

# C++ Tutorials for Beginners

Kouassi Franck Armand Prince

09.05.2021

---

## Contents

<b>1</b>	<b>Getting Started</b>	<b>4</b>
1.1	Understand the Computer Language . . . . .	4
1.2	C++ against other programming languages . . . . .	4
1.3	Summary of C++ . . . . .	5
1.4	Summary . . . . .	6
<b>2</b>	<b>Environment setup</b>	<b>7</b>
<b>3</b>	<b>Your first C++ code</b>	<b>7</b>
3.1	Make your code more readable . . . . .	8
3.2	Summary . . . . .	9
<b>4</b>	<b>Introduction to variables in C++</b>	<b>9</b>
4.1	what is a variable ? . . . . .	9
4.2	Variables Naming conventions . . . . .	10
4.3	Variables types (Data types) . . . . .	10
4.4	Syntax of variable declaration . . . . .	11
4.4.1	Dealing with strings . . . . .	11
4.4.2	Dealing with multiple variables . . . . .	11
4.5	Print message on the screen . . . . .	11
4.6	Scope of Variables . . . . .	12
4.6.1	Global variables . . . . .	12
4.6.2	Local variables . . . . .	12
4.7	Summary . . . . .	13
<b>5</b>	<b>How to handle user input</b>	<b>13</b>
5.0.1	Other variables . . . . .	13
5.1	Problem with space . . . . .	14
5.2	Update a variable . . . . .	15
5.3	Constants . . . . .	15
5.3.1	declare a constant . . . . .	15
5.4	Operators in C++ . . . . .	15
5.4.1	Assignment Operator (=) . . . . .	16
5.4.2	Basic Arithmetic Operators (+, -, *, /, %) . . . . .	16
5.4.3	Assignment Operators . . . . .	16
5.4.4	Auto-increment and Auto-decrement Operators . . . . .	17
5.4.5	Logical Operators . . . . .	17
5.4.6	Relational operators . . . . .	18
5.4.7	Bitwise Operators . . . . .	18
5.4.8	Ternary Operator . . . . .	19
5.5	Summary . . . . .	19

## List of Figures

1	Compiling Process. . . . .	4
2	Programming Language by level. . . . .	5

## List of codes

1	Sample example of C++ programming language . . . . .	6
2	First C++ code . . . . .	7
3	First C++ code Output . . . . .	7
4	Compress C++ code . . . . .	8
5	First short comment . . . . .	8
6	First short comment . . . . .	8
7	Comments in C++ . . . . .	9
8	Variables Syntaxt . . . . .	11
9	Variables declaration . . . . .	11
10	String declaration . . . . .	11
11	single line declaration . . . . .	11
12	Print message on the screen . . . . .	12
13	message output . . . . .	12
14	Global variable . . . . .	12
15	Saving a variable in the memory . . . . .	13
16	Saving a variable in the memory . . . . .	13
17	Other variables . . . . .	13
18	Problem with space example . . . . .	14
19	getline example . . . . .	14
20	getline output . . . . .	15
	codes/assignVar.c++ . . . . .	15
21	declare a constant . . . . .	15
22	Example of Arithmetic Operators . . . . .	16
	codes/assignment.c++ . . . . .	16
23	auto increment/decrement . . . . .	17
24	logic operators example . . . . .	17
25	Bitwise operator example . . . . .	18
26	Ternary operator example . . . . .	19

# 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 have a prior programming background knowledge. All you have to do is to follow along with me 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 efficiently 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.

