C++ Tutorial for Beginners

Kouassi Franck Armand Prince

202024080107

Contents

List of Figures

List of codes

1 Getting Started

This section introduces the basics of C++ programming language and the tools needed to follow this tutorial. The goal of this tutorial is to help beginners getting started with C++ programming language. It does not require you to you to have a prior programming background kownledge. All you have to do is to follow along with me and and try to ***WRITE*** the code on your own machine not just read. Believe me it is easier to read and assume that you have mastered it until you are required to write the code by yourself, that is where it realize you have not properly understood it.

1.1 Understand the Computer Language

Your computer is an incredible and complicated device. Basically, the computer understands one simple language composed of 0 and 1. Thus a message like this "*01001100101001010*" could mean "open a window" for instance. Fortunately, we do not have to learn this language (Binary language). Programmers created languages which are much simpler than binary language. Here you could check *the number of programming languages.*
All programming languages have the same goal, that is being able to easily and efficently communicate with the computer compare to binary language.
Here, is how it works :

1. You write the instructions to be executed by the computer in a programming language (e.g C++)
2. The instructions are translated in binary (0 and 1), the language understood by the computer.
3. The computer can now decode the message and executes your request.

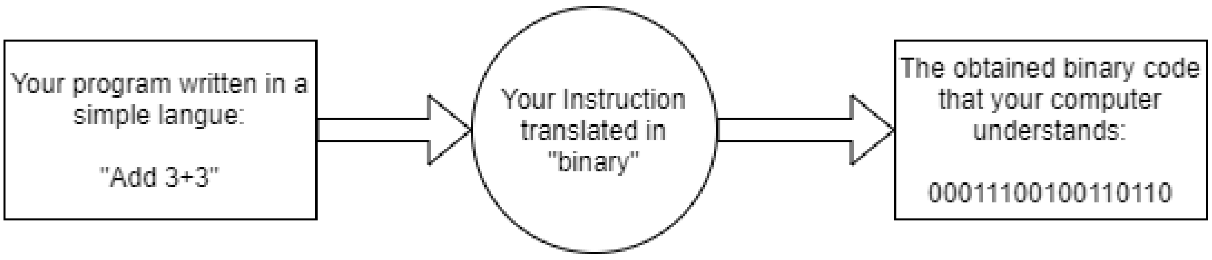
Your program written in a simple langue:
"Add 3+3"
Your Instruction translated in "binary"
The obtained binary code that your computer understands:
00011100100110110

Figure 1: Compiling Process.

1.2 C++ against other programming languages

Before we start talking about why C++ reprsents a powerful language despite its age. let's discuss the the key points to analyze before diving into a language.

There exists numerous programming languages as mentioned in above section, although some languages are interesting, they are seldom used. The main challenge that comes with these languges, is that they do not have a very big community so imagine you working on a project and you are facing a problem, it is difficult to find help since not so many people are using the the language.This explains why C++ represents a good choice for debutant programmers. You are not alone, a lot ressources are availble to guide through your learning process, also C++ is still being widely used.

Another interesting aspect to look at as well is the programming language level. There are of two (02) types: ***high level*** and ***low level***.
high level: is a language that is that is far from binary language and really to humans languge,it allows to easily understand and translate instructions contrary to ***low level*** which a language closed to machine language and generally requires much more effort but gives you more control over what you can do, it is a trade-off.

C++ is a low level language. Do not panic, although coding in C++ might be a little complex, you will have in your possession a very ***powerful*** and particulary ***fast*** language. Infact, if most games are

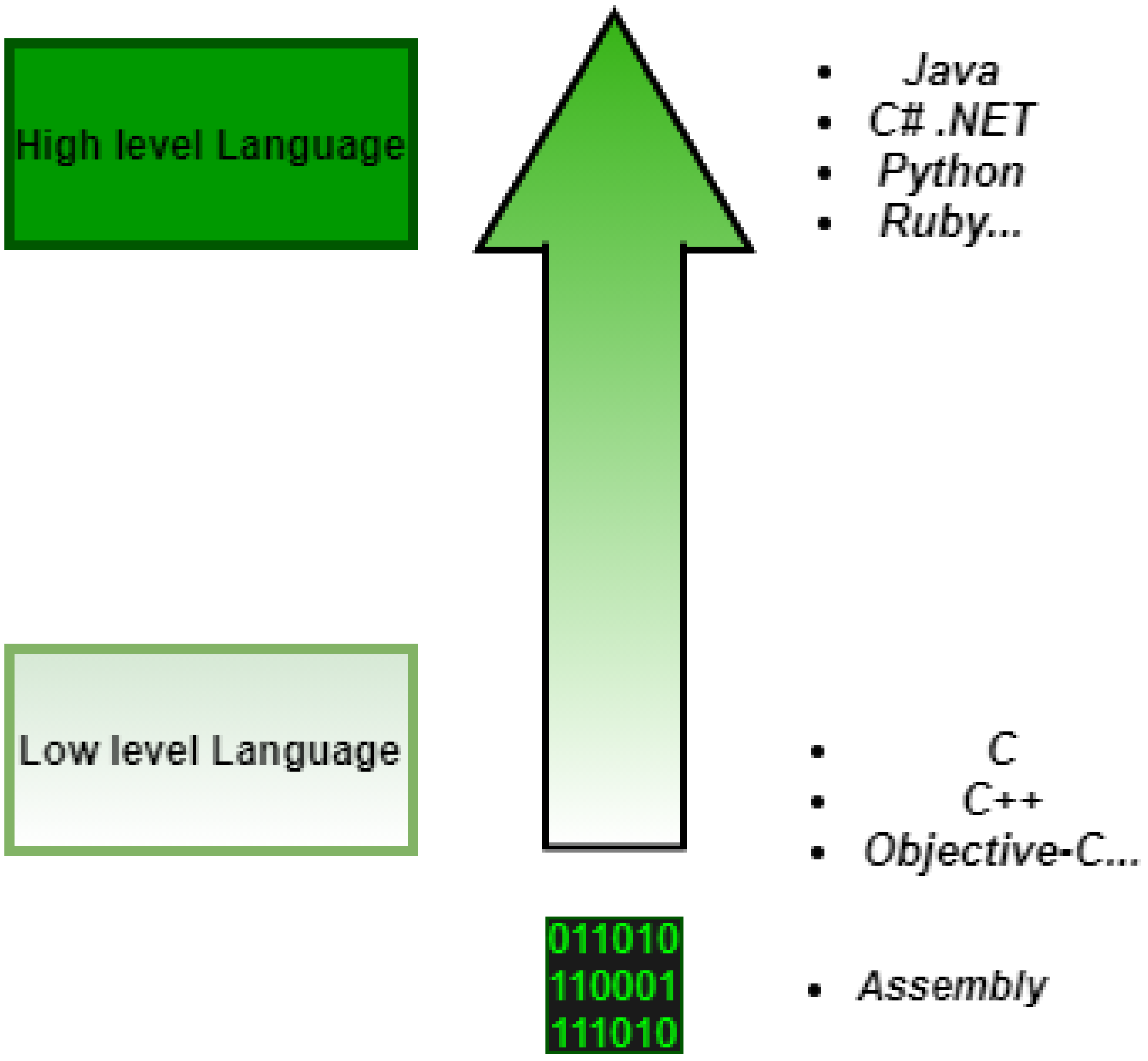

Figure 2: Programming Language by level.

developed in C++, it is because it is the language capable of coupling speed and power, that makes it an essential language.

1.3 Summary of C++

Here we are going to showcase some aspects of C++ that make it an important language regardless of how long it has been since it creation.

