

Computer System Programming

Introduction

Instructor: Yi Fang
TA: Meiguang Jin



Acknowledging : note that some lecture contents are courtesy of Prof. Ozgur Sinanoglu and Prof. Michail Maniatakos
online C++ learning open sources

Overview



- Course aims and outline
- The Importance of computer programming
- A simple computer program “Hello, World!”

The Aims

Teach/learn

- Fundamental programming concepts
- Key useful techniques
- Basic Standard C++ facilities

After the course, you'll be able to

- Write small C++ programs
- Read much larger programs
- Learn the basics of many other languages by yourself
- Proceed with an “advanced” C++ programming course

After the course, you will not (yet) be

- An expert programmer
- A C++ language expert
- An expert user of advanced libraries

Course Format



- Lectures and Labs
- Three Programming assignments
- One Midterm Exam
- One Final Exam

Course Schedule

NEW YORK
UNIVERSITY



ABU DHABI

Week	Lecture Topic	Project	Exam Schedule
1	1. Introduction to computer systems 2. Basics for computer programming <ul style="list-style-type: none">• 2.1 Data Types and Variables• 2.2 Operators• 2.3 Data input and output• 2.4 Statements and flow control		
2	3. Functions <ul style="list-style-type: none">• 3.1 Defining a Function• 3.2 Call a Function• 3.3 Inline Function• 3.4 Overloading Function	Project 1	
3	3. Functions <ul style="list-style-type: none">• 3.4 Defining a Function• 3.5 Function templates		
4	4. Arrays, Strings and Pointers <ul style="list-style-type: none">• 4.1 Arrays• 4.2 Strings• 4.3 Pointers	Project 2	Midterm
5	5. C++ Object Oriented Programming <ul style="list-style-type: none">• 5.1 Classes and Objects		
6	5. C++ Object Oriented <ul style="list-style-type: none">• 5.2 Inheritance• 5.3 Polymorphism	Project3	
7	6. C++ Files and Streams		Final Exam

Grading



- Programming assignments: 30% (10% each)
- Midterm: 30% (Paper and computer based)
- Final: 40% (Paper and computer based)

Importance of programming

- Our civilization runs on software
 - Most engineering activities involve software
- Examples:
 - Ships (Monitoring, engine, hull pumps etc.)
 - Aircrafts (Signal processing, gadget control etc.)
 - Energy (Communications, visualization, etc.)
 - Social Media, Facebook, twitter and so on

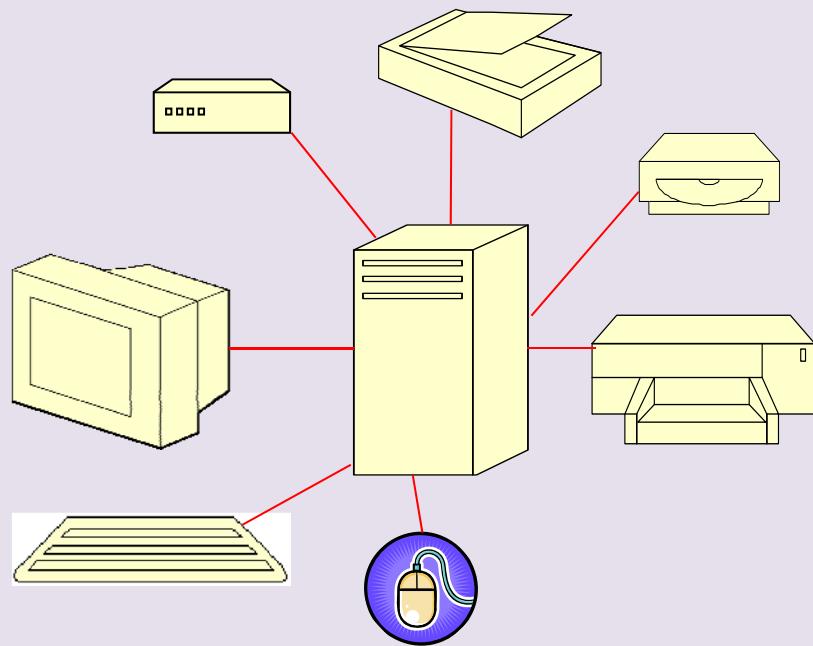


Let's get started !

- We will first learn how a computer operates.

