Programming Fundamental

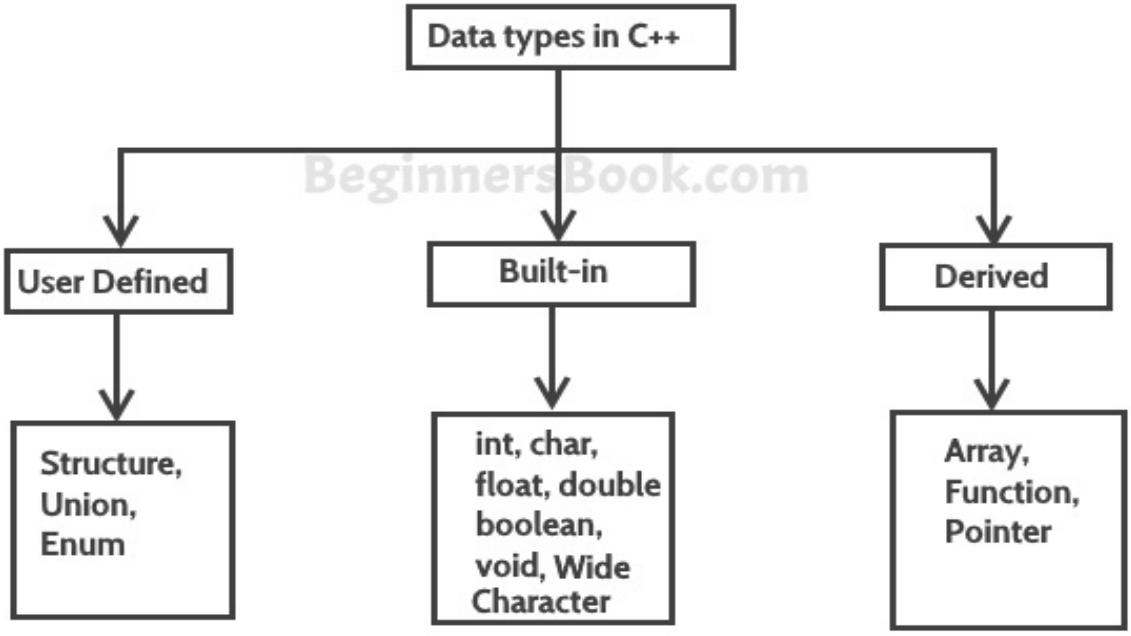
Lab#03

Table of Contents

Datatypes

All variables use data-type during declaration to restrict the type of data to be stored. Therefore, we can say that data types are used to tell the variables the type of data it can store. Whenever a variable is defined in C++, the compiler allocates some memory for that variable based on the data-type with which it is declared. Every data type requires a different amount of memory.

Types of Datatypes



Integer: Keyword used for **integer data types** is int. Integers typically requires 4 bytes of memory space and ranges from -2147483648 to 2147483647.

Character: Character data type is used for storing **characters**. Keyword used for character data type is char. Characters typically requires 1 byte of memory space and ranges from -128 to 127 or 0 to 255.

Boolean: Boolean data type is used for storing **boolean or logical values**. A boolean variable can store either true or false. Keyword used for boolean data type is bool.

Floating Point: Floating Point data type is used for storing **single precision floating point values or decimal values**. Keyword used for floating point data type is float. Float variables typically requires 4 byte of memory space.

Double Floating Point: Double Floating Point data type is used for **storing double precision floating point values or decimal values**. Keyword used for double floating point data type is double. Double variables typically requires 8 byte of memory space.

void: Void means **without any value**. void datatype represents a valueless entity. Void data type is used for those function which does not returns a value.

Datatype Modifiers: As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold. Data type modifiers available in C++ are:

- Signed
- Unsigned
- Short
- Long

Below table summarizes the modified size and range of built-in datatypes when combined with the type modifiers:

DATA TYPE	SIZE (IN BYTES)	RANGE
short int	2	-32,768 to 32,767
unsigned short int	2	0 to 65,535
unsigned int	4	0 to 4,294,967,295
Int	4	-2,147,483,648 to 2,147,483,647
long int	4	-2,147,483,648 to 2,147,483,647
unsigned long int	4	0 to 4,294,967,295
long long int	8	-(2^63) to (2^63)-1
unsigned long long int	8	0 to 18,446,744,073,709,551,615
signed char	1	-128 to 127
unsigned char	1	0 to 255
Float	4	

Double	8	
long double	12	

Note : Above values may vary from compiler to compiler.

