CIS 129
Advanced Computer Programming

Chapter 2: Basic elements of C++ and Input / Output

Mr. Horence Chan

Why use a language like C++?

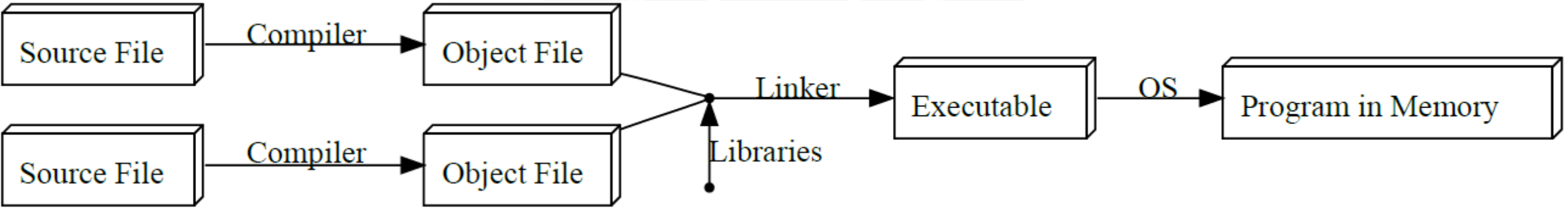
The advantages:

1. **Conciseness**: programming languages allow us to express common sequences of commands more concisely. C++ provides some especially powerful shorthand.
2. **Maintainability**: modifying code is easier when it entails just a few text edits, instead of rearranging hundreds of processor instructions. C++ is object oriented, which further improves maintainability.
3. **Portability**: different processors make different instructions available. Programs written as text can be translated into instructions for many different processors; one of C++'s strengths is that it can be used to write programs for nearly any processor.

C++ is a high-level language: when you write a program in it, the shorthand are sufficiently expressive that you don't need to worry about the details of processor instructions. C++ does give access to some lower-level functionality than other languages (e.g. memory addresses).

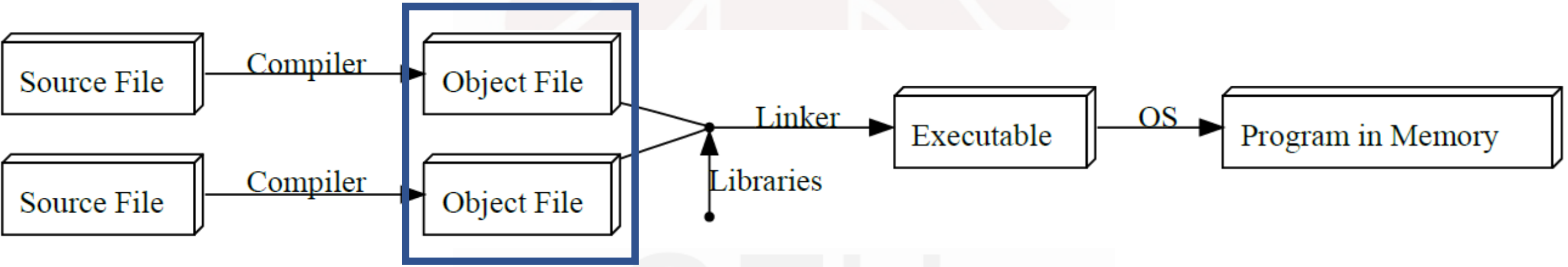
The Compilation Process

- A program goes from text files (or source files) to processor instructions as follows:



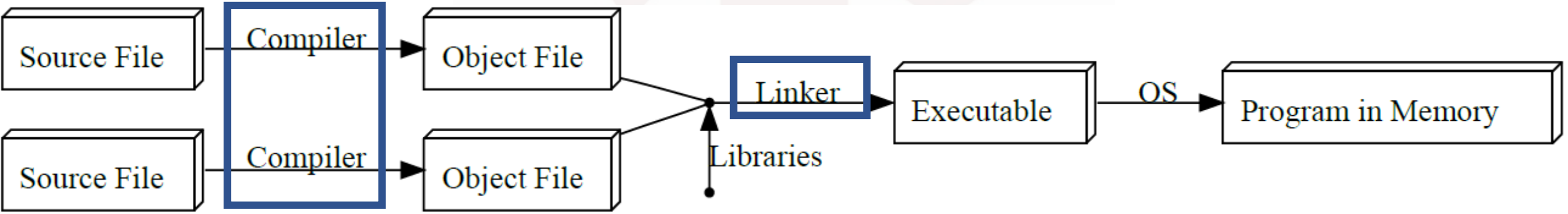
The Compilation Process

- **Object file**: intermediate file that represent an incomplete copy of the program, each source file only expresses a piece of the program, so when it is compiled into an object file, the object file has some markers indicating which missing pieces it depends on.



