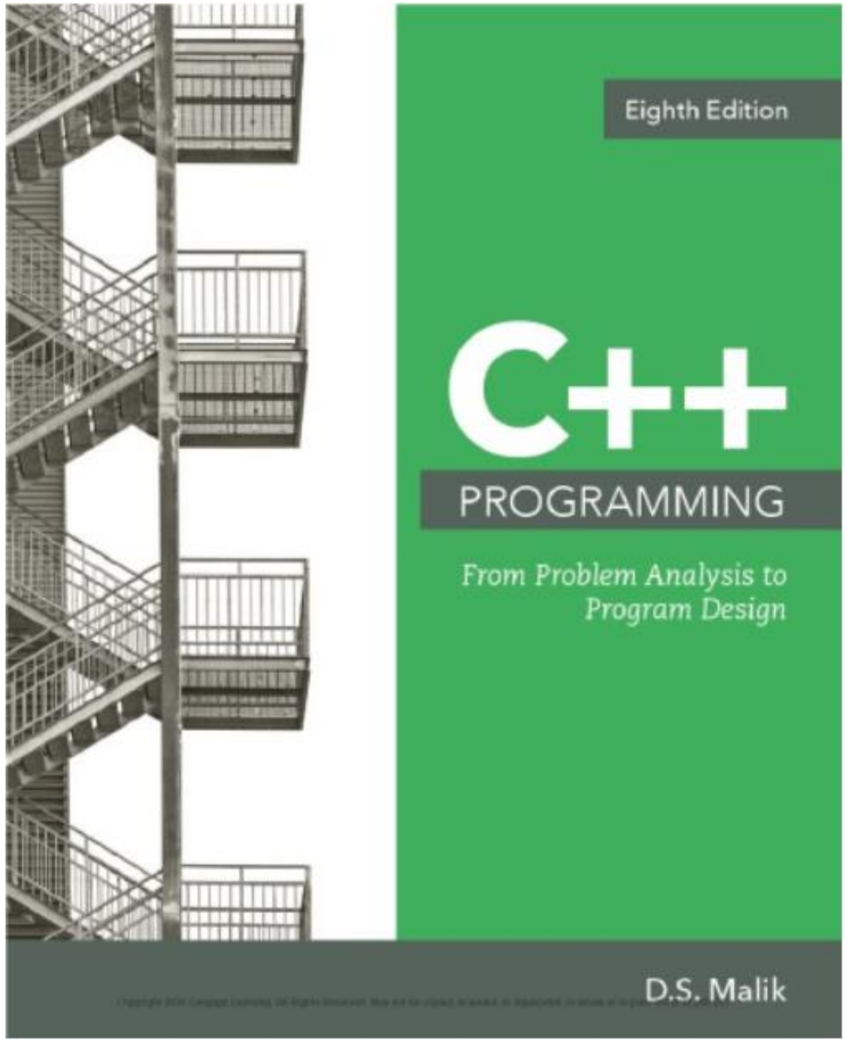
Programming Principles (MT162)

Lecture 2

Dr. Ahmed Fathalla

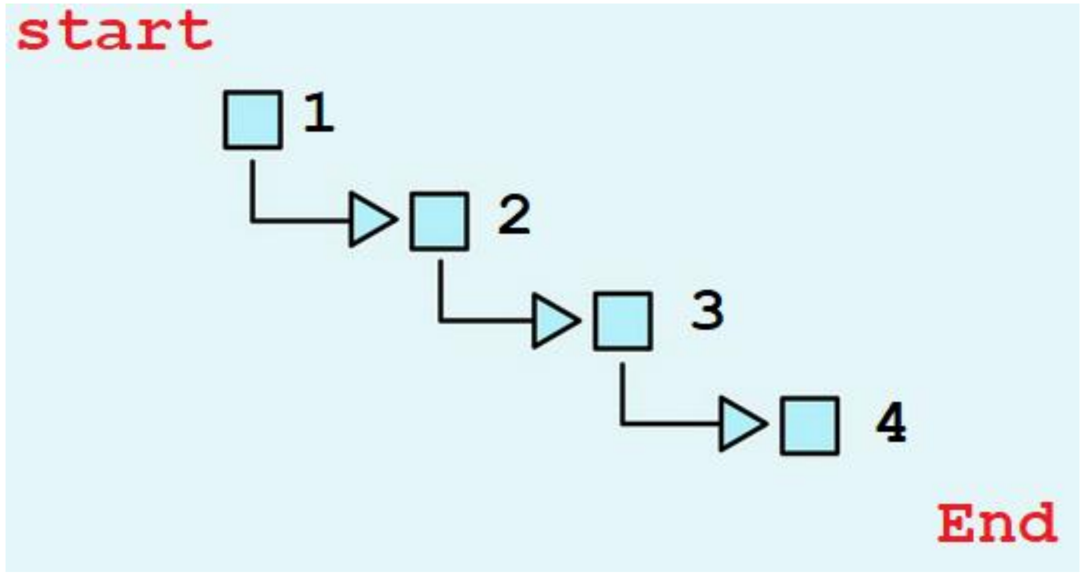
Resources and References

Book: "C++ Programming: From Problem Analysis to Program Design"



Writing a program

- Sequence of instructions.



Exercise

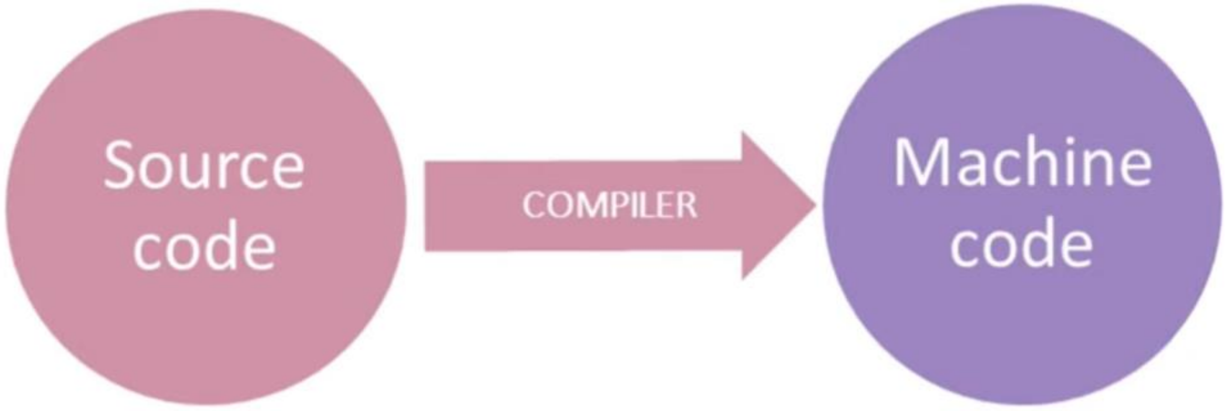
- Ask a user to input two numbers, then divide the first by the second.
    - Expect different types of errors.
        - Division by zero
        - Non numeric values

Machine Languages, Assembly Languages and High-Level Languages

- **Programming Languages**: Fall into three categories
    - Machine languages.
    - Assembly languages (low-level programming language).
    - High-level languages.

The Evolution of Programming Languages

- High-level languages include Basic, FORTRAN, COBOL, **Java**, **C++**, and **C#**.
- Compiler: translates a program written in a high-level language machine language.



Processing a C++ Program (pages 9-10)

- C++ is a **compiled language**, meaning your program's source code must be translated (compiled) before it can be run on your computer.

