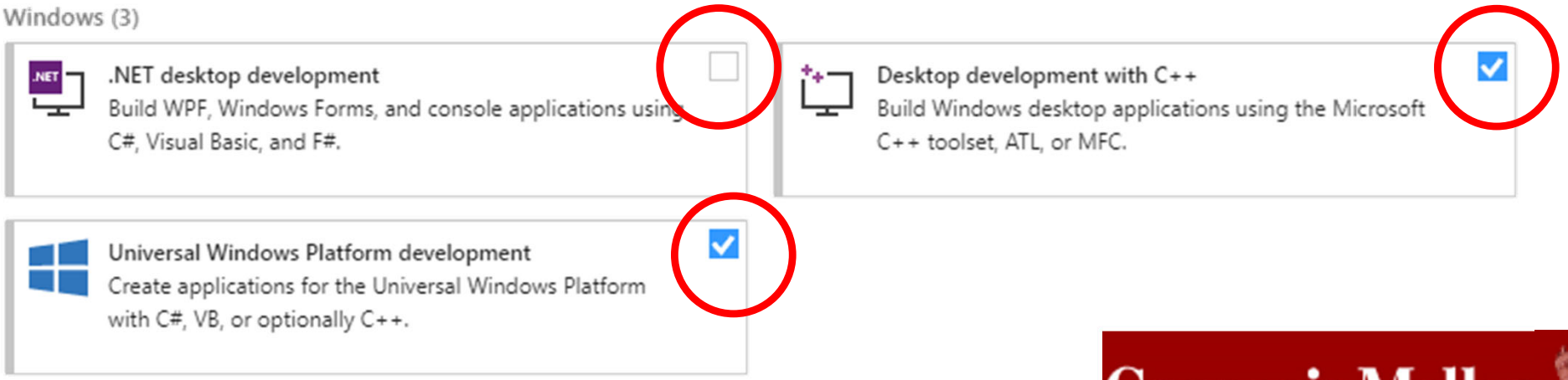
Nestor Gomez, Ph.D., P.E.
nestor@cmu.edu

Carnegie Mellon

Visual Studio 2022 Community Edition

- Download and install (30 minutes?) from: https://visualstudio.microsoft.com/downloads/

- Select “Community Edition” and be sure to include “Universal Platform” and “Desktop development with C++”



Carnegie Mellon

Thought Exercise of the Day

Keep doing this:

Download and install (30 minutes?) from:
https://visualstudio.microsoft.com/downloads/

Select “Community Edition” and be sure to include “Universal Platform” and “Desktop development with C++”

Course Overview

- Introductory programming course
- 12-unit course
  - Lecture: 4 hours per week
  - Other work load (mainly assignments): 8 hours per week
- Main topics
  - C++ programming (w/ object-orientation)
  - Elementary algorithms (data representation, sorting, searching, image processing, etc.)
  - Visualization with OpenGL (mostly 2D with some 3D)
  - Elementary sound processing
  - Multi-threaded programming (parallel processing)

What to get out of this course?

Learn Something

Degree Reqmts

CMU Canvas and Piazza

We use CMU Canvas system to organize the course. You will need to submit assignments via CMU Canvas.

Make sure you can log on and access 24-780-B Canvas.

We have also set up a Piazza page for communication, asking questions, pushing up-loads, providing solutions, etc.

Problem sets (assignments) and solutions will be posted on both Canvas and Pizza

When to Program

- There is a problem to be solved
- Nature of problem is such that computing can help
- Existing software cannot be adapted
- It is impractical to subcontract programming effort
- Want to produce a problem-solving application

- For a good (brief) history of computers and programming, see:
  - http://www.cs.cmu.edu/~guna/15110N12/applications/ln/lecture2.pdf

How does a computer work

- Everything, but everything is binary (ones and zeros, on or off, true or false, respectively)
- Example using integer addition:

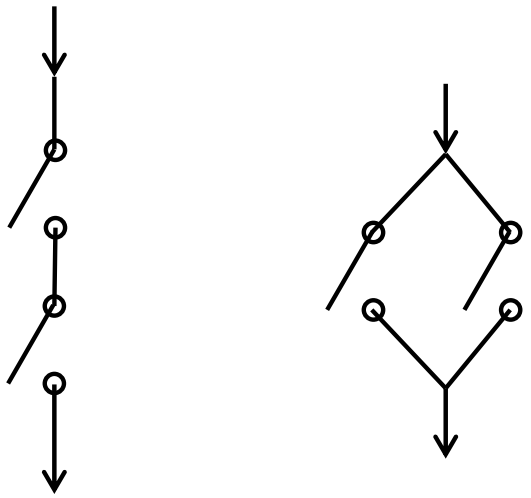
```
  27          111100
 +14           11011
----    +      1110
 41          ------
             101001
```

convert to binary

Carry-over bit

Addition then becomes series of AND/OR operations

AND/OR are circuit “gates”



How does a computer work (cont)

- Every computer operation is actually a series of electron surges through a complex web of and/or gates that ultimately change the state of the memory, which itself can be used as a web of and/or gates
- If a programmer had to plan how all these operations are executed, even small procedures would be challenging.

- We need a “language”

What Language is Best

- List of programming languages
  - https://en.wikipedia.org/wiki/List_of_programming_languages
- Powerful, but not overly cumbersome
- Widely used
- Portable
- Development tools are available (e.g., editors, debuggers, project management)
- Interpreted vs. compiled

Why C++?

C++ has evolved over the decades. Still, most of the old C++ codes work with the newest compilers.

One of the recent problems is too many short-living programming languages.

C++ is a good language to learn. See Tiobe index

https://www.tiobe.com/tiobe-index/

Many languages are specifically designed to the “C-like”

https://en.wikipedia.org/wiki/List_of_C-family_programming_languages

C++ executables are very small (perfect for memory-limited mechatronic devices)

C++ gets the job done!

C++ (a history)

- The C language was developed Brian Kernighan and Dennis Ritchie in the early 1970s

- Bjarne Stroustrup of Bell Labs added classes to C to create C++ in the early 1980s.
- C++ is now an international standard (ISO/IEC 14882)
- Still one of the most popular programming languages (with Python, Java, and C)
- C++ keeps much of C, especially its syntax and low-level memory management, while adding programmer-level constructs.
- C++ also greatly improves string manipulation

Intro to C++ Language

- Computer program
  - a structured combination of data and instructions used to operate a computer
- Programming language
  - a set of instructions, data, and rules that can be used to construct a program
  - Classified by level / machine dependency
    - High-level languages: FORTRAN, BASIC, COBOL, Python, Pascal, Matlab
    - Low-level languages: instructions that are directly tied to one type of computer
    - Middle-level languages: C and C++
  - Compiled language vs. Intepreted language
    - Compiled language: Source code is pre-translated into machine language before the execution (C, C++, FORTRAN, Java)
    - Interpreted language: Source code is translated into machine language during the execution (BASIC, Python, Java, Matlab)
  - Classified by orientation (C++ has both)
    - Procedure-oriented: input data => [process the data] => output results
    - Object-oriented: object = datafields + methods

High-level vs. Low-level language

- High-level language
  - Provides built-in functionalities programmer may want to use.
  - Can perform a complex task with relatively short program if the task can be described as a combination of the built-in functionalities.
  - Limited control of the computer resources (memory, registers, CPUs, etc.)
- Low-level language
  - The language itself has (close to) minimum set of functionalities.
  - Needs to describe detailed steps. The program tends to become longer.
  - Nearly total control of the computer resources.

High-level vs. Low-level language

What about C++? Mid-level language.

- You can write your own functions or import functions written by someone else, which will allow you to achieve complex tasks with a short program.
- You can take nearly full control of the computer resources if you choose to.
- C/C++ is widely used for controlling electronic devices (Arduino http://www.arduino.cc/) because of language maturity and small compiled code size.

Compiled vs. Interpreted language

- Both are a computer program that translates the programming language to the machine language.

Compiled vs. Interpreted language

- Compiled Language
  - The compiler generates a program (from the source code) that is directly understood by the processor
  - The compiler program is not required at run time.
  - Fast. Can be as fast as assembly language.
  - C/C++, Ada, Java
- Interpreted Language
  - The interpreter runs directly off of the source code.
  - The interpreter translates the programming language into the machine language at run time.
  - Interpreter program is required at run time.
  - Slower due to the translation overhead.
  - Python, BASIC, Java

Object-oriented vs. Procedure-oriented

- Procedure-oriented programming
  - Early days (When a computer was really just for numerical computation)
    - Assembly language, BASIC language
    - Any data can be accessed and modified from any place in the program.
    - The programming language helps little to none for organizing the structure of the program or re-using the program modules.
  - Structured programming paradigm
    - C language, PASCAL, Visual BASIC
    - Local variables: Certain data can be accessed only from limited part of the program.
    - The programming language has a mechanism to organize the structure of the program and re-using the program modules.
- Object-oriented programming
    - C++, Java, Objective-C (sort of)
    - Class provides a mechanism for protecting the data.
      - Catching programming errors before shipped to the user.
      - Preventing data loss even after the program is deployed.
    - Enables large-scale and complex programming.

Finally . . . C++ is:

- Mid-level,
- Compiled, and
- Object-Oriented programming language.

However, you *could* code in a low-level, non-object-oriented, and non-structured way, which is NOT recommended.

How to write a computer program

Programming is:

Describing what you want a computer to do for you.

Duty of a compiler or an interpreter:

Translating a programming language (C++, Python, etc.) to a machine language (zeros and ones).

Duty of a programmer:

Translating a human “language” (ideas, processes) to a programming language.

Programming – Procedure-oriented

1. Think what needs to happen. It is important to have a clear “purpose.”
2. Break down the purpose into more detailed steps. There can be multiple levels of detailed steps.
3. Think what data structure is needed.
4. Describe it in a programming language.

This works for a short project.

Programming – Object-oriented

1. Think what needs to happen. It is important to have a clear “purpose.”
2. Think about necessary data structures (or classes) and behaviors of each structure.
3. Think about the chain of events that drives the program. How/why do things happen.
4. Describe it in a programming language.

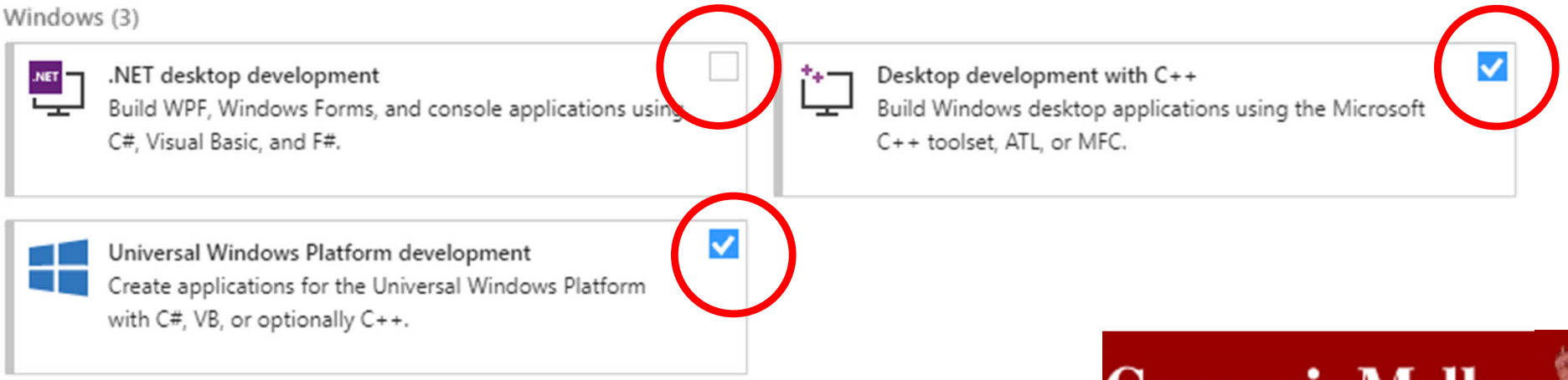
Typically advantageous for larger program.

Installing Visual Studio 2022 Community Edition

Visual Studio 2022 Community Edition

- Download and install (30 minutes?) from: https://visualstudio.microsoft.com/downloads/

- Select “Community Edition” and be sure to include “Universal Platform” and “Desktop development with C++”



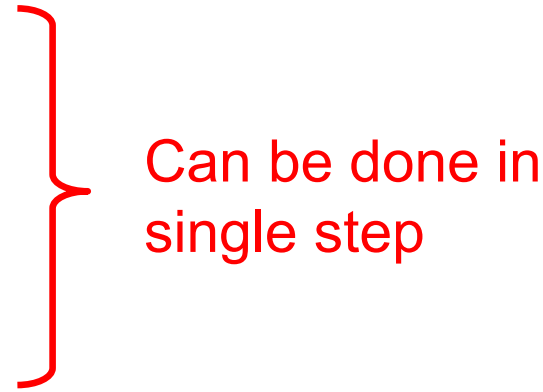
Carnegie Mellon

Using Visual Studio

Five easy steps.