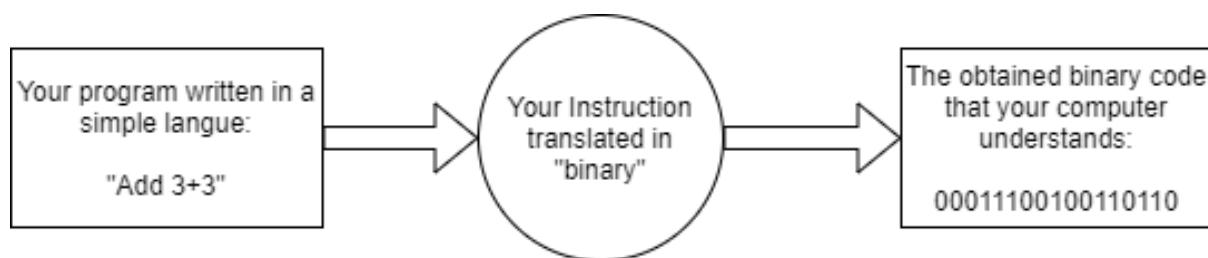


Figure 1: Compiling Process.

## 1.2 C++ against other programming languages

Before we start talking about why C++ represents a powerful language despite its age, let's discuss 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 languages, 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 language. This explains why C++ represents a good choice for debutant programmers. You are not alone, a lot of resources are available 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 far from binary language and really to humans language, it allows to easily understand and translate instructions contrary to **low level** which is 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 particularly **fast** language. Infact, if most games are

