INTRODUCTION TO PROGRAMING

Chapter 2

The Basic Elements of a C++ Program

Khoa Công Nghệ Thông Tin
Trường Đại Học Khoa Học Tự Nhiên
ĐHQG-HCM

GV: Thái Hùng Văn

Objectives

In this chapter, you will:

- Be familiar with the basic components of a C++ program
- Explore simple data types
- Discover how to use operators, evaluates arithmetic expressions
- Learn how to input data into memory using input statements
- Examine ways to output results using output statements
- Investigate how to use preprocessor directives and why they are necessary
- Explore how to properly structure a program, including using comments to document a program
- Write some C++ programs, compile, run and **debug** them.

Outline

- The Basics of a C++ Program
- The Basic Data Types
- The Operators
- Input & Output
- Predefined Functions

The Basics of a C++ Program

The Basics of a C++ Program

- A C++ program is a collection of one or more subprograms (functions)
- Function:
  - Collection of statements
  - Statements accomplish a task
- Every C++ program has a function called **main**
- Programming language is a set of rules /syntax, symbols, operators, statements, data types, keywords, identifiers, reserved words,..

The Basics of a C++ Program – Identifiers

- A C++ **identifier** is is a **name** used to **identify** a function, variable, class, module, or any other **user-defined item**.
- **Standard identifier** is a word used to name an object that comes with the compiler *(ex: cin , cout, printf, scanf)*
- Example:

```
#include <iostream>
using namespace std;
#define PI 3.141592654
int main()  {
   float rad, degree;
   cout<<"Enter the value in degree ";   cin>>degree;
   rad = degree * PI /180;
   cout<<degree<<" degree = "<<rad<<" radian";
   return 0;
}
```

The Basics of a C++ Program – Keywords

- **Keywords** are those words whose meaning is already defined by Compiler
- Most keywords are reserved words (that cannot be used as an identifier) and vice versa.
- There are total 32 keywords in C:

  auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.
- There are some more keywords in C++

The Basics of a C++ Program – Symbols

- A **symbol** is a primitive type whose instances have a unique human-readable form. In some **programming** languages, they are called atoms
- **Symbols** can be used as identifiers or operators
- There are many symbols in C++: + - * / % += -= *= /= %= ++ -- & | ^ ~ && || ! < > <= >= = == != ? : . ( ) { } …

The Basics of a C++ Program – Operators

- **Operators** are the foundation of any programming language.
- An **operator** operates the operands. We can define operators as symbols that help us to perform specific mathematical and logical computations on operands.
- C/C++ has many built-in operator types:
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Bitwise Operators
  5. Assignment Operators
  6. Misc Operators

The Basics of a C++ Program – Statements

- A **simple C++ statement** is each of the **individual instructions** of a program.
- A complex statement is a block that more than one simple statement.
- A **compound statement** is a sequence of zero or more statements enclosed within curly braces. It's frequently used in selection and loop statements.

```
#include <iostream>
using namespace std;
int main()   {
  int number;
  cout << "Enter an integer: ";  cin >> number;
  if ( number > 0) {
     cout << "You entered a positive integer " << endl;
   }
   return 0;
}
```

The Basics of a C++ Program – Data types

- A data-type in C /C++ is a set of values and is determined to act on those values.
- C /C++ provides various types of data-types which allow the programmer to select the appropriate type for the variable to set its value.
- C/C++ Data Types are used to Identify the type of:
  - **a variable when it declared.**
  - **the return value of a function.**
  - **a parameter expected by a function**