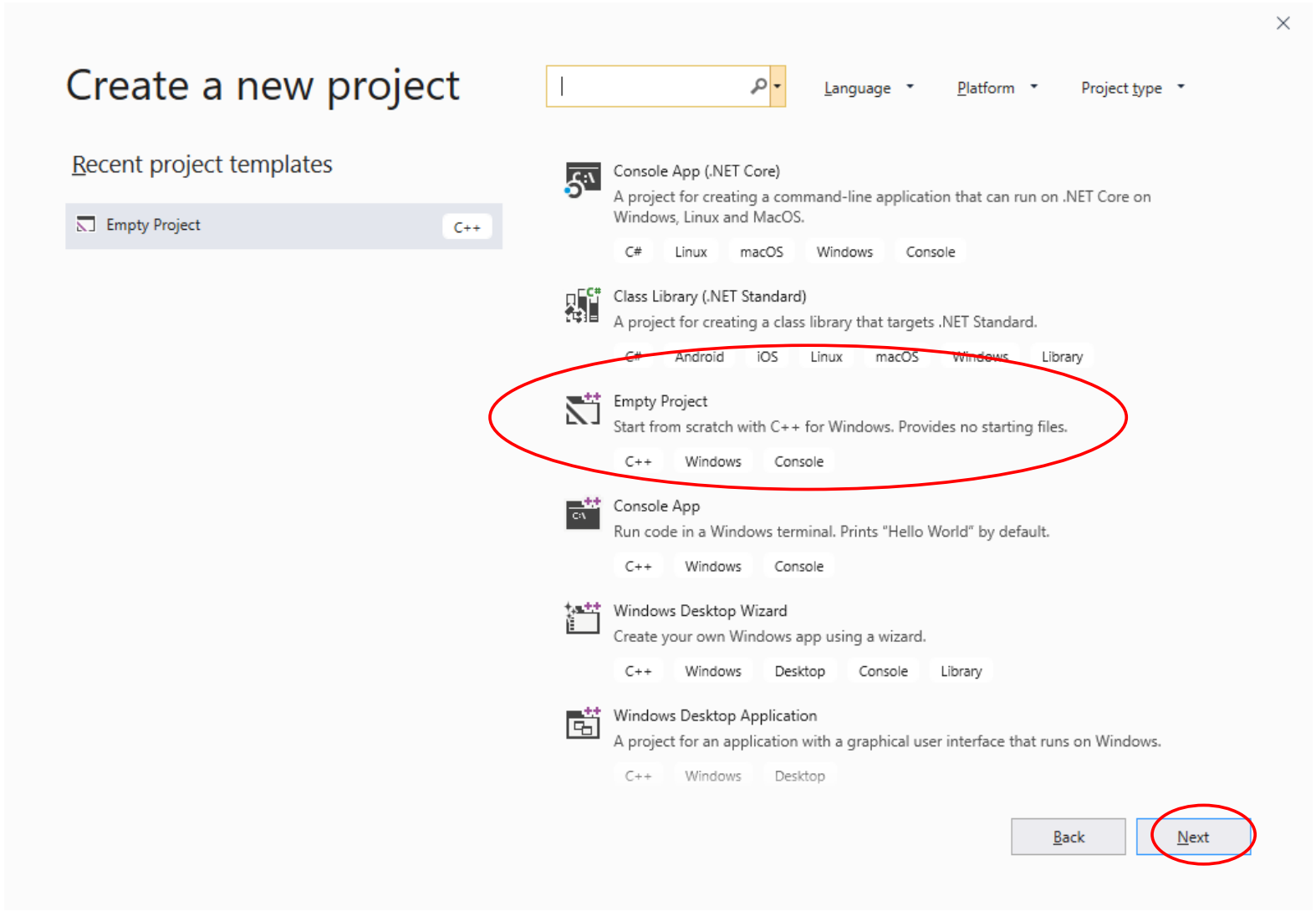
1. Create a project (special folder and .sln file)
2. Create source and header files (.cpp and .h, respectively)
3. Write code for a program
4. Build (compile and link)
5. Run