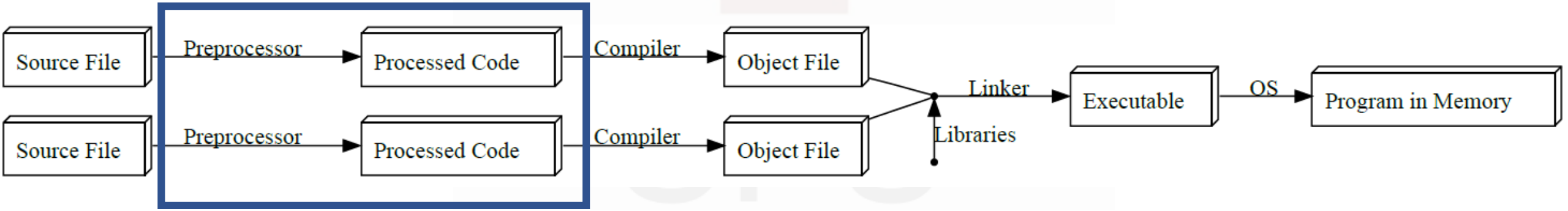
The Compilation Process

- **Linker**: take those object files and the compiled libraries of predefined code that they rely on, fills in all the gaps, and spits out the final program, which can then be run by the operating system (OS).
- The **compiler** and **linker** are just regular programs. The step in the compilation process in which the compiler reads the file is called *parsing*.



The Compilation Process

- In C++, all these steps are performed ahead of time, before you start running a program. In some languages, they are done during the execution process, which takes time. This is one of the reasons C++ code runs far faster than code in many more recent languages.
- C++ actually adds an extra step to the compilation process: the code is run through a **preprocessor**, which applies some modifications to the source code, before being fed to the **compiler**.



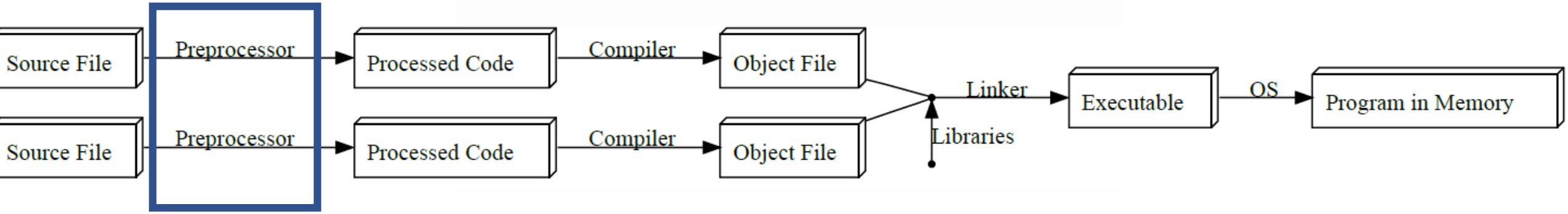
General Notes on C++

- C++ is immensely popular, particularly for applications that require speed and/or access to some low-level features. C++ is a set of extensions to the C programming language. C++ extends C.
- C++ is strong in writing **console programs** (text-based programs). Though you can write graphical programs in C++, it is much hairier and less portable than console programs.
- Everything in C++ is **case sensitive**: `someName` is not the same as `SomeName`.

“Hello World” in C++

```
#include <iostream>
using namespace std;
// My first C++ program!
int main(void)
{
        cout << "hello world!" << endl;
        return 0;
 }
```

- Lines beginning with “#” are preprocessor commands, which usually change what code is actually being compiled.
- `#include` tells the preprocessor to dump in the contents of another file
- `iostream` file, which defines the procedures for _____ and _____.



“Hello World” in C++

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

// My first C++ program!

int main(void)

{
      cout << "hello world!" << endl;
      return 0;
 }
```

- In C++, identifiers can be defined within a context – sort of a directory of names – called a _____.
- This line tells the compiler that it should look in the `std` namespace for any identifier we haven’t defined.
- In this example, we’re telling the compiler to look for `cout` in the `std` namespace, in which many _____ C++ identifiers are defined.

“Hello World” in C++

```
#include <iostream>
using namespace std;
// My first C++ program!
int main(void)
{
	cout << "hello world!" << endl;
	return 0;
}
```

- “//” indicates that everything following it until the end of the line is a _____, it is ignored by the compiler.
- Another way to write a comment is to put it between /* and */
  - e.g. x = 1 + /*sneaky comment here*/ 1;
  - A comment of this form may span multiple lines.
- Comments exist to explain non-obvious things going on in the code. Please use them in your code well!

“Hello World” in C++

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

// My first C++ program!

int main(void)
{
      cout << "hello world!" << endl;
      return 0;
 }
```

- `int main() {…}` defines the code that should execute when the program starts up.
- `void` is function in C++
- The _____ represent grouping of multiple commands into a block.