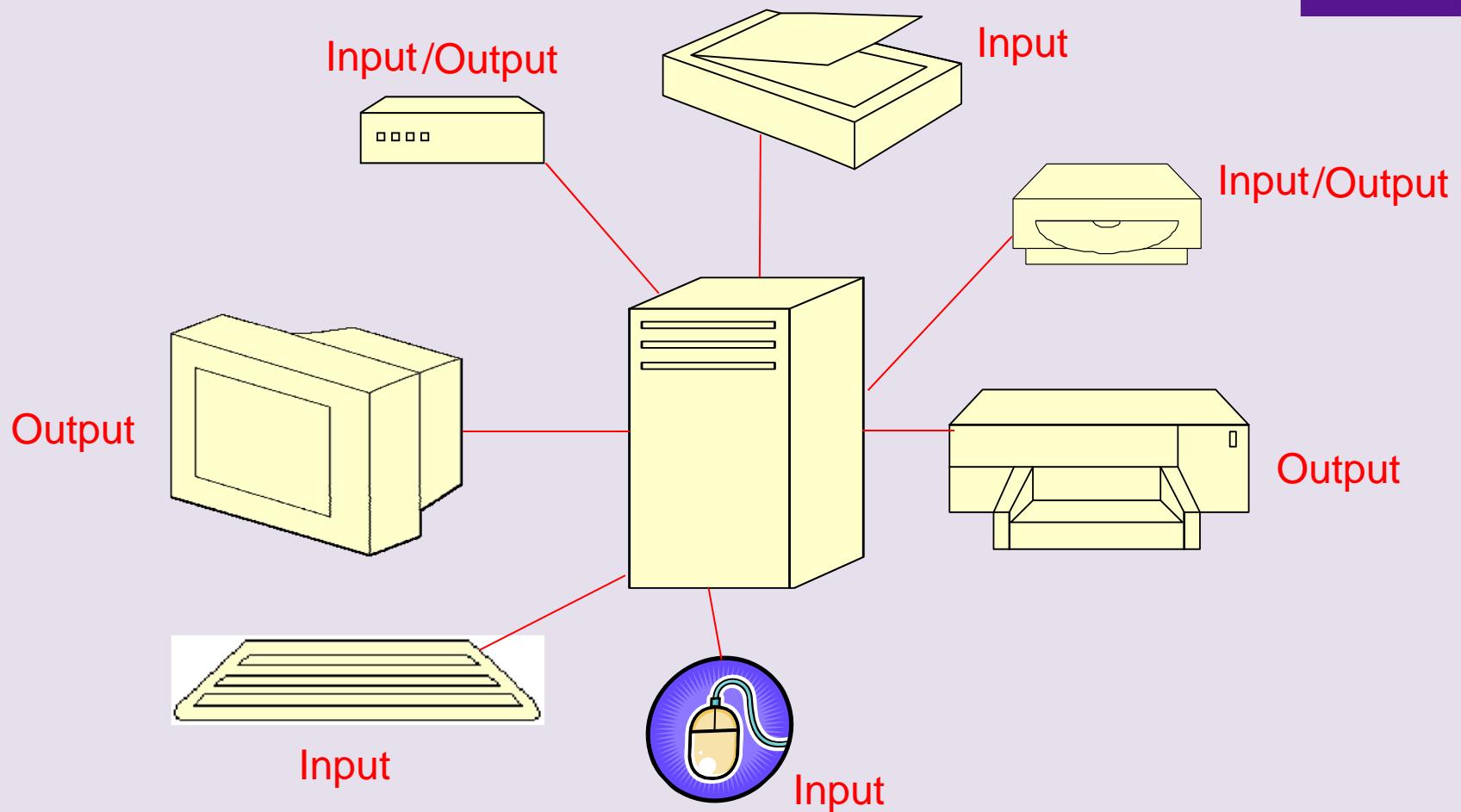


A computer system



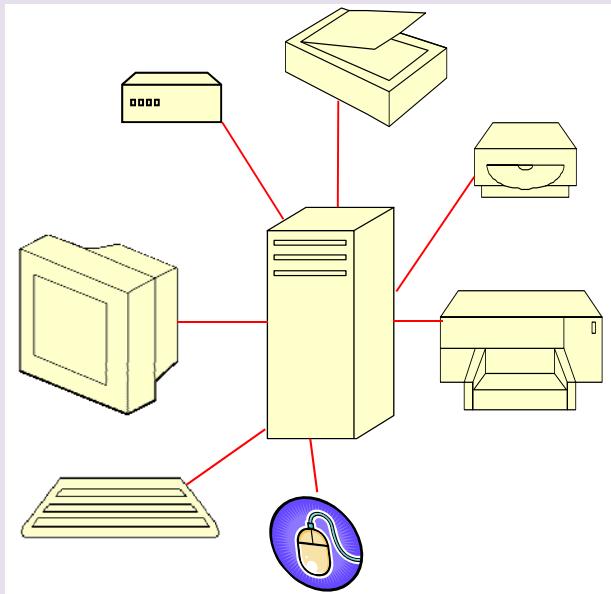
- A computer system is composed of:
 - a monitor,
 - a keyboard,
 - a mouse,
 - and a case (that contains several controlling components such as processor and alike),
 - and also other peripherals like
 - CD player (might have been included in the case),
 - printer,
 - scanner,
 - modem,
 - etc.
- all connected together