Recommended: Debug



Using Visual Studio

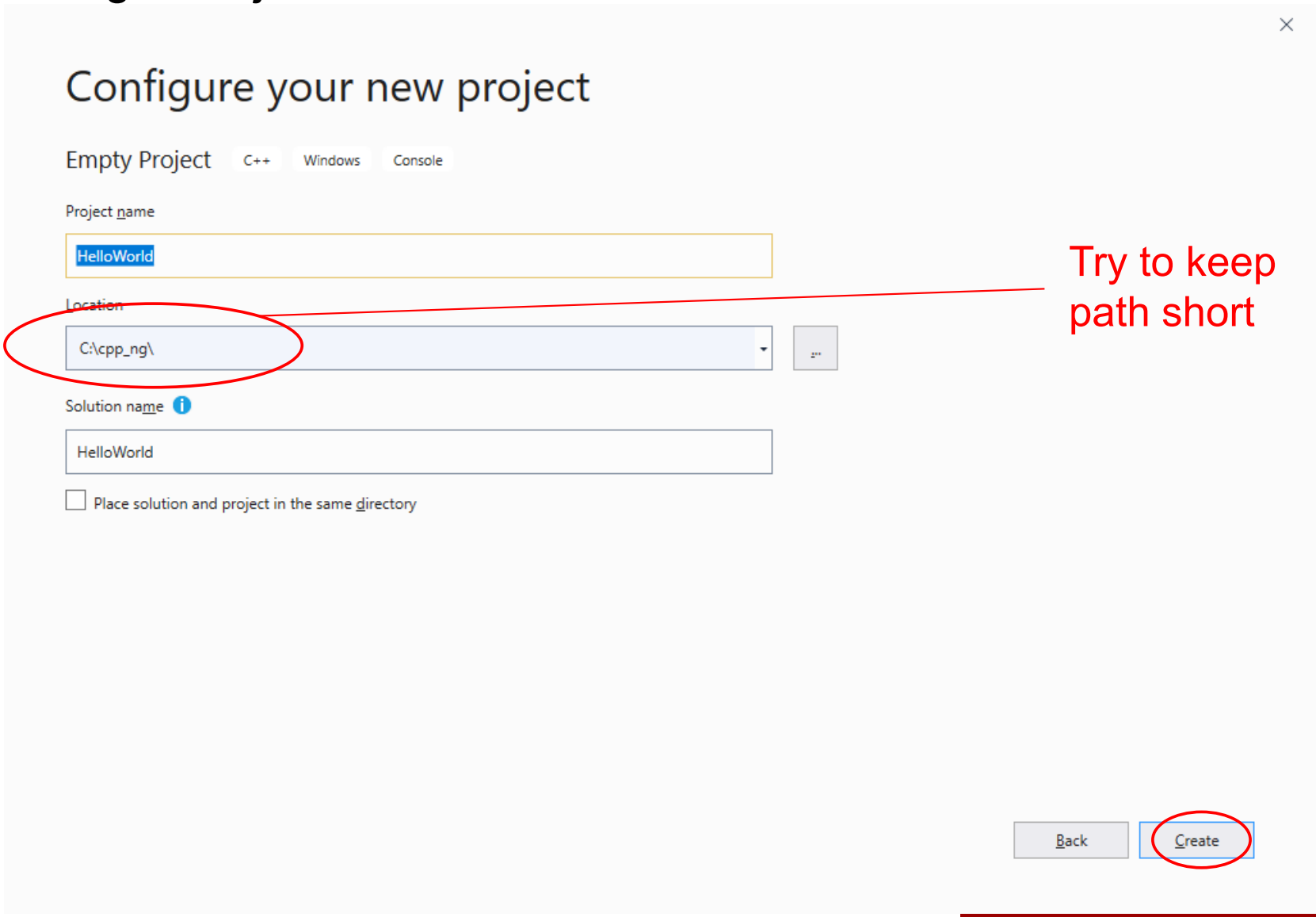
1. Creating a Project



Carnegie Mellon

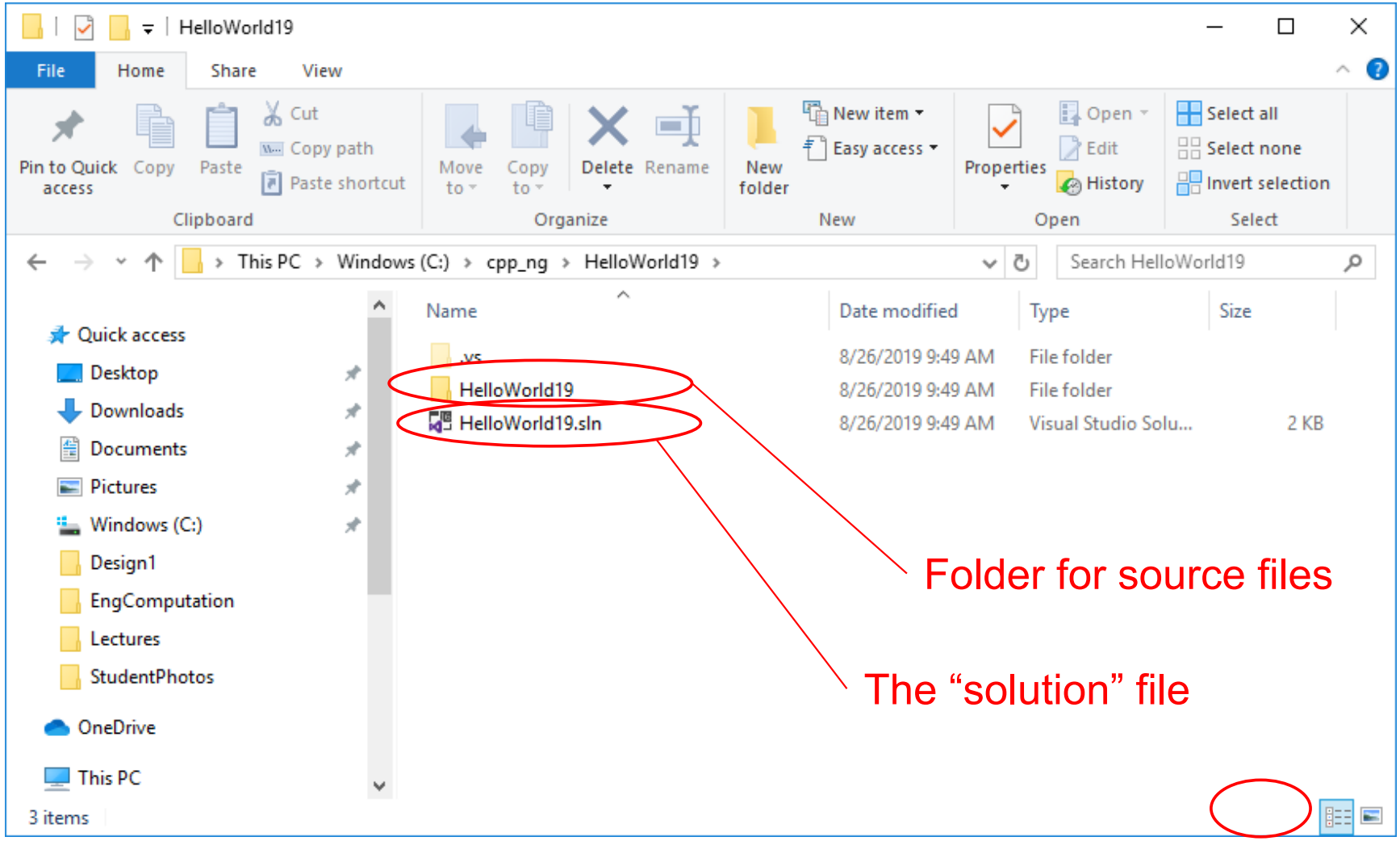
Using Visual Studio

1. Creating a Project



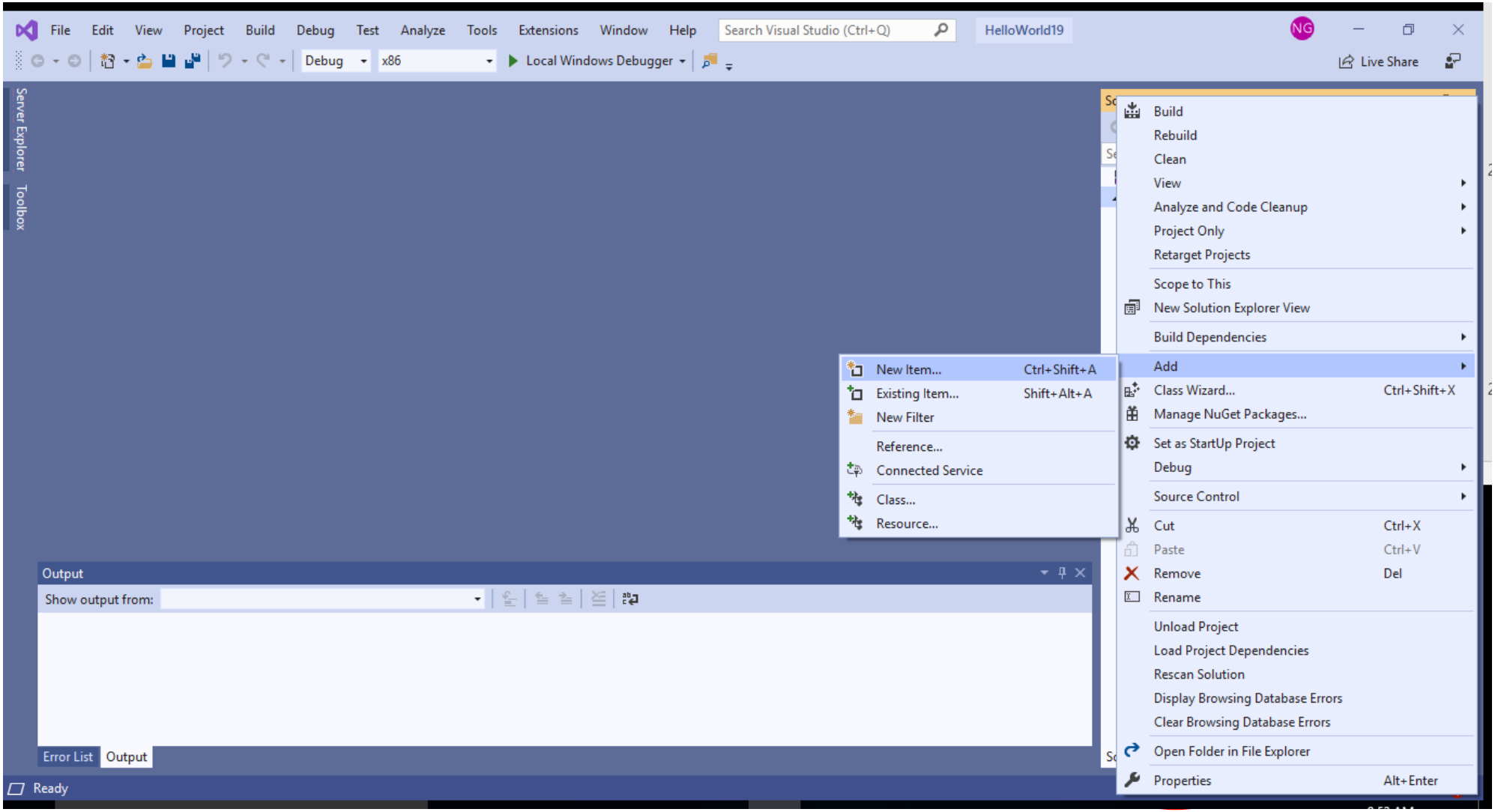
Using Visual Studio

1. Creating a Project (changes to permanent storage)



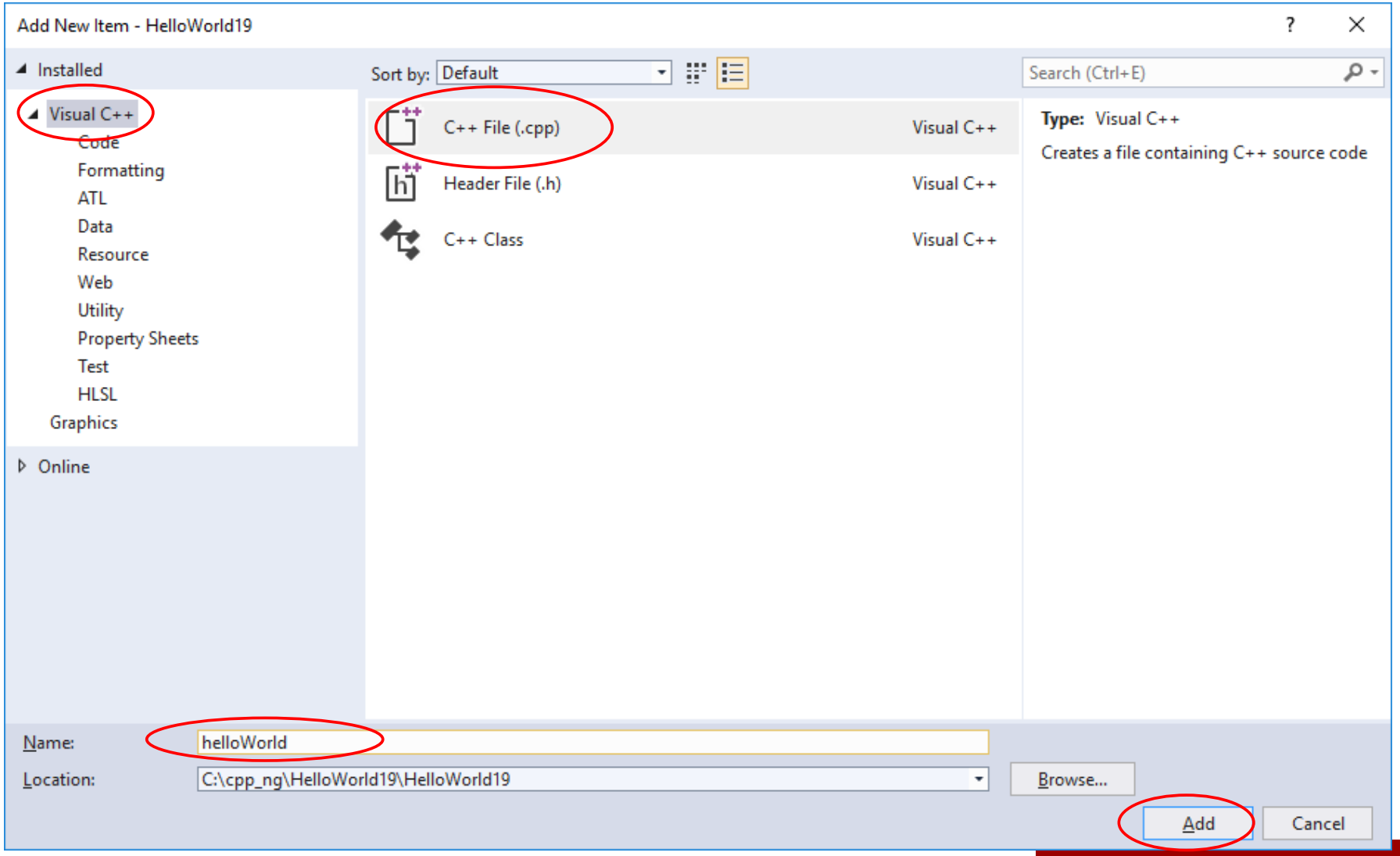
Using Visual Studio

2. Creating a Source File (.CPP file) Right-click on Solution Explorer



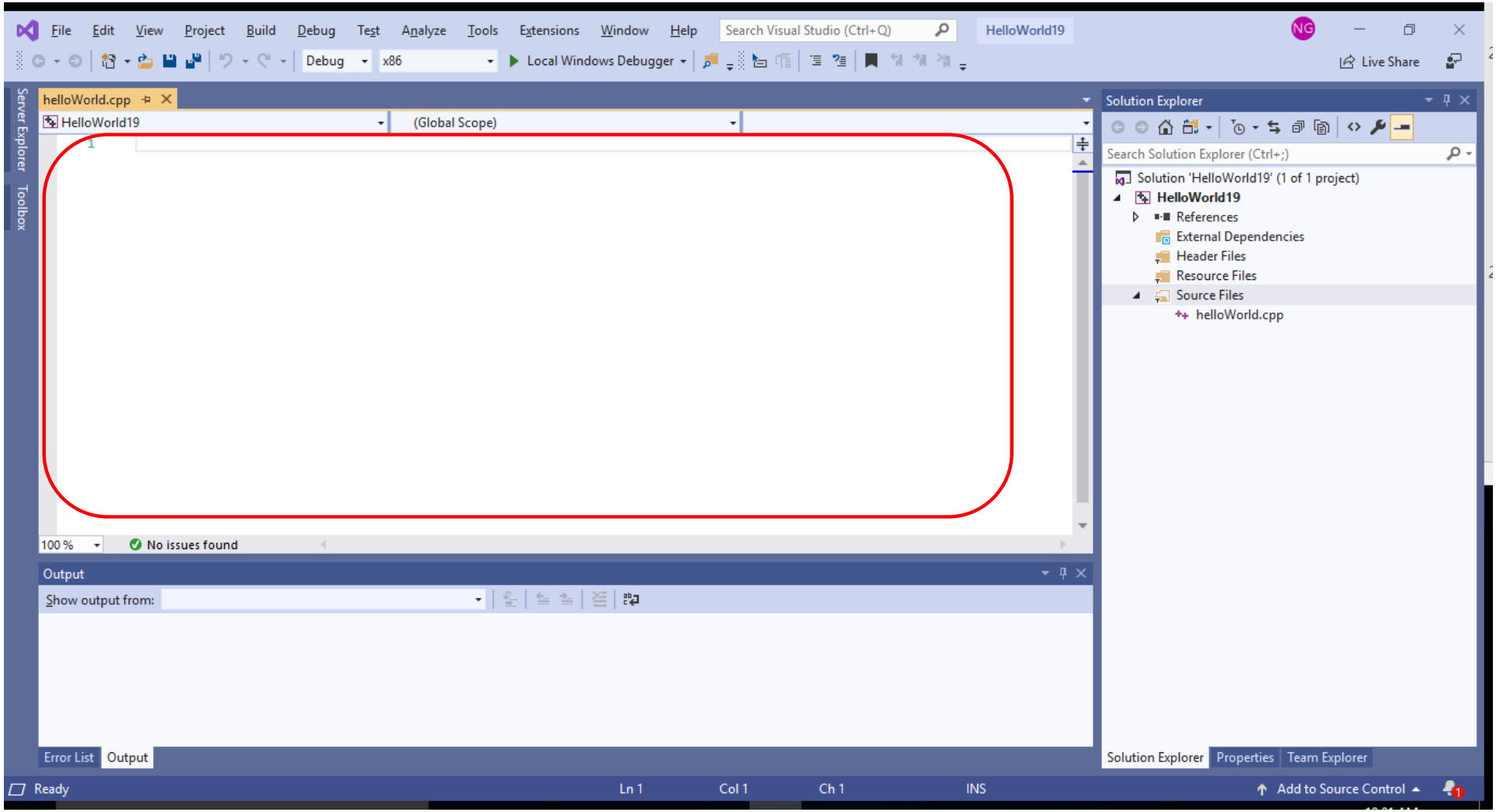
Using Visual Studio

2. Creating a Source File (.CPP file) Right-click on Solution Explorer



Using Visual Studio

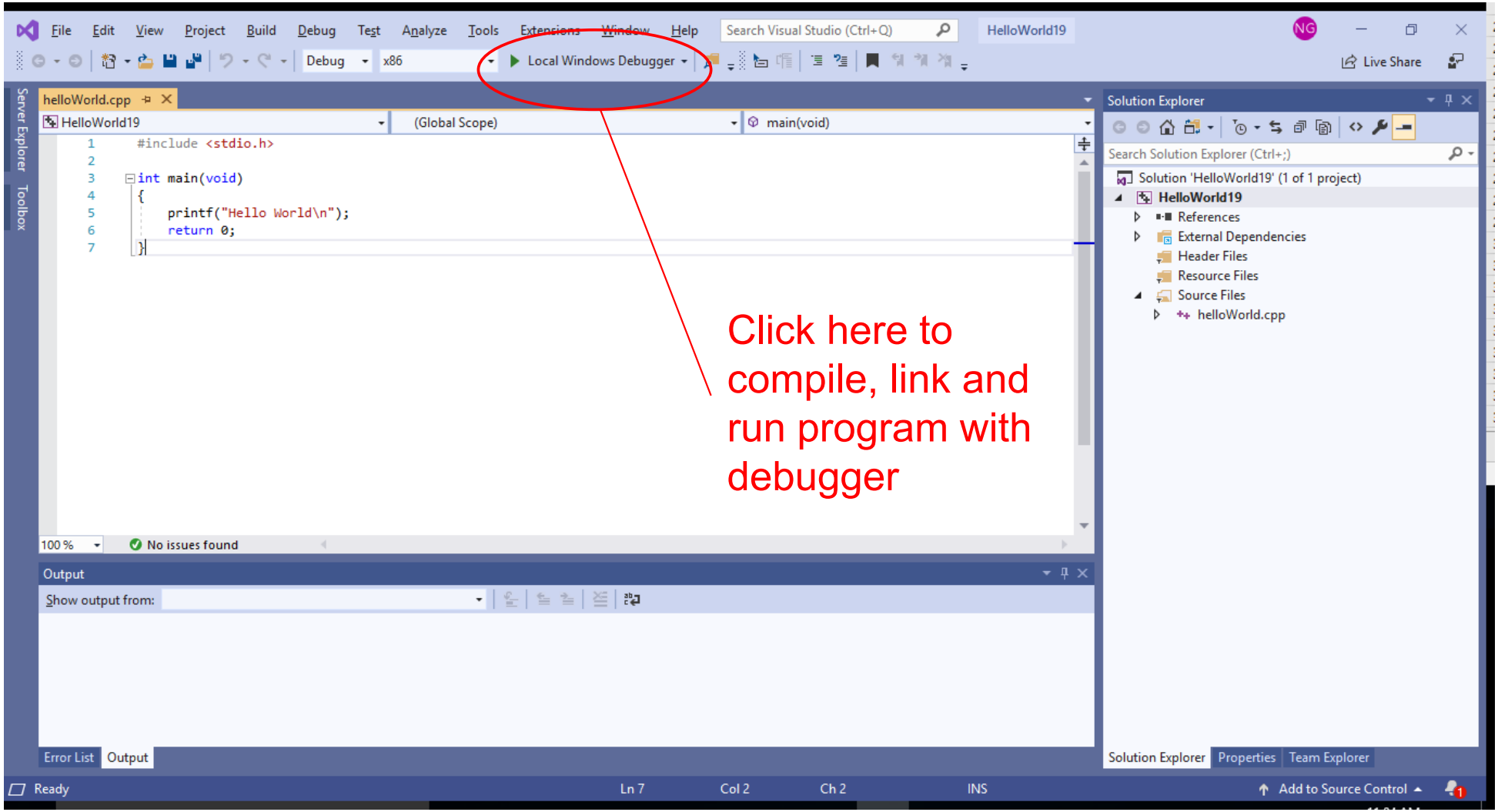
3. Writing a Program



Carnegie Mellon

Using Visual Studio

3. Writing a Program

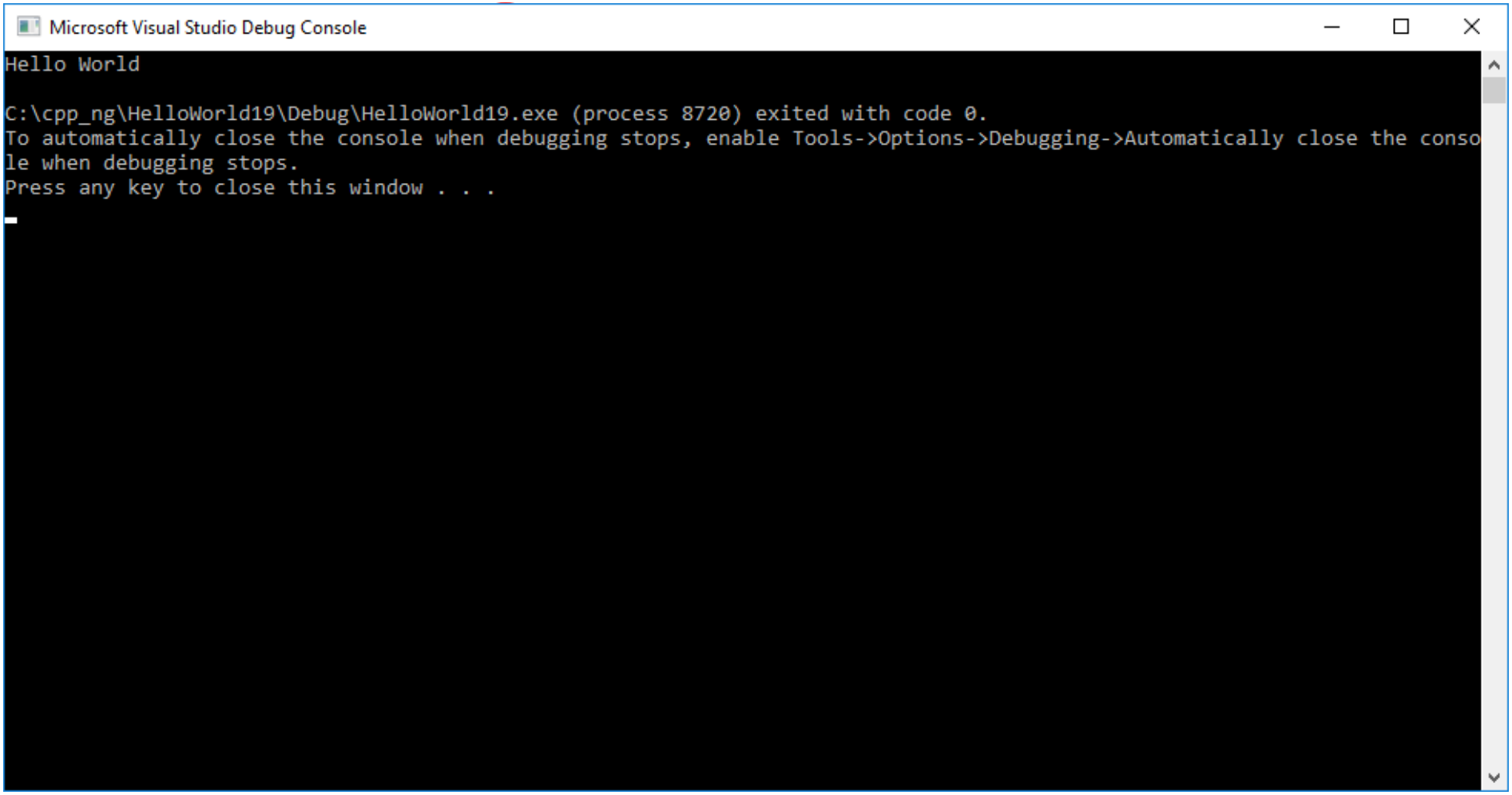


Carnegie Mellon

Using Visual Studio

Result (for now)

Use Ctrl+F5 to bring up console window if you lose it.



