C++ Basics

Contents

- C and C++

- A sample C++ program

- Variables, Expressions, Assignments

- Data types

- Console input/output

- Program style

C and C++

C Programming Language

o High-level general-purpose language developed in **1972** at **AT&T Bell Lab** by **Dennis Ritchie** from two previous programming `BCPL` and `B`

o Originally developed to write the UNIX operating system.

o Hardware independent (portable).

o By late 1970's `C` had evolved to "Traditional `C`"

o The current standard in `C` is `ANSI  C`.

o `C++` is a more advanced version of `C`, incorporating among other things, the object-oriented constructs.

C Programming Language

o Standardization

- Many slight variations of C existed, and were incompatible

- Committee formed to create a "unambiguous, machine-independent" definition

- Standard created in 1989, updated in 1999

- C has become a popular language industry due its power and flexibility

The C Standard Library

o ℂ programs consist of pieces/modules called **functions**

- A programmer can create his **own** functions
  - Advantage: the programmer knows exactly how it works
  - Disadvantage: time consuming
- Programmers will often use the ℂ library functions
  - Use these as building blocks
- Avoid re-inventing the wheel
  - If a pre-made function exists, generally best to use it rather than write your own
  - Library functions carefully written, efficient, and portable

Origins of the C++ Language

- C is somewhere in between the two extremes of a high-level language and a low-level language.
  - strength
  - weakness
- **Bjarne Stroustrup** (AT&T Bell Labs) developed C++ in early 1980s.
  - the 2018 winner of the **Charles Stark Draper Prize for Engineering**, "for conceptualizing and developing the C++ programming language".
- C++: better C (C with Classes).
- Most of C is subset of C++.
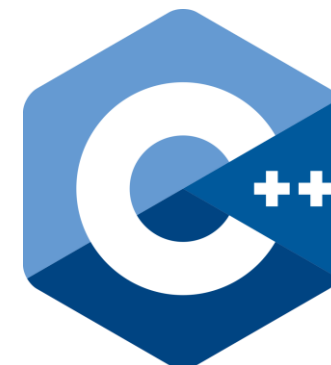
Origins of the C++ Language

o   1985: first commercial implementation released

o   1989: C++ version 2.0

o   1998: C++98 (standardizing version)

o   2003: C++03 (minor update)

o   2011: C++11 (adding numerous features)

o   2014: C++14 (minor update)

o   2017: C++17

o   2020: C++20

Reading

○ Stroustrup'S FAQ:
http://www.stroustrup.com/bs_faq.html#invention

Introduction to C++

C++ Terminology

o **Functions**: procedure-like entities

  - procedures, methods, functions, subprograms

o **Program**: a function called `main`.

A Sample C++ Program

```cpp
//Text-printing program
#include <iostream>
//function main begins program execution
//using namespace std;
int main()
{
    std::cout << "Welcome to C++!\n"; //display message
    return 0; //indicate that program ended successfully

} //end function main
```

Comments

o Comments are for the reader, not the compiler

o Two types:

- Single line

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

- Multiple lines

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

Another Sample C++ Program

```cpp
#include <iostream>

int main()
{
    int numberofLanguages;

    std::cout << "Hello.\n"
              << "Welcome to C++.\n";

    std::cout << "How many programming languages have you used? ";
    std::cin >> numberofLanguages;

    if (numberofLanguages < 1)
        std::cout << "Please read the preface carefully.\n";
    else
        std::cout << "Enjoy the book.\n";

    return 0;
}
```

Layout of a Simple C++ Program

```cpp
1   #include <iostream>
2
3   int main()
4   {
5           Variable_Declarations;
6
7           Statement_1;
8           Statement_2;
9
10          ...
11
12          Statement_n;
13
14          return 0;
15   }
```

Variables, Expressions, Assignment Statements

Identifiers

o **Name** of an item be *declared* or *defined* in a program.

o Consist of letters, digits, and the underscore character (_)

o Must begin with a letter or underscore

  - Avoid starting with the underscore for informally being reserved for system,..

o C++ is **case-sensitive**

  - `NUMBER` is not the same as `number`

o Some predefined identifiers are `main`, `cout` and `cin`

o Unlike reserved words, predefined identifiers may be redefined, but it is not a good idea.