Input and output devices



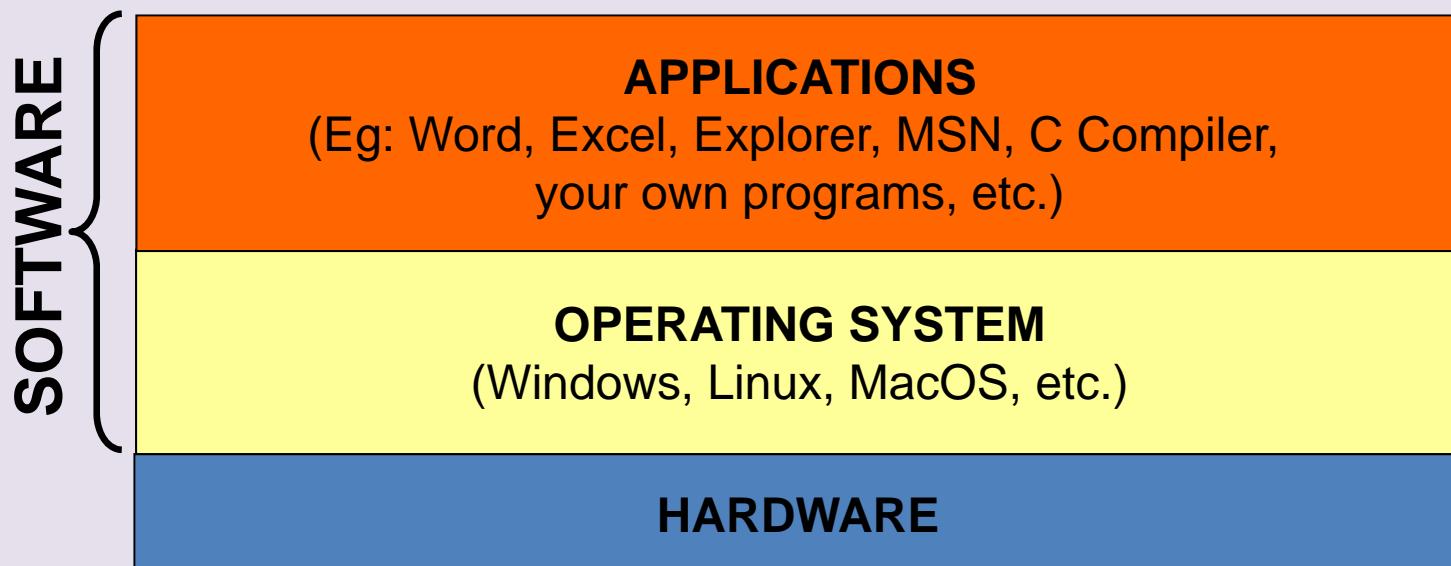
A computer system

- Note that everything could be packed in a single box, but the concepts are the same.



A computer system

- Everything we had in the previous slide is hardware.
 - i.e., physical components that implement what is requested by the software.



A computer system

- In this course, we will learn how to develop our own software (using C++ language), but we need to understand how our programs will be executed by the hardware.

CPU: Central Processing Unit

- In terms of hardware, what is important for us is the CPU.
- It does all processing and control.
 - Everything is controlled and executed by the CPU.



CPU: Central Processing Unit



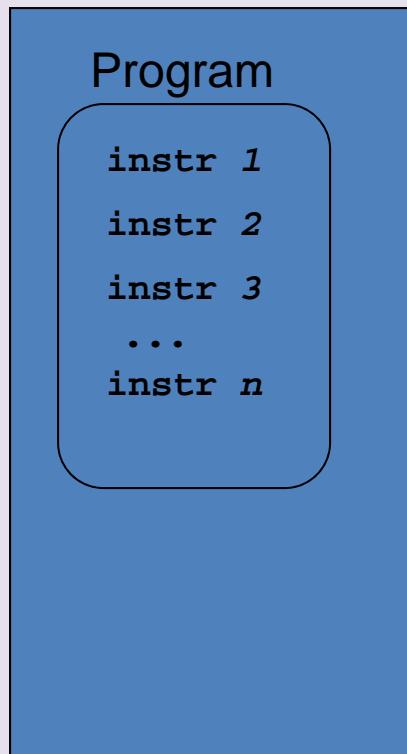
NEW YORK
UNIVERSITY



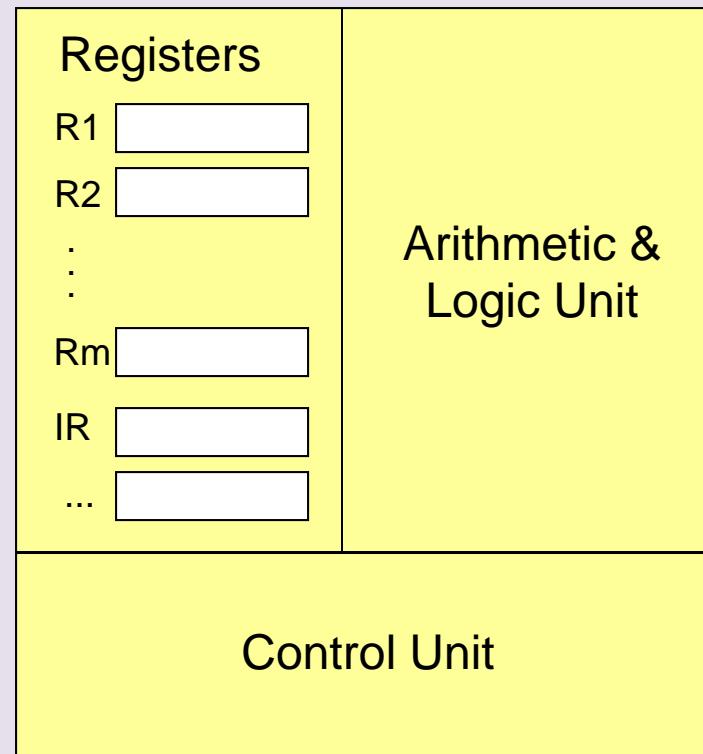
ABU DHABI

How are the instructions executed?

Main Memory



Central Processing Unit (CPU)





How do we write programs?

We write our programs in "*C++ language*" (which is an English-like language)

```
#include <iostream>
int main()
{
    cout << "Hello world!";
    return 0;
}
```

(*source code*)

We use a compiler (such as GCC, Visual C++, etc.) to translate our program from "*C++ language*" to "*machine language*"

Compile & Link

(*object code*)

This is the executable code in "*machine language*."