- **Popularity** : C++ is one of the most popular languages in the world. It is used by some 4.4 million developers worldwide
- **Large Community** : There is a large online community of C++ users and experts that is particularly helpful in case any support is required. There is a lot of resources available on the internet regarding C++.
- **Portable** : Programs developed in C++ can be moved from one platform to another. This is one of the main reasons that applications requiring multi-platform or multi-device development often use C++.

- **Speed** : Programs written in C++ language execute more faster compare to most programming languages

Snippet of C++ To give you an idea of how the code looks, let's look at a simple C++ program displaying "Hello world!" on the screen. Do not try to understand the code just appreciate the beauty and structure. We will go into details in the following sections

```cpp
//==================================================
// Sample example of C++ programming language
//==================================================

#include <iostream>

using namespace std;

int main()
{
    cout << "Hello world!" << endl;
    return 0;
}
//==================================================
```

Listing 1: Sample example of C++ programming language

If you are interested in knowing the story of C++ starting from its creation, you can learn all about C++ from wikipedia

1.4 Summary

- Programs allow us to efficently control actions on the computer: web browsing, text editing etc
- In order to create a program, we write instructions for the computer using a programming; source code
- The source code must be converted in binary by what we could a compiler, it allows the executation of the code.
- C++ is a widely used programming language, it is an evolution of C programming due to the fact that it allows Object Oriented Programming (OOP), a very powerful programming feature.

2 Environment setup

In this section, we are going to introduce the tools needed to follow this tutorial.From our previous discuss, you already know by now an important tool needed, Yes you are right, you need a Compiler, the program that converts your C++ code into the computer readable format.

Aside these, there are additional tools needed for you to code with ease

- **A text editor**: It will allow you to write your source code. On windows we have Notepad or Vi on linux. But of course, it is less recommed because as your code gets bigger and bigger, you might not not be able to fully control it.
- **A compiler**: as mentioned above, it converts your source code into binary format for the computer
- **A debugger**: it helps you find bugs in your programs.

From now on we have two options (02) either we get the programs seperatly which is of course much complicated, but on Linux most programmers prefer to use them in that way, I will not go into much details here, instead we are going to explore the simple way. We can get a program ”3 in 1”, Yes you heard me correctly, a tool capable of handling the 3 listed tools It is commonly refered to as an **IDE** (Integrated Developement Environment). There are of numerous types. In this tutorial we are not going to discuss their similarities. I personnaly recommed Visual Studio Code and here is how you can get started with C++ for Visual Studio Code. So go ahead and install the necessary packages.

3 Your first code

The “Hello World” program is the first but most vital step towards learning any programming language and it is certainly the simplest program you will learn with each programming language. All you need to do is display the message ”Hello World” on the output screen.

```
// Your First C++ Program

#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

Listing 2: First C++ code

Output:

```
Hello world!
```

Listing 3: First C++ code Output

1. // Your first C++ program

In C++, any line starting with // is a comment. Comments are intended for the person reading the code to better understand the functionality of the program. It is completely ignored by the C++ compiler.

2. #include <iostream>

The #include is a preprocessor directive used to include files in our program. The above code is including the contents of the iostream file.

3. int main() {...}

A valid C++ program must have the main() function. The curly braces indicate the start and the end of the function. The execution of code beings from this function.

4. std::cout <<”Hello World!”;

std::cout prints the content inside the quotation marks. It must be followed by ’<<’ followed by the

format string. In our example, ”Hello World!” is the format string.
Note: ; is used to indicate the end of a statement.
5. return 0;
The return 0; statement is the ”Exit status” of the program. In simple terms, the program ends with this statement.

As you might have noticed, the code in 1 looks a little different from the one in 2 but produce the same output do not worry we are going to get to the difference soon.
In 1, line 7 we used using namespace std; to tell the compiler to use standard namespace. Namespace collects identifiers used for class, object and variables. Name space can be used by two ways in a program, either by the use of using statement at the beginning, like we did in above mentioned program or by using name of namespace as prefix before the identifier with scope resolution (::) operator. Thus with namespace std, std::cout <<”Hello World!”; becomes cout <<”Hello World!”; like in our previous example, much simpler right !! it is totally up to you, to define on which style suits you the most.
also, the keyword endl is used to denote the end of a line therefore, the next line of code will be printed on a new line.
It is also important to notice that you could combine instructions into a single one. Here is an example,

```
int main () {cout << "This my first C++ program, in one line.";cout <<"C++
    is fun"}
```

Listing 4: Compress C++ code

This code output the two (02) instructions on one (01) line, you can run the code and see the output.

3.1 Make your code more readable

In order to allow others and yourself to understand your code, it is recommended to add comments to your code. Now we are going to learn how to comment our program. We introduced the concept of concerning comments in the previous section but lets dive depper into it.
There exists two (02) types of comments :

- **Short comments**: as stated in the names they are short and can be written in one line of code. To write a short comment, you just need to start with // following by your comment.

```
// This is my first comment.
```

Listing 5: First short commment

- **Long comments**: if your comments are long enough and can not fit in one line, you can have a block comment. You just need to start and end your block comment with /*.

```
/* The following code is a little more complexe thus I will take my time
    and explain every single line of code, because a week from now I may
    not have forgotten.
*/
```

Listing 6: First short commment

Generally, we do not write too much in the comment section, just the necessary information, unless you have to.

- **Let’s comment our code**

```cpp
//============================================================
// Sample example of C++ programming language
//============================================================

#include <iostream> // include the iostream library

using namespace std;

/*
The role of the "main"
The main function gets called when the program is started

*/

int main()
{
   cout << "Hello world!" << endl; // Prints the message
   return 0; // Ends the program
}

//=============================================================
```

Listing 7: Comments in C++

After you run this code, nothing will change, the output will still be the same because comments are simply ignored by the compiler.

3.2 Summary

- The execution of code begins from the main() function. This function is mandatory. This is a valid C++ program that does nothing.
- The `cout` is used to display a message.
- You can comment your codes in two (02) ways `// Comment` or `/* Comment */`

4 Introduction to variables

So far, you have discovered how to create and compile your first programs in console mode. Right now these programs are very simple. They display messages on the screen...and nothing more. This is mainly due to the fact that your programs do not know how to interact with their users. This is what we will learn how to do in the next chapter. But before that we need to introduce an important concept: **variables**

4.1 what is a variable ?

The one and only thing you need to know is that a variable is a part of the memory that the computer lends us to put values into it. Imagine that the computer has in its entrails a large wardrobe that has thousands (billions!) of small drawers; these are places that we will be able to use to put our variables into.
In the case of a simple calculator, one can usually store only one number at a time. As you can imagine, in the case of a program, we will have to keep more than one thing at the same time. So you need a way to differentiate the variables to be able to access them afterwards. So each variable has a **name**. It is in other words the label that is stuck on the drawer.
The other thing that distinguishes the calculator from the computer is that we would like to be able to store a lot of different things, numbers, letters, sentences, pictures, etc. This is what we call the type of a variable. You can imagine that as the shape of the drawer. We do not use the same drawers to store bottles or books.

4.2 Variables naming conventions

Let's start with the question of variable names. In C++, there are few rules that govern the different names allowed or prohibited.

- Variable names are made up of letters, numbers and the underscore only;
- The first character must be a letter (upper or lower case);
- Spaces in the name is not allowed;

Here are few examples of valid variables: ageZero, first_name also, AGEZERO are allowed variable names. _ageZero in the other hand is not allowed.
To this is added an additional rule, valid for everything written in C++ and not only for variables. The language makes the difference between upper and lower case. In technical terms, it is said that C++ is case sensitive. Thus, myAge, myage, MYAGE and MyAge are all different variables.

Personally, I use a <<convention >>shared by many programmers. In all the big projects with thousands of programmers, there are very strict rules and sometimes difficult to follow. The ones I propose to you here allow to keep a good legibility and above all, they will allow you to understand all the examples in the rest of this course.

- Variable names start with a lower case;
- If the name is composed of several words, they are put together without space;
- Each new word (except the first) begins with a capital letter.

Let us look at this with examples. Let us take the case of a variable that is supposed to contain the age of the user of the program.

- $\boxed{\text{UserAge}}$: no, because the first letter is a capital letter;

- user_age: no, because the words are not seperated;
- ageuser: no, because the second word does not start with a capital letter;
- ageUser: ok

I strongly advise you to adopt the same convention. Making your code readable and easily understandable by other programmers is very important, and it does not just involve formatting.

4.3 Variables types (Data types)

We learned that a variable has a name and a type. We know how to name our variables, now let's see their different types. The computer likes to know what it has in its memory, so you have to indicate what type of element will contain the variable we would like to use. Is it a number, a word, a letter? It must be specified.

Data type	**What it contains**
bool	Data type with two possible values: true or false
char	Data type that holds one character (letter, number, etc.) of data
int	Numeric variables holding whole numbers
unsigned int	A positive or zero integer number.
double	Numeric variables holding numbers with decimal points
string	Data values that are made up of ordered sequences of characters

4.4 Syntax of variable declaration

In order to declare a variable in C++, the following syntaxt should be applied:

```
data_type variable1_name = value1, variable2_name = value2;
```

Listing 8: Variables syntax

As an example, we have:

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

int main()
{

    char myChar = 'A';            // character type
    int myInteger = 1;               // integer type
    float myFloat = 3.14159;      // floating point type
    double myDouble = 6e-4;        // double type (e is for exponential)

    return 0;
}
```

Listing 9: Variables declaration

4.4.1 Dealing with strings

When dealing with strings, the first thing to do is to add a small line at the beginning of your program. The compiler needs to be told that we want to use strings. Without this, it would not include the tools needed to manage them. Below is an example:

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

int main()
{
    string userName = "Albert Einstein";
    return 0;
}
```

Listing 10: String declaration

4.4.2 Dealing with multiple variables

If you have multiple variables of the same type to declare, you can do so on a single line by separating them with a comma (,), just like this:

```cpp
int a = 2, b = 4, c = -1;

string surname = "Albert", givenName= "Einstein";
```

Listing 11: single line declaration

4.5 Print message on the screen

As we discussed earlier, the key to print a value on the screen. Now let's combine that to what we just learn in order to print the value held by our variables. As a remember, the key was $\boxed{\text{cout}}$. let consider this example:

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

int main()
{
    int userAge = 16;
    cout << "Your age is : ";
    cout << userAge;
    return 0;
}
```

Listing 12: Print message on the screen

Output

```
Your age is : 16
```

Listing 13: Output message

4.6 Variables scope

All the variables have their area of functioning, and out of that boundary they do not hold their value, this boundary is called variable scope. For most of the cases its between the curly braces, in which variable is declared that a variable exists, not outside it. We will study the storage classes later, but as of now, we can broadly divide variables into two main types:
• Global Variables
• Local variables

4.6.1 Global variables

Global variables are those, which are once declared and can be used throughout the lifetime of the program. They must be declared outside the main() function. When declared, they can be assigned different values at different time in program lifetime. But even if they are declared and initialized at the same time outside the main() function, then they can also be assigned any value at any point in the program. Here is an example:

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

int x;                  // Global variable declared
int main()

{
    x=10;                      // Initialized once
    cout <<"first value of x = "<< x << endl;
    x=20;                      // Initialized again
    cout <<"Initialized again with value = "<< x << endl;
    return 0;
}
```

Listing 14: Global variable

In this code, the variable x was declared and not initialized (given an initial value) and was updated twice inside the main() function. You can run the code and see the output.

4.6.2 Local variables

Local variables are the variables which exist only between the curly braces, in which its declared. Outside that they are unavailable and leads to compile time error.

4.7 Summary

- A variable is an information stored in the memory.
- There exists various types of variables : $\boxed{\text{bool}}$, $\boxed{\text{char}}$, $\boxed{\text{int}}$ ...
- The value of a variable can be displayed at any time with : $\boxed{\text{cout}}$
- Variables can be declared either globally or locally: Global and Local variables

5 Dealing with user input

In the previous chapter, I explained how to display variables in the console. Now let's see how to do the opposite, which is to ask the user for information to store it in memory. let us look at an example.

```
 1 #include <iostream>
   using namespace std;
 3
   int main()
 5 {
       cout << "How old are you" << endl;
 7
       int ageUser = 0; //We initialized the varaible ageUser
 9
       cin >> ageUser; //We pass in the value of ageUser
11
       cout << "You are " << ageUser << " years old !" <<  endl; //We print
           the value
13
       return 0;
15 }
```

Listing 15: Saving a variable in the memory

Output

```
How old are you?
23
Your are 23 years old!
```

Listing 16: Saving a variable in the memory

The program displayed "how old are are you ?". So far so good, right ? and on line 8, the program ask for to store an integer in the memory then, at this moment, the program ask for user input and updates the variable ageUser initially declared to be holding the value 0. Finally, the program prints a message along with the user input.
When you display the value of a variable, the data comes out of the program, so you use an arrow going from the variable to $\boxed{\text{cout}}$. When we ask the user for information, it is the opposite, the value comes from $\boxed{\text{cin}}$ and goes into the variable.

5.1 Other variables

Obviously, what I presented to you also works with other types of variables. Let us look at it with an example.

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

int main()
{
    cout << "What is your Name ?" << endl;
    string userName = "no name"; //we create a variable with a list of
        characters
    cin >> userName; //We update the variable with the user input

    cout << "What is the value of PI ?" << endl;
    double piUser= -1; //we create a variable with a real number
    cin >> piUser; //we update it with the user input

    cout << "Your name is " << userName << " and you think PI is " <<
       piUser << "." << endl;

    return 0;
}
```

Listing 17: Other variables

I do not think I even need to explain it. But I would encourage you to test it to get a full understanding of what is going on.

5.2 Problem with space

Have you tested the previous code by putting your name and surname? Let's see what happens.

```
What is your name ?
Albert Einstein
What is the value of PI ?
Your name is Albert and you think PI is 0.
```

Listing 18: Problem with space example

It's a space problem. When you press the Enter key, the computer copies what the user wrote into the memory. But it stops at the first space or return to the line. When it comes to a number, there is no problem because there is no space in the numbers.
For Strings, a question arises. There may very well be a space in a string. And so the computer will cut in the wrong place, which is after the first word. In fact, it should be possible to retrieve the whole line rather than just the first word. In order to do so, we use the `getline()` function. Later, we will explain in details what is a function. So if we modify the the previous code by simply replacing the `cin >>userName;` by `getline()`

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

int main()
{
    cout << "What is your Name ?" << endl;
    string userName = "no name"; //we create a variable with a list of
       characters
    getline(cin, userName); //We update the variable with the entire user
       input

    cout << "What is the value of PI ?" << endl;
    double piUser= -1; //we create a variable with a real number
    cin >> piUser; //we update it with the user input

    cout << "Your name is " << userName << " and you think PI is " <<
       piUser << "." << endl;

    return 0;
}
```

Listing 19: getline example

```
What is your name ?
Albert Einstein
What is the value of PI ?
3.14
```

```
Your name is Albert and you think PI is 3.14.
```

Listing 20: getline output

5.3 Update a variable

Let's start by looking at how to change the content of a variable. We use the "=" symbol to make a value change. If I have a type `int` variable whose content I want to change, I write the name of my variable followed by the "=" symbol and finally the new value.