Identifiers

○ Legal identifiers in C++:

- `first`
- `conversion`
- `payRate`

○ Illegal identifiers in C++:

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.

Identifiers

○ Special classes of identifiers: **keywords**, **reserved words**.

○ Examples:

- auto, bool, break, case, const

- ..

○ More keywords in Appendix 1

- Ref: https://en.cppreference.com/w/cpp/keyword

- Ref: https://www.w3schools.in/cplusplus-tutorial/keywords/

Data Types

o Data type: **set of values** together with a **set of operations**

o C++ data types fall into three categories:

- Simple
- Structured
- Pointer

C++'s Data Types

Simple   Structured   Pointers
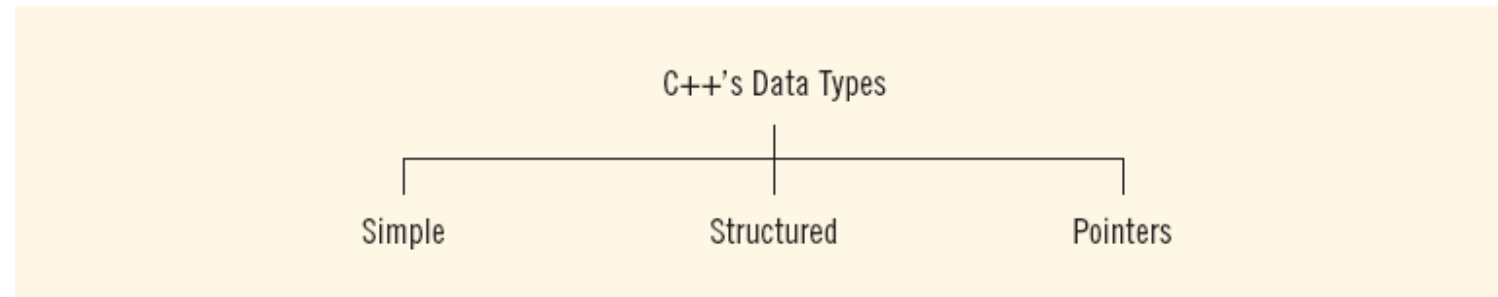
Simple Data Types

o Three categories of simple data

- **Integral**: integers (numbers without a decimal)

- **Floating-point**: decimal numbers

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

Simple Data Types

o Integral data types

- char
- short
- int
- long
- bool
- *unsigned* char
- *unsigned* short
- *unsigned* int
- *unsigned* long

Integral Data Type

- char
- short
- int
- long
- bool
- unsigned char
- unsigned short
- unsigned int
- unsigned long
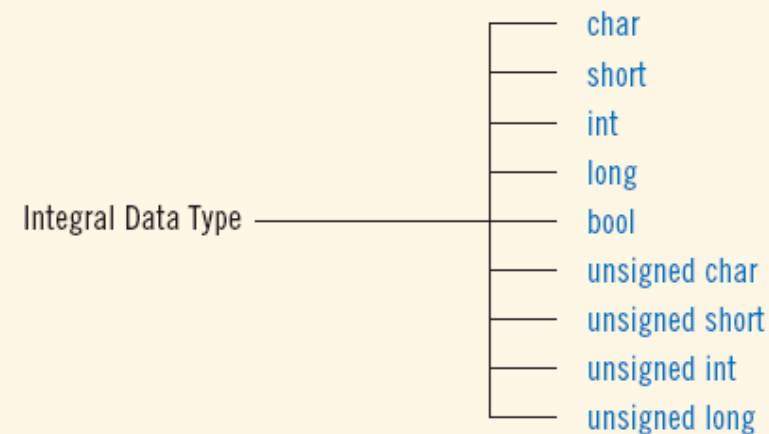
Simple Data Types

o Floating-point data types: `float`, `double`, `long double`

- `float`: *single* precision

- `double`: *double* precision

- C++ uses scientific notation to represent *real* numbers (floating-point notation)

Real Number	C++ Floating-Point Notation
75.924	7.592400E1
0.18	1.800000E-1
0.0000453	4.530000E-5
-1.482	-1.482000E0
7800.0	7.800000E3

○ Different compilers (systems) may allow **different ranges** of values.