The Basics of a C++ Program – Comments

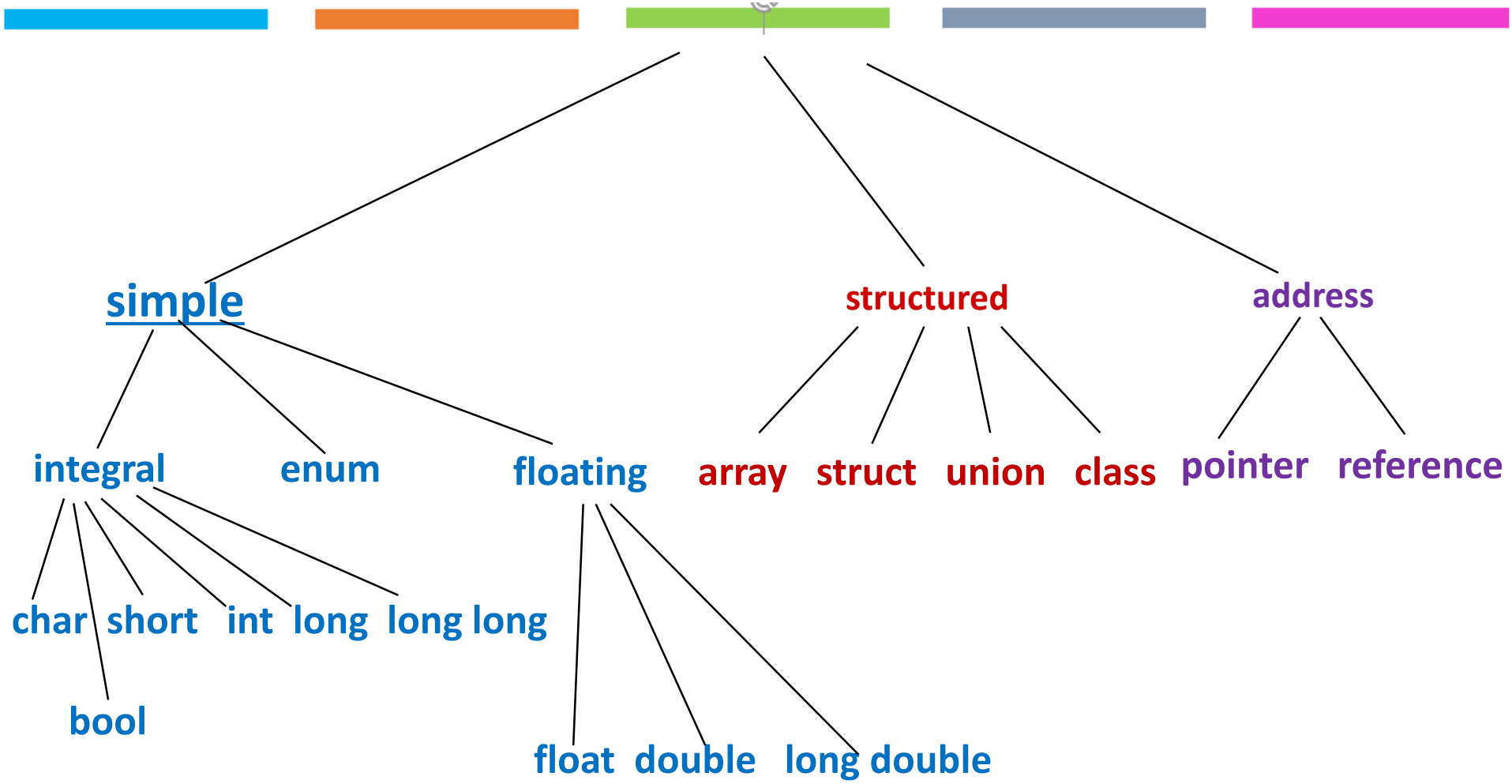
- **Comments** are ignored by the compiler, so they are for the programmer's use only.
- Comments are important part of programming language. You should learn to write comments from the beginning, because it's a good habit for a programmer.
- Comments can be used to explain source code, and to make it more readable. Proper use of commenting can make code maintenance much easier, as well as helping make finding bugs faster.
- Single line comment: Any words in a line after "//" are ignored by the compiler.
- Multi line comment: The /* and */ pair of symbols denotes a C-style multi-line comment. Everything in between them is ignored.
- Tips: In VS, you can use Ctrl_K_C / Ctrl_K_U to make /cancel a comment block.