This is the only thing the computer can understand and run (execute).

```
1110101011001001010
0010101001010000100
1010010101010100010
1001000100101001
```

(*machine code*
or
executable code)

Statement vs. Instruction

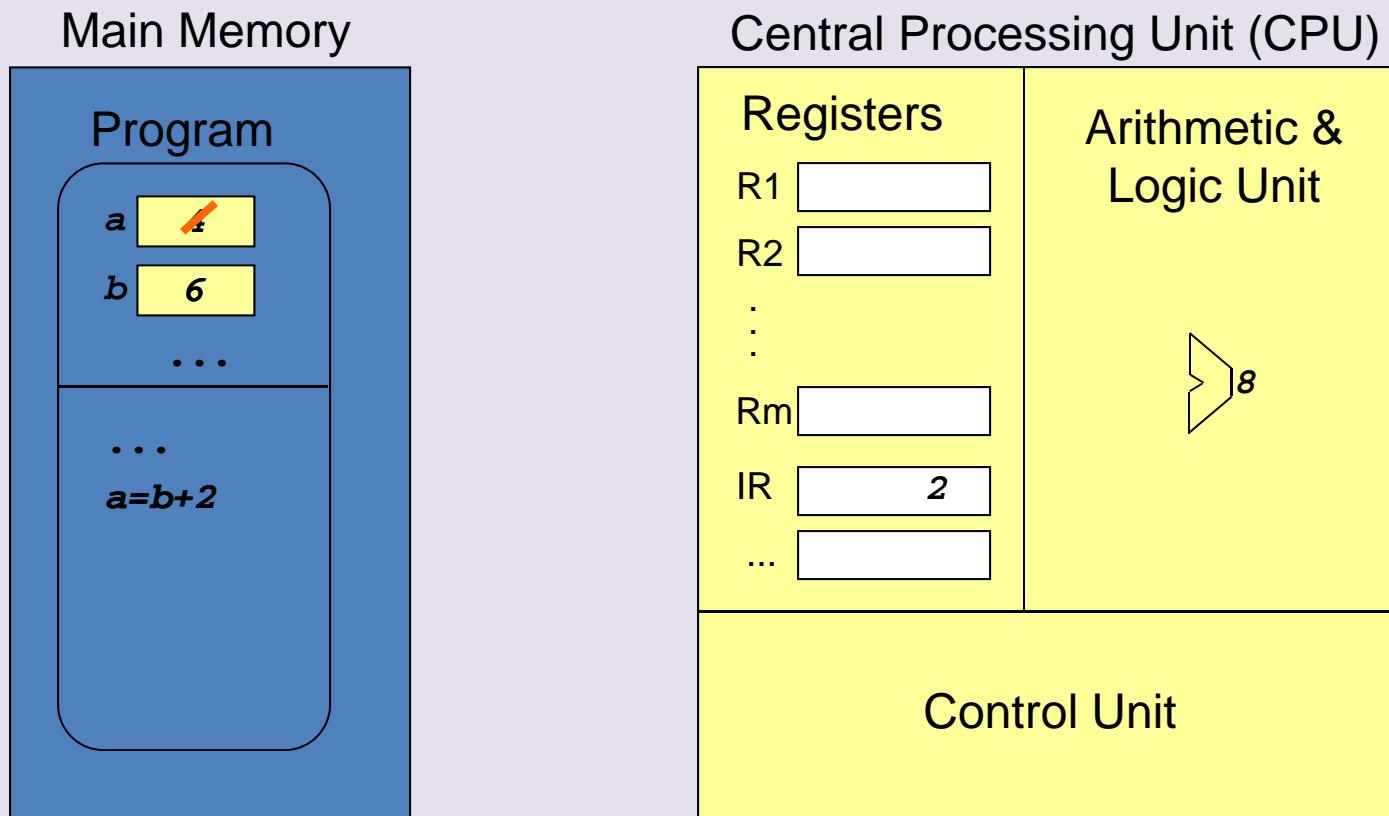
- Our source code (in C++) is composed of statements.
 - Eg: $a = b + c / 2;$
- The corresponding machine code is composed of instructions.
 - Eg: 1101001010110010 (divide c by 2)
 - 0110100100100101 (add it to b)
 - 1010110110111011 (put the result in a)
 - CPU is capable of executing instructions, not statements.
Statements may be too complex.
 - Compiler implements each statement using several instructions.
 - Eg: The statement "a=b+c/2;" can be implemented as
 - » $\text{temp1} = c / 2$
 - » $a = b + \text{temp1}$

Why have input/output?

- A program should not always produce the same output.
 - O/w, you may keep the result and delete the program after you run it for the first time.
- A program should be consistent; i.e., it should not produce random results.
- Therefore, a program should take some input, process it, and produce some output as the result of that input.

Execution of an instruction

- Let's see how an instruction like " $a=b+2$ " is executed.
 - Assume initially a is 4 and b is 6.



Welcome to C++ Programming Language

- Now that we have an overall understanding of the computer, we can start writing C++ programs.

We write our programs in "*C++ language*" (which is an English-like language)

```
#include <stdio.h>
int main()
{
    printf("Hello world!");
    return 0;
}
```

(source code)

We use a compiler (such as GCC, Visual C++, etc.) to translate our program from "*C++ language*" to "*machine language*"

Compile & Link

(object code)

This is the executable code in "*machine language*."

This is the only thing the computer can understand and run (execute).

```
111010101100100101000
101010010100001001010
010101010100010100100
0100101001
```

(*machine code*
or
executable code)

A first program – just the guts...

```
// ...
```

```
int main()
{
    cout << "Hello, world!\n";
    return 0;
}
```

// main() is where a C++ program starts

// output the 13 characters Hello, world!

// followed by a new line

// return a value indicating success

// quotes delimit a string literal

// NOTE: “smart” quotes “ ” will cause compiler problems.