TYPE NAME	MEMORY USED	SIZE RANGE	PRECISION
short (also called short int)	2 bytes	−32,768 to 32,767	Not applicable
int	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
long (also called long int)	4 bytes	−2,147,483,648 to 2,147,483,647	Not applicable
float	4 bytes	approximately 10^{-38} to 10^{38}	7 digits
double	8 bytes	approximately 10^{-308} to 10^{308}	15 digits
long double	10 bytes	approximately 10^{-4932} to 10^{4932}	19 digits
char	1 byte	All ASCII characters (Can also be used as an integer type, although we do not recommend doing so.)	Not applicable
bool	1 byte	true, false	Not applicable

Simple Data Types

o C++11 Fixed-Width Integer Types

o Including <**cstdint**>

TYPE NAME	MEMORY USED	SIZE RANGE
int8_t	1 byte	−128 to 127
uint8_t	1 byte	0 to 255
int16_t	2 bytes	−32,768 to 32,767
uint16_t	2 bytes	0 to 65,535
int32_t	4 bytes	−2,147,483,648 to 2,147,483,647
uint32_t	4 bytes	0 to 4,294,967,295
int64_t	8 bytes	−9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
uint64_t	8 bytes	0 to 18,446,744,073,709,551,615
long long	At least 8 bytes	

bool Data Type

o `bool` type

- Two values: `true` and `false`
- Manipulate logical (Boolean) expressions

o `true` and `false` are called logical values

o `bool`, `true`, and `false` are reserved words.

char Data Type

o The smallest integral data type

o Used for **characters**: letters, digits, and special symbols

o Each character is enclosed in single quotes

- `'A', 'a', '0', '*', '+', '$', '&'`

o A blank space is a character and is written `' '`, with a space left between the single quotes.

Arithmetic Operators

o C++ arithmetic operators:

- + addition

- − subtraction

- * multiplication

- / division

- % modulus operator

o +, -, *, and / can be used with integral and floating-point data types

o Operators can be unary or binary

Arithmetic Operators

C++ operation	C++ arithmetic operator	Algebraic expression	C++ expression
Addition	+	$f + 7$	`f + 7`
Subtraction	−	$p - c$	`p - c`
Multiplication	*	bm or $b \cdot m$	`b * m`
Division	/	x / y or $\dfrac{x}{y}$ or $x \div y$	`x / y`
Modulus	%	$r \bmod s$	`r % s`

Arithmetic Operators

o Rules of operator precedence

- Same manner as in algebraic expressions
- Operators in **parentheses** evaluated first
  - Nested/embedded parentheses
    - Operators in innermost pair first

- Multiplication, division, modulus applied next
  - Operators applied from left to right

- Addition, subtraction applied last
  - Operators applied from left to right

Variables

o Location in memory where value can be stored

o **Declare** variables with name and data type **before use.**

- **int** integer1;
- **int** integer2;
- **float** sum;
- **int** numberofBeans;

o Can declare several variables of same type in one declaration (Comma-separated list)

- **int** num1, num2, count;
- **double** oneWeight, totalWeight;

Variable Declarations

o Syntax:

```
Type_Name Variable_Name_1, Variable_Name_2,..;
```

o Examples:

```
int count, numberofDragons, numberofTrolls;
double distance;
```

Assignment Statements

o   To change the value of a variable.

o   Equal sign (=) used as the **assignment operator**.

o   Syntax:

$$\texttt{Variable = Expression;}$$

- Expression can be a variable, a number or a more complicated expression (made up of variables, numbers, operators, function invocations,..)
- Expression is evaluated and its value is assigned to the variable on the left side.