Using Visual Studio

- When you need to continue working on an existing project, you can open from:

File -> Open -> Project/Solution

Or find on “recent” projects list

Compiling with XCode, Mac OS X

XCode

- Primary developing system for Mac OS X and iPhone.
- Freely available from:
  http://developer.apple.com/technologies/xcode.html
- You will need to register as an Apple developer. The registration for the basic package is free.
- Similar IDE (Integrated Developing Environment) to Visual Studio.
- Based on GCC version 4.2, which is also a very good C++ compiler.

Compiling with XCode on Mac OS X

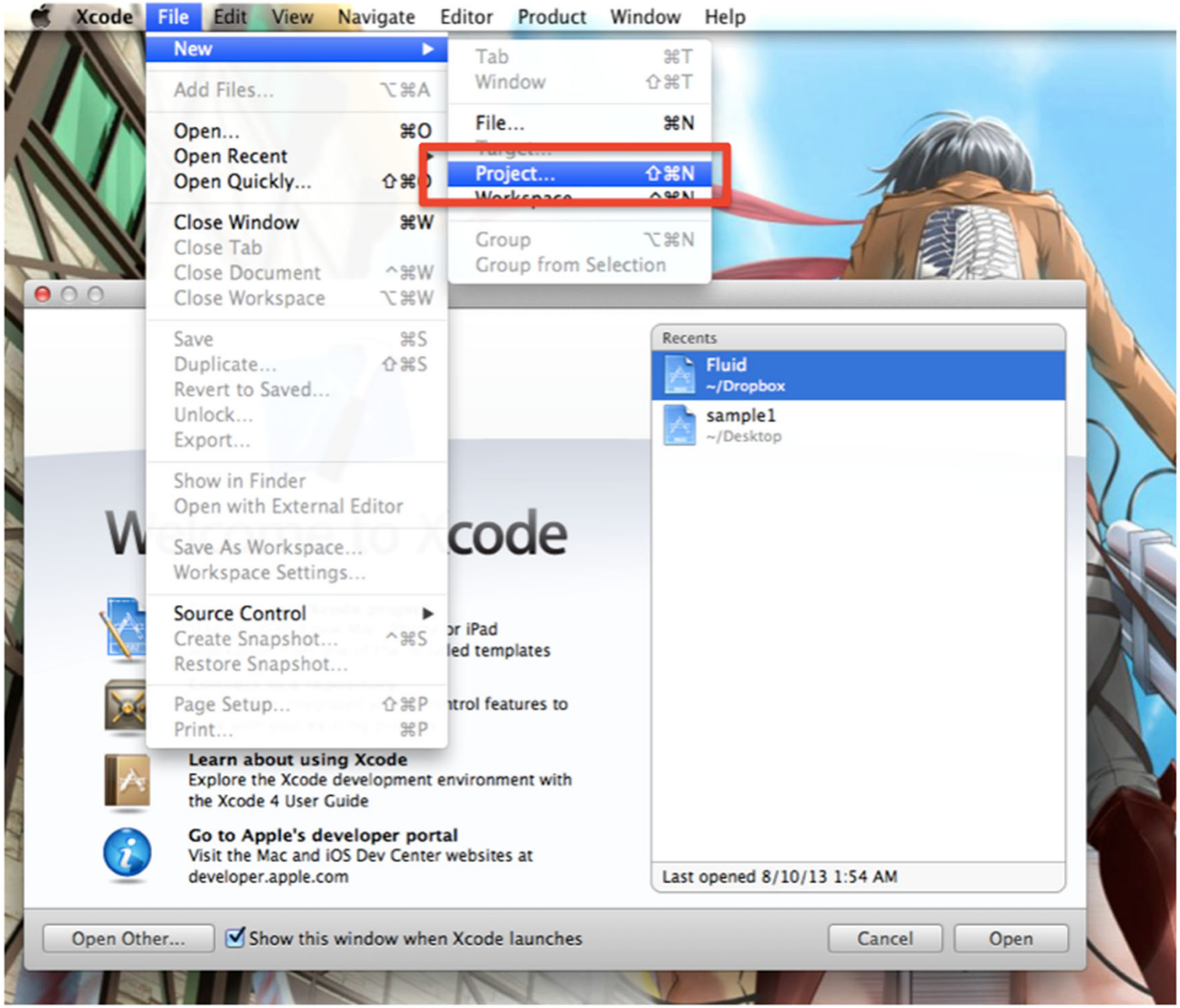
XCode (old version)

- Freely available from:

https://developer.apple.com/downloads/index.action

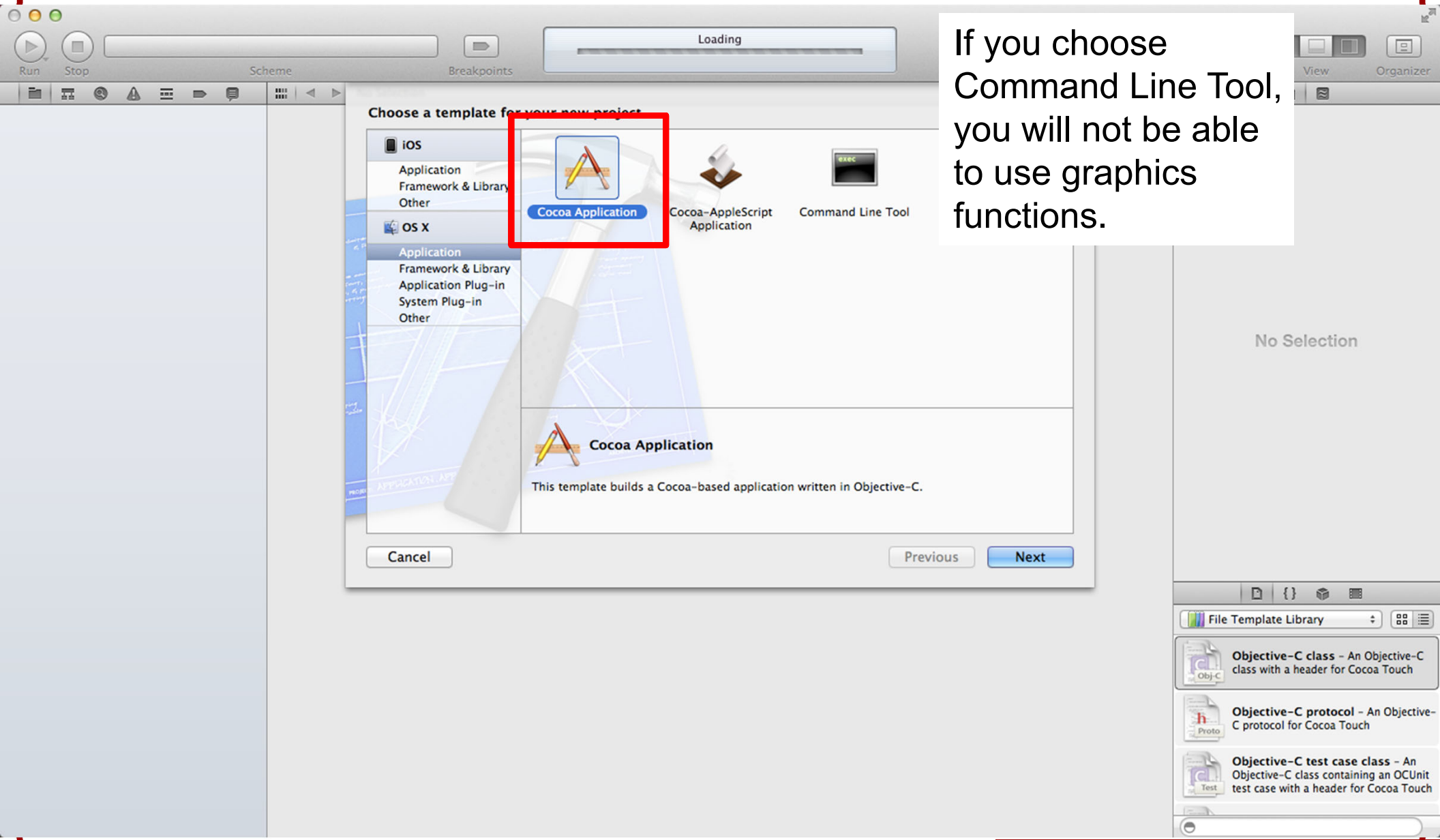
Mac OS X	XCode version
Yosemite 10.10.x	XCode 6.x
Mavericks 10.9.x	XCode 5.x