“Hello World” in C++

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

// My first C++ program!

int main(void)

{
        cout << "hello world!" << endl;
        return 0;
 }
```

- `cout <<` : This is the syntax for _____ some piece of text to the screen.
- `endl`: _____, if there is anything need to output afterwards, it will start a new line.
- **Strings:** A sequence of characters such as `hello world` is known as a *string*. A string that is specified explicitly in a program is a *string literal*.
- Every statement ends with a _____ (except preprocessor commands and blocks using {}).
- Forgetting these semicolons is a common mistake among new C++ programmers.

“Hello World” in C++

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

// My first C++ program!

int main(void)

{
        cout << "hello world!" << endl;
        return 0;
 }
```

- `return 0` indicates that the program should tell the operating system it has _____.
- This syntax will be explained in the chapter of functions; for now, just include it as the last line in the `main` block.

Tokens

Tokens are the minimal chunk of program that have meaning to the compiler – the smallest meaningful symbols in the language.

Token type	Description/Purpose	Examples
Keywords	Words with special meaning to the compiler	
Identifiers	Names of things that are not built into the language	
Literals	Basic constant values whose value is specified directly in the source code	`"Hello, world!", 24.3, 0, 'c'`
Operators	Mathematical or logical operations	
Punctuation/ Separators	Punctuation defining the structure of a program	
Whitespace	Spaces of various sorts; ignored by the compiler	Spaces, tabs, newlines, comments

Values and Statements

- A ***statement*** is a unit of code that does something – a basic building block of a program.
- An ***expression*** is a statement that has a ***value*** – for instance, a number, a string, the sum of two numbers, etc.
  - 4 + 2, x - 1, and "Hello, world!\n" are all expressions.
- Not every statement is an expression. It makes no sense to talk about the value of an `#include` statement, for instance.

Operators

- We can perform arithmetic calculations with *operators*. Operators act on expressions to form a new expression. For example, we could replace "`Hello, world!\n`" with `(4 + 2) / 3`, which would cause the program to print the number 2. In this case, the “+” operator acts on the expressions “4” and “2” (its operands).
- Operator types examples:
  - **Mathematical**: +, -, *, /, and () have their usual mathematical meanings, including using - for negation. % (the modulus operator) takes the remainder of two numbers: 6 % 5 evaluates to 1.
  - **Logical**: used for “and,” “or,” and so on.

Data Types

- Every expression has a type – a formal description of what kind of data its value is.
- For instance:
  - `0` is an integer
  - `3.142` is a *floating-point* (decimal) number
  - `"Hello, world!\n"` is a *string* value (a sequence of characters).
- Data of different types take a different amounts of memory to store.
- Examples of data types:

Type Names	Description	Size	Range
`char`	Single text character or small integer. Indicated with single quotes (`'a'`, `'3'`).	1 byte	signed: -128 to 127 unsigned: 0 to 255
`int`	Larger integer.	4 bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
`bool`	Boolean (true/false). Indicated with the keywords `true` and `false`.	1 byte	Just true (1) or false (0).
`float`	Floating point number.	4 bytes	+/- 3.4e +/- 38 (6 digits)

Data Types

- An operation can only be performed on compatible types. You can add `34` and `3`, but you can't take the remainder of an integer and a _____.
- An operator also normally produces a value of the same type as its operands; thus, `1 / 4` evaluates to _____ because with two integer operands, `/` truncates the result to an _____. To get 0.25, you'd need to write something like _____.

Data Types

- What is the output of the statements?

```
int x = 2147483647;
cout << x + 1;
```

PSY - GANGNAM STYLE (강남스타일) M/V

officialpsy

Subscribe 7,600,830

-2142584554

Add to Share More

8,761,309 1,139,833

Variables

- We might want to give a value a name so we can refer to it later.
- We do this using *variables*. A variable is a named location in memory.
- The name of a variable is an identifier token. Identifiers may contain numbers, letters, and underscores (_), and may not start with a _____.
- For example, say we wanted to use the value `4 + 2` multiple times. We might call it `x, num, number_1, etc`.