Assignment Statements

○ Examples:

```
distance = rate * time;
count = count + 2;
```

Assignment Statements

○ More Examples:

```
int num1, num2;
double sale;
char first;
string str;

num1 = 4;
num2 = 4 * 5 - 11;
sale = 0.02 * 1000;
first = 'D';
str = "It is a sunny day.";
```

Assignment Statements

○ More Examples:

```
int num1, num2;
int num3;

num1 = 18;
num2 = num1 + 27;
num2 = num1;
num3 = num2 / 5;
num3 = num3 / 4;
```

Assignment Statements

o Assignment statements can be used as expressions.

- DO NOT use (coding errors).

o Examples:

- n = (m = 2);
- n = m = 2;

Named Constant

o **Named constant**: memory location whose content **cannot change** during execution.

o The syntax to declare a named constant is:

```
const dataType identifier = value;
```

o `const` is a reserved word

o Examples:

- `const double CONVERSION = 2.54;`
- `const int NO_OF_STUDENTS = 20;`
- `const char BLANK = ' ';`
- `const double PAY_RATE = 15.75;`

Initializing Variables in Declarations

o Syntax:

```
Type_Name    Variable_Name_1 (Expression_for_Value_1),
Variable_Name_2 (Expression_for_Value_2),...;
```

o Examples:

```
int count(0), limit(10);
double distance(999.99);
```

More Statement Assignments

o Syntax:

```
variable Operator = Expression
variable = variable Operation Expression
```

o Examples:

```
count += 2;
total -= discount;
bonus *= 2;
time /= rushFactor;
change %= 100;
amount *= cnt1 + cnt2;
```

Assignment Compatibility

o General rule: cannot store a value of one type in a variable of another type.

o Errors?

```
int intVar;
intVar = 2.99; //2.99 is of type double.
```

- Depends on *compilers*.

Assignment Compatibility

o `int` **to** `double`:

- `double doubleVar;`
- `doubleVar = 2;`

o integers and Booleans:

- `int` values can be assigned to `bool` type.
  - **nonzero:** `true`; **zero:** `false`

- `bool` values can be assigned to `int` type.
  - `true`: **1**; `false`: **0**.

Type Casting

o **Type cast**: changing a value of one type to a value of another type.

o **Implicit type coercion:** when value of one type is automatically changed to another type

- `double d = 5; //converting 5 to 5.0`
- `int t = 5/2; //value of t?`
- `double res = 5.0/2; //value of res?`

o **Cast operator:** provides explicit type conversion

Type Casting

○ Four kinds of type casting:

- **static_cast**<*Type*> *(Expression)*
- **const_cast**<*Type*> *(Expression)*
- **dynamic_cast**<*Type*> *(Expression)*
- **reinterpret_cast**<*Type*> *(Expression)*

Type Casting

Expression	Evaluates to
`static_cast<int>(7.9)`	`7`
`static_cast<int>(3.3)`	`3`
`static_cast<double>(25)`	`25.0`
`static_cast<double>(5+3)`	`= static_cast<double>(8) = 8.0`

`static_cast<double>(15) / 2`	`= 15.0 / 2`
	(because `static_cast<double>(15) = 15.0`)
	`= 15.0 / 2.0 = 7.5`

`static_cast<double>(15 / 2)`	`= static_cast<double>(7)` (because `15 / 2 = 7`) `= 7.0`

`static_cast<int>(7.8 +`
`static_cast<double>(15) / 2)`
	`= static_cast<int>(7.8 + 7.5)`
	`= static_cast<int>(15.3)`
	`= 15`

`static_cast<int>(7.8 +`
`static_cast<double>(15 / 2))`
	`= static_cast<int>(7.8 + 7.0)`
	`= static_cast<int>(14.8)`
	`= 14`

Type Casting

Example 01:

```cpp
int i1 = 4, i2 = 8;
std::cout << i1 / i2 << std::endl;
std::cout << (double)i1 / i2 << std::endl;
std::cout << i1 / (double)i2 << std::endl;
std::cout << (double)(i1 / i2) << std::endl;
```

