

# Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University

INTRODUCTION TO  
PROGRAMMING  
WITH

A large, stylized blue logo for C++ programming. It features a large 'C' followed by two '+' signs.

Third Edition

Contents are based on book by Y. Daniel Liang

# Elementary programming

- This chapter focuses on learning elementary programming techniques to solve problems.
- Writing a program involves designing algorithms and translating them into programming instructions, or code.
- An algorithm describes how a problem is solved by listing the actions that must be taken and the order of their execution.
- Algorithms can be described in natural languages (human languages) or in pseudocode (natural language mixed with some programming code).

# Pseudocode

- The algorithm for calculating the area of a circle can be described as follows:
  1. Read in the circle's radius.
  2. Compute the area using the following formula:  
$$\text{area} = \text{radius} * \text{radius} * p$$
  3. Display the result.
- Translate an algorithm into a program.


```
int main()
{
    // Step 1: Read in radius
    double radius;
    // Step 2: Compute area
    // Step 3: Display the area
}
```

- In order to store the radius, the program needs to declare a symbol called a **variable**.
- A variable represents a value stored in the computer's memory.

# Prompt

- A string "Enter a radius: " to the console. This is known as a prompt.
- It directs the user to input something. Your program should always tell the user what to enter when expecting input from the keyboard.
- The **cin** object reads a value from the keyboard.
- **cin** (pronounced see-in) stands for console input.
- The >> symbol, referred to as the stream extraction operator, assigns an input to a variable.

```
cout << "Enter a radius: ";  
cin >> radius;
```

Enter a radius: 2.5 

- The **cin** object causes a program to wait until data is entered at the keyboard and the Enter key is pressed. C++ automatically converts the data read from the keyboard to the data type of the variable.

# Declaring variables

- C++ provides simple data types for representing **integers**, **floating-point numbers** (i.e., numbers with a decimal point), **characters**, and **Boolean** types. These types are known as **primitive data types** or **fundamental types**.

```
int main()
{
    // Step 1: Read in radius
    double radius;
    cout << "Enter a radius: ";
    cin >> radius;

    // Step 2: Compute area
    double area = radius * radius * 3.14159;

    // Step 3: Display the area
    cout << "The area is " << area << endl;

    return 0;
}
```

The area is 19.6349

- See List2\_3.cpp

# Declaring variables

- Variables are used to represent data of a certain type.
- To use a variable, you declare it by telling the compiler its name as well as the type of data it can store.
- The variable declaration tells the compiler to allocate appropriate memory space for the variable based on its data type.
- Variable naming rules
  1. Start with a letter or an underscore(\_); it cannot start with a digit.
  2. Cannot be a reserved word. (main, cout, cin,...)
  3. Can be of any length, but your C++ compiler may impose restriction. Use identifiers of 31 characters or fewer to ensure portability.
- Same rules for identifiers, which are the names that identify elements such as variables and functions in a program.

# Assignment Statements and Assignment Expressions

- After a variable is declared, you can assign a value to it by using an assignment statement.
- In C++, the equal sign (=) is used as the assignment operator.

```
variable = expression;
```

- An expression represents a computation involving values, variables, and operators that, taking them together, evaluates to a value

```
int y = 1; // Assign 1 to variable y
double radius = 1.0; // Assign 1.0 to variable radius
int x = 5 * (3 / 2); // Assign the value of the expression to x
x = y + 1; // Assign the addition of y and 1 to x
area = radius * radius * 3.14159; // Compute area
```

- You can use a variable in an expression. A variable can also be used in both sides of the = operator.

```
x = x + 1;
```

# Assignment expression and Named Constants

- An assignment statement is also known as an assignment expression. For example, the following statement is correct:

```
cout << x = 1;
```

which is equivalent to

```
x = 1;  
cout << x;
```

- A named constant is an identifier that represents a permanent value.
- The syntax for declaring a constant:

```
const datatype CONSTANTNAME = value;  
const double PI = 3.14159;
```



# Numeric Types

- Every data type has a range of values. The compiler allocates memory space for each variable or constant according to its data type.
- C++ uses three types for integers: short, int, and long. Each integer type comes in two flavors: signed and unsigned.
- C++ uses three types for floating-point numbers: float, double, and long double.
- The double type is usually twice as big as float. So, the double is known as double precision, while float is single precision.

```
cout << "The size of int: " << sizeof(int) << " bytes" << endl;  
cout << "The size of long: " << sizeof(long) << " bytes" << endl;  
cout << "The size of double: " << sizeof(double) << " bytes" << endl;
```

```
-----  
The size of int: 4 bytes  
The size of long: 4 bytes  
The size of double: 8 bytes
```

- A nu'meric literal is a character-string whose characters are selected from the digits 0 through 9, a sign character (+ or -), and the decimal point.

