# Lecture Two – C++ Data Types

## Introduction

A program can use several data to solve a given problem, for example, characters, integers, or floating-point numbers. Since a computer uses different methods for processing and saving data, the data type must be known. The type defines:

1. The internal representation of the data.
2. The amount of memory to allocate.

## Built Data Types in C++

Built-in types (also called *fundamental types*) are specified by the C++ language standard and are built into the compiler. Built-in types aren't defined in any header file. The following table shows the built-in data type and the maximum and minimum value which can be stored in the designated variables. These are:-

1. Integer Data Type
2. Floating-point Data Type
3. Characters Data Type
4. Strings Data Type
5. Boolean Data Type
6. void Data Type

| Type | Size | Range of Values (decimal) |
|---|---|---|
| `char` | 1 byte | -128 to +127 or 0 to 255 |
| `unsigned char` | 1 byte | 0 to 255 |
| `signed char` | 1 byte | -128 to +127 |
| `int` | 2 byte resp. | -32768 to +32767 resp. |
| | 4 byte | -2147483648 to +2147483647 |
| `unsigned int` | 2 byte resp. | 0 to 65535 resp. |
| | 4 byte | 0 to 4294967295 |
| `short` | 2 byte | -32768 to +32767 |
| `unsigned short` | 2 byte | 0 to 65535 |
| `long` | 4 byte | -2147483648 to +2147483647 |
| `unsigned long` | 4 byte | 0 to 4294967295 |

## 1. Integer Data

Integer data types represent whole numbers without a fractional or decimal part. They can be signed (positive, negative, or zero) or unsigned (only positive or zero). The int type is the default basic integer type. It can represent all of the whole numbers over an implementation-specific range.

- A signed integer representation is one that can hold both positive and negative values. It's used by default, or when the signed modifier keyword is present.
- The unsigned modifier keyword specifies an unsigned representation that can only hold non-negative values.

A size modifier specifies the width in bits of the integer representation used. The language supports:

1. short, A short type must be at least 16 bits wide.
2. long, A long type must be at least 32 bits wide.
3. long long, A long long type must be at least 64 bits wide.

The int keyword may be omitted when signed, unsigned, or size modifiers are specified. The modifiers and int type, if present, may appear in any order. For example, short unsigned and unsigned int short refer to the same type.

Example:
```
int signedInt = -42;
unsigned int unsignedInt = 123;
```

### Integer Literals

When integer values are stored and represented as literal, then such literals are known as integer literals. There are two types of integer literal:

1. Prefixes
2. Suffixes
3. **Prefixes:** The bases of the integer values are represented through the prefix of the integer literals. For example, 0x80 = 128, here 0x represents Hexa decimal base and the value in decimal is 128. There are four types of prefixes used to represent integer literals:
   i. **Decimal-literal:** Decimal-literals have base 10, which does not contain any prefix for representation. It contains only decimal digits (0,1,2,3,4,5,6,7,8,9). For example, 10, 22, 34 etc.
   ii. **Octal-literal:** The base of the octal-literals is 8 and uses 0 as prefix for representation. It contains only octal digits (0, 1, 2, 3, 4, 5, 6, 7). For example, 010,022,034 etc.
   iii. **Hex-literal:** The base of the Hex-literals is 16 and uses 0x or 0X as the prefix for representation. It contains only hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a or A, b or B, c or C, d or D, e or E, f or F). For example, 0x80, 0x16, 0x4A etc.
   iv. **Binary-literal:** The base of the Binary-literals is 2 and uses 0b or 0B as the prefix for representation. It contains only binary digits (0, 1). For example, 0b11, 0b110, 0B111 etc.