Creating a project in XCode



Carnegie Mellon

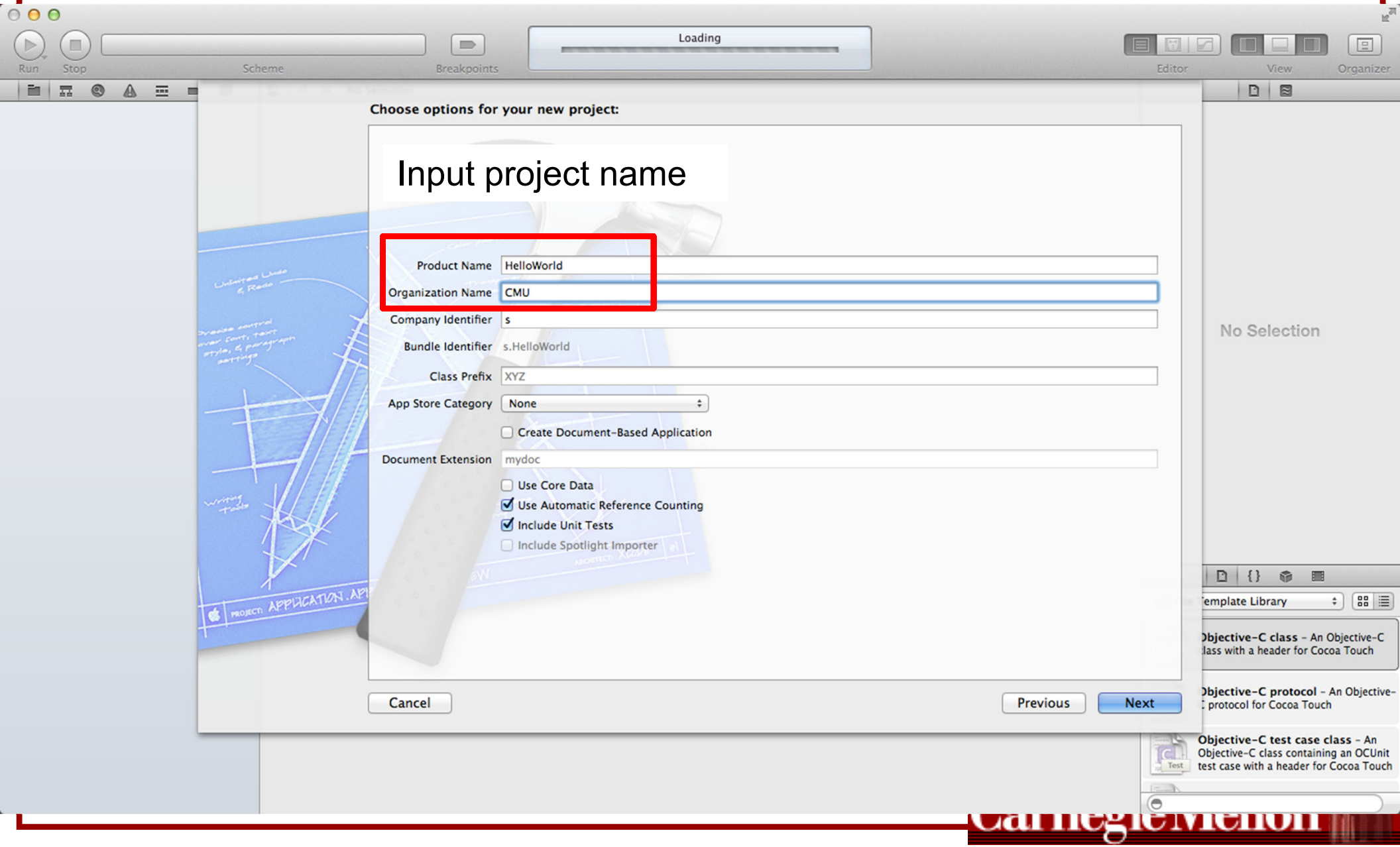
Creating a project in XCode



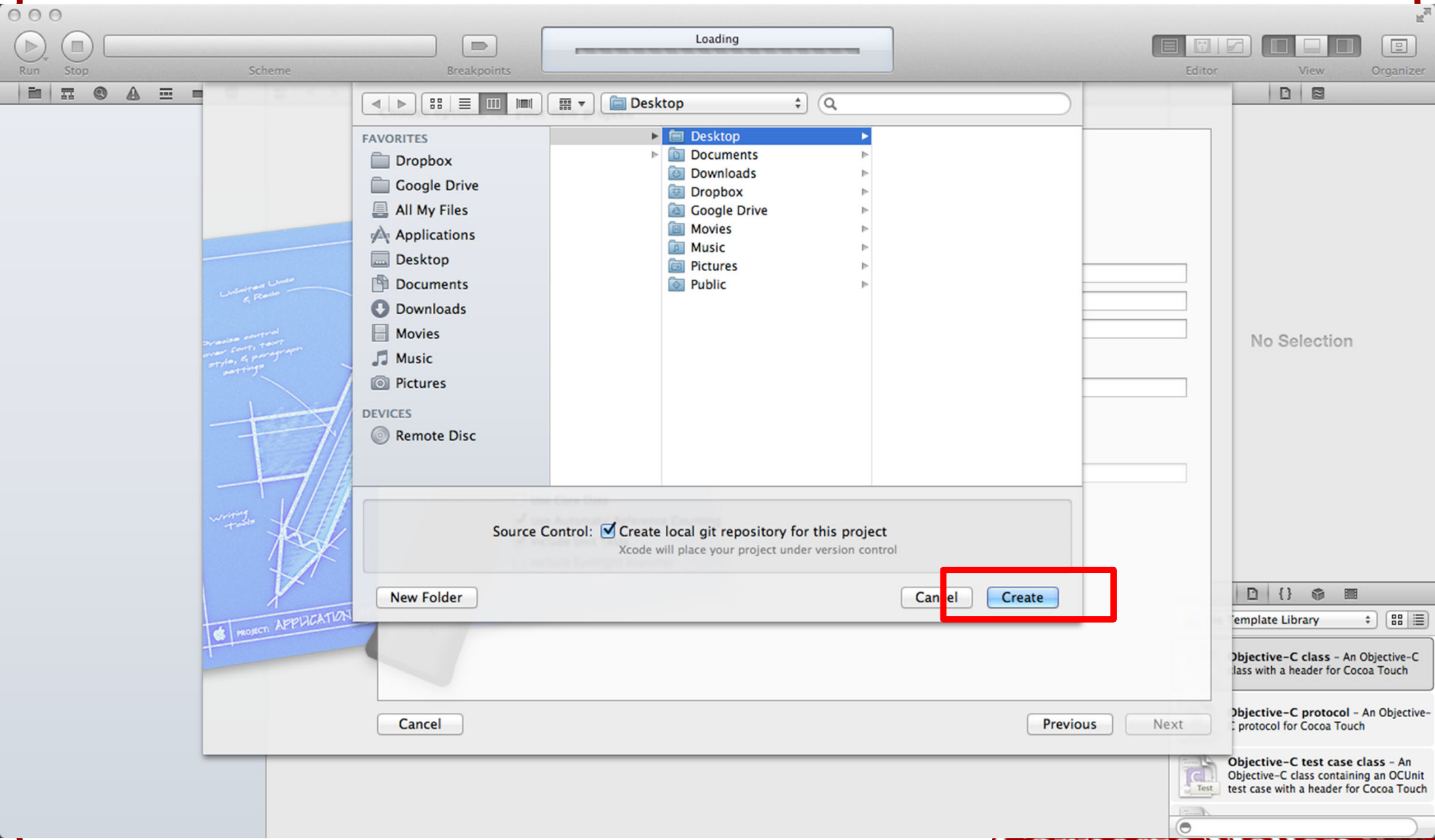
If you choose Command Line Tool, you will not be able to use graphics functions.