# Numeric Literals

- An integer literal is a decimal integer number.
- To denote an octal integer literal, use a leading 0 (zero)
- To denote a hexadecimal integer literal, use a leading 0x or 0X (zero x).
- For example, the following code displays the decimal value 65535 for hexadecimal number FFFF and decimal value 8 for octal number 10.

```
cout << 0xFFFF << `` `` << 010;
```

- Floating-point literals can be written in scientific notation in the form of  $a \times 10^b$ .
- A special syntax is used to write scientific notation numbers. For example,  $1.23456 \times 10^2$  is written as 1.23456E2 or 1.23456E+2 and  $1.23456 \times 10^{-2}$  as 1.23456E-2.
- When a number is converted into scientific notation, its decimal point is moved (i.e., floated) to a new position.

# Numeric Operators

Operator	Name	Example	Result
+	Addition	34 + 1	35
-	Subtraction	34.0 - 0.1	33.9
*	Multiplication	300 * 30	9000
/	Division	1.0 / 2.0	0.5
%	Modulus	20 % 3	2

- When both operands of a division are integers, the result of the division is the quotient and the fractional part is truncated. For example,  $5 / 2$  yields 2, not 2.5.
- The % operator, known as modulo or remainder operator, works only with integer operands and yields the remainder after division.
- See List2\_7.cpp

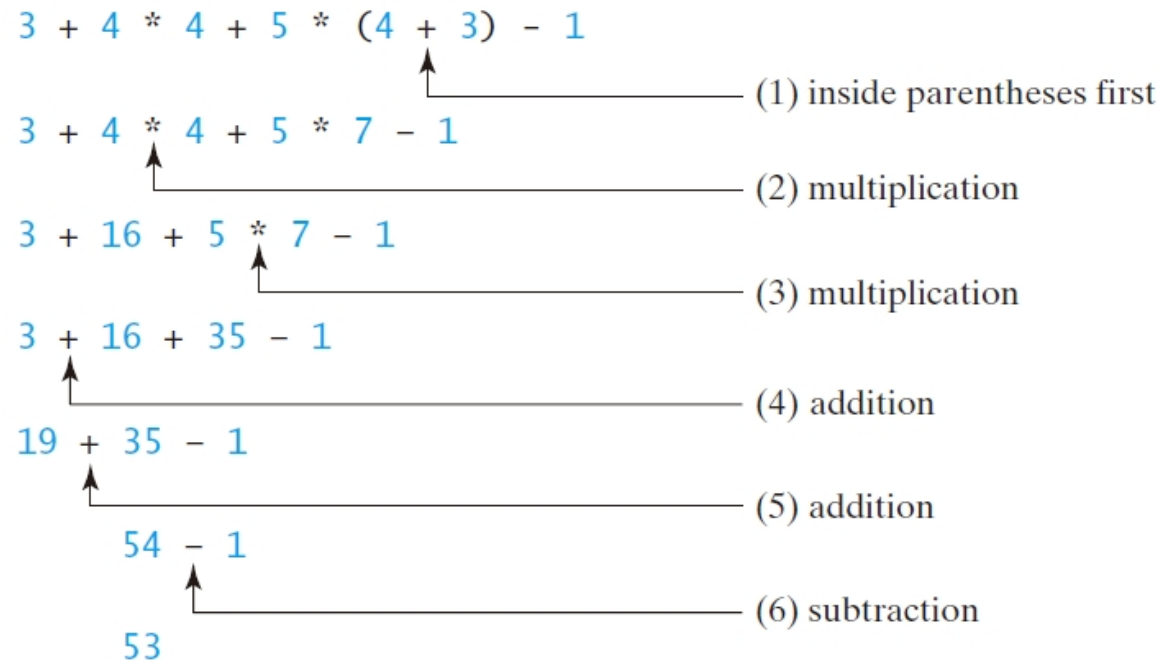
# Exponent Operations

- The `pow(a, b)` function can be used to compute  $a^b$ .
- `pow` is a function defined in the `cmath` library.  
`#include <cmath>`

Homework 2.1: Programming exercise, 2.13.