```
int myNumber = 0; //I create a variable and initialized it with value 0

myNumber = 5; //I update it with value 5

/*We can also do this in this way: */

int a = 4, b = 5; // declare two variables
a = b; // giving the value of b to a
```

Listing 21: update variables

5.4 Constant variables

I told you how to modify variables right, I hope you understood! Because we are going to do the opposite. I will show you how to declare non-modifiable variables (constants).
Think of a calculator that requires the constant π. This value never changes π will always be 3.14. There are also variables whose value never changes but whose value is not known in advance. Let's take the result of an operation in a calculator. Once the calculation is done, the result does not change. The variable that contains the result is therefore a constant.

5.4.1 Declare a constant

It's very simple, you declare a normal variable and add the keyword `const` between the type and the name. Also, it applies to all variables types.

```
int const daysInYear = 365;
string const password = "wAsTZsaswQ";
double const pi(3.14);
unsigned int const playerLifeTime = 100;
```

Listing 22: declare a constant

5.5 Operators

Operators are special type of functions, that take one or more arguments and produces a new value. For example : addition (+), substraction (-), multiplication (*) etc, are all operators. Operators are used to perform various operations on variables and constants.

5.5.1 Assignment Operator (=)

Operates '=' is used for assignment, it takes the right-hand side (called rvalue) and copy it into the left-hand side (called lvalue). Assignment operator is the only operator which can be overloaded but cannot be inherited. As an example, we have `x = 10;` This statement assigns the integer value 10 to the variable x.
Note: The assignment operation always takes place from right to left, and never the other way around. Another example: `a = b = c = 10` It assigns 5 to the all three variables: a, b and c; always from right-to-left.

5.5.2 Basic Arithmetic Operators (+, -, *, /, %)

Operator	Description
+	Addition
*	Multiplication
-	Subtraction
/	Division
%	Modulo

Note: Modulo operator returns remainder, for example 20 % 5 would return 0.

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

int main(){

  int num1 = 240;
  int num2 = 40;

  cout << "num1 + num2: " << (num1 + num2) << endl; //num1 + num2: 280
  cout << "num1 - num2: " << (num1 - num2) << endl; //num1 - num2: 200
  cout << "num1 * num2: "<< (num1 * num2) << endl; //num1 * num2: 9600
  cout << "num1 / num2: " << (num1 / num2) << endl; //num1 / num2: 6
  cout << "num1 % num2: " << (num1 % num2) << endl; //num1 % num2: 0

  return 0;
}
```

Listing 23: Example of Arithmetic Operators

5.5.3 Assignment Operators

Assignments operators in C++ are: =, +=, -=, *=, /=, %=
num2 = num1 would assign value of variable num1 to the variable.
num2+=num1 is equal to num2 = num2 + num1;
num2-=num1 is equal to num2 = num2 - num1;
num2/=num1 is equal to num2 = num2 / num1;
num2%=num1 is equal to num2 = num2 % num1;

by applying this to the previous example we have:

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

int main(){

 int num1 = 240;
 int num2 = 40;

 num2 = num1;
 cout << "= Output: " << num2 << endl; // = Output: 240
 num2 += num1;
 cout << "+= Output: " << num2 << endl; // += Output: 480
 num2 -= num1;
 cout << "-= Output: " << num2 << endl; // -= Output: 240
 num2 *= num1;
 cout << "*= Output: " << num2 << endl; // *= Output: 57600
 num2 /= num1;
```

```
  cout << "/= Output: " << num2 << endl; // /= Output: 240
  num2 %= num1;
  cout << "%= Output: " << num2 << endl; // %= Output: 0

  return 0;
}
```

Listing 24: Assignment Operators

5.5.4 Increment and decrement Operators

Here we are going to talk about the following operators: ++ and - -.
num++ is equivalent to num = num + 1;
num- - is equivalent to num = num - 1;

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

int main(){

  int num1 = 240;
  int num2 = 40;

  num1++; num2--;
  cout <<"num1++ is: " << num1 <<  endl; //num1++ is: 241
  cout <<"num2-- is: " << num2; //num2-- is: 39

  return 0;
}
```

Listing 25: auto increment/decrement

5.5.5 Logical Operators

Logical Operators are used with binary variables. They are mainly used in conditional statements and loops for evaluating a condition. We have &&,|| and ! Given two boolean variables b1 and b2.
• b1&&b2 will return true if both b1 and b2 are true else it would return false.
• b1||b2 will return false if both b1 and b2 are false else it would return true.
• !b1 would return the opposite of b1, that means it would be true if b1 is false and it would return false if b1 is true.

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

int main(){

   bool b1 = true;
   bool b2 = false;

   cout << "b1 && b2: " << (b1&&b2) << endl; //b1 && b2: 0
   cout << "b1 || b2: " << (b1||b2) << endl; //b1 || b2: 1
   cout << "!(b1 && b2): " << !(b1&&b2); //!(b1 && b2): 1

   return 0;
}
```

Listing 26: logic operators example

5.5.6 Relational operators

We have six relational operators in C++: ==, !=, >, <, <=, >=.

Data type	What it contains
==	returns true if both the left side and right side are equal
!=	returns true if left side is not equal to the right side of operator.
<	returns true if left side is less than right side.
>	returns true if left side is greater than right.
>=	returns true if left side is greater than or equal to right side.
<=	returns true if left side is less than or equal to right side

5.5.7 Bitwise Operators

The bitwise Operators: &, ||,<<, >>, Bitwise operator performs bit by bit processing.
num1 = 11; //equal to 00001011 and num2 = 22; // equal to 00010110
num1 & num2 compares corresponding bits of num1 and num2 and generates 1 if both bits are equal, else it returns 0. In our case it would return: 2 which is 00000010 because in the binary form of num1 and num2 only second last bits are matching.
num1 || num2 compares corresponding bits of num1 and num2 and generates 1 if either bit is 1, else it returns 0. In our case it would return 31 which is 00011111.
num1 () num2 compares corresponding bits of num1 and num2 and generates 1 if they are not equal, else it returns 0. In our example it would return 29 which is equivalent to 00011101.
num1 is a complement operator that just changes the bit from 0 to 1 and 1 to 0. In our example it would return -12 which is signed 8 bit equivalent to 11110100
num1 >>2 is left shift operator that moves the bits to the left, discards the farleft bit, and assigns the rightmost bit a value of 0. In our case output is 44 which is equivalent to 00101100.
Note: In the example below we are providing 2 at the right side of this shift operator that is the reason bits are moving two places to the left side. We can change this number and bits would be moved by the number of bits specified on the right side of the operator. Same applies to the right side operator.
num1 >>2 is right shift operator that moves the bits to the right, discards the far right bit, and assigns the leftmost bit a value of 0. In our case output is 2 which is equivalent to 00000010

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

int main(){

    int num1 = 11;  // 11 = 00001011
    int num2 = 22;  // 22 = 00010110
    int result = 0;

    result = num1 & num2;
    cout<<"num1 & num2: "<<result<<endl; //num1 & num2: 2

    result = num1 | num2;
    cout<<"num1 | num2: "<<result<<endl; //num1 | num2: 31

    result = num1 ^ num2;
    cout<<"num1 ^ num2: "<<result<<endl; //num1 ^ num2: 29

    result = ~num1;
    cout<<"~num1: "<<result<<endl; //~num1: -12

    result = num1 << 2;
    cout<<"num1 << 2: "<<result<<endl; //num1 << 2: 44

```

```cpp
    result = num1 >> 2;
    cout<<"num1 >> 2: "<<result; //num1 >> 2: 2

    return 0;
}
```

Listing 27: Bitwise operator example

5.5.8 Ternary Operator

This operator evaluates a boolean expression and assign the value based on the result.
here is the syntaxt $\boxed{\text{variable num1 = (expression) ? value if true : value if false}}$.
If the expression results true then the first value before the colon (:) is assigned to the variable num1 else the second value is assigned to the num1.