Example 02:

```cpp
double d1 = 5.5, d2 = 6.6;
std::cout << (int)d1 / i2 << std::endl;
std::cout << (int)(d1 / i2) << std::endl;
d1 = i1;
std::cout << d1 << std::endl;
i2 = d2;
std::cout << i2 << std::endl;
```

Increment and Decrement Operators

o Increment operator: increment variable by 1

- Pre-increment: `++variable`

- Post-increment: `variable++`

o Decrement operator: decrement variable by 1

- Pre-decrement: `--variable`

- Post-decrement: `variable—`

o What is the difference between the following?

```
x = 5;
y = ++x;
```

```
x = 5;
y = x++;
```

Increment and Decrement Operators

○ `n++`: first returns the value of `n`; then the value of `n` is increased by 1.

○ `++n`: first the value of `n` is increased by 1; the returns the value of increased `n`.

○ `n--`: first returns the value of `n`; then the value of `n` is decreased by 1.

○ `--n`: first the value of `n` is decreased by 1; the returns the value of decreased `n`.

○ Only applied to a single variable. Others (e.g, `(x + y)++, 5++, ..`) are illegal.

Console Input/Output

Console Input/Output

o Using these objects: `std::cin`, `std::cout`, `std::cerr` of `iostream`

o Declaring before use:

`#include <`**`iostream`**`>`

Input

o Data must be loaded into main memory before it can be manipulated

o Storing data in memory is a two-step process:

- Instruct computer to allocate memory

- Include statements to put data into memory

Input Using `std::cin`

○ `std::cin` is used with >> to gather input

○ The stream **extraction operator** is >>

○ Using more than one variable in `std::cin` allows more than one value to be read at a time

○ Examples:
- `std::cin    >> miles;`
- `std::cin    >> numberofLanguages;`
- `std::cin    >> dragrons >> trolls;`
- `std::cin    >> dragrons`
  `            >> trolls;`

Input Using `std::cin`

o `std::in` stops when getting white spaces.

o Try to use function `getline` of `std::cin`.

Input Using `std::cin`

```cpp
#include <iostream>
#include <cstring>
int main()
{
    char name[80];
    std::cout << "Input your name: ";
    std::cin.getline(name, 80);
    std::cout << "Your name is " << name << "\n";
    return 0;
}
```

Output Using `std::cout`

o Any combinations of variables and strings can be output.

o `std::cout` is used with $<<$ to output.

o The stream **insertion operator** is $<<$

o Expression evaluated and its value is printed at the current cursor position on the screen.

Output Using `std::cout`

○ The new line character is '`\n`'. May appear anywhere in the string.

○ `std::endl` causes insertion point to move to beginning of next line.

Output Using `std::cout`

o Commonly used escape sequences:

	Escape Sequence	Description
\n	Newline	Cursor moves to the beginning of the next line
\t	Tab	Cursor moves to the next tab stop
\b	Backspace	Cursor moves one space to the left
\r	Return	Cursor moves to the beginning of the current line (not the next line)
\\	Backslash	Backslash is printed
\'	Single quotation	Single quotation mark is printed
\"	Double quotation	Double quotation mark is printed

Output Using `std::cout`

o Formatting for Numbers:

```
std::cout.setf(std::ios::fixed);
std::cout.setf(std::ios::showpoint);
std::cout.precision(2);
```

```
std::cout.setf(std::ios::fixed);
std::cout.setf(std::ios::showpoint);
std::cout.precision(2);

std::cout << 7.9999 << " " << 10.5 << std::endl;
```

```
8.00 10.50
```

File Input

Reading From a Text File

o using `std::ifstream`.