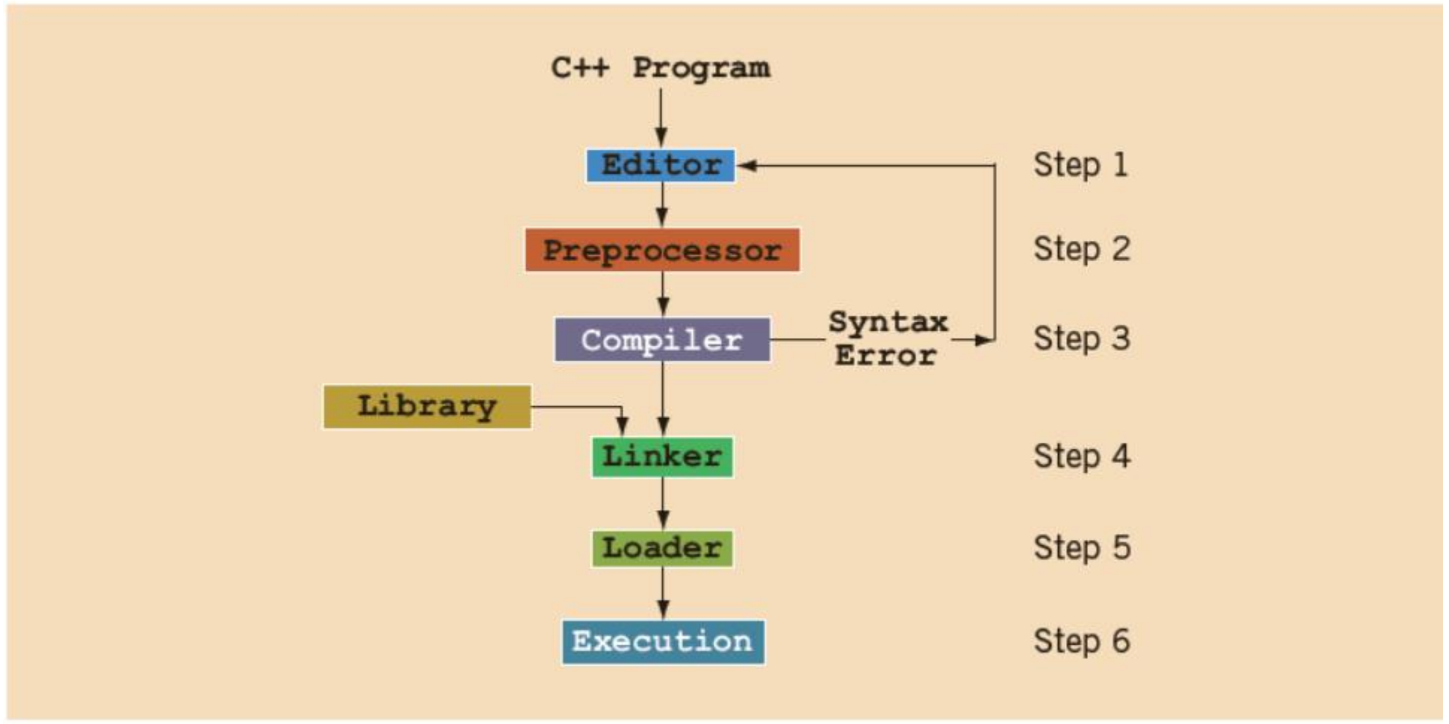


FIGURE 1-2 Processing a C++ program

Programming with the Problem Analysis–Coding–Execution Cycle.

(Page 11-14)

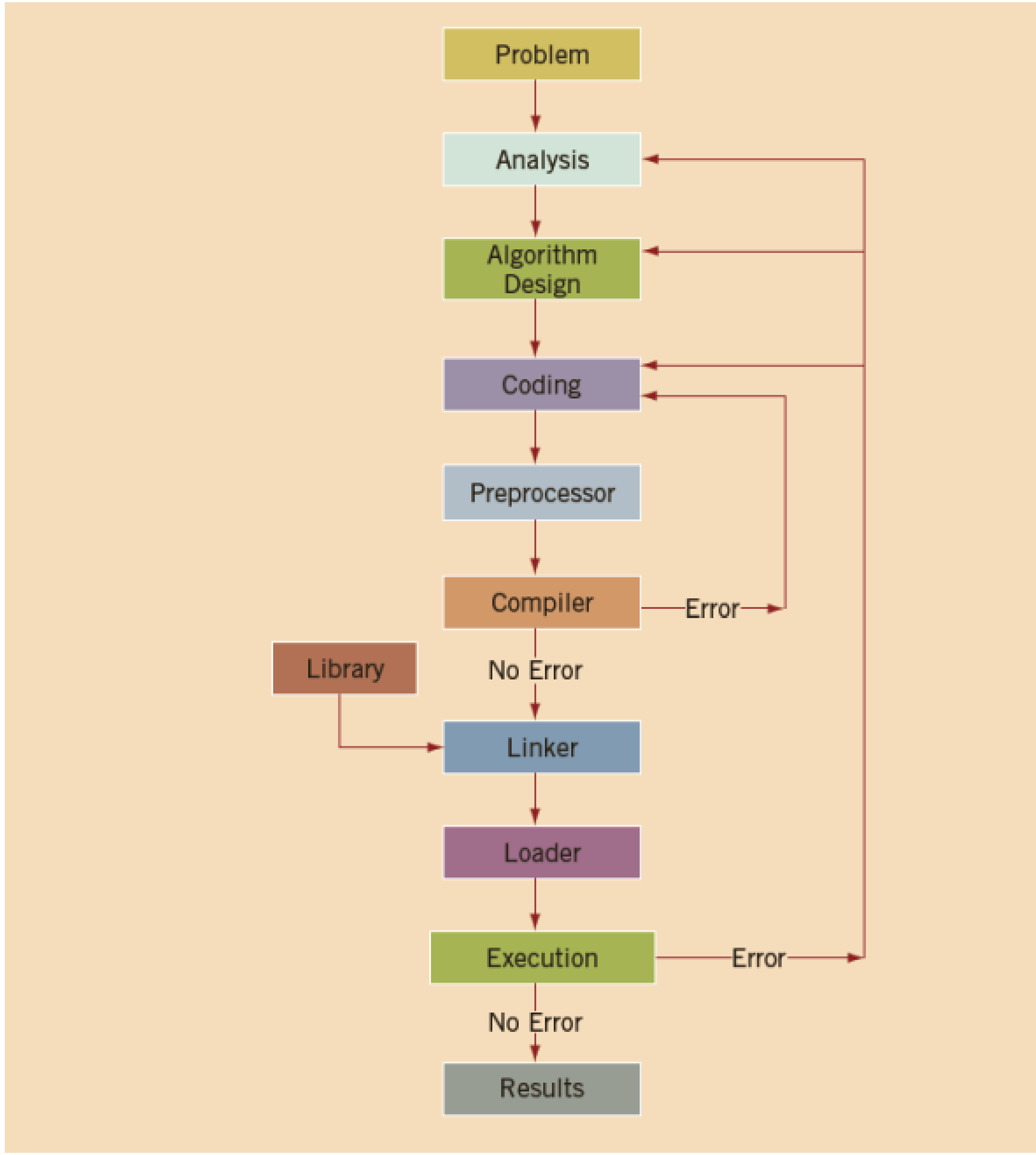


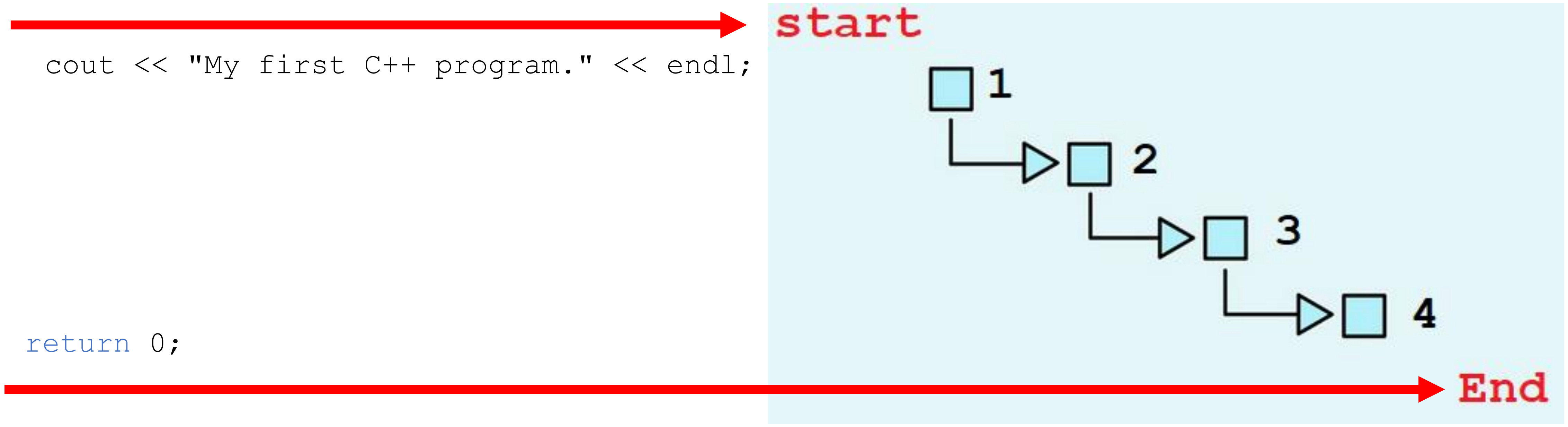
FIGURE 1-3 Problem analysis–coding–execution cycle