Example:
```
#include <iostream>
using namespace std;
int main(){
    const int DECIMAL = 128; //defining decimal-literal
    const int OCTAL = 0200;//defining octal-literal
```

```
    const int HEX = 0x80;//defining hex-literal
    const int BINARY = 0b10000000;//defining binary-literal
cout<<"Decimal Literal: "<<DECIMAL<<"\n literals ";
//display of result through
    cout<<"Octal Literal: "<<OCTAL<<"\n";
    cout<<"Hex Literal: "<<HEX<<"\n";
    cout<<"Binary Literal: "<<BINARY<<"\n";}
```

Output:

```
Decimal Literal: 128
Octal Literal: 128
Hex Literal: 128
Binary Literal: 128
```

2. **Suffixes:** The type of integer values are represented through the suffixes of the integer literals. For example, 3826382382688LL, 2836263826823909ULL etc. In the above example, LL represents long long int for the value 3826382382688 and ULL represents unsigned long long int for the value 2836263826823909. Following are the types of suffixes used to represent integer literals:
   i. **int:** It is the default integer type and hence its representation does not need any suffix. The value of integer literals ranges from -2147483648 to 2147483647.
   i. **unsigned int:** It is the integer type that does not contain negative int values. The value of unsigned integer literals ranges from 0 to 4294967295. A compiler error will be triggered if any negative value gets assigned to unsigned integer literals. The literal contains u or U as the suffix for its representation.
   ii. **long int:** The value of long integer literals ranges from -2,147,483,648 to 2,147,483,647. The literal contains l or L as the suffix for its representation.
   iii. **unsigned long int:** The value of unsigned long integer literals ranges from 0 to 4,294,967,295. The literal contains ul or UL as the suffix for its representation.
   iv. **long long int:** The value of long long integer literals ranges from -(2^63) to (2^63)-1. The literal contains ll or LL as the suffix for its representation.
   v. **unsigned long long int:** The value of unsigned long long integer literals ranges from 0 to 18,446,744,073,709,551,615. The literal contains ull or ULL as the suffix for its representation.

Examples:

```
long b = 4523232;
long int c = 2345342;
long double d = 233434.56343;
short d = 3434233; // Error! out of range
unsigned int a = -5;    // Error! can only store positive numbers or 0
```

Example:

```
#include <iostream>
using namespace std;
int main(){
    //defining integer-literal
    const int INTEGER = 128;
    //defining unsigned integer-literal
```

```
    const unsigned int UNSIGNED = 3147483647U;
    //defining long integer-literal
    const long int LONG = 2147483646L;
    //defining unsigned long integer-literal
    const unsigned int UNSIGNED_LONG = 4294967294UL;
    //defining long long integer-literal
    const long long int LONG_LONG = 5294967294LL;
    //defining unsigned long long integer-literal
const unsigned long long int UNSIGNED_LONG_LONG = 18446744073709551610ULL;
    //display of result through literals
    cout<<"Integer Literal: "<<INTEGER<<"\n";
    cout<<"Unsigned Integer Literal: "<<UNSIGNED<<"\n";
    cout<<"Long Integer Literal: "<<LONG<<"\n";
    cout<<"Unsigned Long Integer Literal: "<<UNSIGNED_LONG<<"\n";
    cout<<"Long Long Int Literal: "<<LONG_LONG<<"\n";
    cout<<"Unsigned Long Long Int Literal: "<<UNSIGNED_LONG_LONG<<"\n";}
```

Output:

```
Integer Literal: 128
Unsigned Integer Literal: 3147483647
Long Integer Literal: 2147483646
Unsigned Long Integer Literal: 4294967294
Long Long Int Literal: 5294967294
Unsigned Long Long Int Literal: 18446744073709551610
```

## 3. Floating-Point Data

The floating-point data types contain the real numbers. The real numbers hold an integer part, a real part and a fractional part and an exponential part in it.
The following table lists the floating-point types in C++ and the comparative restrictions on floating-point type sizes.

| Type | Size | Range of Values | Lowest Positive Value | Accuracy (decimal) |
|---|---|---|---|---|
| float | 4 bytes | –3.4E+38 | 1.2E—38 | 6 digits |
| double | 8 bytes | –1.7E+308 | 2.3E—308 | 15 digits |
| long double | 10 bytes | –1.1E+4932 | 3.4E—4932 | 19 digits |

Numbers with a fraction part are indicated by a decimal point in C++ and are referred to as floating-point numbers. In contrast to integers, floating-point numbers must be stored to a preset accuracy. The following three types are available for calculations involving floating-point numbers:

1. float for simple accuracy
2. double for double accuracy
3. long double for high accuracy

The value range and accuracy of a type are derived from the amount of memory allocated and the internal representation of the type.