The Basics of a C++ Program – Rules

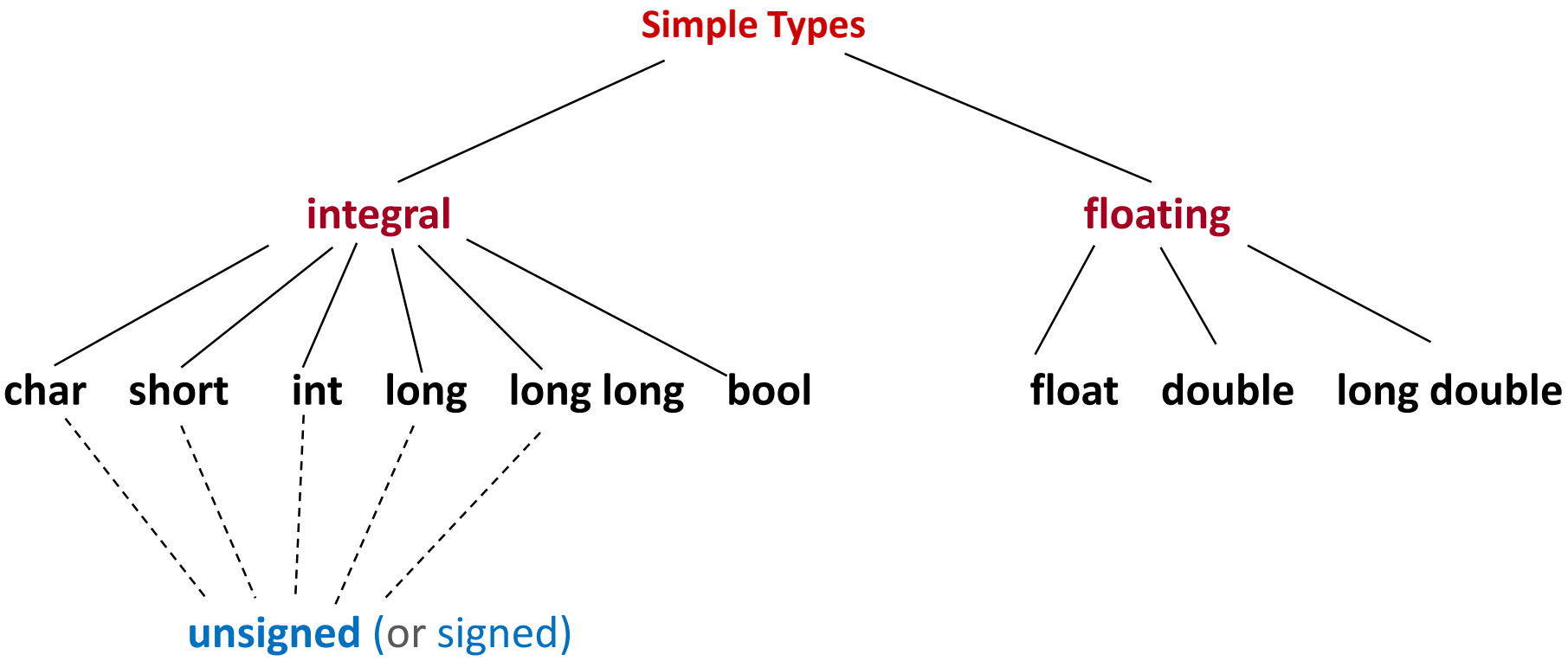
- Rules for writing valid identifiers:
  - An identifier must begin with an alphabet letter (a..z /A..Z), or (_)
  - After that, you can append multiple characters and numbers to create a string that makes sense. (C/C++ is case-sensitive)
  - There is no limit on the number of characters that an identifier may have. But, most compilers only recognize the first 31 characters.
- Rules for writing valid arithmetic expressions.
- Rules for evaluating arithmetic expressions.