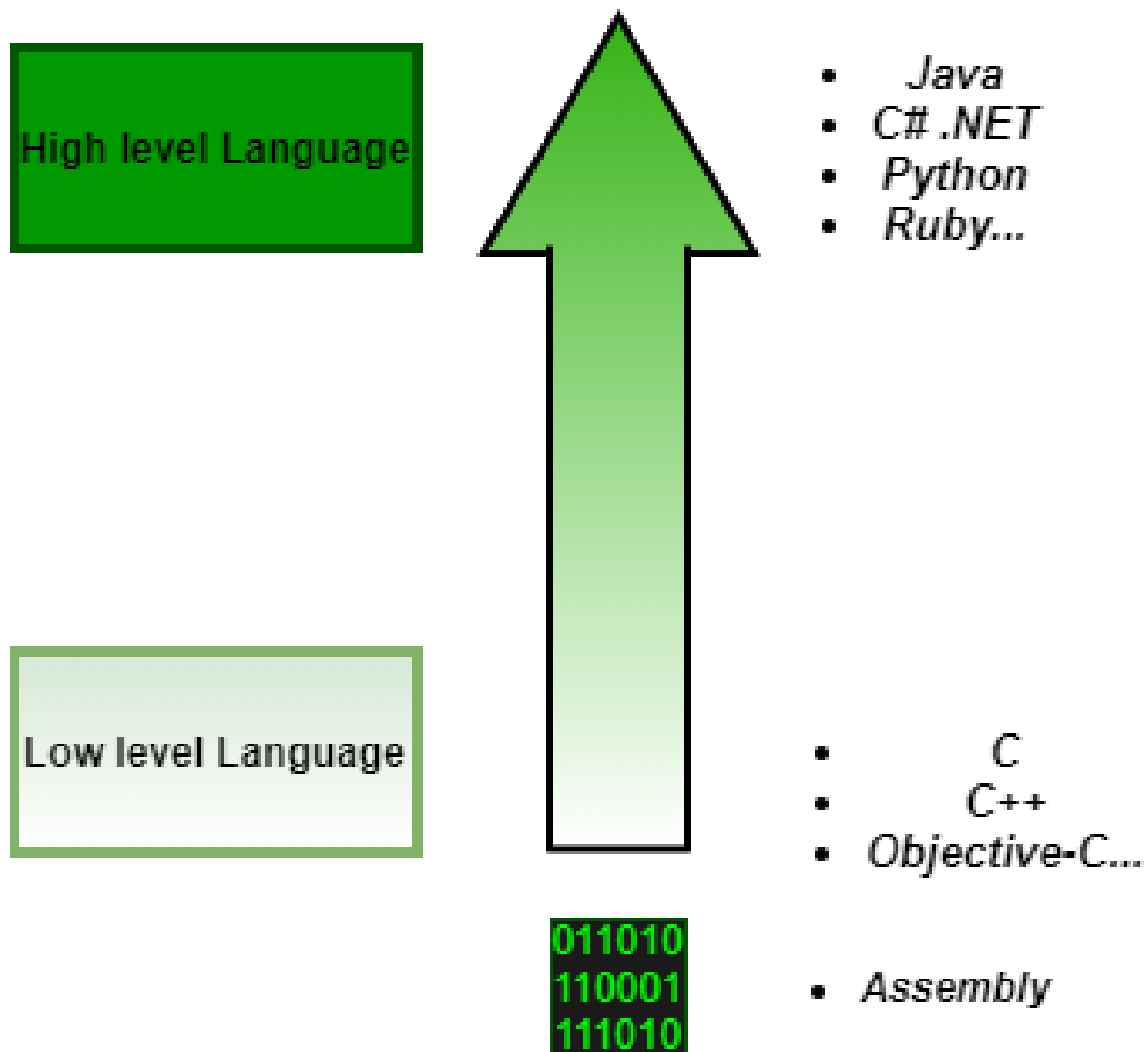


Figure 2: Programming Language by level.

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 its 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

```

1 //=====
2 // Sample example of C++ programming language
3 //=====
4
5 #include <iostream>
6
7 using namespace std;
8
9 int main()
10 {
11     cout << "Hello world!" << endl;
12     return 0;
13 }
14 //=====

```

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 efficiently 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 execution 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 recommended because as your code gets bigger and bigger, you might 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 separately 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 referred to as an **IDE** (Integrated Development Environment). There are of numerous types. In this tutorial we are not going to discuss their similarities. I personally recommend 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 C++ 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.

```
2 // Your First C++ Program
4 #include <iostream>
6 int main() {
8     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 begins 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. for example,

```
1 || 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 02 lines, 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 the in the previous section but lets dive deeper 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.

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

Listing 5: First short comment

- **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 `/*`.

```
1 || /* 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  
3 || not have forgotten.  
   || */
```

Listing 6: First short comment

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

- **Let's comment our code**



```

2 //=====
// Sample example of C++ programming language
//=====
4
6 #include <iostream> // include the iostream library
8
10 using namespace std;
12
14 /*
The role of the "main"
The main function gets called when the program is started
*/
16 int main()
18 {
    cout << "Hello world!" << endl; // Prints the message
    return 0; // Ends the program
20 }
//=====

```

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 in C++

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 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 a 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. 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.

- `UserAge`: no, because the first letter is a capital letter;
- `user_age`: no, because the words are not separated;
- `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 doesn't 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
<code>bool</code>	data type with two possible values: true or false
<code>char</code>	data type that holds one character (letter, number, etc.) of data
<code>int</code>	numeric variables holding whole numbers
<code>unsigned int</code>	A positive or zero integer number.
<code>double</code>	numeric variables holding numbers with decimal points
<code>string</code>	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 syntax should be applied:

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

Listing 8: Variables Syntax

As an example, we have:

```
1 || #include <iostream>
   || using namespace std;
3 ||
   || int main()
5 || {
   ||
   ||     char myChar = 'A';           // character type
   ||     int myInteger = 1;           // integer type
   ||     float myFloat = 3.14159;     // floating point type
   ||     double nyDouble = 6e-4;     // double type (e is for exponential)
11 ||
   ||     return 0;
13 || }
```

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:

```
1 || #include <iostream>
   || #include <string>
3 || using namespace std;
   ||
5 || int main()
   || {
7 ||     string userName = "Albert Einstein";
   ||     return 0;
9 || }
```

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:

```
1 || int a = 2, b = 4, c = -1;
3 || 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 `cout`. let consider this example:

```

1  #include <iostream>
   using namespace std;
3
   int main()
5  {
       int userAge = 16;
7      cout << "Your age is : ";
       cout << userAge;
9      return 0;
   }

```

Listing 12: Print message on the screen

### Output

```

|| Your age is : 16

```

Listing 13: message output

## 4.6 Scope of Variables

All the variables have their area of functioning, and out of that boundary they don't hold their value, this boundary is called scope of the variable. 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. If only 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 also they can be assigned any value at any point in the program. Here is an example:

```

1  #include <iostream>
   using namespace std;
3
   int x;                      // Global variable declared
5  int main()
7  {
       x=10;                    // Initialized once
9      cout <<"first value of x = "<< x << endl;
       x=20;                    // Initialized again
11     cout <<"Initialized again with value = "<< x << endl;
       return 0;
13 }

```

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 : `bool`, `char`, `int` ...
- The value of a variable can be displayed at any time with : `cout`
- There exists two types of variables : Global and Local variables

## 5 How to handle 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's look at an example.

