Lecture 1.2

Topics
1. Basic (Data) Types – Introduction
2. Variables – Introduction
3. C++ Constants – Introduction
4. C++ Basic Operations – Introduction
5. C++ Variable Declaration, Assignment, & Usage

---

1. C++ Basic (Data) Types -- Introduction

Recall that data are represented by sequences of **0**s and **1**s. These are called bits; these bits will be arranged into some bit patterns for values. A bit is the smallest data unit in computer.

A byte is a group of 8 bits and it is actually used to represent more meaningful and workable data while developing computer code.

1.1 Data Types

In general, C++ has many data types that can be grouped in three categories as follows,

(i) Simple Data Type

(ii) Structured Data Type

(iii) Pointers

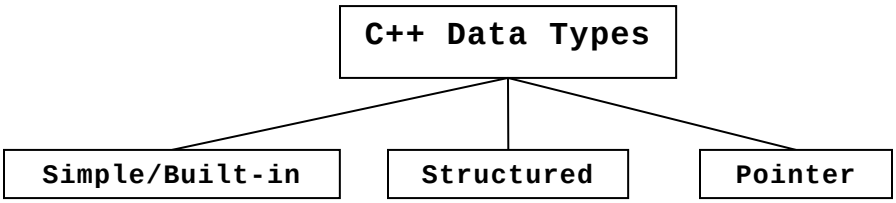

Figure 1 C++'s data types

Simple or built-in data types will be considered in the next several lectures. Structured and pointer data types will be considered in later discussions. Built-in types are depicted in the following pictures.

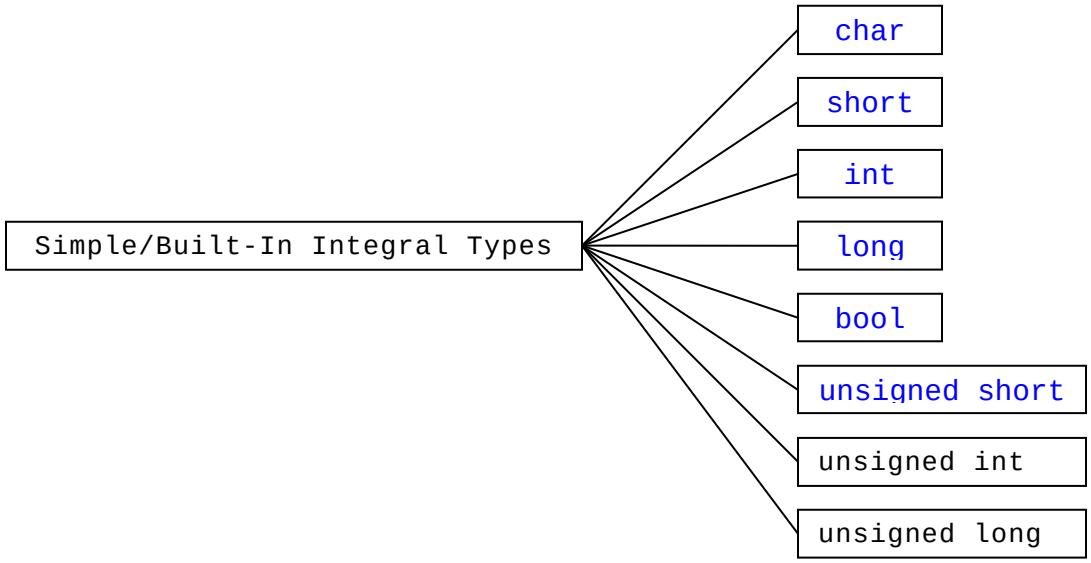

Figure 2 Built-in integral data types

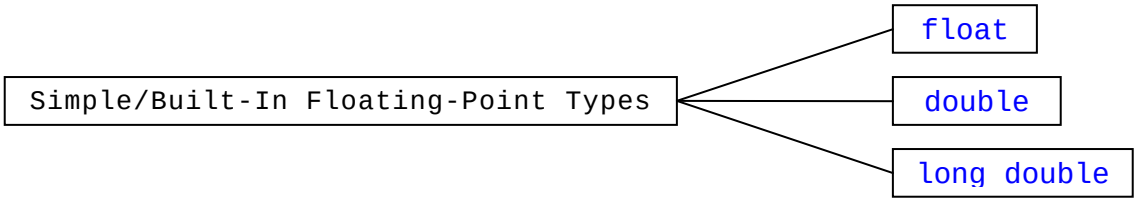

Figure 3 Built-in floating-point data types

1.2 Integer Data Types

Integer data are values that do not have decimal part or portion.