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

int main(){

  int num1, num2; num1 = 99;
  /* num1 is not equal to 10 that's why
   * the second value after colon is assigned
   * to the variable num2
   */
  num2 = (num1 == 10) ? 100: 200; //num2: 200

  cout << "num2: " << num2 << endl;
  /* num1 is equal to 99 that's why
   * the first value is assigned
   * to the variable num2
   */
  num2 = (num1 == 99) ? 100: 200;
  cout << "num2: "<<num2; //num2: 100

  return 0;
}
```

Listing 28: Ternary operator example

5.6 Summary

- To ask the user to enter information, we use $\boxed{\text{cin >>variable;}}$
- There is a difference between $\boxed{\text{cin >>}}$ and $\boxed{\text{cout <<}}$
- The function $\boxed{\text{getline()}}$ is used to get the entire user input (specially input with space)
- The computer is indeed a super calculator that performs multiple operations (arithmetic etc.)

6 Decision making

Programs must be able to make decisions. To achieve this, developers use so-called control structures. This name mainly hides in fact two elements that we will see in this chapter:
• Conditions : they allow you to write rules in the program like If this happens, then do this.
• Loops : they allow a series of instructions to be repeated several times.

6.1 if - condition

The if statement can be presented and use in different forms depending on the task we are trying to solve and the conditions to be tested. Here are the various forms that the if condition can take:
* *if statement*
* *nested if statement*
**if-else statement*
**if-else-if statement*

6.1.1 if statement

If statement consists a condition, followed by statement or a set of statements. Below is the basic structure of a simple if statement.

```
if(condition){
  Statement(s);
}
```

Listing 29: Basic if statement structure

The statements inside if parenthesis (usually referred as if body) gets executed only when the given condition is true. If the condition is false then the statements inside if body are completely ignored.

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

int main(){

  int num=70;

  if( num < 100 ){
     /* This cout statement will only execute,
      * if the above condition is true
      */
     cout<<"number is less than 100";
  }

  if(num > 100){
     /* This cout statement will only execute,
      * if the above condition is true
      */
     cout<<"number is greater than 100";
  }
  // Program Output: number is less than 100; because num=70 is less than
      100

  return 0;
}
```

Listing 30: if statement example

6.1.2 Nested if statement

An if statement inside another if statement is called nested if statement then it is called the nested if statement. Here is the basic structure of the nested if statement.

```
if(condition_1) {
   Statement1;

   if(condition_2) {
      Statement2;
   }
}
```

Listing 31: Nested if statement structure

Statement1 would execute if the condition_1 is true. Statement2 would only execute if both the conditions(condition_1 and condition_2) are true. let's see an example.

```
if(condition_1) {
   Statement1;

   if(condition_2) {
      Statement2;
   }
}
```

Listing 32: Nested if example

6.1.3 if...else statement

The general form of a simple if...else statement is,

```
if(expression)
{
    statement1;
}
else
{
    statement2;
}
```

Listing 33: if...else structure

If the 'expression' is true or returns true, then the 'statement1' will get executed, else 'statement1' will be skipped and 'statement2' will be executed. Here is an example.

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

int main(){

   int num=10;

   if( num < 5 ){
      //This would be executed if above condition is true
      cout << "num is less than 5";
   }
   else {
      //This would run if above condition is false
      cout<<"num is greater than or equal 5";
   }
```

```
    // Program Output: num is greater than or equal 50
    /* This is because  num = 10 is greater than 10
       So the if condition does not get to be executed.
   */

    return 0;
}
```
Listing 34: if...else example

6.1.4 if-else-if statement

if-else-if statement is used when we need to check multiple conditions. This is how it looks:

```
if(condition_1) {
   /*if condition_1 is true execute this*/
   statement;
}
else if(condition_2) {
   /* execute this if condition_1 is not met and
    * condition_2 is met
    */
   statement;
}
else if(condition_3) {
   /* execute this if condition_1 & condition_2 are
    * not met and condition_3 is met
    */
   statement;
}
.
.
.
else {
   /* if none of the condition is true
    * then these statements gets executed
    */
   statement(s);
}
```
Listing 35: if-else-if structure

Note: The most important point to note here is that in if-else-if, as soon as the condition is met, the corresponding set of statements get executed, rest gets ignored. If none of the condition is met then the statements inside "else" gets executed. Here is an example of if-else-if statement.

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

int main(){

   int num;

   cout<<"Enter an integer number between 1 & 99999: ";
   cin>>num;

   if(num <100 && num>=1) {
      cout<<"Its a two digit number";
   }
```

```
    else if(num <1000 && num>=100) {
       cout<<"Its a three digit number";
    }
    else if(num <10000 && num>=1000) {
       cout<<"Its a four digit number";
    }
    else if(num <100000 && num>=10000) {
       cout<<"Its a five digit number";
    }
    else {
       cout<<"number is not between 1 & 99999";
    }
    /* Program output :
     Enter an integer number between 1 & 99999: 8976
     Its a four digit number
    */
    return 0;
}
```

Listing 36: if-else-if example

6.2 while loop

In while loop, condition is evaluated first and if it returns true then the statements inside while loop execute, this happens repeatedly until the condition returns false. When condition returns false, the control comes out of loop and jumps to the next statement in the program after while loop. Here is the basic structure of a while loop.

```
variable initialization;

while (condition)
{
   statement;
   variable increment or decrement;
}
```

Listing 37: while loop structure

One important point is to increment or decrement statement inside while loop so that the loop variable gets changed on each iteration to stop the loop from running indefinitely.

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

int main(){

   int i = 1; // we initialize the variable

   /* The loop would continue to print
    * the value of i until the given condition
    * i<=6 returns false.
    */
   while(i <= 6){
      cout << "Value of variable i is:  "<< i << endl;
      i++; // we increment i until the condition is not true and exit the
         loop
   }

   return 0;
}
```

Listing 38: while loop example

6.3 do-while loop

In some situations it is necessary to execute body of the loop before testing the condition. Such situations can be handled with the help of do-while loop. do statement evaluates the body of the loop first and at the end, the condition is checked using while statement. General format of do-while loop is : $\boxed{\text{do statement while (condition);}}$. let's see an example:

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

int main(){

   int num = 1; // initialize variable

   do{
      cout << "Value of num: " << num << endl;
      num++;
   }while(num <= 6);

   /* Output:
    * Value of num: 1
    * Value of num: 2
    * Value of num: 3
    * Value of num: 4
    * Value of num: 5
    * Value of num: 6
    */

   return 0;
}
```

Listing 39: do-while loop example

6.4 for loop

for loop is used to execute a set of statement repeatedly until a particular condition is satisfied. The General format is : $\boxed{\text{for (initialization; condition; increment/decrement) statement;}}$. Here is an example.