Variables

```
1 # include <iostream>
2 using namespace std ;
3 int main () {
4       int x;
5       x = 4 + 2;
6       cout << x / 3 << ' ' << x * 2;
7 return 0;
8 }
```

- Line 4 is the _____ of the variable `x`. We must tell the compiler what type `x` will be so that it knows how much memory to reserve for it and what kinds of operations may be performed on it.
- Line 5 is the _____ of `x`, where we specify an initial value for it. This introduces a new operator: `=`, the assignment operator. We can also change the value of `x` later on in the code using this operator.
- A single statement can be replaced that does both _____ and _____:

```
int x = 4 + 2;
```

- Note how we can print a sequence of values by "chaining" the `<<` symbol. (line 6)

Input and output

```
1 # include <iostream>
2 using namespace std ;
3 int main () {
4     int x;
5     cin >> x;
6     cout << x / 3 << ' ' << x * 2;
7 return 0;
8 }
```

- `cout <<` (line 6) is the syntax for _____ values.
- `cin >>` (line 5) is the syntax for _____ values.

Input and output

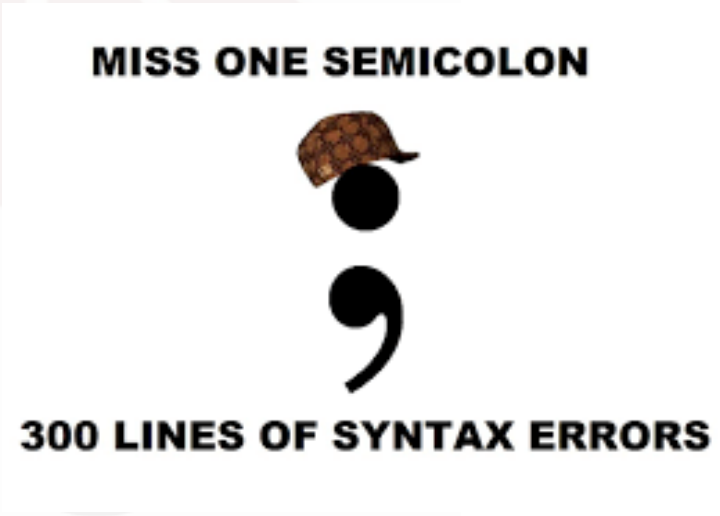
```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4 int main()
5 {
6     string str;
7     cout << "Please enter some words: ";
8     getline(cin, str);
9     cout << "The words enter are: ";
10    cout << str<< endl;
11    return 0;
12}
```

- `# include <string>` is used to declare _____ variable

- `getline()` is used when we need to received _____ in the same line and put them into _____ variable