// so make sure your quotes are of the style “ ”

// \n is a notation for a new line

A first program – complete

// a first program:

```
#include "../std_lib_facilities.h" // get the library facilities needed for now
```

```
int main() // main() is where a C++ program starts
{
    cout << "Hello, world!\n"; // output the 13 characters Hello, world!
                                // followed by a new line
    return 0; // return a value indicating success
}
```

// note the semicolons; they terminate statements

// curly brackets { ... } group statements into a block

// main() is a function that takes no arguments ()

// and returns an int (integer value) to indicate success or failure

A second program

// modified for Windows console mode:

```
#include "../std_lib_facilities.h" // get the facilities for this course

int main()                                // main() is where a C++ program starts
{
    cout << "Hello, world\n";               // output the 13 characters hello, world!
                                                // followed by a new line
    keep_window_open();                   // wait for a keystroke
    return 0;                            // return a value indicating success
}

// without keep_window_open() the output window will be closed immediately
// before you have a chance to read the output (on Visual C++ 2010)
```

Hello, world!

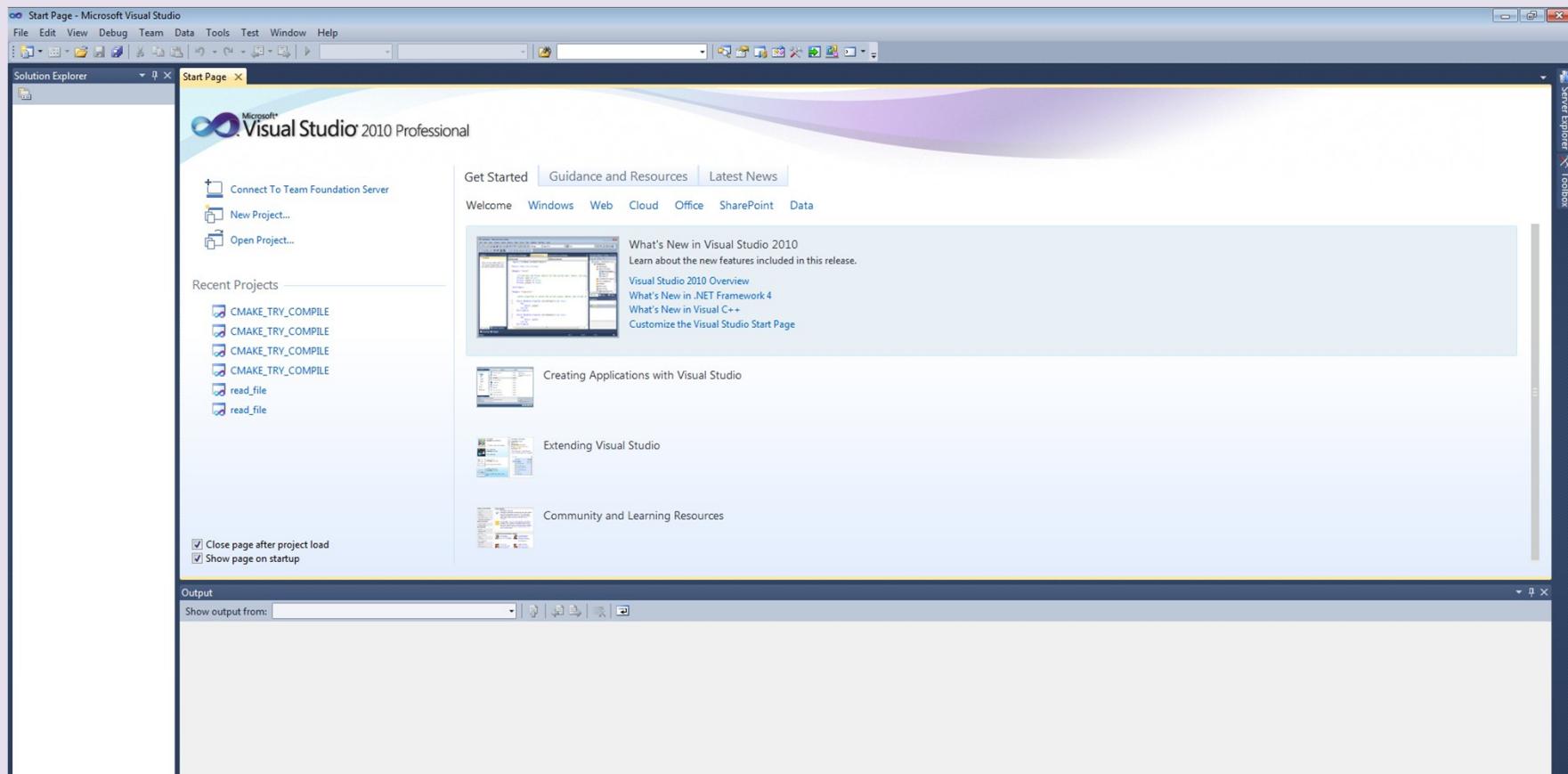
- “Hello world” is a very important program
 - Its purpose is to help you get used to your tools
 - Compiler
 - Program development environment
 - Program execution environment
 - Type in the program carefully
 - After you get it to work, please make a few mistakes to see how the tools respond; for example
 - Forget the header
 - Forget to terminate the string
 - Misspell return (e.g. retrun)
 - Forget a semicolon
 - Forget { or }
 - ...