For examples, `5, -5, 1000, -1000`.

The data type will be specified by one of the following keywords: `short, int, long`. For examples,

```
int iCounter = 5;
long lSum;
```

Character may also be considered as integer type and being implied with the integer equivalent value indicated in the ASCII Character Table.

For examples,

```
char cValue1 = 'A'; /* 'A' := 65 */
char cValue2 = 'Z'; /* 'Z' := 90 */
char cValue3 = 'a'; /* 'a' := 97 */
char cValue4 = 'z'; /* 'z' := 122 */
char cValue5 = '0'; /* '0' := 48 */
char cValue6 = '9'; /* '9' := 57 */
```

1.3 Floating-Point Data Types

Floating-point data are numerical values that have the decimal parts.

For examples, `5.5, -5.4, 1000.0, -1000.000`.

The type will be specified by one of the following keywords: `float` and `double`.

For examples,

```
float fVar1 = 5.5;
double dVar2 = -5.4;
```

1.4 Sizes of Data Types

C++ types are used to indicate how a computer must take on a value, translate, and store in memory.

The simplest interpretation of the storage is just to refer to the size of each data type (or the size of the memory block reserved for a value of a given type).

The sizes are given as follows,

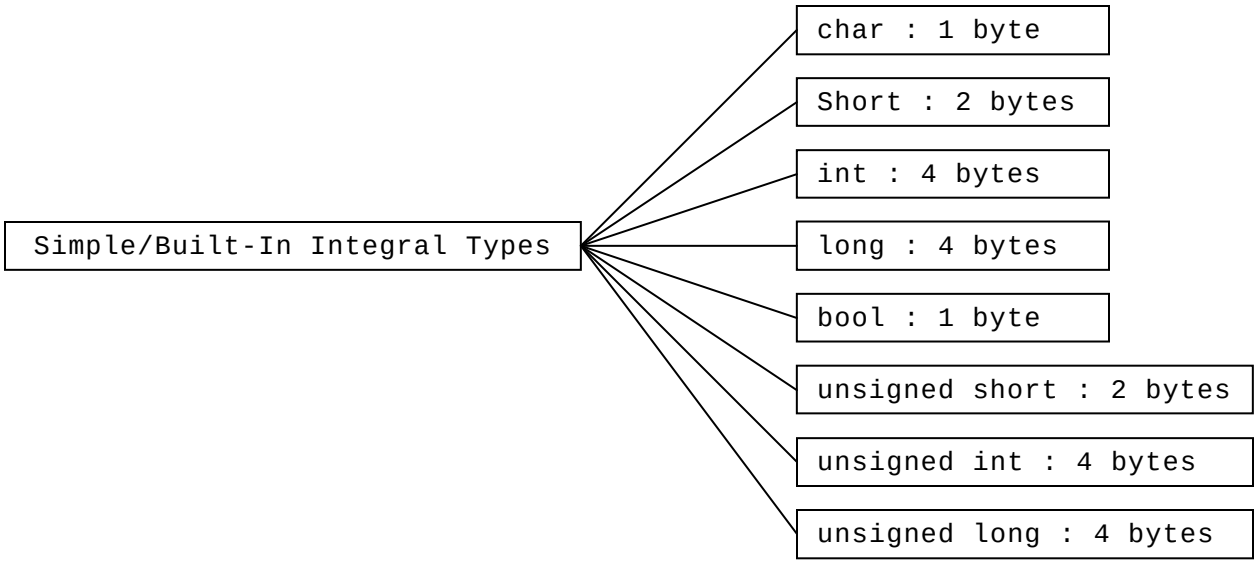

Figure 4 Data sizes for integral types

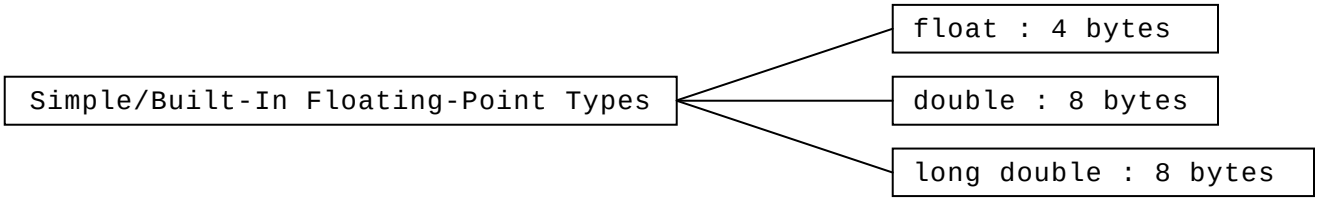

Figure 5 Data sizes for floating-point types

2. Variables – Introduction

In C++, every data value must also have and/or inherit a type.