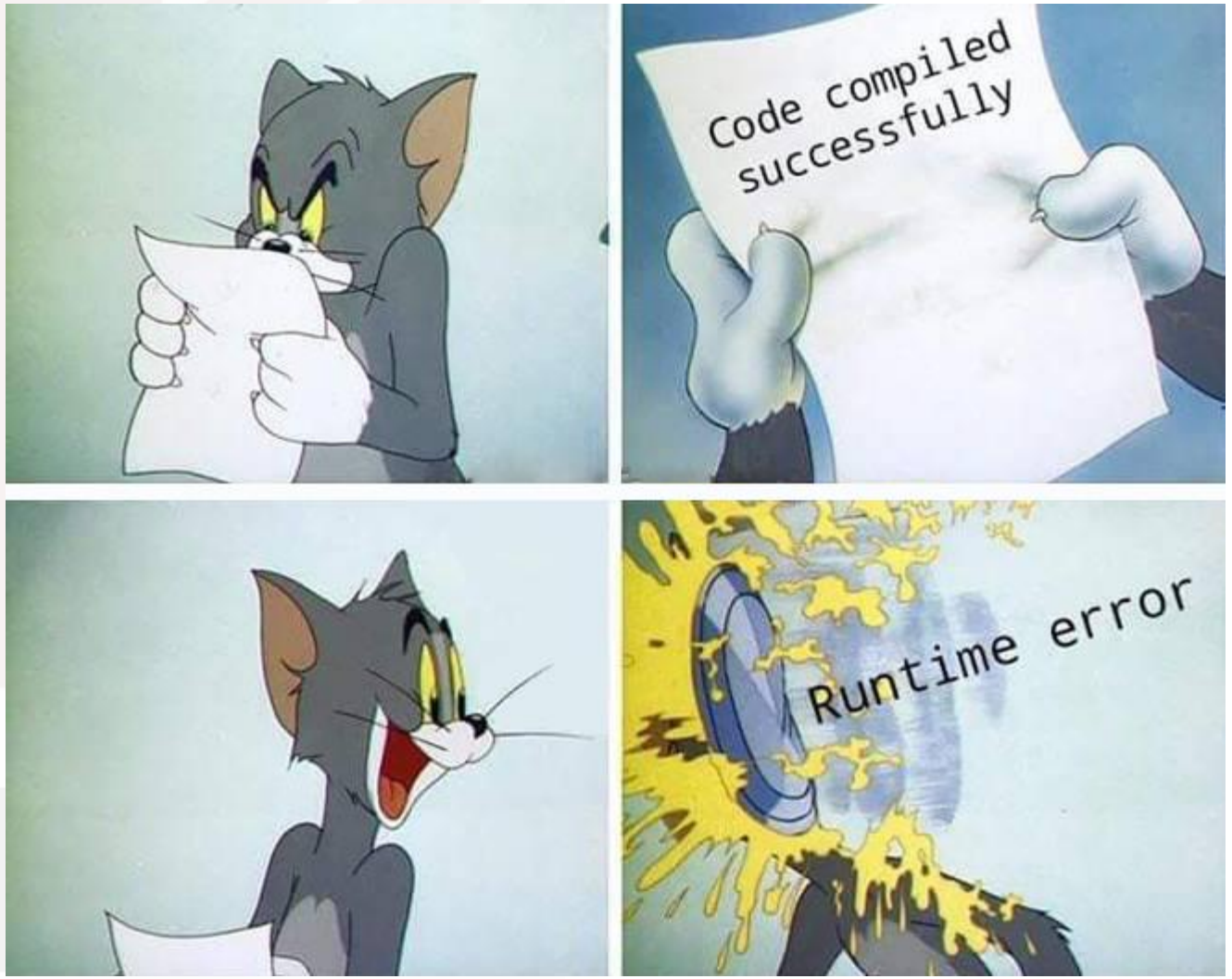
Debugging

- There are two kinds of errors you'll run into when writing C++ programs
- _____ : problem raised by the compiler, generally resulting from violations of the syntax rules or misuse of types. These are often caused by typos and the like.



Debugging

- _____: problem that you only spot when you run the program, you did specify a legal program, but it doesn’t do what you wanted it to. These are usually more tricky to catch, since the compiler won’t tell you about them.



Manipulators

- When using manipulators, we need to type `#include <iomanip>` in the beginning
- **setprecision():** sets the _____ to be used to format floating-point values on output operations.
- **fixed**: write floating-point values in _____ notation.
- **showpoint**: outputs floating-point numbers with a _____ and _____.

Manipulators

```
1 # include <iostream>
2 # include <iomanip>
3 using namespace std ;
4 int main () {
5 float x = 12.3456;
6 cout << fixed << showpoint;
7 cout << setprecision(1) << x << endl;
8 cout << setprecision(3) << x << endl;
9 }
```

- This is a method to round off a number to decimal places
- The output should be a fixed decimal format with decimal point

- Round off to 1 decimal places
- Round off to 3 decimal places

Output:

12.3

12.346

Manipulators

- **setw()**: formats the output of an expression in a specific number of _____; the default output is right-justified.
- **setfill()**: fill the _____ columns on an output device with character.
- **left**: _____ the output
- **right**: _____ the output

Manipulators

- For example, here is a price list of a restaurant
- All the food are align to the left
- All the prices (numbers) are align to the right
- In order to align the above items probably, a certain number of dots “.” and blanks “ ” are placed between the food, “$” sign and numbers

```
Price list of a restaurant
Cheese burger: ..... $  14
Fries: ............. $   9
Coke: .............. $   4
Ice cream: ......... $  11
Angus Steak: ....... $ 100
```

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
 cout << "Price list of a restaurant"<<endl;
 cout << setw(20)
      << setfill('.')
      << left
      << "Cheese burger: "
      << " $"
      << setw(4)
      << setfill(' ')
      << right
      << "14"
      << endl;
}
```

```
Price list of a restaurant
Cheese burger: ..... $  14
```

//set the _____ of the output "Cheese burger: " be 20

//fill the _____ of the output with ____

//align "Cheese burger: " to the _____

//set the _____ of the output price (14) be 4

//fill the _____ of the output with _____

//align (14) to the _____

Try to type the remaining parts with similar fashion!

Location of .cpp file in Visual Studio

- Right click the project file
- Click “Open Folder in File Explorer”
- .cpp file is shown in the directory