Suggested softwares for C++

- Windows/Desktop
  - Visual studio.
  - CodeBlocks
- Online compiler:
  - https://www.onlinegdb.com/online_c++_compiler
  - https://onecompiler.com/cpp
  - https://riju.codes/cpp
- Mobile application:
  - Cxxdroid - C++ compiler IDE
  - CppDroid - C/C++ IDE

Processing a C++ Program

```
#include <iostream>
using namespace std;
int main()
{
    cout << "My first C++ program." << endl;




  return 0;
}
```



Sample Run:

```
My first C++ program.
```

Program Description

- In a C++ program, statements that begin with the symbol # are called **preprocessor directives**.

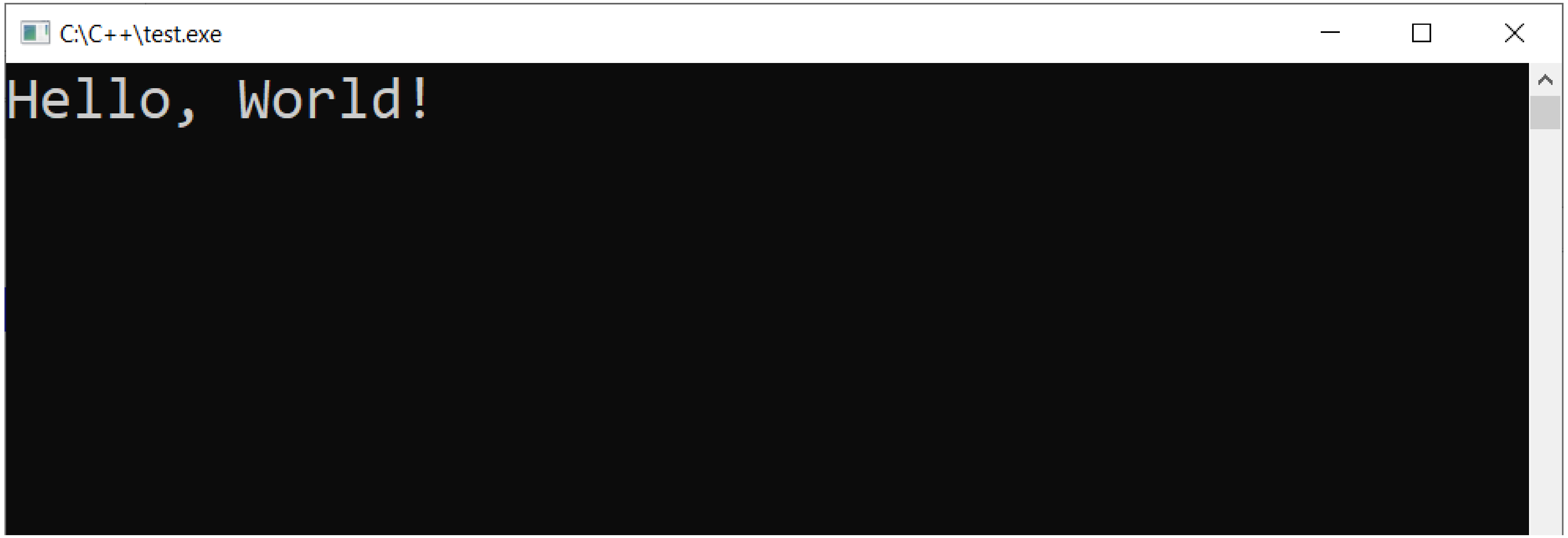
  # is a directive to use the header file "iostream" which contains a description of "**cout**" & "**endl**" function.

- "using namespace" statement allows using **cin**, **cout**, and **endl** without using the prefix **std::**

- "main" is a **function** (it is necessary)

- "endl" means newline.

