**Overview of Basic Concepts in C++ - Chapter 2**

**Table of Contents**

[Hello World 2](#_Toc49598350)

[Character Variables 5](#_Toc49598351)

[Escape Sequences 6](#_Toc49598352)

[Input 7](#_Toc49598353)

[Arithmetic Precedence 8](#_Toc49598354)

[const Qualifier 9](#_Toc49598355)

[#define Directive 10](#_Toc49598356)

[The setw Manipulator 11](#_Toc49598357)

[Unsigned Data Types 13](#_Toc49598358)

[Casts 14](#_Toc49598359)

[Header Files and Library Files 16](#_Toc49598360)

## Hello World

The following is the code for the Hello World program in C++:

#include<iostream>  
using namespace std;  
int main()  
{  
 cout<<"Hello World";  
 return 0;  
}

C++

There are a large number of similarities between this code and one written in the C language. In fact, code for the C language can be used in a C++ program, either alone or mixed in, without any problems. The first two lines are known as directives. The first line simply includes a header file for some basic functions such as taking user input and displaying output.

The second line declares which namespace is being used. In object-oriented programming, objects are divided into different namespaces. For example, the object cout is part of the standard namespace, and thus we have declared at the top of our code that the standard namespace is being used by default. Using different namespaces helps in avoiding conflicts between objects with the same name. If we did not include this line, every time we used cout or some other object belonging to the standard namespace, we would explicitly have to declare which namespace it belonged to like this:

std::cout<<”Hello World”;

C++

If we are using one namespace by default, and we wish to use an object from a different namespace, the namespace the new object belonged to would have to explicitly be declared.

Extra note: The string within the double quotes, “Hello World”, is known as a string constant, since it is not stored in a variable and cannot be changed.

The following is another example:

#include<iostream>  
using namespace std;  
int main()  
{  
 int var = 10;  
 cout<<var;  
 cin>>var;  
 cout<<"\nThe value of var is "<<var<<endl;  
}

C++

Here we can see that the integer variable var was directly printed using cout. The closest parallel from the C language is the printf() function, but that required a format specifier to declare what type of value was being printed. cout does not need this. Similarly, cin can be used to directly take input from a user and store it into var, without need for a format specifier. Notice that the symbols after cout point *towards* cout and the symbols after cin point *away from* cin. This is related to how the streams work. The identifier cout is predefined in C++ to correspond to the standard output stream. A stream is an abstraction that refers to a flow of data. The standard output stream normally flows to the screen display, but it can be redirected. Streams are discussed in detail in Chapter 12.

The operator << is called the insertion or put to operator. It directs the contents of the variable to its right to the object to its left. From C, the << operator was used as the left-shift bitwise operator. It has been overloaded to perform its new function here. Operator overloading is discussed in detail in Chapter 8.

The last line makes use of a process called cascading. This is when a mixture of strings and variables are printed in a single line one after the other. Cascading can also be used with cin, but this is not usually done. One thing to note here is, at the beginning of the line \n has been used, which is simply a newline character, but at the end of the line endl has been used which is new. endl is a manipulator. Manipulators send additional instructions to the output stream to manipulate the output. Here, it performs the same function as \n, but it also clears the buffer. Details of what this means and why it is useful will be discussed later, but for now it is enough to know that its use is encouraged in the programming community.

## Character Variables

The variable type char can store integers in the range -128 to 127. 0 to 127 are used to represent ASCII characters. This can be extended to 255 to accommodate foreign language characters, but the numbers 128 to 255 are not standardized. This, along with the fact that char only uses one byte, which is too small for some foreign characters, causes problems with many systems. Standard C++ provides a larger character type wchar\_t to handle foreign characters.

## Escape Sequences

Some characters such as ‘\n’ and ‘\t’are known as escape sequences. They do not represent the normal n or t characters, but instead represent the newline and tab characters respectively. Their normal behaviour is ‘escaped’ using the backslash. There are more characters that work like this. We would have to escape the normal behaviour of quotation marks in this way for example, if we wanted them to appear on the screen along with our text.

Escape sequences are also used to print characters with hexadecimal ASCII codes. A solid rectangle for example, has an ASCII code of 178, which translates to B2. Thus, it can be printed using ‘\xB2’. This is the format that must be used for hexadecimal ASCII codes.