1. In the decimal form, it is necessary to add a decimal point, exponent part or both, or else it will give an error.
   For example: `-2.0, 0.0000234`

```
float myFloat = 3.14159;
double myDouble = 3.141592653589793;
float area = 64.74;
double volume = 134.64534;
```

2. In the exponential form, it is necessary to add the integer part, fractional part or both, or else, it will give an error.
   For example: `-0.22E-5`
   Note: `E-5 = 10-5`

## Example:-

```
// To display hexadecimal integer literals and
// decimal integer literals.
//
#include <iostream>
using namespace std;
int main()
{
// cout outputs integers as decimal integers:
cout << "Value of 0xFF = " << 0xFF << " decimal"
<< endl;                    // Output: 255 decimal
// The manipulator hex changes output to hexadecimal
// format (dec changes to decimal format):
cout << "Value of 27 = " << hex << 27 <<" hexadecimal"
<< endl;                 // Output: 1b hexadecimal
return 0;}
```

## 4. Character Data Types

The C++ language uses the char data type to represent characters. When a single character enclosed by a single quote is stored and represented as a literal, then the literal is known as a character literal. char type: All the characters belonging to the ASCII table can be represented and stored through this type of literal.

Example:

```
char myChar = 'X';
const char VARA = 'A'; // constant char literal
char test = 'h';
```

As you might have guessed, signed char can store both positive and negative integers, while unsigned char can only store positive integers (including 0).

```
// plain char
char plain_1 = 65;
char plain_2 = 0;
```

```
// plain char with negative value might cause problems with some compilers
char plain_3 = -56;
// signed char
signed char  sin_1 = 12;
signed char  sin_2 = 0;
signed char  sin_3 = -12;

// unsigned char
unsigned char  unsin_1 = 85;
unsigned char  unsin_2 = 0;
```

The C++ language does not specify any particular characters set, although in general a character set that contains the ASCII code (American Standard Code for Information Interchange) is used. This 7-bit code contains definitions for 32 control characters (codes 0 – 31) and 96 printable characters (codes 32 – 127). The char (character) type is used to store character codes in one byte (8 bits). This amount of storage is sufficient for extended character sets,.

## 5. String Data Types

When more than one character is stored in double-quotes and represented as literals. Such a literal is known as a string literal. It can store the entire special as well as escape sequence characters.

## Internal representation of a string literal

String literal: `"Hello!"`

Stored byte sequence:

| 'H' | 'e' | 'l' | 'l' | 'o' | '!' | '\0' |
|---|---|---|---|---|---|---|

Example:

```
#include <iostream>
using namespace std;
int main () {
   char ch[12] = {'H', 'e', 'l', 'l', 'o', ' ', 'w','o','r','l','d'};
   string st = "Welcome to C++";
   string std_st = "Happy Learning";
   cout << ch << endl;
   cout << st << endl;
   cout << std_st << endl;   return 0;}
Output:
Hello world
Welcome to C++
Happy Learning
```

## 6. Boolean Data Types

Boolean is a data type in C++ that represents true or false values. It is commonly used in programming to control program flow, make decisions, and evaluate conditions. In C++, a Boolean is a data type that can have two possible values: true or false. Booleans are commonly used in conditional statements, loops, and other control structures to determine whether a particular condition is true or false. To declare a Boolean variable in C++, we use the bool keyword. Here's an example:

```
bool myBoolean = true;
```

In this example, myBoolean is a Boolean variable that has been assigned the value true. We can also declare Boolean variables without assigning them a value, like this:

```
bool myBoolean;
```

In this case, the initial value of myBoolean will be false.
Example:

```
bool a = true;
bool b = false;
bool c = true;
bool result1 = a && b; // result1 is false
bool result2 = a || b; // result2 is true
bool result3 = !a;    // result3 is false
bool result4 = !b;    // result4 is true
bool result5 = (a || b) && c; // result5 is true
```

## 7. void type

The void type describes an empty set of values. No variable of type void can be specified. The void type is used primarily to declare functions that return no values or to declare generic pointers to untyped or arbitrarily typed data. The void keyword indicates an absence of data.