- Open file for reading: `open`

- Close file after reading: `close`

- Take input (same as `cin`, extractor operator): `>>`

o including `fstream`

Examples

```cpp
#include <iostream>
#include <fstream>

int main()
{
    std::ifstream fIn;

    fIn.open("Data01.txt");
    if (fIn.is_open() == false)
    {
        std::cout << "File does not exist" << std::endl;
        return 1;
    }

    int N, i;
    int A[100];

    fIn >> N;
    for (i = 0; i < N; i++)
        fIn >> A[i];

    for (i = 0; i < N; i++)
        std::cout << A[i] << "\t";
    std::cout << "\n";

    fIn.close();
    std::cout << "Done\n";
    return 0;
}
```

Program Style

Program Style

o Every C++ program has a function `main`

o All C++ statements end with a **semicolon**

o `{` and `}` are not C++ statements

o Follow the syntax rules

o Commas separate items in a list

Program Style

o Identifiers can be self-documenting:

- CENTIMETERS_PER_INCH

o Avoid run-together words :

- annualsale

- Solution:
  - Capitalize the beginning of each new word
    - annualSale
  - Inserting an underscore just before a new word
    - annual_sale

Use of Blanks

o Use one or more blanks to separate numbers when data is input.

o Used to separate reserved words and identifiers from each other and from other symbols.

o Must never appear within a reserved word or identifier.

Prompt Lines

o Prompt lines: executable statements that inform the user what to do.

```
std::cout << "Please enter a number between 1 and 10 and "
    << "press the return key" << std::endl;
std::cin >> num;
```

Documentation

o A well-documented program is easier to understand and modify

o You use comments to document programs

o Comments should appear in a program to:

- Explain the purpose of the program
- Identify who wrote it
- Explain the purpose of particular statements

Syntax Errors

o Errors in syntax are found in compilation

```
int x;          //Line 1
int y           //Line 2: error
double z;       //Line 3


y = w + x;      //Line 4: error
```

Reading

o Programming Convention

Libraries and Namespaces

Libraries

o C++ has a small number of operations

o Many functions and symbols needed to run a C++ program are provided as collection of libraries.

o Every **library** has a name and is referred to by a **header file**.

o **Preprocessor directives** are commands supplied to the preprocessor

o All preprocessor commands begin with **#**

o **No semicolon** at the end of these commands.

Libraries

o C++ includes a number of standard libraries.

- Ref: Appendix 4

o Using preprocessor directive `include`:

- `#include <Library_Name>`
- `#include <Header_File>`

```
#include <iostream>
#include <cmath>
#include <string>
```

Namespaces

o A **namespace** is a collection of name definitions.

o In modern C++, all of the functionality in the C++ standard library is defined inside namespace `std`.

o Using the definitions, insert the `using` directive.

```
using namespace std;
using std::cin;
using std::endl;
```

Namespaces

o `cin` and `cout` are declared in the header file `iostream`, but within `std` namespace.

o To use `cin` and `cout` in a program, use the following two statements:

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

Namespaces

o Best practice:

- Use **explicit** namespace prefixes to access identifiers defined in a namespace.

o Warning:

- Many texts, tutorials, and even some compilers recommend or use a *using directive* at the top of the program. However, used in this way, this is a **bad practice**, and highly discouraged.

Namespaces

○ An error example:

```cpp
#include <iostream> // imports the declaration of std::cout

using namespace std; // makes std::cout accessible as "cout"
int cout() // declares our own "cout" function
{
    return 5;
}
int main()
{
    cout << "Hello, world!"; // Compile error!
    return 0;
}
```

Namespaces

○ A good example:

```cpp
#include <iostream> // imports the declaration of std::cout

//using namespace std; // makes std::cout accessible as "cout"
int cout() // declares our own "cout" function
{
    return 5;
}
int main()
{
    std::cout << "Hello, world!" << std::endl;
    return 0;
}
```

Namespaces

```cpp
#include <iostream>
namespace test{
    void cout()
    {
        std::cout << "inside test::cout" << std::endl;
    }
}
int main()
{
    std::cout << "Hello World" << std::endl;
    test::cout();
    return 0;
}
```

Programming Example

Programming Example

o Write a program that takes as input a given length expressed in feet and inches

- Convert and output the length in centimeters

o **Input**: length in feet and inches

o **Output:** equivalent length in centimeters

o Lengths are given in feet and inches

o Program computes the equivalent length in centimeters

o One foot is equal to `12` inches.

o One inch is equal to `2.54` centimeters.

Questions and Answers