The Basic Data Types

C/C++ Data Types



C/C++ Simple Data Types



C/C++ Simple Data Types – Size & Value

Data Type	Value	Storage (in bytes)
bool	true & false	1
char / unsigned char	-128..127 / 0..255	1
short / unsigned short	-32768..32767 / 0..65535	2
long / unsigned long	-2^31 .. 2^31 – 1 / 0 .. 2^32 - 1	4
long long/ unsigned longlong	-2^63 .. 2^63 – 1 / 0 .. 2^64 - 1	8
int / unsigned (int)	$-2^{8N-1}..2^{8N-1}-1$ / 0.. $2^{8N}-1$	N (2 or 4)
float	-3.4E+38 to 3.4E+38	4
double	-1.7E+308 to 1.7E+308	8
long double	= **double** *(on most new compilers)*	8

Note:

An **integer** type (in C /C++) is **signed** by default.
So, **signed char = char, signed int = int,** ..

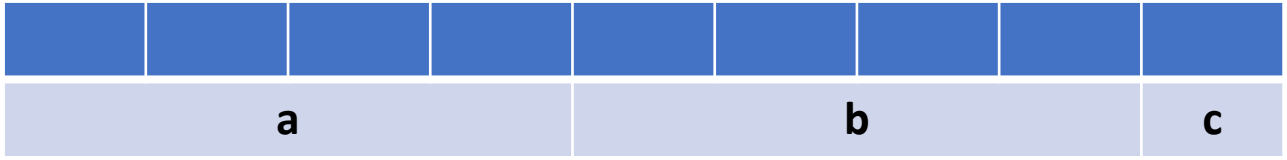
Variables

- A variable is a memory address where data can be stored and changed.
- Declaring a variable means specifying both its name and its data type.
  - A declaration tells the compiler to allocate enough memory to hold a value of this data type, and to associate the identifier with this location.

long **a**;

float **b**;

char **c**;



Variable Declaration

There are several ways to declare variables:

dataType variableName ;

//This declares a variable, declares its data type, and reserves memory for it

dataType variableName = initialValue ;

// This declares a variable and puts an initial value into that memory.

dataType variableNameOne, variableNameTwo ;

// This declares two variables, both of the same data type

dataType variableNameOne = initialValueOne,
variableNameTwo = initialValueTwo ;

// This declares two variables and puts an initial value in each.

If you have several variables of different types, use several declaration statements.

Variable Declaration Examples

```
char a;
int b=2019;
float c, d;
unsigned      e=10,
              f=22;
long g=2019, h=2020;
long i;
double j;
```

The Operators

C /C++ Operators

- An **operator** operates the operands. It is a symbol that tells the compiler to perform specific mathematical or logical computations on operands.
- C /C++ is rich in built-in operators that provide the following types:
  1. Arithmetic Operators
  2. Relational Operators
  3. Logical Operators
  4. Bitwise Operators
  5. Assignment Operators
  6. Misc Operators

Arithmetic Operators

- The following table shows all the arithmetic operators supported by the C /C++ language. Assume variable **A** *(integer type)* holds 3 and variable **B** *(integer type)* holds 7 then −

Operator	Description	Example
+	Adds two operands.	A + B = 10
–	Subtracts second operand from the first.	A – B = -4
*	Multiplies both operands.	A * B = 21
/	Divides numerator by de-numerator.	B / A = 2
%	Modulus Operator, remainder of after an integer division.	B % A = 1
++	Increment operator increases the integer value by one.	A++ = 4
--	Decrement operator decreases the integer value by one.	A-- = 2

Relational Operators

- The table shows the relational operators supported by C/C++. Assume variable **A** holds 3 and variable **B** holds 7 then −

Operator	Description	Example
==	Checks if the values of two operands are equal or not.	(A == B) is false.
!=	Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true.	(A > B) is false.
<	Checks if the value of left operand is less than right operand..	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to right operand.	(A >= B) is false.
<=	Checks if the value of left operand is less than or equal to right operand.	(A <= B) is true.