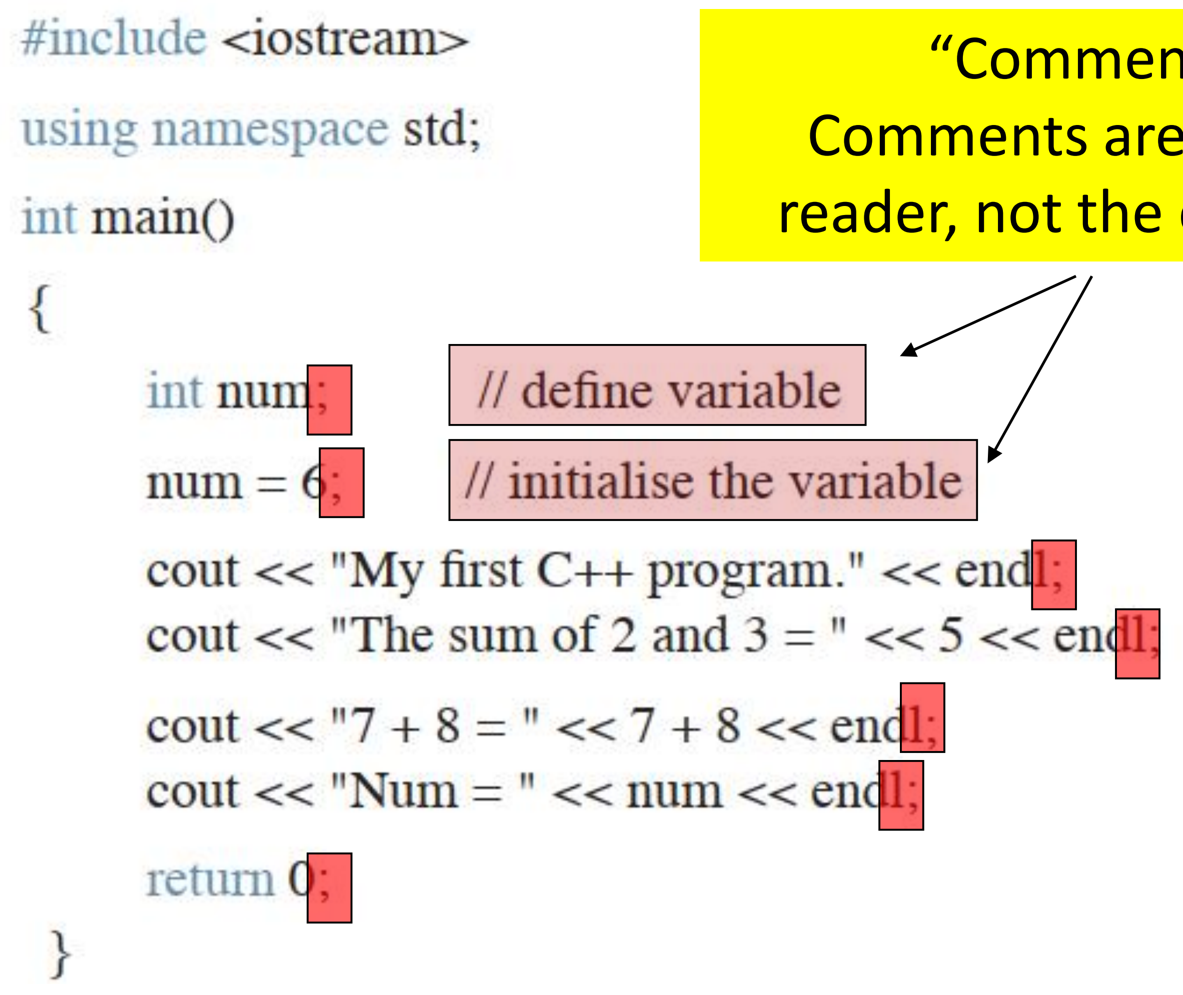
Output


C:\C++\test.exe

```
Hello, World!
```

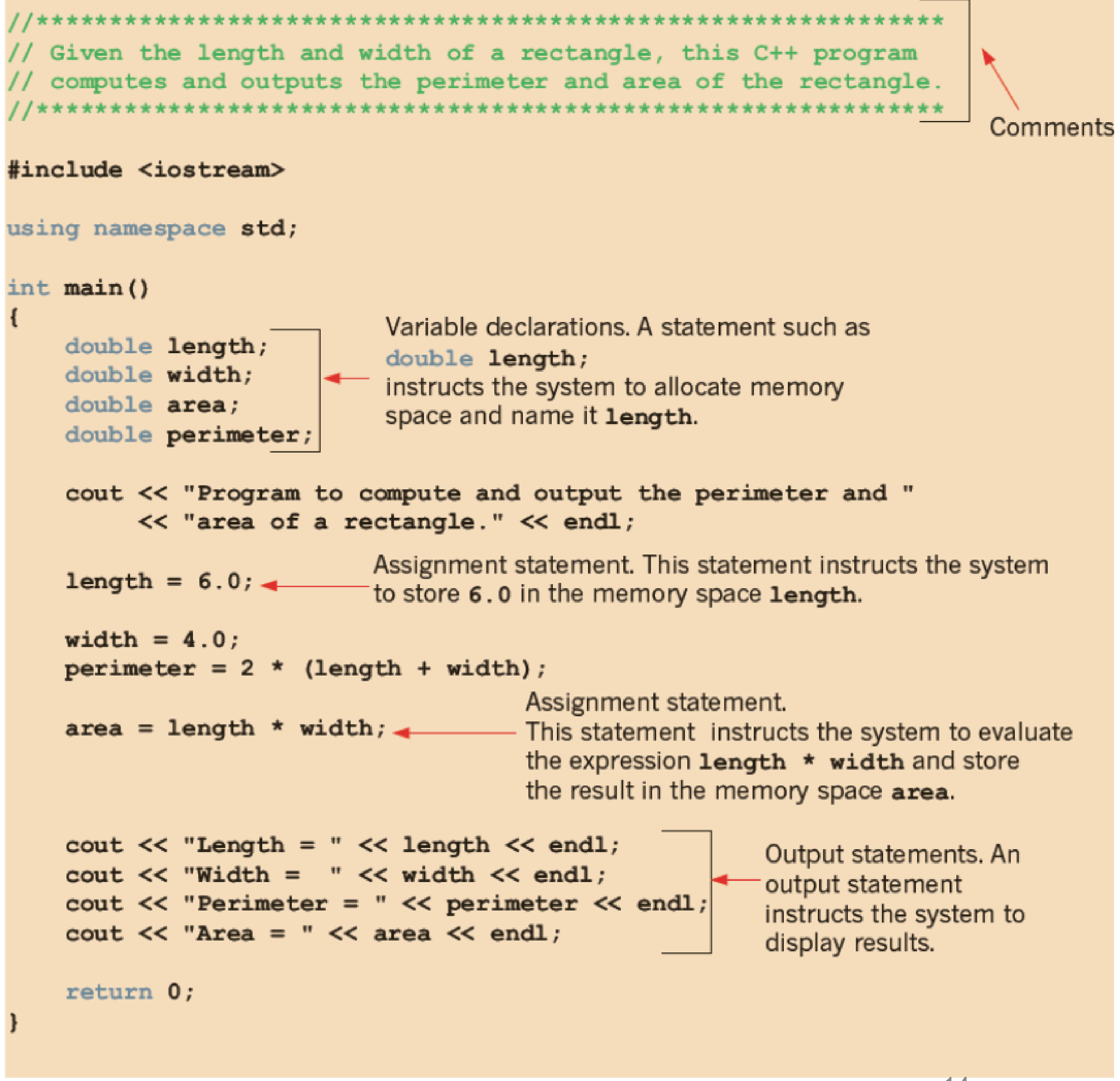
Sample code

“Comment”:
Comments are for the reader, not the compiler



```
#include <iostream>
using namespace std;
int main()
{
        int num;          // define variable
        num = 6;          // initialise the variable
        cout << "My first C++ program." << endl;
        cout << "The sum of 2 and 3 = " << 5 << endl;
        cout << "7 + 8 = " << 7 + 8 << endl;
        cout << "Num = " << num << endl;
        return 0;
 }
```

Sample code



```cpp
//********************************************************************
// Given the length and width of a rectangle, this C++ program
// computes and outputs the perimeter and area of the rectangle.
//********************************************************************

#include <iostream>

using namespace std;

int main()
{
    double length;
    double width;
    double area;
    double perimeter;

    cout << "Program to compute and output the perimeter and "
         << "area of a rectangle." << endl;

    length = 6.0;

    width = 4.0;
    perimeter = 2 * (length + width);

    area = length * width;

    cout << "Length = " << length << endl;
    cout << "Width =  " << width << endl;
    cout << "Perimeter = " << perimeter << endl;
    cout << "Area = " << area << endl;

    return 0;
}
```

Comments

Variable declarations. A statement such as `double length;` instructs the system to allocate memory space and name it **`length`**.

Assignment statement. This statement instructs the system to store **`6.0`** in the memory space **`length`**.

Assignment statement. This statement instructs the system to evaluate the expression **`length * width`** and store the result in the memory space **`area`**.

Output statements. An output statement instructs the system to display results.

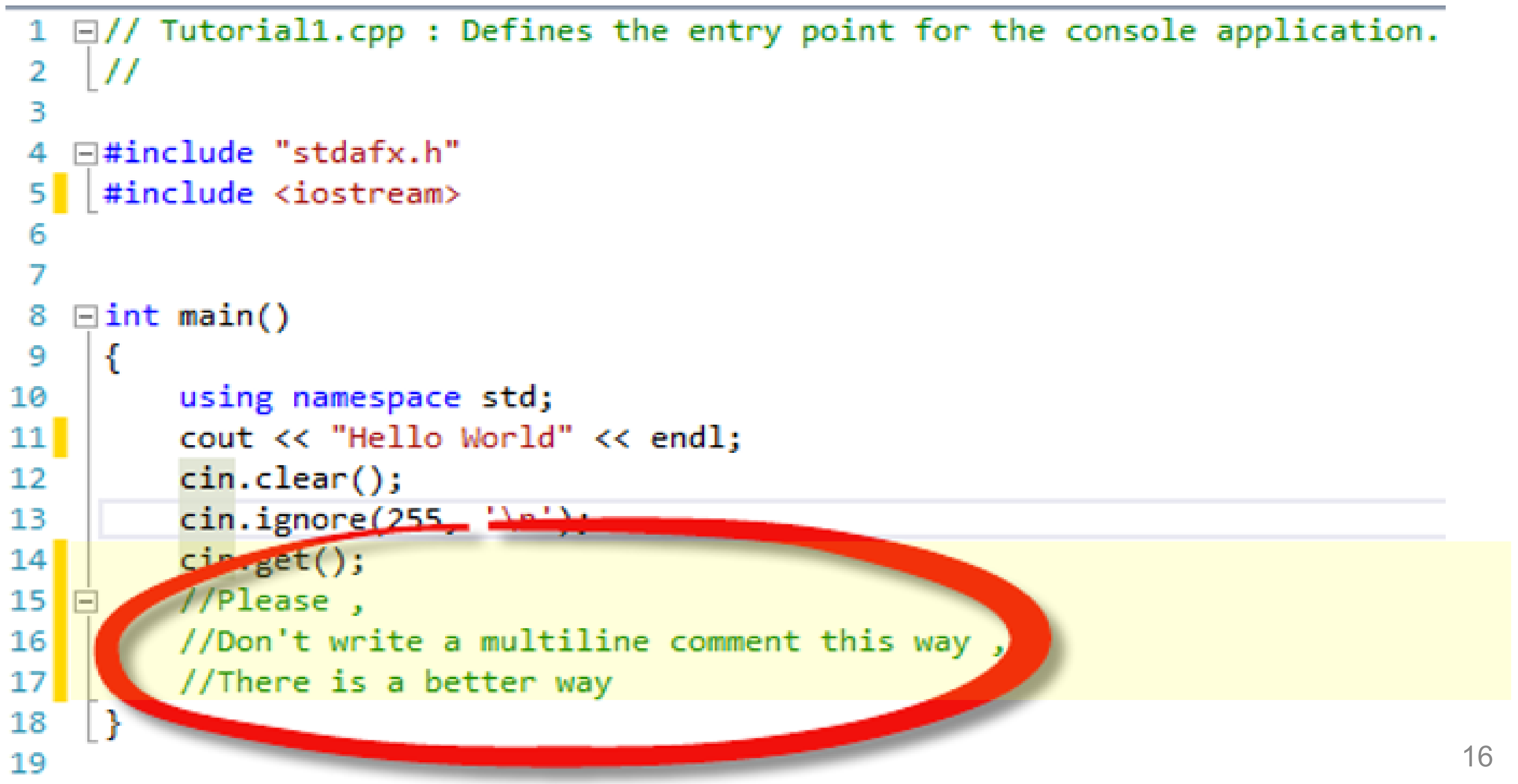
FIGURE 2-1 Various parts of a C++ program