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

int main(){

   for(int i = 1; i <= 6; i++){

      /* This statement would be executed
       * repeatedly until the condition
       * i<=6 returns false.
       */
      cout<<"Value of variable i is: "<<i<<endl;
   }

   /* Output:
    * Value of variable i is: 1
```

```
     * Value of variable i is: 2
     * Value of variable i is: 3
     * Value of variable i is: 4
     * Value of variable i is: 5
     * Value of variable i is: 6
     */

   return 0;
}
```

Listing 40: for loop example

6.5 Jumping out of a loop

Sometimes, while executing a loop, it becomes necessary to skip a part of the loop or to leave the loop as soon as certain condition becocmes true, that is jump out of loop.

6.5.1 Continue

It causes the control to go directly to the test-condition and then continue the loop process. On encountering continue, cursor leave the current cycle of loop, and starts with the next cycle.

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

int main(){

   for (int num=0; num<=6; num++) {

      /* This means that when the value of
       * num is equal to 4 this continue statement
       * would be encountered, which would make the
       * control to jump to the beginning of loop for
       * next iteration, skipping the current iteration
       */

      if (num == 4) {
          continue;
      }
      cout << num << " "; // Print result on a single line
   }

   return 0;
}
```

Listing 41: continue example

6.5.2 break

When break statement is encountered inside a loop, the loop is immediately exited and the program continues with the statement immediately following the loop. let's see an example

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

int main(){

   int var;
   for (var=20; var>=10; var --) {
      cout << "var: "<< var << endl;
      if (var == 18) {
         break;
      }
   }
   cout << "I'm out of the loop";

    /*
     * In this example we have a for loop
     * running from 20 to 10 but since we
     * have a break statement when the
     * variable is equal to 18, which
     * terminates the program.
     * Program Output is :
     * var:20
     * var:19
     * var:18
     * I'm out of the loop
     */

   return 0;
}
```

Listing 42: break example

6.6 switch

Switch case statement is used when we have multiple conditions and we need to perform different action based on the condition. When we have multiple conditions and we need to execute a block of statements when a particular condition is satisfied.In such case either we can use lengthy if..else-if statement or switch case. Here is the syntaxt of a switch case statement

```
switch (variable or an integer expression)
{
     case constant_1:
     //C++ code 1
     break;

     case constant_2:
     //C++ code 2
     break;
    .
    .
    .
     default:
     //C++ code

}
```

Listing 43: Switch syntaxt

Switch checks the value of the expression and verify if it is equal to constant_1, if the value happens to be the same, c++ code 1 gets executed until it reaches the the break statement sand terminate the program. In case the expression wa not equal to constant_1, the following condition gets tested and executed. In case none of them satistied the conditions, the default condition gets executed.

```
#include <iostream>
```

```
using namespace std;

int main(){

   int i=2;

   switch(i) {

      case 1:
        cout<<"Case1 "<<endl;
        break;

      case 2:
        cout<<"Case2 "<<endl;
        break;

      case 3:
        cout<<"Case3 "<<endl;
        break;

      case 4:
        cout<<"Case4 "<<endl;
        break;

      default:
        cout<<"Default "<<endl;
    }
    // Program Output: Case2
    return 0;
}
```

Listing 44: Switch example

In the above program, the integer i corresponds to the case 2, therefore the code block of that case gets executed.

6.7 Summary

- The conditions allow to test the value of the variables and to modify the behavior of the program accordingly.
- Loops allow you to repeat instructions many times.
- You can also use characters in switch case.

7 Functions

A function is a block of code which is used to perform a particular task, for example let's say you are writing a large C++ program and in that program you want to do a particular task several number of times, in order to do that you have to write few lines of code and you need to repeat these lines every time. In order to do that, you can write a function and call that function every time you want to perform that specific task. This would make you code simple, readable and reusable. Also, functions are used to provide modularity to a program. Creating an application using function makes it easier to understand, edit, check errors etc.

```
return-type function-name(parameter_1, parameter_2, ...)
{
    // function-body
}
```

Listing 45: Function structure

•**return-type:** defines what the function will return. It can be int, char etc. Also, there exist functions that can do not return anything, they are mentioned with a return-type: void.
• **Function Name:** is the name of the function, using the function name it is called.
• **Parameters:** are variables to hold values of arguments passed while function is called. A function may or may not contain parameter list.
• **Function body:** it contains your code statement are written. Let's look at a concrete example for functions. We are going to create a simple function that adds to numbers.

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

int sum(int num1, int num2) // The sum function that add 2 values
{
    int num3;
    num3 = num1 + num2;
    return num3;
}

int main()
{
    int result;
    // calling the function with the function name 'sum'
    result = sum (2, 3);
    cout << " The result is " << result;

   /* Output: The Program
    *  add 2+3 and return the result: 5
    */

    return 0;
}
```

Listing 46: Function example

As we can see from the above code, the function sum() was created and called before the main function, and there is a reason for that, we will get to that later. For the meantime less understand the code. The sum() function takes two (02) arguments which are num1 and num2. Inside the the body of the sum() function we notice an arithmetic operation that is the addition of num1 and num2 and the result being kept in another variable, num3 and finally the function return the final result of the operation kept in num3.
The main() function simply calls the the sum() function and pass it the two arguments (2, 3) which represents num1 and num2 for the sum() function and adds the result into the variable result.

7.1 Passing parameters to a function

Functions are called by their names as we saw in the previous example. When the function does not have an argument, it can be called directly using its name. On the contrary, for functions with arguments we have two ways to pass them: Pass by Value and Pass by reference.

7.1.1 Pass by Value

In this calling technique we pass the values of arguments which are stored or copied into the formal parameters of functions. Hence, the original values are unchanged only the parameters inside function changes.

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

void calculator(int x);

int main()
{
    int x = 10;
    calculator(x);
    printf("%d", x);
    return 0;
}

void calculator(int x)
{
    x = x + 10 ;
}
```

Listing 47: Pass by value example

In this case the actual variable x is not changed, because we pass argument by value, hence a copy of x is passed, which is changed, and that copied value is destroyed as the function ends(goes out of scope). So the variable x inside main() still has a value 10.

7.1.2 Pass by reference

Here, we pass the address of the variable as arguments. In this case the formal parameter can be taken as a reference, in both the case they will change the values of the original variable.

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

void calculator(int *p);

int main()
{
    int x = 10;
    calculator(&x);      // passing address of x as argument
    printf("%d", x);
    return 0;
}

void calculator(int *p)
{
    *p = *p + 10;
}
```

```
                         // The Program output: 20
```
Listing 48: Pass by reference example

Do not worry if you don not understand what this code does, we will learn pointer later on.

7.2 Default arguments

When we mention a default value for a parameter while declaring the function, it is said to be as default argument. In this case, even if we make a call to the function without passing any value for that parameter, the function will take the default value specified. Here is an example:

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

int sum(int a, int b = 10, int c = 2);

int main(){
   /* In this case a value is passed as
    * 1 and b and c values are taken from
    * default arguments.
    */
   cout << sum(1) << endl;

   /* In this case a value is passed as
    * 1 and b value as 2, value of c values is
    * taken from default arguments.
    */
   cout << sum(1, 2) << endl;

   /* In this case all the three values are
    * passed during function call, hence no
    * default arguments have been used.
    */
   cout<< sum(1, 2, 3) << endl;

   return 0;
}
int sum(int a, int b, int c){
   int z;
   z = a + b + c;


   /* Program Output:
    * 13
    * 5
    * 6
    */

   return z;
}
```
Listing 49: Default argument

7.2.1 Rules of default arguments

In order to use default arguments, there exist a rule that must be followed otherwise, the compiler will display an error message. so let's see some valid and invalid default arguments declaration.

1–: Only the last argument must be given default value. You cannot have a default argument followed by non-default argument.