Logical & Bitwise Operators

- The table shows the relational operators supported by C/C++. Assume variable **A**=10=1010b and variable **B**=12=1100b then −

Operator	Description	Example
&& ||	Called Logical AND /OR operator. If both the operands are true, then the condition becomes true.	(A>0) && (B<9) is false; (A>0) || (B<9) is true.
!	Logical NOT Operator is used to reverse the logical state	!(A >0) is true.
& |	Binary AND / OR Operator copies a bit to the result if it exists in both operands.	(A & B) = 8 = 0000 1000b (A | B) = 14=0000 1110b
^	Binary XOR copies the bit if it is set in only one operand	A^B = 6 = 0000 0110b
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(00001010b) = 11110101b
<< >>	Binary Left /Right Shift. The left operands is moved left /right by the number of bits in the right operand.	A << 2 = 101000b A >> 1 = 101b

Assignment Operators

- The following table lists the assignment operators –

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand	C = A + B will assign the value of A + B to C
+= -=	Add /Subtract AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand.	C -= A is equivalent to C = C - A
*= /= %=	Multiply /Divide /Modulus AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand.	C /= A is equivalent to C = C / A
<<= >>=	Left /Right shift AND assignment operator.	C <<= 2 is same as C = C << 2
&= |= ^=	Bitwise AND assignment operator. Bitwise inclusive OR and assignment operator Bitwise exclusive OR and assignment operator	C &= 2 is same as C=C&2

Misc Operators

- Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :**

Operator	Description	Example
sizeof()	Returns the size of a datatyp/ variable.	sizeof(short) = 2
&	Returns the address of a variable.	**&a**; returns the actual address of **a**
*	Pointer to a variable.	*a;
? :	Conditional Expression. Syntax: **<Condition> ? X : Y**	If Condition is true ? then value X : otherwise value Y
etc..		

Operator Precedence

- Besides the operators discussed above, there are a few other important operators including **sizeof** and **? :**

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right

Expression

- An expression is a valid arrangement of variables, constants, and operators.
- Each expression can be evaluated to compute a value of a given type
- In C++, an expression can be:
  - A variable or a constant
  - An operation
  - Function call
- Example:

Operators - Examples

```
x = 10; y = x++;    // x = 11 , y = 10
x = 10; y = ++x;    // x = 11 , y = 11
x = 10; y = x++ + ++x;      // x = 12 , y = 22
x = 1 / 2; y = 1.0/2;       // x = 0 , y = 0.5
x = (1 > 2) ? 3456 : 7890; // x = 7890
1 < 2 ? y = 3 : y = 4;      // y = 3
```

Input and Output

C /C++ Operators

- C++ treats input and output as a **stream** of characters.
- **stream** : sequence of characters (printable or nonprintable)
- The functions to allow standard I/O are in **iostream** header file or **iostream.h**.
- Thus, we start every program with

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

- **include:** directive copies that file into your program
- **namespace:** a collection of names and their definitions. Allows different namespaces to use the same names without confusion

Output

- Build-in identify **cout** is predefined to denote an output stream that goes to the standard output device (display screen).
- The insertion operator << called "put to" takes 2 operands.
- The left operand is a stream expression, such as **cout**. The right operand is an **expression** of simple type or a **string constant** */unsetf, dec/hex/oct,…*
- Syntax:

cout << expression/manipulator [<< expression/ manipulator];

(expression is evaluated and value is printed, manipulator is used to format the output)

- Common manipulators:

endl, setprecision, fixed, showpoint, setw, setfill, left/right

Output

- String constants (in double quotes) are to be printed as is, without the quotes.

  cout<<"Enter the number of candy bars ";

  OUTPUT: Enter the number of candy bars

- **"Enter the number of candy bars "** is called a **prompt**.
- All user inputs must be preceded by a **prompt** to tell the user what is expected.
- You must insert **spaces** inside the quotes if you want them in the output.
- Do not put a string in quotes on multiple lines.

Output

- All expressions are computed and then outputted.

  cout << "The answer is " << 3 * 4 ;

  => OUTPUT: The answer is 12

- The backslash is called the escape character.
- It tells the compiler that the next character is "escaping" it's typical definition and is using its secondary definition.
- Examples:
  - new line: **\n**
  - horizontal tab: **\t**
  - backslash: \\
  - double quote \"

Output

- All expressions are computed and then outputted.

  cout << "The answer is " << 3 * 4 ;

  => OUTPUT: The answer is 12

- The backslash is called the escape character. It tells the compiler that the next character is "escaping" it's typical definition and is using its secondary definition.
  - new line: **\n**
  - horizontal tab: **\t**
  - backslash: \\
  - double quote \"
- **cout<<"\n"** and **cout<<endl** both are used to bring cursor to newline.