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

int main()
{
    cout << "Size of char : " << sizeof(char)
      << " byte" << endl;
    cout << "Size of int : " << sizeof(int)
      << " bytes" << endl;
    cout << "Size of short int : " << sizeof(short int)
      << " bytes" << endl;
    cout << "Size of long int : " << sizeof(long int)
       << " bytes" << endl;
    cout << "Size of signed long int : " << sizeof(signed long int)
       << " bytes" << endl;
    cout << "Size of unsigned long int : " << sizeof(unsigned long int)
       << " bytes" << endl;
    cout << "Size of float : " << sizeof(float)
       << " bytes" <<endl;
    cout << "Size of double : " << sizeof(double)
       << " bytes" << endl;
    cout << "Size of wchar_t : " << sizeof(wchar_t)
       << " bytes" <<endl;
   System("pause");
    return 0;
}

Output:

Size of char : 1 byte
Size of int : 4 bytes
Size of short int : 2 bytes
Size of long int : 8 bytes
Size of signed long int : 8 bytes
Size of unsigned long int : 8 bytes
Size of float : 4 bytes
Size of double : 8 bytes
Size of wchar_t : 4 bytes
```

Input(cin)

In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is cin.

For formatted input operations, cin is used together with the extraction operator, which is written

as >> (i.e., two "greater than" signs). This operator is than followed by the variable where the extracted data is stored. For example:

```
int age;
cin>>age;
```

The first statement declares a variable of type *int* called age, and the second extracts from *cin* a value to be stored in it. This operation makes the program wait for input from *cin*; generally, this means that the program will wait for the user to enter some sequence with the keyboard. In this case, note that the characters introduced using the keyboard are only transmitted to the program when the *ENTER (or RETURN)* key is pressed. Once the statement with the extraction operation on *cin* is reached, the program will wait for as long as needed until some input is introduced.

The extraction operation on *cin* uses the type of the variable after the >> operator to determine how it interprets the characters read from the input; if it is an integer, the format expected is a series of digits, if a string a sequence of characters, etc.

Output(cout)

On most program environments, the standard output by default is the screen, and the C++ stream object defined to access it is *cout*.

For formatted output operations, *cout* is used together with the *insertion operator*, which is written as << (i.e., two "less than" signs).

```
cout<<"output sentence";
cout<<120;
cout<<x;
```

The << operator inserts the data that follows it into the stream that precedes it. In the examples above, it inserted the literal string *Output sentence*, the number *120*, and the value of variable x into the standard output stream *cout*. Notice that the sentence in the first statement is enclosed in double quotes (") because it is a string literal, while in the last one, x is not. The double quoting is what makes the difference; when the text is enclosed between them, the text is printed literally; when they are not, the text is interpreted as the identifier of a variable, and its value is printed instead.

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

int main ()
{
 int i;
 cout << "Please enter an integer value: ";
 cin >> i;
 cout << "The value you entered is " << i;
```

```
 cout << " and its double is " << i*2 << ".\n";
 return 0;
}
```
Output:
Please enter an integer value: 702
The value you entered is 702 and its double is 1404.

Comments

A well-documented program is a good practice as a programmer. It makes a program more readable and error finding become easier. One important part of good documentation is Comments.

- In computer programming, a comment is a programmer-readable explanation or annotation in the source code of a computer program
- Comments are statements that are not executed by the compiler and interpreter.

Types of comments

1. Single line comment
2. Multi-line comment

1. Single line Comment

Represented as // double forward slash. It is used to denote single line comment. It apply comment to a single line only. It is refereed as C++-style comments as it is originally part of C++ programming.
For example:

```
#include<iostream>
int main()
{

    // Single line Welcome user comment
    cout<<"Welcome to PFlab";
    return 0;
}
Output:
Welcome to PFlab
```

2. Multi-line comment

Represented as /* any_text */ start with forward slash and asterisk (/*) and end with asterisk and forward slash (*/). It is used to denote multi-line comment. It can apply comment to more than a single line. It is referred as C-Style comment as it was introduced in C programming.

```
#include<iostream>
int main()
{
```

```
    /* Multi-line Welcome user comment
    written to demonstrate comments
    in C/C++ */
cout<<"Welcome to PF lab";
    return 0;
}
Output:
Welcome to PF lab
```

Reference:

https://beginnersbook.com/2017/08/cpp-data-types/

https://www.geeksforgeeks.org/c-data-types/

http://www.cplusplus.com/doc/tutorial/basic_io/

https://www.geeksforgeeks.org/basic-input-output-c/