Comment (Single-line)

```
// This is a C++ program. It prints the sentence:
// Welcome to C++ Programming.

#include <iostream>
using namespace std;
int main()
{
  // testing the program.
  cout << "My first C++ program." << endl;
  return 0;
}
```

Comment (Multi-line)

```
/*
   You can include comments that can
   occupy several lines.
*/
```



```
// Tutorial1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include <iostream>


int main()
{
    using namespace std;
    cout << "Hello World" << endl;
    cin.clear();
    cin.ignore(255, '\n');
    cin.get();
    //Please ,
    //Don't write a multiline comment this way ,
    //There is a better way
}

```

Braces, brackets, and parentheses

{ } Braces ("curly braces")
[ ] Brackets ("square brackets").
( ) Parentheses.
“ “ Quotation marks.
, Comma
: Colon

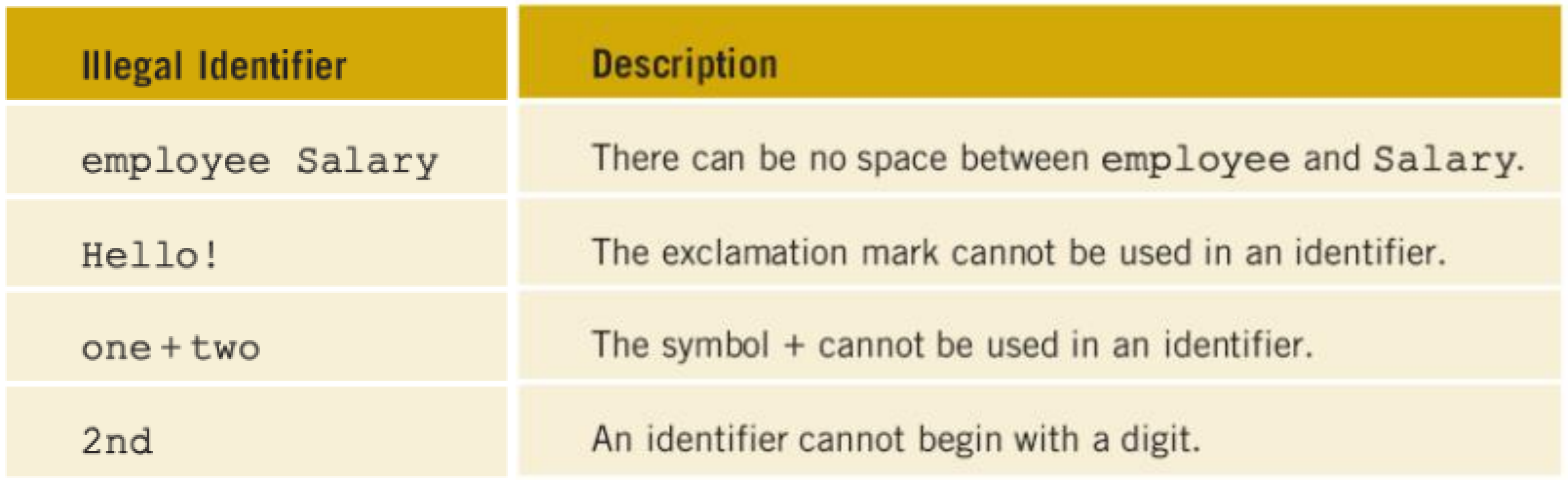
; Simi-colon
! Exclamation mark
_ Underscore
<, > Angle brackets
<< Insertion operator.
>> Extraction operator.

Identifiers

- The C++ identifier is a name used to identify a **variable**, **function**, **class**, **module**, or any other user-defined item. (It's better to be meaningful names)
- Consist of letters, digits, and the underscore character (_).
- Must begin with a letter or underscore.
- C++ is **case sensitive**
  - `NUMBER` is not the same as `number`

Identifiers

TABLE 2-1 Examples of Illegal Identifiers



Illegal Identifier	Description
`employee Salary`	There can be no space between `employee` and `Salary`.
`Hello!`	The exclamation mark cannot be used in an identifier.
`one+two`	The symbol + cannot be used in an identifier.
`2nd`	An identifier cannot begin with a digit.

Simple data types (page 38-43)

1. **Integral**: deals with integers, or numbers without a decimal part includes:

    char
    short
    Int
    long
    bool
    unsigned char
    unsigned short
    unsigned int
    unsigned long

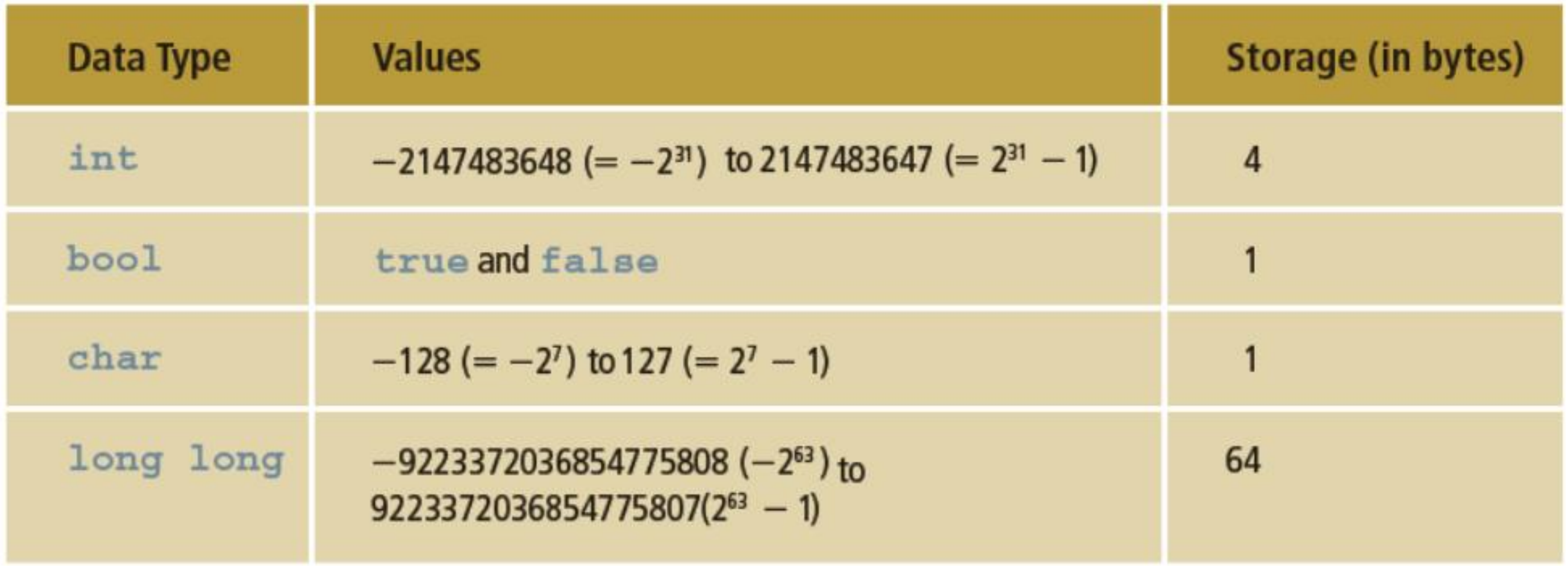
2. **Floating-point**: deals with decimal numbers includes:

    float
    double
    long double

3. **Enumeration**: user-defined data type

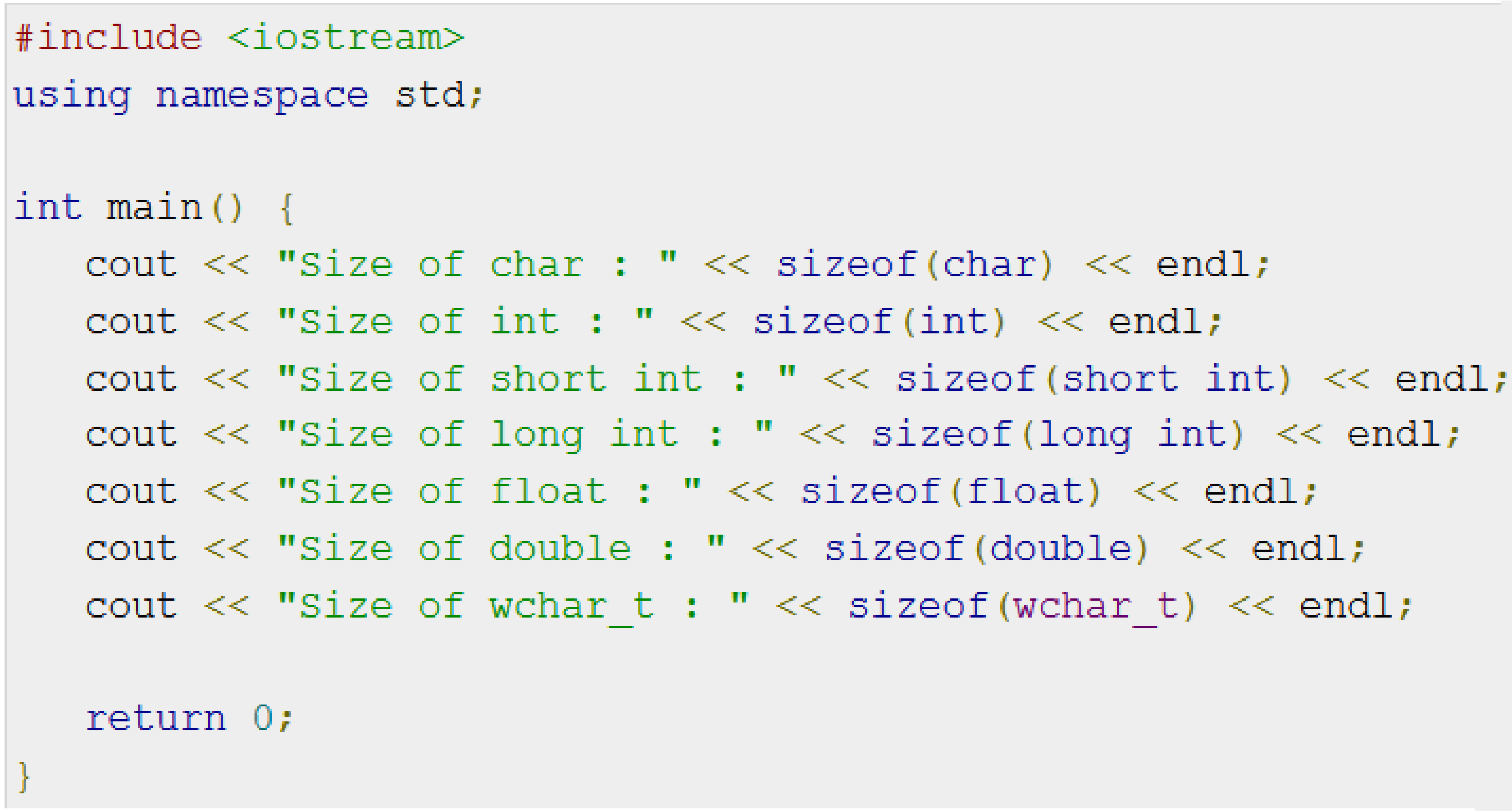
Simple data types

TABLE 2-2 Values and Memory Allocation for Simple Data Types

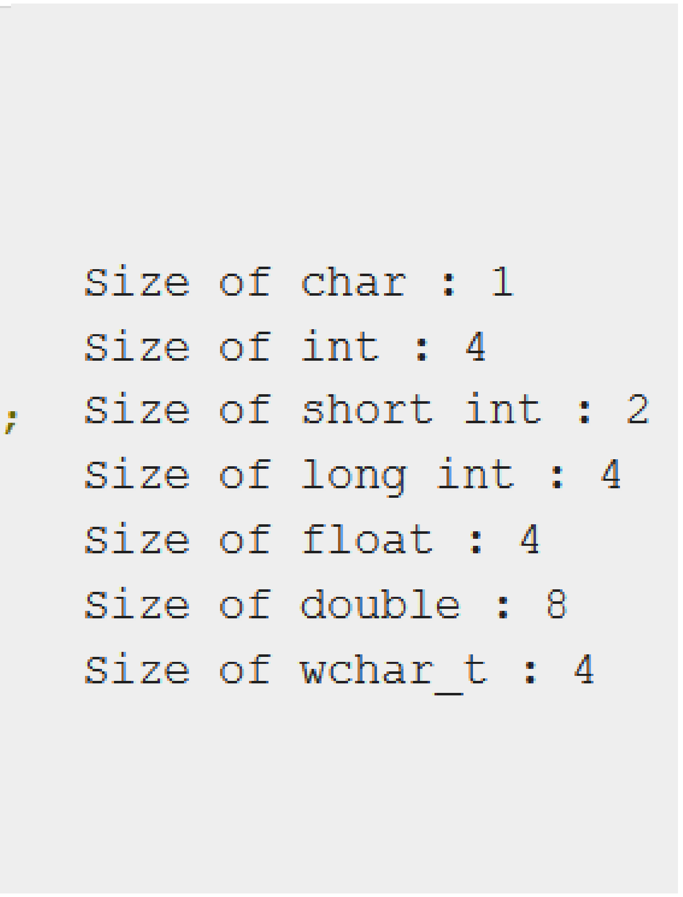


Data Type	Values	Storage (in bytes)
`int`	-2147483648 ($= -2^{31}$) to 2147483647 ($= 2^{31} - 1$)	4
`bool`	`true` and `false`	1
`char`	-128 ($= -2^{7}$) to 127 ($= 2^{7} - 1$)	1
`long long`	-9223372036854775808 (-2^{63}) to 9223372036854775807($2^{63} - 1$)	64