```cpp
int sum(int x, int y);
int sum(int x, int y = 0);
int sum(int x = 0, int y);  // This is Incorrect
```

Listing 50: Default argument rule 1

2–: If you default an argument, then you will have to default all the subsequent arguments after that.

```cpp
int sum(int x, int y = 0);
int sum(int x, int y = 0, int z);  // This is incorrect
int sum(int x, int y = 5, int z=3);  // Correct
```

Listing 51: Default argument rule 2

Note: You can give any value a default value to argument, compatible with its datatype.

7.3 Function overloading

Function overloading is usually used to enhance the readability of the program. If you have to perform one single operation but with different number or types of arguments, then you can simply overload the function. Overloading allows you to use the same name for different functions, to perform, either same or different functions.

7.3.1 Overloading with different number of Arguments

In this type of function overloading we define two functions with same names but different number of parameters of the same type. Here is an example

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

// first definition
int sum (int x, int y)
{
    cout << x + y;
}

// second overloaded defintion
int sum(int x, int y, int z)
{
    cout << x + y + z;
}

int main()
{
    // sum() with 2 parameter will be called
    sum (10, 20);

    //sum() with 3 parameter will be called
    sum(10, 20, 30);

   /* Program Output:
    * 30
    * 60
    */
    return 0;
}
```

Listing 52: Overloading with different number of arguments

Here sum() function is said to overloaded, as it has two defintion, one which accepts two arguments and another which accepts three arguments. Which sum() function will be called, depends on the number of arguments.

7.3.2 Overloading with different return type

In this type of overloading we define two or more functions with same name and same number of parameters, but the return type of is different. For example in this program, we have two sum() function, first one gets two integer arguments and second one gets two double arguments.

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

// first definition which returns an integer
int sum(int x, int y)
{
    cout << x + y;
}

// second overloaded defintion which returns a double
double sum(double x, double y)
{
    cout << x + y;
}

int main()
{
    sum(5, 2);
    sum(5.5, 2.5);

    /* Program Output:
    * 7
    * 8.0
    */

    return 0;
}
```

Listing 53: Overloading with different return type

7.4 Summary

- A function is a portion of code that contains instructions and has a specific role.
- All programs have at least one function: the main() function. This is the one that runs when the program starts.
- The same function can be called several times during. the execution of a program.
- Arguments can be passed by two ways : pass by reference and pass by reference.
- Default arguments are called when a parameter is not initialized
- Default arguments must be declared following a rule.
- Function overload allows you to use the same function to perform multiple task.

8 Arrays

An array is a collection of similar items stored in contiguous memory locations. In programming, sometimes a simple variable is not enough to hold all the data. We are going to see how an array is created and initialized.

```cpp
// Method 1 :
int arr[5];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
arr[3] = 40;
arr[4] = 50;

// Method 2 :
int arr[] = {10, 20, 30, 40, 50};

// Method 3 :
int arr[5] = {10, 20, 30, 40, 50};
```

Listing 54: Array declaration

As you can see in the above code, an array can be declared based on differents methods. Each entry in the array can be specified starting from index 0 as shown in method 1. Also, an array can be declared without spacifying the size, like in method 2 and finally the contrary can also be done, like in method 3.

8.1 Accessing elements in an array

Array index starts with 0, which means the first array element is at index 0, second is at index 1 and so on. We can use this information to display the array elements.

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

int main(){

   int arr[] = {10, 25, 39, 40, 54};

   cout << arr[0] << endl;
   cout << arr[1] << endl;
   cout << arr[2] << endl;
   cout << arr[3] << endl;
   cout << arr[4] << endl;

  /* Output:
   * 10
   * 25
   * 39
   * 40
   * 54
   */

   return 0;
}
```

Listing 55: Accessing Array elements

Although this helps us accessing the array elements, but it is not efficent, if you have thousands of elements this method falls short. An efficent way will be to use a loop. We introduced loops earlier, with a loop this is how we could easily access the array elements.

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

int main(){

   int arr[] = {10, 25, 39, 40, 54};
   int n=0;

   while(n <= 4){

      cout << arr[n] << endl;
      n++;
   }

   /* Output:
   * 10
   * 25
   * 39
   * 40
   * 54
   */

   return 0;
}
```

Listing 56: Loop through the array elements

8.2 Multidimensional Arrays

Multidimensional arrays are also known as array of arrays. The data in multidimensional array is stored in a tabular form.

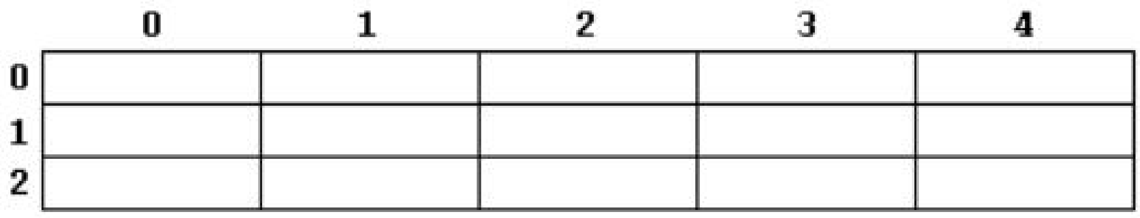

Figure 3: Multidimensional array.

A two-dimensional array for instance looks like this: $\boxed{\text{int arr[2][3]}}$ this array has 2*3 = 6 elements. similary, a three-dimensional array : $\boxed{\text{int arr[2][3][2]}}$ this array has 2*3*2 = 12 elements.

8.2.1 Declare and initialize a two-dimensional array

Consider the following two-dimensional array $\boxed{\text{int myArray[2][3];}}$ we can initialize this following the normal procedure like this: $\boxed{\text{int myArray[2][3] = \{15, 18 ,12 ,22 ,80 ,50\}}}$; another method to initialize will be: int $\boxed{\text{myArray[2][3] = \{\{15, 18 ,12\}, \{22 ,80 ,50\}\}}}$;. In this array the elements are arranged following this structure:
arr[0][0] – first element
arr[0][1] – second element
arr[0][2] – third element
arr[1][0] – fourth element

arr[1][1] – fifth element
arr[1][2] – sixth element
Here is how this code looks like :

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

int main(){

   int arr[2][3] = {{15, 18, 12}, {22, 80, 50}};

   for(int i = 0; i < 2; i++){
      for(int j = 0; j < 3; j++){
        cout << "arr["<< i << "][" << j << "]: " << arr[i][j] << endl;
      }
   }
   /* Output:
   * 15
   * 18
   * 12
   * 22
   * 80
   * 50
   */
   return 0;
}
```

Listing 57: two-dimensional array example

8.2.2 Passing array to a function

You can pass array as an argument to a function just like you pass variables as arguments. In order to pass array to the function you just need to mention the array name during function: $\boxed{\text{function\_name(array\_name);}}$

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

/* This function adds the corresponding
 * elements of both the arrays and
 * displays it.
 */

void sum(int arr1[], int arr2[]){

   int temp[5];

   for(int i = 0; i < 5; i++){
      temp[i] = arr1[i] + arr2[i];
      cout << temp[i] << endl;
   }
}
int main(){

   int a[5] = {10, 20, 30, 40 ,50};
   int b[5] = {1, 2, 3, 4, 5};