```
1 #include <iostream>
2 using namespace std;
3
4 int main()
5 {
6     cout << "How old are you" << endl;
7
8     int ageUser = 0; //We initialized the variable ageUser
9
10    cin >> ageUser; //We pass in the value of ageUser
11
12    cout << "You are " << ageUser << " years old !" << endl; //We print
13        the value
14
15    return 0;
16 }
```

Listing 15: Saving a variable in the memory

### Output

```
1 How old are you?
2 23
3 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 `cout`. When we ask the user for information, it is the opposite, the value comes from `cin` and goes into the variable.

### 5.0.1 Other variables

Obviously, what I presented to you also works with other types of variables. Let's look at that with a small example.

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
```

```

5 int main()
6 {
7     cout << "What is your Name ?" << endl;
8     string userName = "no name"; //we create a variable with a list of
9     characters
10    cin >> userName; //We update the variable with the user input
11
12    cout << "What is the value of PI ?" << endl;
13    double piUser= -1; //we create a variable with a real number
14    cin >> piUser; //we update it with the user input
15
16    cout << "Your name is " << userName << " and you think PI is " <<
17        piUser << "." << endl;
18
19    return 0;
20 }

```

Listing 17: Other variables

I don't think I even need to explain it. But I would encourage you to test it to get a full understanding of what's going on.

## 5.1 Problem with space

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

```

1 What is your name ?
2 Albert Einstein
3 What is the value of PI ?
4 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()`

```

1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 int main()
6 {
7     cout << "What is your Name ?" << endl;
8     string userName = "no name"; //we create a variable with a list of
9     characters
10    getline(cin, userName); //We update the variable with the entire user
11    input
12
13    cout << "What is the value of PI ?" << endl;
14    double piUser= -1; //we create a variable with a real number
15    cin >> piUser; //we update it with the user input
16
17    cout << "Your name is " << userName << " and you think PI is " <<
18        piUser << "." << endl;
19
20    return 0;
21 }

```

```

16 |         return 0;
18 |     }

```

Listing 19: getline example

```

2 | What is your name ?
  | Albert Einstein
  | What is the value of PI ?
4 | 3.14
  | Your name is Albert and you think PI is 3.14.

```

Listing 20: getline output

## 5.2 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.

```

1 | int myNumber = 0; //I create a variable and initialized it with value 0
3 | myNumber = 5; //I update it with value 5
5 | /*We can also do this in this way: */
7 | int a = 4, b = 5; // declare two variables
  | a = b; // giving the value of b to a

```

## 5.3 Constants

I told you how to modify variables. I hope you understood! Because we're going to do the opposite. I will show you how to declare non-modifiable variables (constants).

Pensez à une calculatrice, qui aura besoin de la constante  $\pi$ . This value never changes  $\pi$  will always 3.14.

But that's not all. 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.

So let's see how to declare such a variable.

### 5.3.1 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.

```

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

```

Listing 21: declare a constant

## 5.4 Operators in C++

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

### 5.4.1 Assignment Operator (=)

Operator '=' 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 10 to the all three variables: a, b and c; always from right-to-left.