Size of simple data types



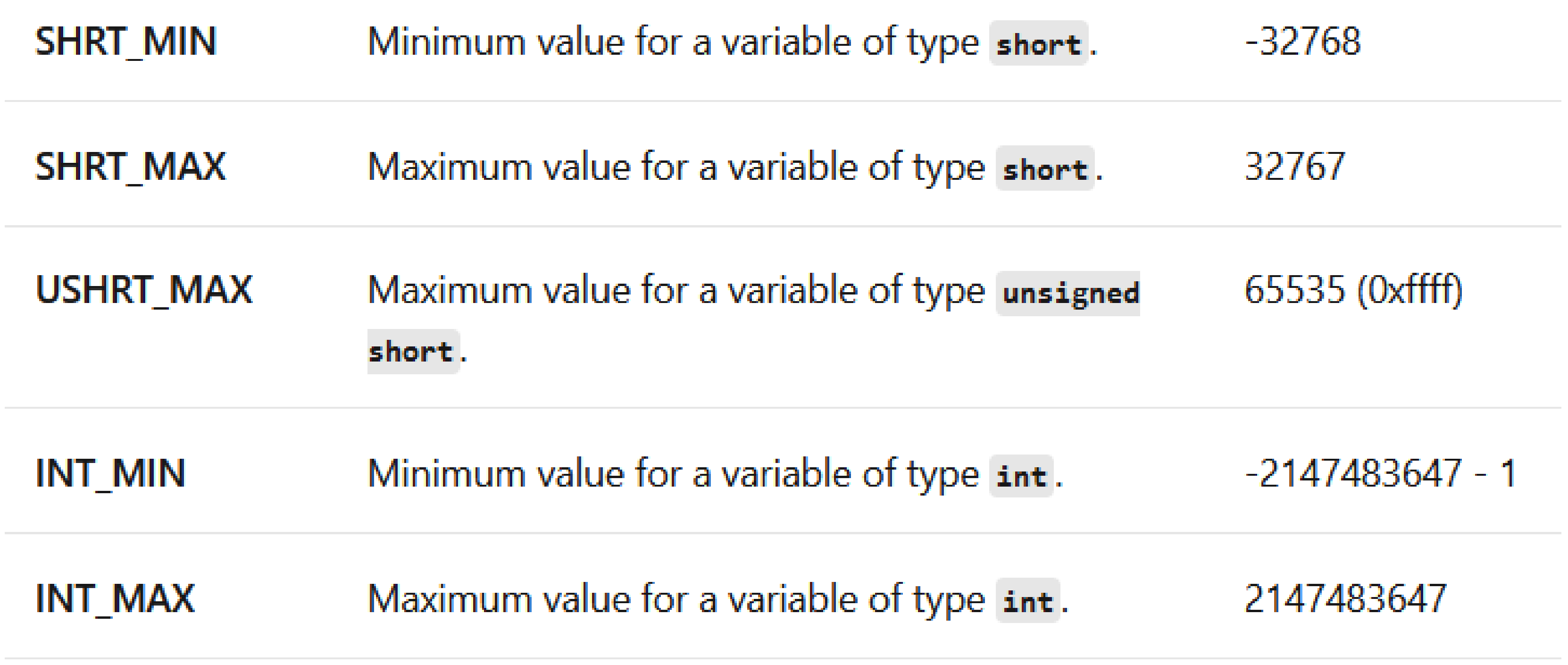
```
#include <iostream>
using namespace std;

int main() {
   cout << "Size of char : " << sizeof(char) << endl;
   cout << "Size of int : " << sizeof(int) << endl;
   cout << "Size of short int : " << sizeof(short int) << endl;
   cout << "Size of long int : " << sizeof(long int) << endl;
   cout << "Size of float : " << sizeof(float) << endl;
   cout << "Size of double : " << sizeof(double) << endl;
   cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

   return 0;
}
```



```
Size of char : 1
Size of int : 4
Size of short int : 2
Size of long int : 4
Size of float : 4
Size of double : 8
Size of wchar_t : 4
```

The limits for integer types in C and C++

https://docs.microsoft.com/en-us/cpp/c-language/cpp-integer-limits?view=msvc-160 (optional reference)



SHRT_MIN	Minimum value for a variable of type `short`.	-32768
SHRT_MAX	Maximum value for a variable of type `short`.	32767
USHRT_MAX	Maximum value for a variable of type `unsigned short`.	65535 (0xffff)
INT_MIN	Minimum value for a variable of type `int`.	-2147483647 - 1
INT_MAX	Maximum value for a variable of type `int`.	2147483647

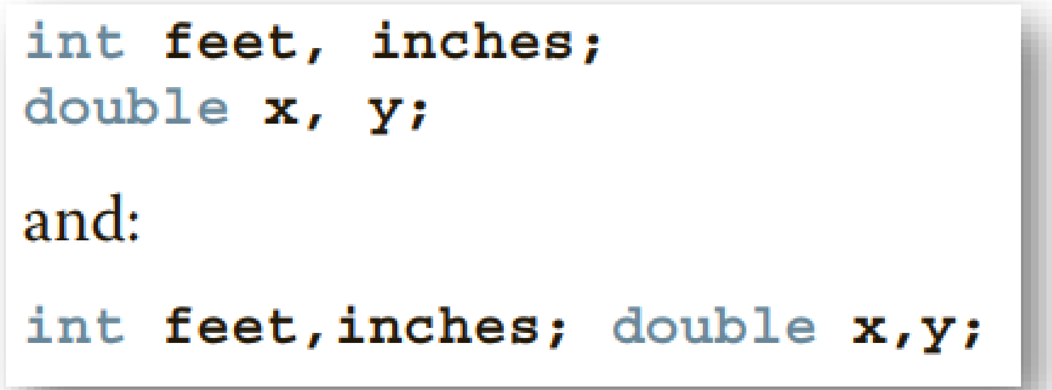
Define a variable

- Define and initialize

```
int var_1 = 5;
```

- Define then initialize

```
int var_1;
var_1 = 5;
```



```
int feet, inches;
double x, y;
```

and:

```
int feet,inches; double x,y;
```

Arithmetic Operators

C++ arithmetic operators:

+ addition

- subtraction

* multiplication

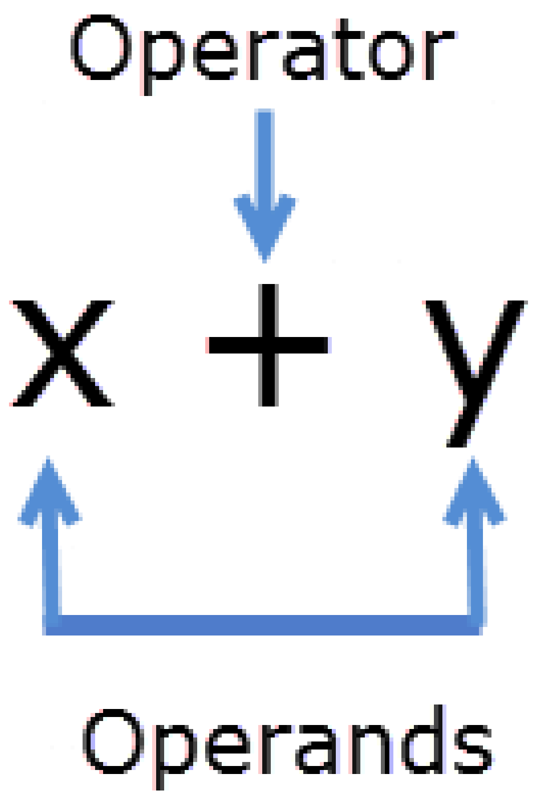
/ division

% modulus operator

++, -- Increment and Decrement Operators (reading tasks)

- Prefix Increment and Decrement Operators
- Postfix Increment and Decrement Operators.

Arithmetic Expression



Arithmetic Operators

- Operators can be **unary** or **binary**.

- Examples of **<u>Unary</u>** operators:
  - unary minus (-)
  - Increment (++)
  - Decrement (- -)
  - NOT (!)