- One should not just say "`5` is a number" because it is an incomplete specification in C++.
- Even saying "`5` is an integer value" is somewhat incomplete.
- A complete statement may be as "`5` is an integer of type `int`".

Data type is a mechanism that C++ enforces how a value should be stored. One can then retrieve and use the value depending on need (or required operations).

The process and storing and retrieving values will most likely involve variable.

What is a variable?

2.1 Definition

In C++, a variable is a name that the user can specify and use to store data of a given type. A variable is used to store data and, thus, one should be able to identify its storage and value.

User can choose any name for the variable, but then the user must also select a data type for the name to correctly represent a value associating with the variable.

A variable (name) can be made up with combination of alphabets, digits, and in some cases with the underscore ('_') and/or dollar sign ('$'). The use of underscore and dollar sign is not recommended and should not be used if possible.

There are some suggestions for making up variable name (as one of the identifier) as follows,

(1) A variable name should start with a lowercase alphabet, and

(2) A variable name can have several words; each word should start with an uppercase alphabet except the first word; and

(3) Digits can be used in combination with alphabets

Note that a variable must be declared before one can use it.

2.2 Examples

```
int iCounter;    //iCounter is a variable of type int
long lSum;       //lSum is a long


iCounter = 0;    //Using iCounter – assigning 0 to iCounter

float fVar1;     //fVar1 is a float
double dVar2;    //dVar2 is a double


char cValue1;    //cValue is char
```

3. C++ Constants – Introduction

Constants are values that cannot be changed during the execution of the program. Constants may have types (as being implied) when these values were coded into the code. There are three types of constants of

- **Integer,**
- **Floating-point**, and
- **String**.

Let's consider them next.

3.1 Integer Constants

An integer constant, as coded with a series of digits, has

- the signed integer (**`int`**) type, or
- the signed long integer (**`long`**) type if the number is large, and
- the default type is **`int`**.

One can override the defaults by using the suffixes **`u`** (or **`U`**) and **`l`** (or **`L`**). **`U`** and **`L`** are preferred to avoid any misunderstanding or confusion.

The following examples show the types of the constants.

Literal	Value	Type
123	123	int
-123	-123	int
-123L	-123	long int
123LU	123	unsigned long int

3.2 Floating-Point Constants

The default type for any floating-point constant is **`double`**. One can use the format specifiers **`l`** (**`L`**) and **`f`** (**`F`**) to indicate whether it is a **`long double`** or a **`float`**.

Some examples are given below.

Literal	Value	Type
`123.`	`123.0`	`double`
`-0.123`	`-0.123`	`double`
`123F`	`-123.0`	`float`
`123.0L`	`123`	`long double`

In addition to these numeric constants, C++ has two other constants: **character** and **string**.

3.3 Character Constants

Character constants are enclosed between two single quotes. In C++, there are two groups of characters: (1) Printable characters, and (2) Non-printable characters.

Printable Characters

Normally, character constants would have shown with only one character quoted. These characters are mostly called **printable characters** that means one can print these characters on paper or see them displayed on screen.

For examples,

```
'a' 'A' 'o' 'O' 'z' 'Z' '1' '9' '0'
```

Note that the digits enclosed in the single quotes should be understood as characters and not values.

Non-Printable Characters

There are other characters that have a backslash (\) added in front of characters (only one character after the backslash). These are called **nonprintable characters**; their values or effects would be either seen or heard when the system tries to display them.

For examples,

`'\0'`	Null character	`'\a'`	Bell
`'\b'`	Backspace	`'\t'`	Horizontal tab
`'\n'`	Newline	`'\''`	Single quote
`'\"'`	Double quote	`'\\'`	Backslash

3.4 String Constants

A string constant is a sequence of zero or more characters enclosed in a pair of double quotes.