Carnegie Mellon

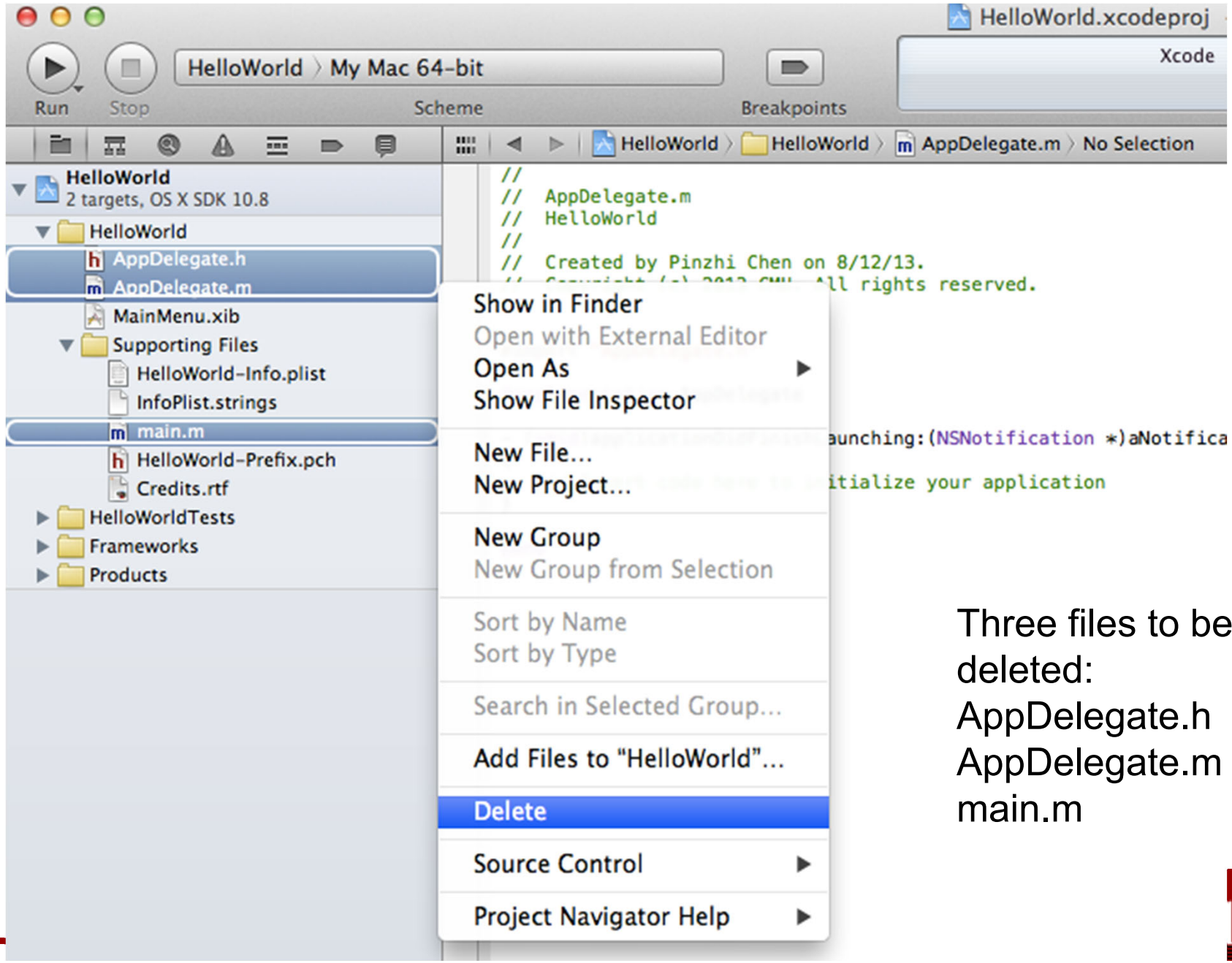
Creating a project in XCode



Creating a project in XCode

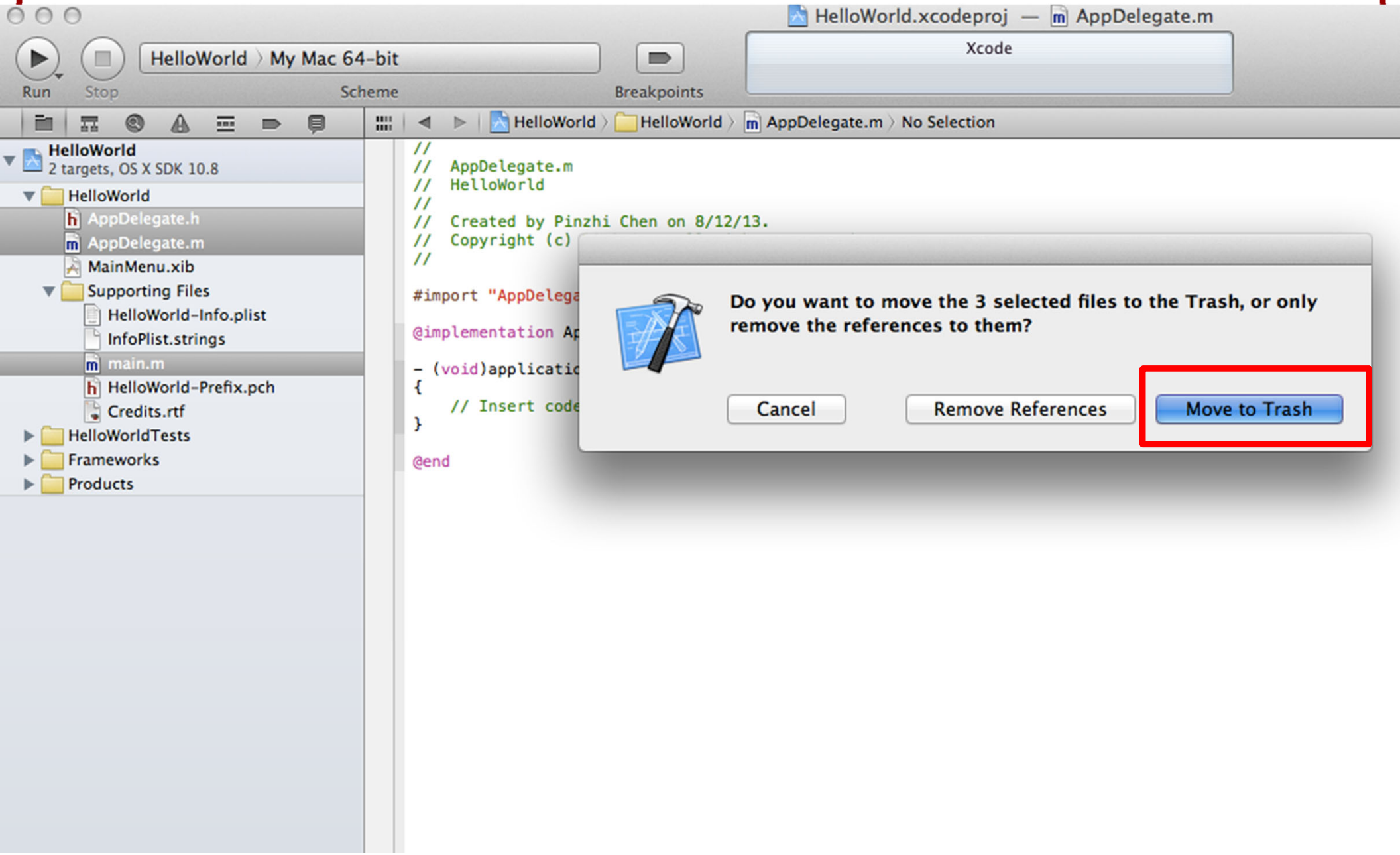


Deleting default source files

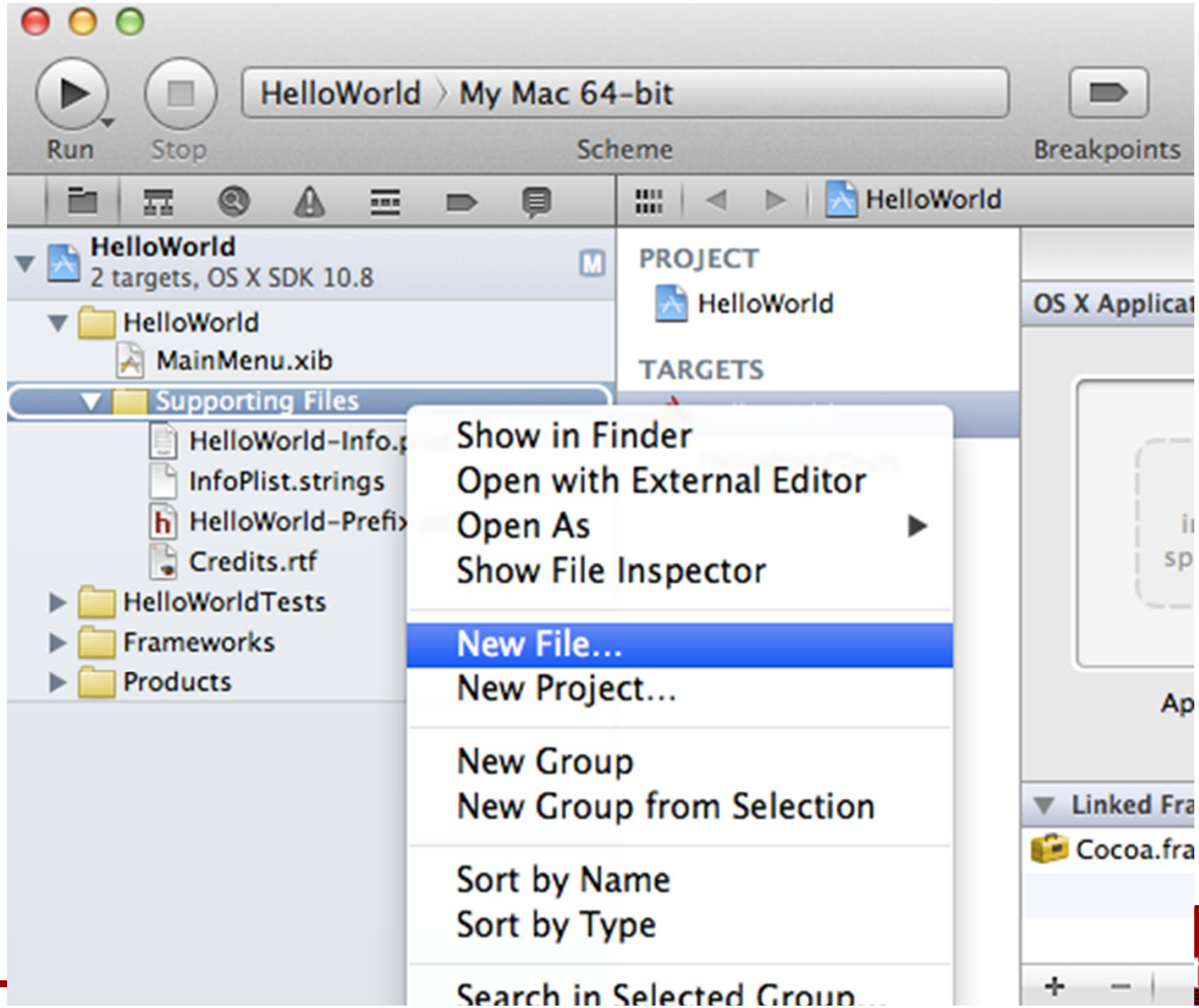


Three files to be deleted:
AppDelegate.h
AppDelegate.m
main.m

Deleting default source files

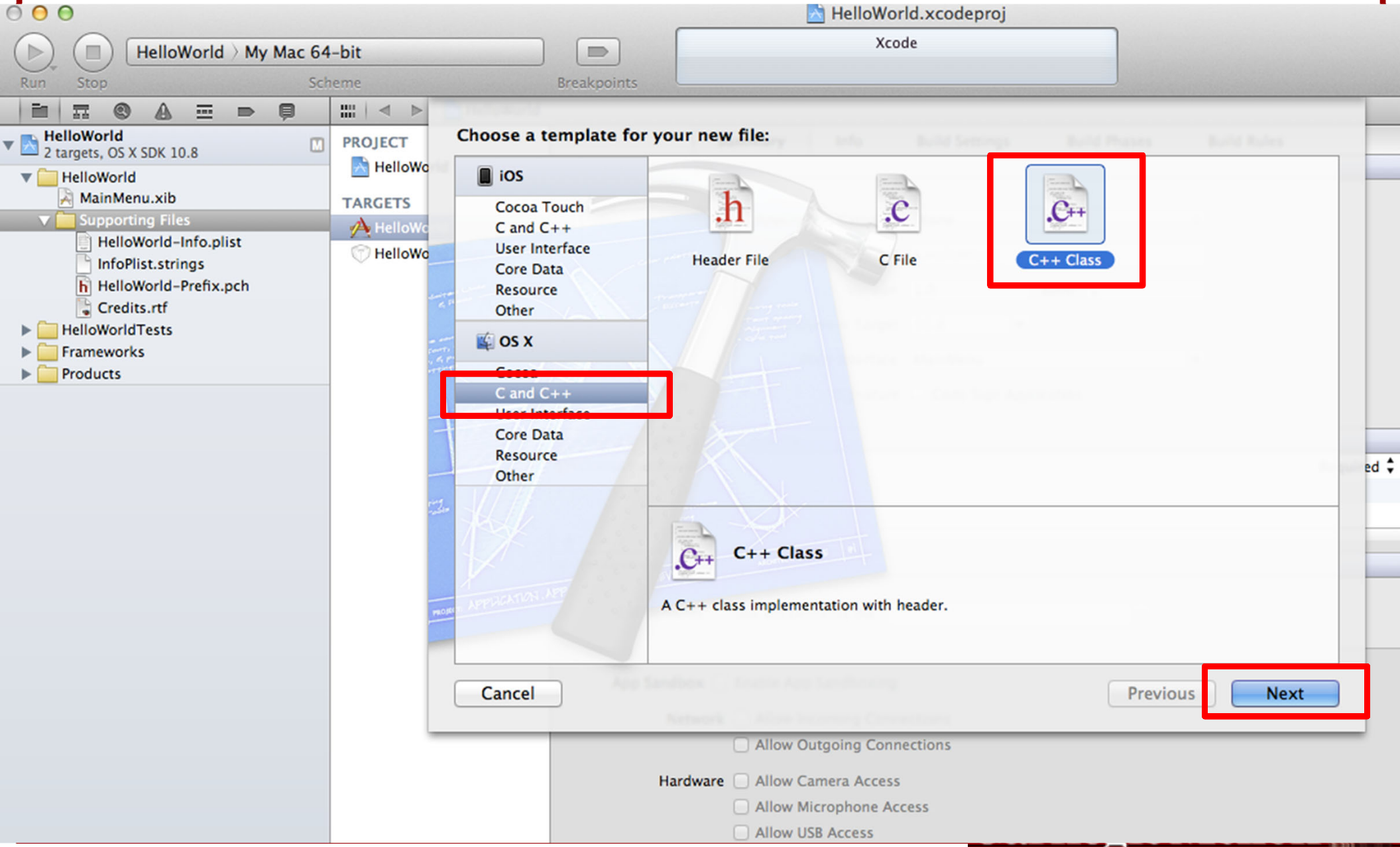