Output

- Use the three format statements (magic formula) to format to fixed decimal notation.

  cout.setf(ios::fixed);

  cout.setf(ios::showpoint);

  cout.precision(2);

- **setf** "set flag" means that all real output will be formatted according to the function, until changed by either unsetting the flag or a new **setf** command.
- **ios::** means the functions from the **iostream**

```
#include <iostream>
int main() {
    std::cout << "The number 2019 in octal:   " << std::oct << 2019 << std::endl
              << "The number 42 in decimal: " << std::dec << 42 << std::endl
              << "The number 42 in hex:     " << std::hex << 42 << std::endl;
}
```

Input

- Variable **cin** is predefined to denote an input stream from the standard input device (the keyboard)
- The extraction operator >> called "get from" takes 2 operands. The left operand is a **stream expression**, such as **cin**--the right operand is a variable of simple type.
- Operator >> attempts to extract the next item from the input stream and store its value in the right operand variable.
- Input is not entered until user presses <ENTER> key.
- Allows backspacing to correct.
- Skips whitespaces (space, tabs, etc.)

Input

- Multiple inputs are stored in the order entered:

  cin>>num1>>num2;

  User inputs: **3 4**

  Assigns num1 = 3 and num2 = 4

- Leading blanks for numbers are ignored.
- If the type is double, it will convert integer to double.
- Keeps reading until blank or <ENTER>.
- Remember to **prompt** for inputs

Input

- Data must be loaded into main memory before it can be manipulated
- Storing data in memory is a two-step process:
  - Instruct computer to allocate memory
  - Include statements to put data into memory

```
#include <iostream>
using namespace std;
int main() {
    int firstNumber, secondNumber, sumOfTwoNumbers;

    cout << "Enter two integers: ";
    cin >> firstNumber >> secondNumber;
    // sum of two numbers in stored in variable sumOfTwoNumbers
    sumOfTwoNumbers = firstNumber + secondNumber;
    // Prints sum
    cout << firstNumber << " + " <<  secondNumber << " = " << sumOfTwoNumbers;
}
```

Predefined Functions

What is Function?

- A function is a block of code that performs a specific task.
- There are two types of function:
  - **Standard library (predefined) functions**
  - **User-defined functions**
- The StandardLibrary provides a rich collection of functions for performing common useful operations *(input/output, mathematical calculations, string processing, error checking,...)*
- However, the predefined functions are not enough. Programmers will often have to build and use user-defined functions.

Predefined Function

- The standard library functions are built-in functions in C/C++ programming language. These are already declared and defined in C/C++ libraries.
- Predefined functions are organized into separate libraries:
  - I/O functions are in **`iostream`** header,
  - Math functions are in **`cmath`** header, …
- You need to include appropriate header files to use them.
- Example:

```
#include <iostream>
#include <cmath>
int main() {
    std::cout << "Square root of 2019 is : " << sqrt (2019);
    return 0;
}
```

Using Predefined Function

- To use a predefined function, you need the name of the appropriate header file; and must use the **#include** directive with the header file name in every **.cpp** file
- You also need to know *Function name, Number of parameters required, Type of each parameter, What the function is going to do*
- For #include <filename> , the preprocessor searches it in directories pre-designated by the compiler/IDE. This method is normally **used to include standard library header files.**
- For #include "filename" the preprocessor searches first in the same directory as the file containing the directive, and then follows the search path used for the #include <filename> form. This method is normally **used to include programmer-defined header files.**

Example

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
  double number, squareRoot;
  cout << "Enter a number: ";     cin >> number;
  // sqrt() is a library function to calculate square root
  squareRoot = sqrt(number);
  cout << "Square root of " << number << " = " << squareRoot;
  return 0;
}
```

Reference

- **Thinking in C**, Bruce Eckel, E-book, 2006.
- **Theory and Problems of Fundamentals of Computing with C++**, John R.Hubbard, Schaum's Outlines Series, McGraw-Hill, 1998.