I/O Streams and Standard I/O Devices

- **Stream**: sequence of characters from source to destination
- **I/O**: sequence of bytes (stream of bytes) from source to destination
  - Bytes are usually characters, unless program requires other types of information
- **Input stream**: sequence of characters from an **input device** to the **computer**
- **Output stream**: sequence of characters from the **computer** to an **output device**

I/O Streams and Standard I/O Devices (continued)

- Use `iostream` header file to **extract** data from keyboard and **send** output to the screen
  - Contains definitions of two data types:
    - `istream` - input stream
    - `ostream` - output stream
  - Has two variables:
    - `cin` - stands for common input
    - `cout` - stands for common output

cin and the Extraction Operator >>

- The syntax of an input statement using `cin` and the extraction operator `>>` is:

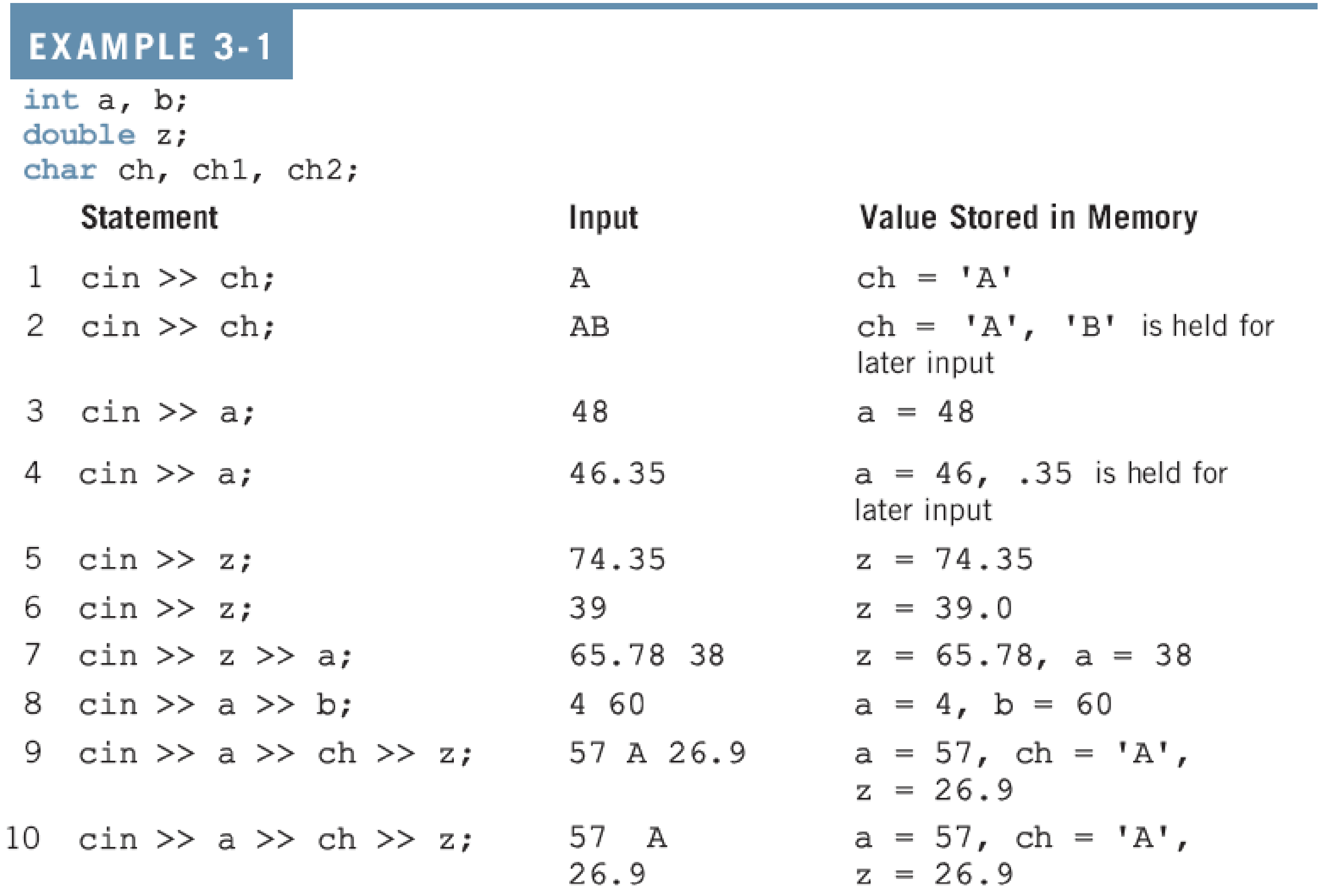
```
cin >> variable >> variable...;
```

- The extraction operator `>>` is “binary operator”
  - Left-side **operand** is an input stream variable
    - Example: `cin`
  - Right-side **operand** is a variable

cin and the Extraction Operator >> (continued)

- No difference between a single `cin` with multiple variables and multiple `cin` statements with one variable
- When scanning, `>>` skips all whitespace
  - Blanks and certain nonprintable characters
- `>>` distinguishes between character `2` and number `2` by the right-side operand of `>>`
  - If type `char` or `int` (or `double`), the 2 is treated as a character or as a number `2`

`cin` and the Extraction Operator >> (continued)



EXAMPLE 3-1

```
int a, b;
double z;
char ch, ch1, ch2;
```

	Statement	Input	Value Stored in Memory
1	`cin >> ch;`	`A`	`ch = 'A'`
2	`cin >> ch;`	`AB`	`ch = 'A', 'B'` is held for later input
3	`cin >> a;`	`48`	`a = 48`
4	`cin >> a;`	`46.35`	`a = 46, .35` is held for later input
5	`cin >> z;`	`74.35`	`z = 74.35`
6	`cin >> z;`	`39`	`z = 39.0`
7	`cin >> z >> a;`	`65.78 38`	`z = 65.78, a = 38`
8	`cin >> a >> b;`	`4 60`	`a = 4, b = 60`
9	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`

10	`cin >> a >> ch >> z;`	`57  A`
		`26.9`
			`a = 57, ch = 'A', z = 26.9`

EXAMPLE 3-1

```
int a, b;
double z;
char ch, ch1, ch2;
```

11	`cin >> a >> ch >> z;`	57
		A
		26.9
			`a = 57, ch = 'A',`
			`z = 26.9`

12	`cin >> a >> ch >> z;`	57A26.9	`a = 57, ch = 'A',`
			`z = 26.9`

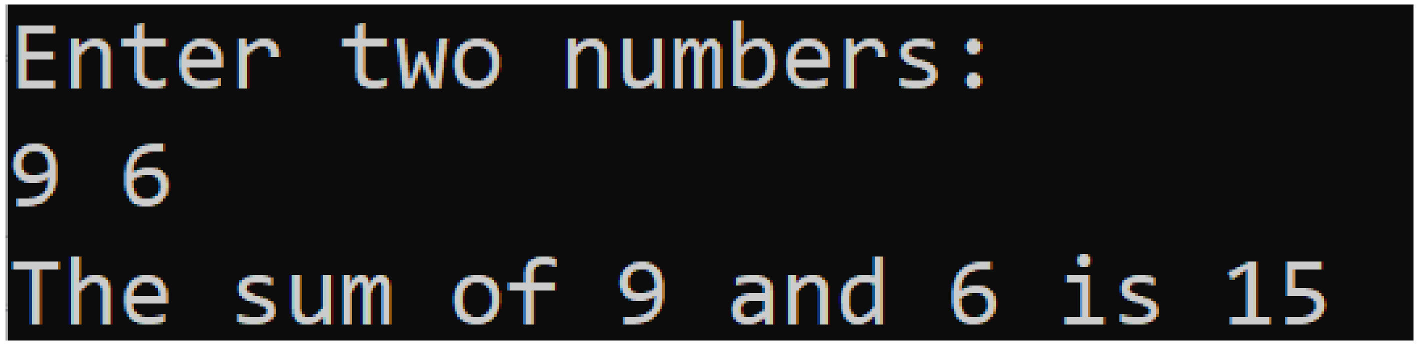
13	`cin >> z >> ch >> a;`	36.78B34	`z = 36.78, ch = 'B',`
			`a = 34`

14	`cin >> z >> ch >> a;`	36.78
		B34
			`z = 36.78, ch = 'B',`
			`a = 34`

15	`cin >> a >> b >> z;`	11 34	`a = 11, b = 34,` computer waits for the next number
16	`cin >> a >> z;`	46 32.4 68	`a = 46, z = 32.4, 68` is held for later input
17	`cin >> a >> z;`	78.49	`a = 78, z = 0.49`
18	`cin >> ch >> a;`	256	`ch = '2', a = 56`
19	`cin >> a >> ch;`	256	`a = 256,` computer waits for the input value for `ch`
20	`cin >> ch1 >> ch2;`	A B	`ch1 = 'A', ch2 = 'B'`

Exercise_1

- Ask the user to input two numbers, then print the sum of the two numbers.



```
Enter two numbers:
9 6
The sum of 9 and 6 is 15
```

Solution

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

int main()
{
   int a,b;
   cout<<"Enter two numbers: \n";
   cin>>a>>b;
   cout<<"The sum of "<<a<<" and "<<b<<" is "<<a+b;
   return 0;
}
```