Example: void main()

## Sizes of built-in types

The amount of memory needed to store an object of a certain type can be ascertained using the sizeof operator: sizeof(name)yields the size of an object in bytes, and the parameter name indicates the object type or the object itself. For example, sizeof(int) represents a value of 2 or 4 depending on the machine. Most built-in types have implementation-defined sizes. The following table lists the amount of storage required for built-in types in Microsoft C++. In particular, long is 4 bytes even on 64-bit operating systems.

| Data Types | Memory Size | Range |
|---|---|---|
| char | 1 byte | -128 to 127 |
| signed char | 1 byte | -128 to 127 |
| unsigned char | 1 byte | 0 to 127 |
| short | 2 byte | -32,768 to 32,767 |
| signed short | 2 byte | -32,768 to 32,767 |
| unsigned short | 2 byte | 0 to 32,767 |
| int | 2 byte | -32,768 to 32,767 |
| signed int | 2 byte | -32,768 to 32,767 |
| unsigned int | 2 byte | 0 to 32,767 |
| short int | 2 byte | -32,768 to 32,767 |
| signed short int | 2 byte | -32,768 to 32,767 |
| unsigned short int | 2 byte | 0 to 32,767 |
| long int | 4 byte | |
| signed long int | 4 byte | |
| unsigned long int | 4 byte | |
| float | 4 byte | |
| double | 8 byte | |
| long double | 10 byte | |

# Exercises

## Exercise 1

The sizeof operator can be used to determine the number of bytes occupied in memory by a variable of a certain type. For example, sizeof(short) is equivalent to 2. Write a C++ program that displays the memory space required by each fundamental type on screen.

### Solution 1

```
#include <iostream>
using namespace std;
int main()
{
cout << "\nSize of Fundamental Types\n"
<< "  Type          Number of Bytes\n"
<< "---------------------------------" << endl;
cout << "  char:          " << sizeof(char) << endl;
cout << "  short:         " << sizeof(short)<< endl;
cout << "  int:           " << sizeof(int)  << endl;
cout << "  long:          " << sizeof(long) << endl;
cout << "  float:         " << sizeof(float)<< endl;
cout << "  double:        " << sizeof(double)<<endl;
cout << "  long double:   " << sizeof(long double)
<< endl; return 0;}
```

## Exercise 2

Write a C++ program to generate the screen output shown below:-

```
                    I
                              "RUSH"
                                       \TO\
                              AND
                    /FRO/
```

## Solution 2

```
// Usage of escape sequences
#include <iostream>
using namespace std;
int main(){
cout << "\n\n\t I"              // Instead of tabs
"\n\n\t\t \"RUSH\""       // you can send the
"\n\n\t\t\t \\TO\\"       // suited number
"\n\n\t\t AND"            // of blanks to
"\n\n\t /FRO/" << endl;    // the output.
return 0; }
```

## Exercise 3

Which of the variable definitions shown below is invalid or does not make sense?

Defining and initializing variables:

```
int a(2.5); const long large;
int b = '?'; char c('\');
char z(500); unsigned char ch = '\201';
int big = 40000; unsigned size(40000);
double he's(1.2E+5); float val = 12345.12345;
```

## Solution 3

Incorrect:

```
int a(2.5);              // 2.5 is not an integer value
const long large;        // Without initialization
char z(500);             // The value 500 is too large
// to fit in a byte
int big = 40000;         // Attention! On 16-bit systems
// int values are <= 32767
double he's(1.2E+5);     // The character ' is not
// allowed in names
float val = 12345.12345;  // The accuracy of float
// is only 6 digits
```

## Exercise 4

Write a C++ program that two defines variables for floating-point numbers and initializes them with the values 123.456 and 76.543. Then display the sum and the difference of these two numbers on screen.

## Solution 4

```
// Defining and initializing variables
#include <iostream>
```

```
using namespace std;
int main()
{ float x = 123.456F,              // or double
y = 76.543F,sum;
sum = x + y;
cout << "Total:        "
<< x << " + " << y << " = " << sum << endl;
cout << "Difference:  "
<< x << " — " << y << " = " << (x — y) << endl;
return 0; }
```