Creating your source file

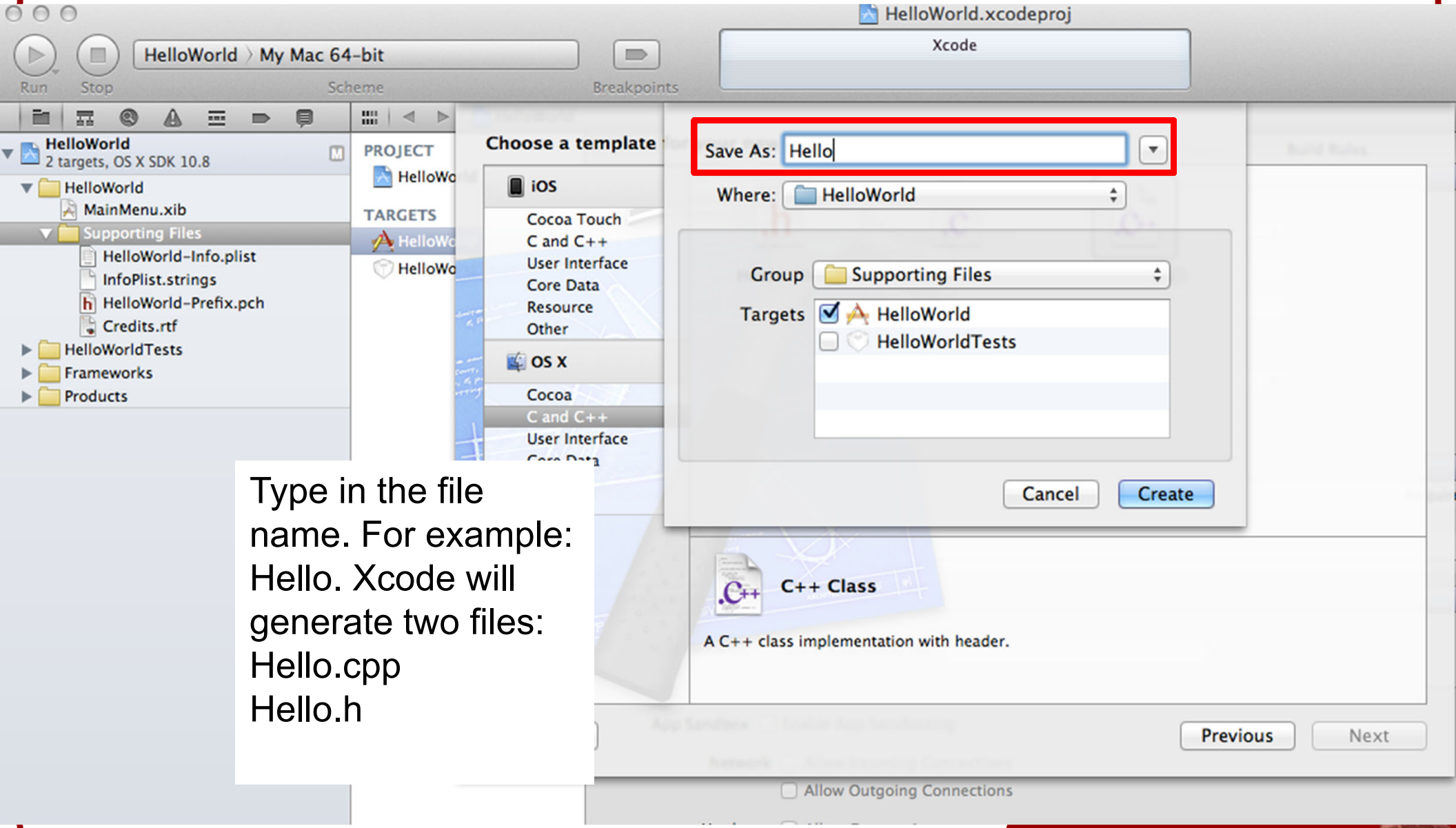


Right-click on "Supporting Files" and select Add -> New File

Creating your source file

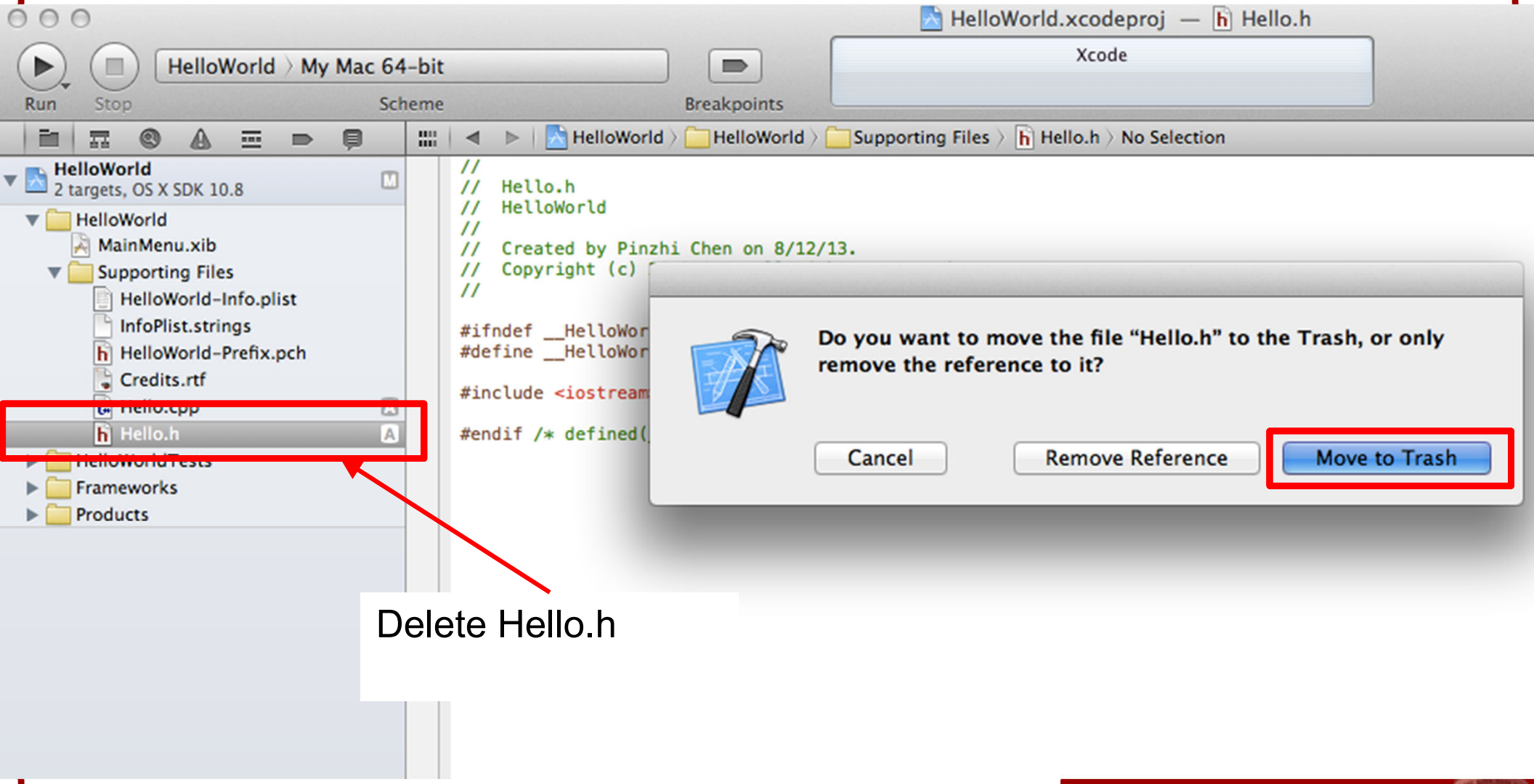


Creating your source file



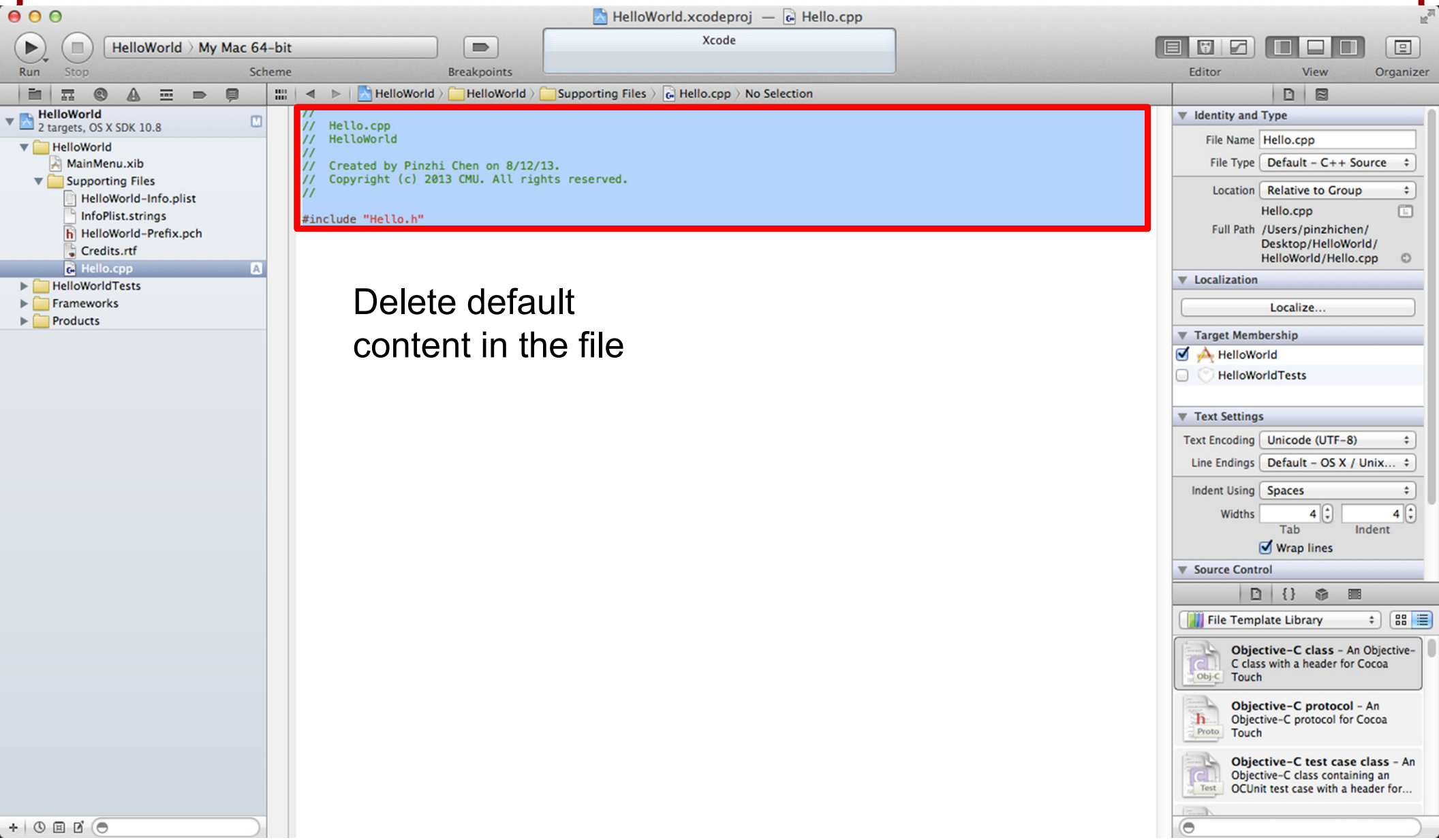
Type in the file name. For example: Hello. Xcode will generate two files:
Hello.cpp
Hello.h