   //Passing arrays to function
   sum(a, b);
```

```
    /* Output:
    * 15
    * 18
    * 12
    * 22
    * 80
    * 50
    */
    return 0;
}
```

Listing 58: Array as an argument

In this example, we are passing two arrays a & b to the function sum(). This function adds the corresponding elements of both the arrays and display them.

8.3 Summary

- Array is a dynamic way of storing and accessing elements.
- Arrays can be declared following different structures.
- There exist multidimensional arrays

9 Pointers

Pointer is a variable in C++ that holds the address of another variable. They have data type just like variables, for example an integer type pointer can hold the address of an integer variable and an character type pointer can hold the address of char variable. The basic syntaxt of pointer is : $\boxed{\text{data\_type *pointer\_name;}}$. To declare a pointer, we can proceed in this way: $\boxed{\text{int *p, var}}$, This pointer p can hold the address of an integer variable, here p is a pointer and var is just a simple integer variable. To assign the address of variable to pointer we use ampersand symbol (&). just like this $\boxed{\text{p = \&var;}}$ this is how you assign the address of another variable to the pointer Lets take a simple example to understand what we discussed above.

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

int main(){

   //Pointer declaration
   int *p, var=23;

   //Assignment
   p = &var;

   cout << "Address of var: " << &var <<endl;
   cout<< "Address of var: " << p << endl;
   cout << "Address of p: " << &p << endl;
   cout << "Value of var: " << *p;

   /* Output:
   * Address of var: 0x7fff5dfffc0c
   * Address of var: 0x7fff5dfffc0c
   * Address of p: 0x7fff5dfffc10
   * Value of var: 23
   */

   return 0;
}
```

Listing 59: Pointer example

9.1 Pointer and arrays

While handling arrays with pointers you need to take care few things. First and very important point to note regarding arrays is that the array name alone represents the base address of array so while assigning the address of array to pointer don't use ampersand sign(&), here is the correct form $\boxed{\text{p = arr;}}$

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

int main(){

   //Pointer declaration
   int *p;

   //Array declaration
   int arr[]={1, 2, 3, 4, 5, 6};

   //Assignment
```

```
    p = arr;

    // Loop through array
    for(int i = 0; i < 6; i++){
      cout << *p << endl;
      //++ moves the pointer to next int position
      p++;
    }

    /* Output:
    * 1
    * 2
    * 3
    * 4
    * 5
    * 6
    */
    return 0;
}
```

Listing 60: Pointer and arrays example

9.2 Pointer's address and value increment

When we are accessing the value of a variable through pointer, sometimes we just need to increment or decrement the value of variable though it or we may need to move the pointer to next int position. In the above code we incremented using one way here we are discussing few more cases.

```
// Pointer moves to the next int position (as if it was an array)
p++;
// Pointer moves to the next int position (as if it was an array)
++p;

/* All the following three cases are same they increment the value
 * of variable that the pointer p points to.
 */
++*p;
++(*p);
++*(p)
```

Listing 61: Pointer increment

9.3 Summary

- Pointers can be used to to point to various elements.
- Pointers can be pass as an argument to a function.
- Pointers can be combine with arrays.

10 Introduction to OOP(Object Oriented Programming)

Object oriented programming is a way of solving complex problems by breaking them into smaller problems using objects. Before Object Oriented Programming (commonly referred as OOP), programs were written in procedural language, they were nothing but a long list of instructions. On the other hand, the OOP is all about creating objects that can interact with each other, this makes it easier to develop programs in OOP as we can understand the relationship between them.
In Object oriented programming we write programs using classes and objects utilising features of OOPs such as **abstraction, encapsulation, inheritance** and **polymorphism.**

10.1 Class and Objects

A class is like a blueprint of data member and functions and object is an instance of class. For example, lets say we have a class Car which has data members (variables) such as speed, weight, price and functions such as gearChange(), slowDown(), brake() etc. Now lets say I create a object of this class named FordFigo which uses these data members and functions and give them its own values. Similarly we can create as many objects as we want using the blueprint(class).

```
//Class name is Car
class Car
{
    //Data members
    char name[20];
    int speed;
    int weight;

public:
    //Functions
    void brake(){
    }
    void slowDown(){
    }
};

int main()
{
   //ford is an object
   Car ford;
}
```

Listing 62: Class example

Now that we now how to create a class and object we also need to know about access modifiers. C++ has three new keywords introduced, namely public, private and protected

10.1.1 Public Access Modifier

Public, means all the class members declared under public will be available to everyone. The data members and member functions declared public can be accessed by other classes too.

```
class PublicAccess
{
    // public access modifier
    public:
    int x;             // Data Member Declaration
    void display();    // Member Function decaration
}
```

Listing 63: Public Access example

10.1.2 Private Access Modifier

Private keyword, means that no one can access the class members declared private, outside that class. If someone tries to access the private members of a class, they will get a compile time error. By default class variables and member functions are private.

```cpp
class PrivateAccess
{
    // private access modifier
    private:
    int x;              // Data Member Declaration
    void display();     // Member Function decaration
}
```

Listing 64: Private Access example

10.1.3 Protected access modifier

Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class. But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A. We will learn about inheritance later.)

```cpp
class ProtectedAccess
{
    // protected access modifier
    protected:
    int x;              // Data Member Declaration
    void display();     // Member Function decaration
}
```

Listing 65: Protected Access example

10.2 Abstraction

Abstraction is a process of hiding irrelevant details from user. For example, When you send an sms you just type the message, select the contact and click send, the phone shows you that the message has been sent, what actually happens in background when you click send is hidden from you as it is not relevant to you.

10.3 Encapsulation

Encapsulation is a process of combining data and function into a single unit like capsule. This is to avoid the access of private data members from outside the class. To achieve encapsulation, we make all data members of class private and create public functions, using them we can get the values from these data members or set the value to these data members.

10.4 Inheritance

Inheritance is a feature using which an object of child class acquires the properties of parent class.

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

class ParentClass {

  //data member
  public:
    int var1 =100;
};
class ChildClass: public ParentClass {
  public:
  int var2 = 500;
};
int main(void) {

  ChildClass obj;

}
```

Listing 66: Protected Access example

Now this object obj can use the properties (such as variable var1) of ParentClass.

10.5 Polymorphism

Function overloading and Operator overloading are examples of polymorphism. Polymorphism is a feature in which an object behaves differently in different situation. In function overloading we can have more than one function with same name but numbers, type or sequence of arguments.

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

class Sum {

  public:

    int add(int num1,int num2){
      return num1 + num2;
    }

    int add(int num1, int num2, int num3){
      return num1 + num2 + num3;
    }
};
int main(void) {

   //Object of class Sum
   Sum obj;

   //This will call the second add function
   cout << obj.add(10, 20, 30) << endl;

   //This will call the first add function
   cout << obj.add(11, 22) << endl;

  // Output :
  // 60
  // 30
   return 0;
}
```

Listing 67: Polymorphism example

10.6 Summary

- OOP allows complex problems to be broken into chunks and process sequentially.

- abstraction, encapsulation, inheritance and Polymorphism are the beasic feature of OOP.

11 Conclusions

Yaaay !!! first I would like to congratulate you for sticking with me so far, I know it has not been an easy journey but you did, together we covered the basics you need to get started with C++ programming and introduced some advance concept like OOP. You can call yourself a C++ wizard. In this tutorial I tried as much as possible to provide an example to every single concept along with the explanation to let you understand in a more deeper way. Try as much as possible to write the code by yourself and not just copy and paste it in your IDE. In the next tutorial we are going to focus on big projects we can do together. We will break it into chunks and walk through the process together.
Thanks for you ! For any questions, comments and feedback feel free to contact me, you can find me at: Armand21@hotmail.fr