# Evaluating Expressions and Operator Precedence

- Operators contained within pairs of parentheses are evaluated first. Parentheses can be nested, in which case the expression in the inner parentheses is evaluated first.
- Multiplication, division, and remainder operators are applied next.
- Addition and subtraction operators are applied last.



# Augmented Assignment Operators

- The operators `+`, `-`, `*`, `/`, and `%` can be combined with the assignment operator (`=`) to form augmented operators.
- Often, the current value of a variable is used, modified, and then reassigned back to the same variable.

```
count = count + 1;
```

- C++ allows you to combine assignment and addition operators using an augmented assignment operator.

```
count += 1;
```

Operator	Name	Example	Result
<code>+=</code>	Addition assignment	<code>i += 8</code>	<code>i = i + 8</code>
<code>-=</code>	Subtraction assignment	<code>i -= 8</code>	<code>i = i - 8</code>
<code>*=</code>	Multiplication assignment	<code>i *= 8</code>	<code>i = i * 8</code>
<code>/=</code>	Division assignment	<code>i /= 8</code>	<code>i = i / 8</code>
<code>%=</code>	Modulus assignment	<code>i %= 8</code>	<code>i = i % 8</code>

# Augmented Assignment Operators

- The augmented assignment operator is performed last after the other operators in the expression are evaluated.

`x /= 4 + 5.5 * 1.5; i.e. x = x / (4 + 5.5 * 1.5);`

- There are no spaces in the augmented assignment operators. For example, `+` `=` should be `+=`.

**In-Class Exercise:** Implement 2.21.

# Increment and Decrement Operators

- The increment (++) and decrement (--) operators are for incrementing and decrementing a variable by 1.

```
int i = 3, j = 3;  
i++; // i becomes 4  
j--; // j becomes 2
```

- i++ is pronounced as i plus plus and i-- as i minus minus.
- These operators are known as postfix increment (postincrement) and postfix decrement (postdecrement), because the operators ++ and -- are placed after the variable.
- ++i increments i by 1 and --j decrements j by 1. These operators are known as prefix increment (preincrement) and prefix decrement (predecrement).

```
int i = 3, j = 3;  
++i; // i becomes 4  
--j; // j becomes 2
```



# Increment and Decrement Operators

- `var++`/postincrement/Increment `var` by 1 and use the original `var` value in the statement

```
int i = 10;  
int newNum = 10 * i++;  
cout << "i is " << i  
<< ", newNum is " << newNum;
```

-----  
i is 11, newNum is 100

- `++var`/preincrement/Increment, `var` by 1, but use the new `var` value in the statement

```
int i = 10;  
int newNum = 10 * ++i;  
cout << "i is " << i  
<< ", newNum is " << newNum;
```

-----  
i is 11, newNum is 110

**In-Class Exercise:** 2.23, 2.24.

# Numeric Type Conversions

- When assigning a floating-point value to an integer variable, the fractional part of the floating-point value is truncated (not rounded).

```
int    i = 34.7; // i becomes 34
double f = i;    // f is now 34
double g = 34.3; // g becomes 34.3
int    j = g;    // j is now 34
```

- C++ also allows you to convert a value from one type to another explicitly by using a casting operator.

```
static_cast<type>(value)
```

```
cout << static_cast<int>(1.7);      // 1
cout << static_cast<double>(1) / 2; // 0.5
```

- Casting a variable of a type with a small range to a variable of a type with a larger range is known as widening a type.
- Casting a variable of a type with a large range to a variable of a type with a smaller range is known as narrowing a type.

**In-Class Exercise:** 2.28.

**Homework 2.2:** Programming exercise, 2.1, Read 2.16