## Input

cin >> var;

C++

cin is an object predefined in C++ to correspond to the standard input stream, which takes data from the keyboard unless it is redirected. The >> used with cin is called the extraction or get from operator. It takes the value from the stream object on its left and places it in the variable on its right.

## Arithmetic Precedence

For operators that have the same precedence, like \* and /, the one on the left is carried out first. Thus, 5/9\*2 != 5/(9\*2).

## const Qualifier

It is possible to ensure that a particular variable is never edited, even by mistake. This is done by using the keyword const while initializing the variable as:

const int a = 5;

C++

## #define Directive

This can be used to define values for particular keywords. When that keyword is found in the code, the compiler will replace the keyword with whatever value is assigned to it.

#define **PI** 3.14159

C++

Whenever the keyword **PI** is found in the program, it will be replaced with the value 3.14159. However, this method is not commonly used in C++.

## The setw Manipulator

By default, any text being printed has a certain width. If we print out the number 123, it will take up the width of three characters. This can make things ugly at times, for example if we wanted to make a list of cities and their populations, we would need to add spaces or tabs to separate the names of the cities from their populations. Further, if the populations were unknown, then the list could not be aligned properly since the numbers could be very large or very small.

Cout<<”LOCATION\t”<<”POP.”<< endl;  
cout<<”Portcity\t”<<pop1<<endl;  
cout<<”Hightown\t”<<pop2<<endl;  
cout<<”Lowville\t”<<pop3<< endl;

C++

This code gives the following result:

LOCATION POP.

Portcity 2425785

Hightown 47

Lowville 9761

As you can see, the numbers are difficult to compare. This can be solved using the setw manipulator. It gives the character set a fixed width, adding spaces where necessary to fill up the width, and aligns it along the right-hand side.

Cout<<setw(8)<<”LOCATION”<<setw(12)<<”POPULATION”<<endl;  
cout<<setw(8)<<”Portcity”<<setw(12)<<pop1<<endl;  
cout<<setw(8)<<”Hightown”<<setw(12)<<pop2<<endl;  
cout<<setw(8)<<”Lowville”<<setw(12)<<pop3<<endl;

C++

Here, the city names take up the width of 8 characters, and the populations take up the width of 12 characters.

LOCATION POPULATION

Portcity 2425785

Hightown 47

Lowville 9761

Note that the definitions for the manipulators other than endl are not in the iostream header file. We must include the iomanip header file.

## Unsigned Data Types

We can force data types to be unsigned, so that they do not include negative values. This allows us to take twice the size in positive values.

unsigned int a;

C++

## Casts

Casts, also known as type casts, are used to convert one data type to another. Conversions may be done automatically by the compiler, but the automatic conversion follows a specific order. To convert between any data types we want, we can use casts.

Casts can be of several types, such as static casts, dynamic casts and const casts, but we will only look at static casts for now.

aCharVar = static\_cast<char>(anIntVar);

C++

The above line converts an integer to a character type and stores it in another variable.

Casts can also help us bypass the problem of storing values that are too large. For example, if we have an integer a = 1500000000, and we try to perform the calculation, a = (a\*10)/10; both the original value and the actual result of this calculation are small enough to be stored in the variable a. However, in the middle of the calculation, after the multiplication, we get a value 15000000000, which is far too big to be stored as an integer. This results in the wrong answer. If we do this instead:

a = (static\_cast<double>(a) \* 10)/10;

C++

the results will be correct. This is because the variable a is converted to a temporary variable of type double which has enough memory to store the results of the multiplication that follows. This is called coercion.

Previously, casts could be performed in this way:

aCharVar = (char) anIntVar;

C++

This style will still work, but it is discouraged since it is difficult to find, even with the search function.

Casting in general should be used only if necessary since they are a controlled way to bypass type safety, a mechanism that makes sure variables do not change types by mistake. Casting can cause trouble since they sometimes make it impossible for the compiler to spot problems.

## Header Files and Library Files

We have frequently used header files to include functions from outside of our program. What we are actually doing is including what are known as library functions. These library functions are available in library files. However, to be able to use them, out code needs to have some additional information about them as well. This information, along with the links to the library files, are given in the header files. So header files do not actually contain the functions we use, they just contain directions to those functions.