Creating your source file



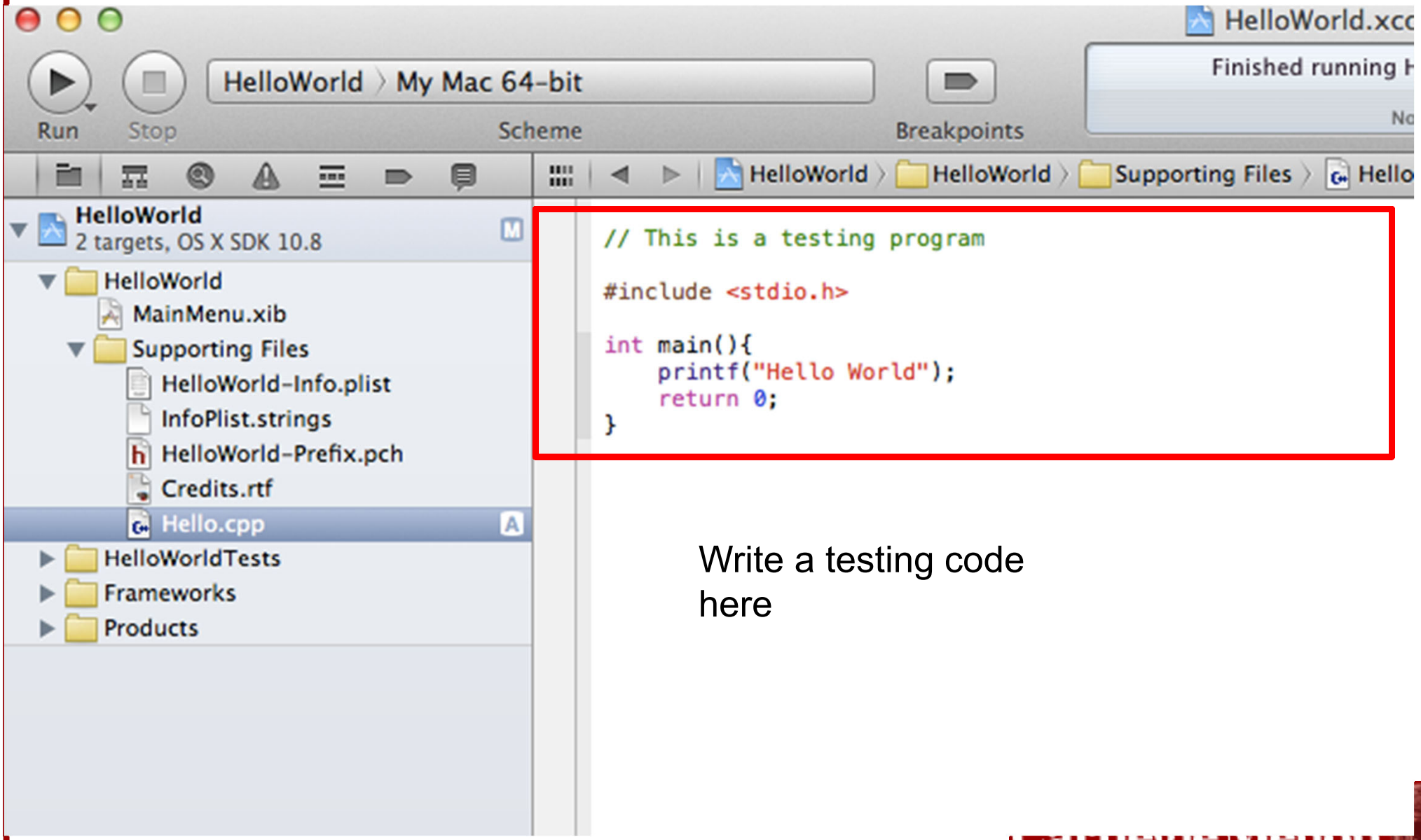
Delete Hello.h

Creating your source file



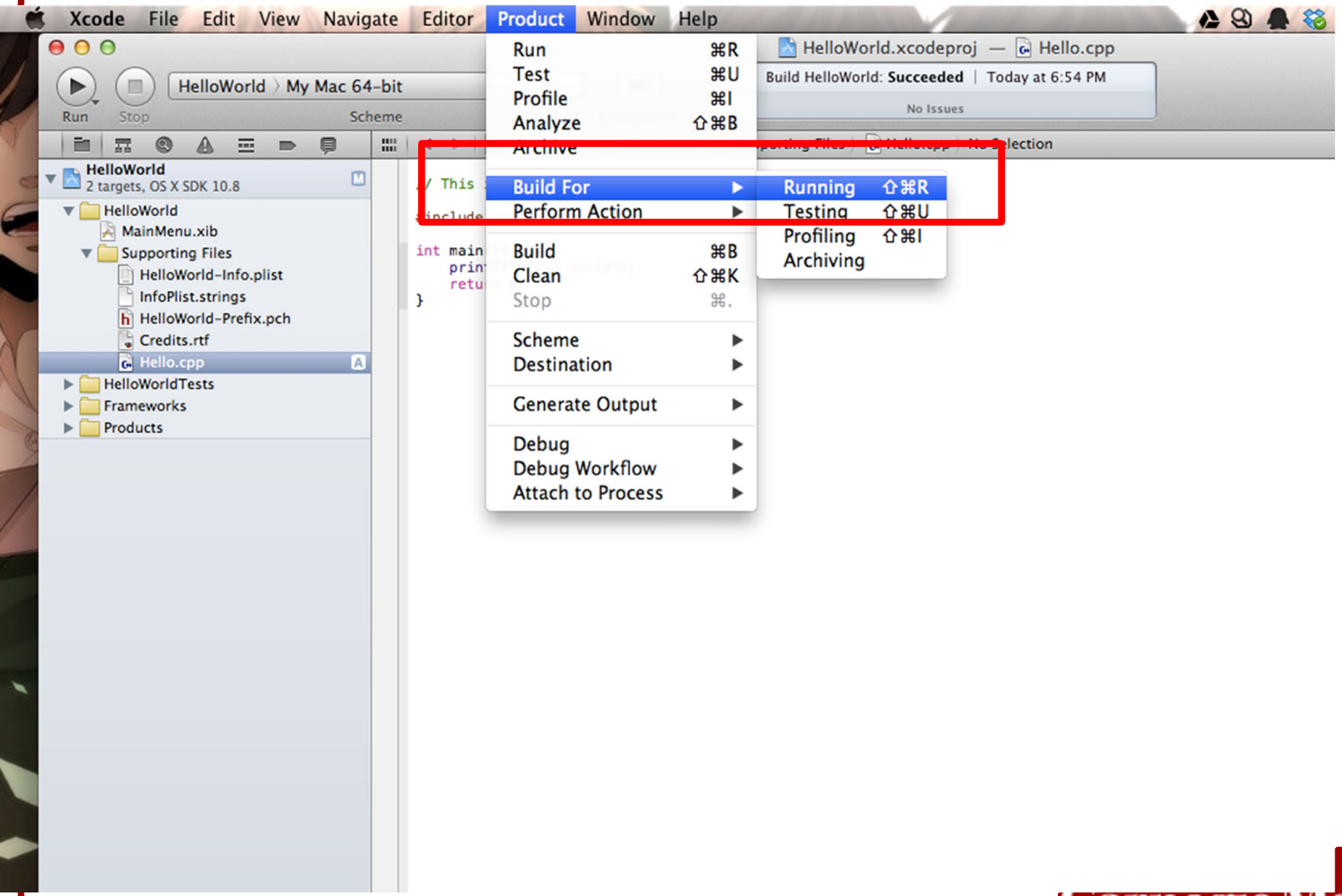
Delete default content in the file

Creating your source file

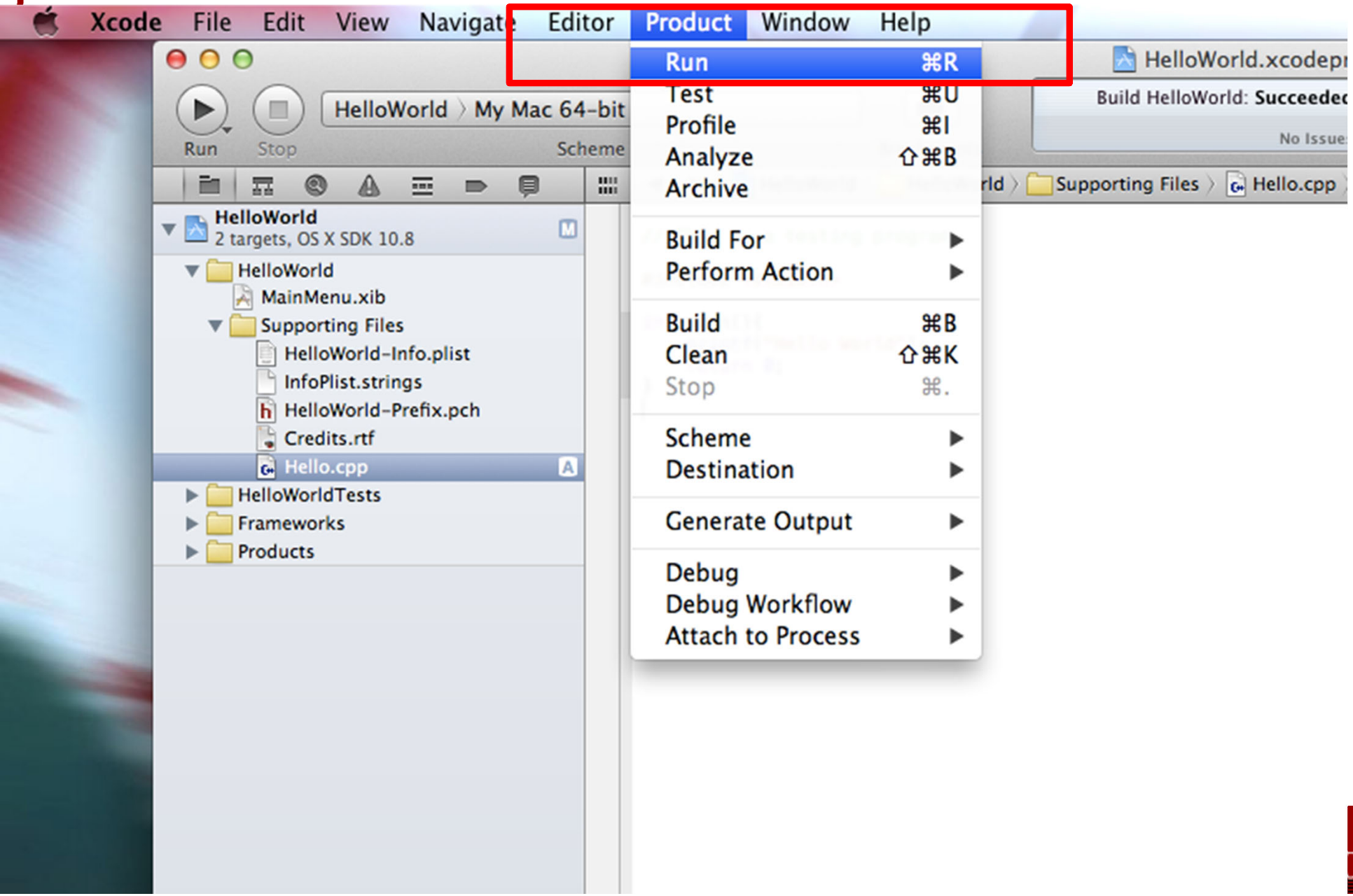


Write a testing code here

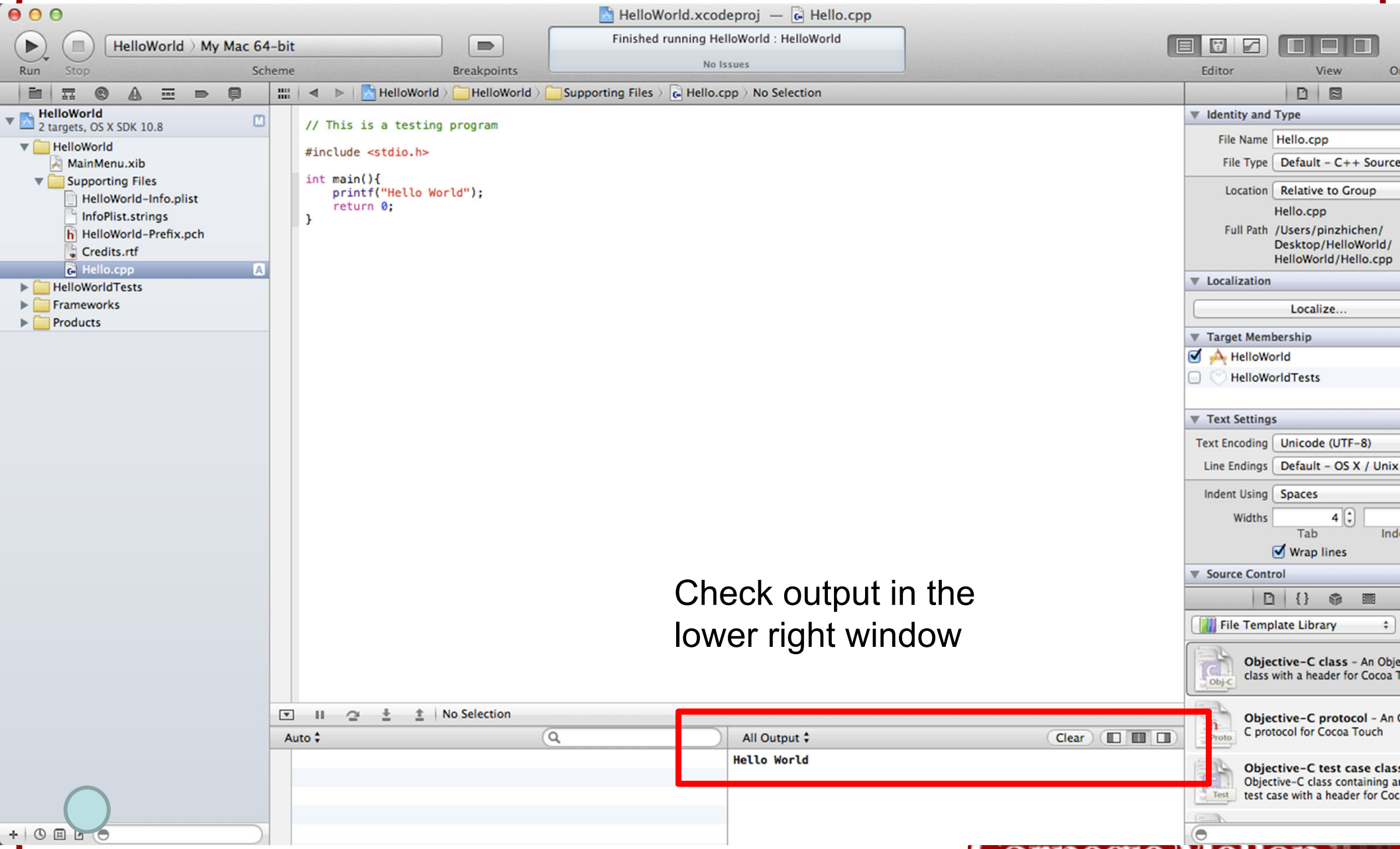
Build and Run



Bulid and Run



Build and Run



Check output in the lower right window

Minimum Requirements for a Useful Program

- Input
- Processing
- Output

Console output

Using printf function

Including a header file

Main function: The program starts from here.

Printf function writes a text string in the console window

```
#include <stdio.h>
int main(void)
{
    printf("Hello World!\n");
    printf("(It's a language tradition)\n");

    return 0;
}
```

Tabs and carriage returns have no meaning in C/C++, but are critical for human use.

Why not cout? Programmer preference, although fancy number formatting may be easier with printf()

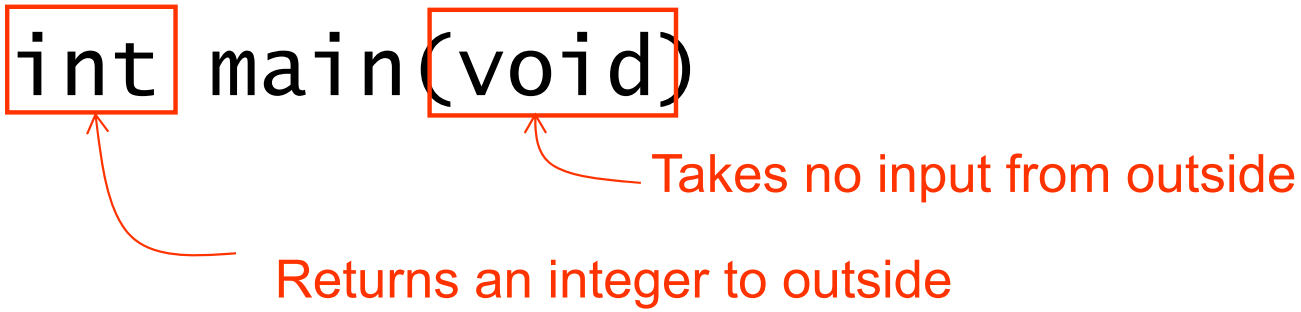
Including a header file

```
#include <stdio.h>
```

This tells the C++ compiler which set of tools you want to use in your program. Example of the tools are:

stdio.h	Standard Input and Output
string.h	Text-String handling
stdlib.h	Standard library (utility)
math.h	Mathematical library
time.h	Timer library

Defining the entry point of program



Every procedure in C language is in the form of a *function*. When the operating system launches the program, the OS calls this **main()** function after the program is initialized.

Thus, a program must have *one and only one* main() function

Writing text string on console window

```
printf("Hello world!\n");
```

Don't forget semi-colon at the end

New line

This function writes a text string in the console window.

Writing text string on console window

```
return 0;
```

This will return zero to the outside. In this case, outside is whoever started the program.

You could return other values, but zero is usually taken to mean “OK”

Alternative Console output

Using cout

Including a header file

"std" is the standard library (more on namespaces later)

Main function: The program starts from here.

cout is the output datastream for the console that you "push" things into

```
#include <iostream>
using namespace std;
int main(void)
{
    cout << "Hello World! " << endl;
    cout << "(It's a language tradition)\n";

    return 0;
}
```

Actually not needed in C++ main function, but kept in for tradition

Better for printing all kinds of stuff even when you don't know the type.
printf() is from C-language, streams are from C++

Basic Data Types and Variables

Integer and Floating Point

```
#include <stdio.h>

int main(void)
{
   int a;
   double b;

   a=5;
   b=5.0;

   printf("a=%d, b=%lf\n",a,b);


   return 0;
}
```

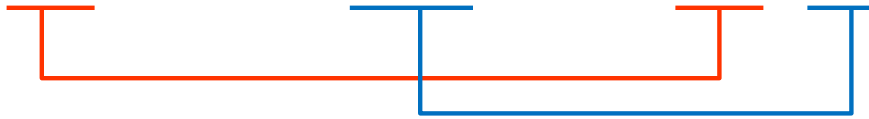
Defining variables

Assigning values

Printing values

Printing integer and double-precision floating point

```
printf("a=%d, b=%lf\n",a,b);
```



"%d" is for an integer, and "%lf" is for a double-precision floating point.

This is a lowercase "L" (for endline)

Or (easier)

```
cout << a << ", " << b << endl;
```

Primitive Data Types

- C++ enforces data typing, so it is important that every value is *declared*.
  - bool: simplest data type, only one bit of storage, 0=false, 1=true
  - char: stores alphanumeric character using ASCII values, 8 bits (one byte) of storage
  - **int**, long long: stores an integer value (use “int” almost always, 4 bytes, upto ± 2 billion)
  - float, **double**: stores a floating point value (use “double” almost always, 8 bytes, 16 decimal places, but float is good for graphics)

- Size of data types change according to system and compiler, so be careful if you are pushing the limits.

Variables and Data Types

- A *variable* is a storage space for a value of a certain data type

Declarations:

```
<type> <variableName> [optional_Initiator];
char mychar;
int countOfThings = 0;
double gravity = 9.81;
```

Yes, “normal” rules for variable name:
- Must start with letter
- No spaces (underscores OK)
- Numbers OK
- No symbols

Strings

- A string is actually an array of char and can be declared like this:

More on this later today

```
char *firstName; // no space allocated
char secondName[25]; // value to be assigned later
char bestNameEver[] = “Nestor Gomez”;
```

- But that’s the C way. C++ is a bit better

```
string bestNameEver = “Nestor Gomez”,
                soSoName = “Nolen Keeys”;
```

Operators

+ - * / all perform as normal

% is the modulo operator (remainder, 17 % 3 returns a value of 2)

Use pow(number, exponent) to raise to a power, sqrt(number) for square root.

Common error:

= is the assignment operator *radius = 6,371*

== is the comparison operator *if (radius==6,371)*

When dividing two integers, the result is also an integer:

`2 / 3` results in **zero**

Be sure to "cast" appropriately by assuring that at least one of the values is a double:

`(double)2 / 3` results in 0.666666667

can also use `double(2) / 3` or `2. / 3`

More Operators

This is two of the vertical bar (above the Enter key, called “pipe”)

- Boolean:

  Not → ! And → && Or → ||

  Also have <= >= !=

- Aggregate operators:

```
overallSum = overallSum + nextItem;
overallSum += nextItem;
```

  Also -=, *=, /=

- Increments:

```
counter = counter + 1;
counter++;
```

  Also have --

- Operator precedence is like in math:

  Please, please my dear Aunt Sally.

Parentheses, power, multiplication/division, addition/subtraction