Hello world

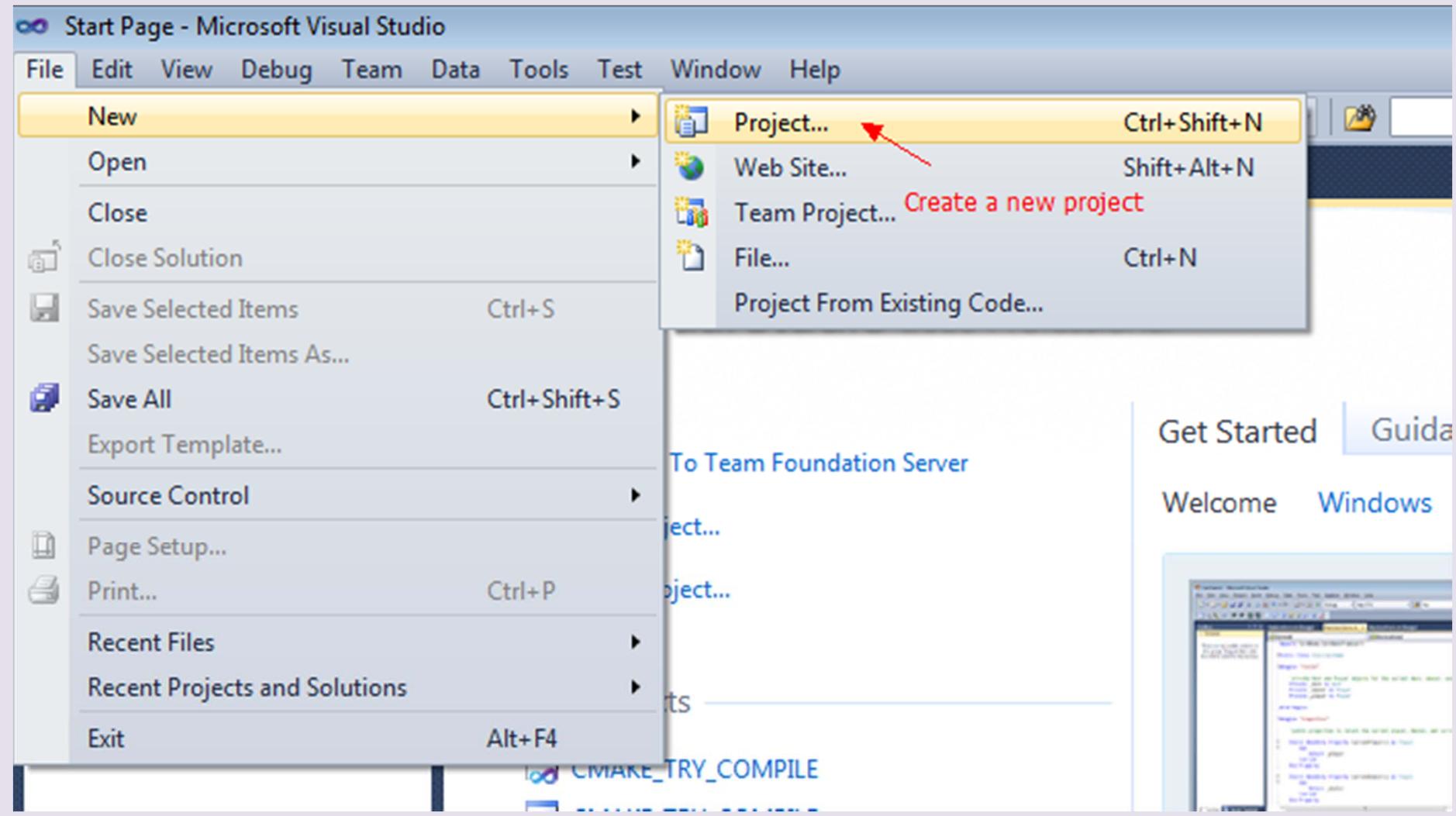
- It's almost all "boiler plate"
 - Only cout << "Hello, world!\n" directly does anything
- That's normal
 - Most of our code, and most of the systems we use simply exist to make some other code elegant and/or efficient
 - "real world" non-software analogies abound
- "Boiler plate," that is, notation, libraries, and other support is what makes our code simple, comprehensible, trustworthy, and efficient.
 - Would you rather write 1,000,000 lines of machine code?
- This implies that we should not just "get things done"; we should take great care that things are done elegantly, correctly, and in ways that ease the creation of more/other software:
 - Style Matters!