For examples,

`""`	Null string
`"CIS xxx"`	A seven (7) character string
`"CIS xxx\n"`	An eight (8) character string

3.5 Boolean Constants

The **`bool`** value can either be **`true`** or **`false`**. The **`true`** constant can be thought of having a value **`1`** and a **`false`** is having a value of **`0`**.

4. C++ Basic Operations

C++ also provides basic operations that comprise of arithmetic operations, comparison operations, logic operations, etc.

Arithmetic Operations

```
+    -    *    /    %
```

Comparison Operations

```
>    >=   <    <=   ==   !=
```

Logical Operations

```
|| (AND)       && (OR)
```

There are other operators in C++ as well (e.g., `<<, >>, new, delete`, etc.).

The use of variables and operators will produce many expressions and/or statements that are from very simple to very complex.

For examples,

```
int iCounter;
long lSum;


iCounter = 0;
lSum = 1 + 8;


lSum = iCounter + 5;
lSum = iCounter * 5 + lSum – 9;
```

Note that a declared variable may not be used without being initialized or assigned with a known value.

Note also that objects in computer programming refer to data. In object-oriented programming, objects have both data and operations

And again, data are stored in memory. One may or may not be able to access to these memory locations; and one may or may not be able to change data values at these locations.

5. C++ Variable Declaration, Assignment, & Usage

Consider the following statement,

```
int iValue;
```

What does the statement mean? What is the value of `iValue`? Where is `iValue` stored in memory?

5.1 Declaration

The above statement declares `iValue` as a variable of type `int`. That means `iValue` can be used to store integral values with a specific format or type of `int`.

In general, a declaration will have the following form,

```
Type variableName;
```

Where

`Type` is any existing and predefined type definition
`variableName` can be any valid name

There are some rules that can be used to make up variable name. They are as follows,

(i) A variable must start with an alphabet or an underscore, and
(ii) A variable may be a combination of alphabets, digits, and underscore, and
(iii) A variable may have up to 31 characters (or more depending on compilers).

Some recommendations for variable name:

(1) Starting the variable with a lowercase character/letter, and
(2) Avoiding the underscore, and
(3) Using multiword name. In this case, every first character of the word should be capitalized except the first word, and
(4) Using sensible and meaningful variable name.

For most of the 32-bit platforms, a computer or system will reserve a 4-byte block for `iValue`. This 4-byte block will have integral values stored in it. Because of having just 4 bytes (or 32 bits) to store a value, there is a range of values that can be stored in this block with out error or lost.

During the execution of the program, the computer will search for this 4-byte block of memory and assign a declared variable to this memory block. No other variables can share this memory block. Every memory block can also be referred to by its address. The computer will assign this address and the user/programmer cannot ask where the block should be located at.

For a simple variable declaration just declared such as above, there is no known and valid value stored in the variable. That means one should not use these uninitialized or unassigned variables.

5.2 Assigning Value to Variable

By declaring a variable:

- A value can then be assigned to a declared variable. The assigned value can be constant or result of some expression (to be evaluated to a value).
- Assigning a value to a variable is referring to data storage. A variable should be assigned with a value before it may be used (or you will have garbage).
- Using a variable will involve
    - Retrieving the value being stored in the variable, and
    - Then performing some operations on the retrieved value.

For examples,

```
iValue = 5;
iValue = 5 + 100;
```

The process of variable assignment will almost always have the variable on the left hand side of the equal sign.

5.3 Location of Declaration

In C++, all variables do not have to be declared at once or at the beginning of the program. A variable needs to be declared just before using it. The computer (or system) will have its own way to handle all existing variables and the newly declared variables.

In general, all declared variables may not be allocated with memory blocks that are contiguous. But it is true that each individual memory block will have its size set and placed at a proper location.

5.4 Using Variable

- Using a variable really means retrieving the value that was stored in the variable.
- By using the variable, it is some where on the right hand side of the equal sign.

For examples,

```
int iSum;
iSum = iValue + 9;
iSum = iSum + iValue;
```

5.5 Variable Notation and Convention

There are conventions and styles for making up variable names that would also always include the above rules and recommendations.

One style, as given in the Hungarian notation, would have the first character (or, a group of starting characters) implying the data type that a variable should have. Explanation is given in the Hungarian Convention article.

For examples,

```
int iVal;
double dSum;
char cToken;
int iCounterBean;
int iSumOdd;
double dAveTemperature;
```