### 5.4.2 Basic Arithmetic Operators (+, -, \*, /, %)

Operator	Description
+	addition
*	Multiplication
/	Division
%	Modulo

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

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5
6     int num1 = 240;
7     int num2 = 40;
8
9     cout << "num1 + num2: " << (num1 + num2) << endl; //num1 + num2: 280
10    cout << "num1 - num2: " << (num1 - num2) << endl; //num1 - num2: 200
11    cout << "num1 * num2: " << (num1 * num2) << endl; //num1 * num2: 9600
12    cout << "num1 / num2: " << (num1 / num2) << endl; //num1 / num2: 6
13    cout << "num1 % num2: " << (num1 % num2) << endl; //num1 % num2: 0
14
15    return 0;
16 }
```

Listing 22: Example of Arithmetic Operators

### 5.4.3 Assignment Operators

Assignment 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:

```
1 #include <iostream>
2 using namespace std;
3
4 int main(){
5
6     int num1 = 240;
7     int num2 = 40;
```



```

8   num2 = num1;
10  cout << "= Output: " << num2 << endl; // = Output: 240
    num2 += num1;
12  cout << "+= Output: " << num2 << endl; // += Output: 480
    num2 -= num1;
14  cout << "-= Output: " << num2 << endl; // -= Output: 240
    num2 *= num1;
16  cout << "*= Output: " << num2 << endl; // *= Output: 57600
    num2 /= num1;
18  cout << "/= Output: " << num2 << endl; // /= Output: 240
    num2 %= num1;
20  cout << "%= Output: " << num2 << endl; // %= Output: 0
22  return 0;
    }

```

#### 5.4.4 Auto-increment and Auto-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;

```

1  #include <iostream>
   using namespace std;
3
   int main(){
5
       int num1 = 240;
       int num2 = 40;
7
       num1++; num2--;
       cout << "num1++ is: " << num1 << endl; //num1++ is: 241
11      cout << "num2-- is: " << num2; //num2-- is: 39
13
       return 0;
   }

```

Listing 23: auto increment/decrement

#### 5.4.5 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.

```

1  #include <iostream>
2  using namespace std;
4
   int main(){
6
       bool b1 = true;
       bool b2 = false;
8

```

```

10 cout << "b1 && b2: " << (b1&&b2) << endl; //b1 && b2: 0
11 cout << "b1 || b2: " << (b1||b2) << endl; //b1 || b2: 1
12 cout << "!(b1 && b2): " << !(b1&&b2); //!(b1 && b2): 1
13
14 return 0;
}

```

Listing 24: logic operators example

## 5.4.6 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

## 5.4.7 Bitwise Operators

There are six 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>
2 using namespace std;
4 int main(){
6     int num1 = 11; // 11 = 00001011
7     int num2 = 22; // 22 = 00010110
8     int result = 0;
10     result = num1 & num2;
11     cout<<"num1 & num2: "<<result<<endl; //num1 & num2: 2
12
13     result = num1 | num2;

```

```

14  cout<<"num1 | num2: "<<result<<endl; //num1 | num2: 31
16  result = num1 ^ num2;
17  cout<<"num1 ^ num2: "<<result<<endl; //num1 ^ num2: 29
18
19  result = ~num1;
20  cout<<"~num1: "<<result<<endl; //~num1: -12
21
22  result = num1 << 2;
23  cout<<"num1 << 2: "<<result<<endl; //num1 << 2: 44
24
25  result = num1 >> 2;
26  cout<<"num1 >> 2: "<<result; //num1 >> 2: 2
27
28  return 0;
}

```

Listing 25: Bitwise operator example

### 5.4.8 Ternary Operator

This operator evaluates a boolean expression and assign the value based on the result.

here is the syntax `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.

```

1  #include <iostream>
2  using namespace std;
3
4  int main(){
5
6      int num1, num2; num1 = 99;
7      /* num1 is not equal to 10 that's why
8       * the second value after colon is assigned
9       * to the variable num2
10     */
11     num2 = (num1 == 10) ? 100: 200; //num2: 200
12
13     cout << "num2: " << num2 << endl;
14     /* num1 is equal to 99 that's why
15      * the first value is assigned
16      * to the variable num2
17     */
18     num2 = (num1 == 99) ? 100: 200;
19     cout << "num2: "<<num2; //num2: 100
20
21     return 0;
22 }

```

Listing 26: Ternary operator example

## 5.5 Summary

- To ask the user to enter information, we use the