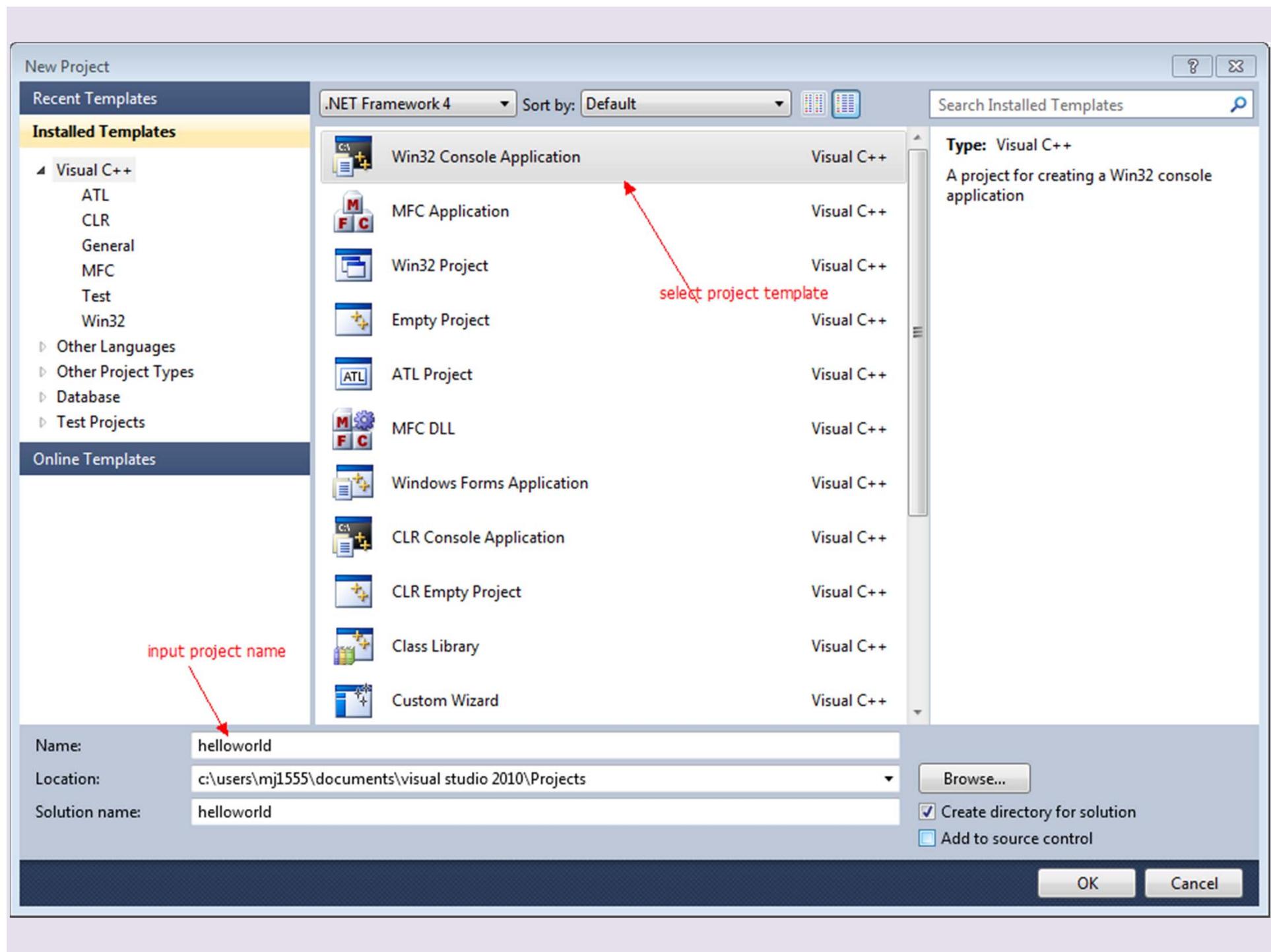
How to set up “hello world” in Visual Studio 2010

Open Visual Studio



Create a new project







Welcome to the Win32 Application Wizard

Overview

Application Settings

These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

next

< Previous

Next >

Finish

Cancel



Application Settings

Overview

Application Settings

Application type:

- Windows application
- Console application
- DLL
- Static library

Additional options:

- Empty project
- Export symbols
- Precompiled header

select empty project

Add common header files for:

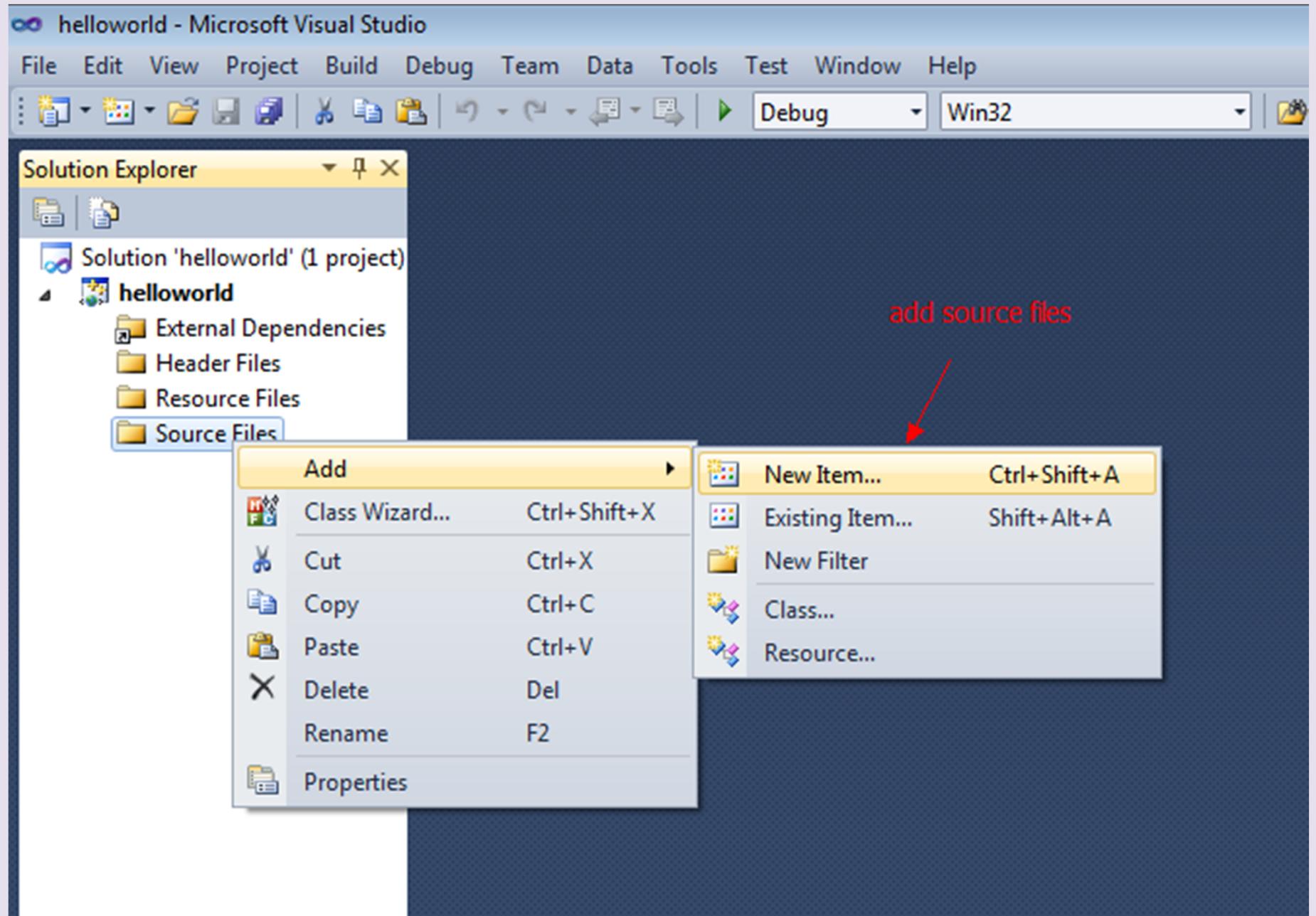
- ATL
- MFC

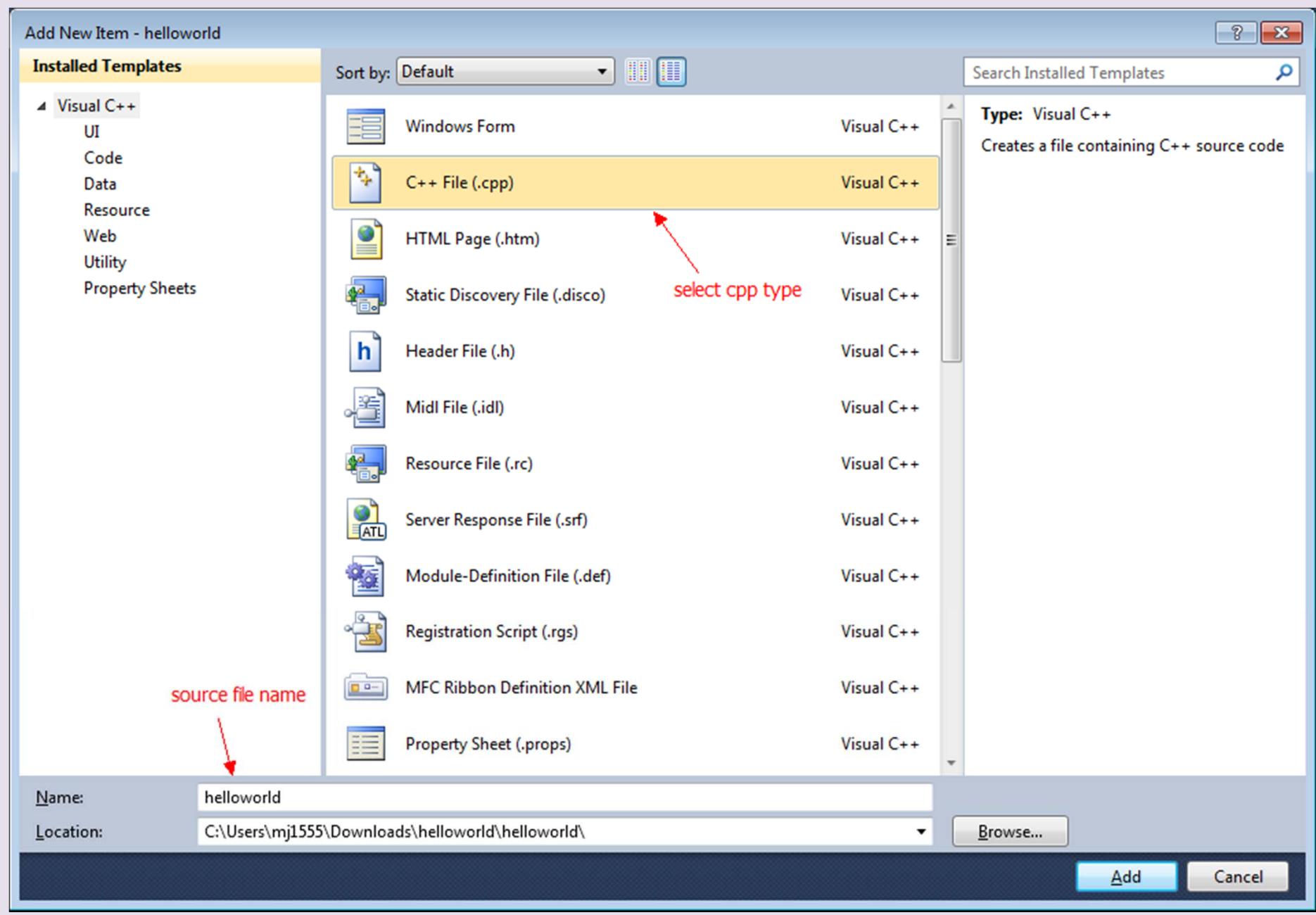
< Previous

Next >

Finish

Cancel





helloworld - Microsoft Visual Studio

File Edit View Project Build Debug Team Data Tools Test Window Help

Solution Explorer helloworld.cpp X

(Global Scope)

```
#include <stdio.h>
int main()
{
    printf("hello world!\n");
    return 0;
}
```

run the program

C:\Windows\system32\cmd.exe

hello world!
Press any key to continue . . .

output

The next lecture

- We will take about Data Types and Variables, Operators, Data input and output