Ministry of Higher Education and Scientific Research

Amar Telidji University of Laghouat

Faculty of Technology

Department of Electrotechnics

programming in C++

Lecture notes

Dr.BELKHIRI Ahmed

a.belkheiri@lagh-univ.dz

Table of Contents

1. Introduction

A computer program is a sequence of instructions or commands given to a computer to perform specific actions. These instructions are usually quite basic. For example, they may include operations such as addition, multiplication, or other fundamental mathematical tasks, making the computer a highly efficient calculating machine. More complex instructions also exist, such as those for comparing values or processing characters.

Creating a program is essentially the process of using a series of basic instructions to accomplish a specific task. All programs are developed in this way: an MP4 player instructs the computer to play videos, a chat application allows communication with others over a network, and an operating system provides the computer with instructions on how to utilize hardware resources effectively.

The computer relies on a specialized electronic component designed to execute these instructions: the processor is responsible for carrying out small, basic operations, known as instructions, which serve as the foundation for everything a computer does. These instructions are stored in the computer as binary digits which are sequences of zeros and ones stored in the computer, which the processor interprets as commands to execute. These sequences are called machine language, which is inherently difficult for humans to understand. To address this, more advanced programming languages have been developed. These languages, often designed to resemble natural human language, are easier to learn and use compared to machine language.

There are many programming languages, such as Fortran, Pascal, C, and C++. The focus of this work is on C++. This lecture notes are primarily intended for 3rd year of a Licence degree in Automation. It aims to help students become familiar with the C++ language and serves as an introduction to Object-Oriented Programming (OOP).

1.2 History

The C language was created in the early 1970s in the laboratories of AT&T in the United States. Its designer, Dennis Mac Alistair Ritchie, aimed to improve an existing language by adding new features. By 1973, C was nearly fully developed and began to be

distributed the following year. Its success among computer scientists was so great that in 1989, ANSI, followed by ISO in 1990, decided to standardize it, meaning they established official international rules for the language. Today, there are three standards: ANSI C89 or ISO C90, ISO C99, and ISO C11.

Dr. Bjarne Stroustrup from Bell Laboratories (again) is the creator of C++, which was developed in the early 1980s. The goal was to create a language that supported object-oriented programming while maintaining high performance. It was also designed to appeal to a wide audience. Stroustrup created C++ based on C, ensuring compatibility: any C program can be compiled in C++. Most of the additions to C++ were inspired by the Simula67 and Algol68 languages.

Currently, C++ is one of the most widely used languages in the world, both for scientific applications and software development. As the successor to the C language, the "++" operator signifies incrementation, meaning the increase of variable types. Additionally, C++ allows the definition of new variable types, represented by the concepts of classes and objects, and later introduces object-oriented programming techniques.

1.3 Integrated Development Environment (IDE)

An Integrated Development Environment (IDE) for C++ is a software application that provides comprehensive tools for software development. It helps programmers write, test, and debug their code more efficientlyFor C++ development, several IDEs are commonly used, offering a range of features that can streamline the development process.

Here are some popular C++ IDEs:

1. Visual Studio (Windows)
    - A powerful and widely used IDE developed by Microsoft, offering a range of tools for C++ development, including debugging, refactoring, and performance profiling.
2. Code::Blocks (Cross-platform)

- A free, open-source IDE that is highly customizable and works across multiple platforms (Windows, Linux, Mac). It supports multiple compilers, such as GCC and Clang.

3. Dev-C++ (Windows)
    - A free, lightweight IDE for C++ that is simple and easy to use. It's based on the GCC compiler and is suitable for beginners.
4. Eclipse IDE for C/C++ Developers (Cross-platform)
    - A well-known open-source IDE with support for C/C++ development through the CDT (C/C++ Development Tools) plugin. It is highly customizable and extensible.
5. CLion (Cross-platform)
    - A commercial IDE from JetBrains, specifically designed for C and C++ development. It offers features like smart code completion, an integrated debugger, and a user-friendly interface.
6. Xcode (MacOS)
    - Apple's official IDE, primarily for macOS development, but it also supports C++ with features like debugging, syntax highlighting, and compiling.
7. Online IDEs:
    - Replit: An online IDE that allows you to write, compile, and run C++ programs directly from your web browser.
    - Ideone: Another online tool that supports C++ programming, allowing you to compile and execute code quickly.

Each IDE provides different features, and the choice of which to use depends on your operating system, the complexity of your projects, and your personal preference.

IDEs typically include a code editor, a compiler, a debugger, and other useful tools.

1- Code Editor

The code editor is used to write or modify code containing a C++ program, which must be saved with the ".CPP" extension. These files are referred to as source code files. The file extension helps the compiler recognize that the file contains a C++ program.

Note: A C++ program generally consists of multiple source files, which can be of two types:

- Files containing executable instructions: These use the .cpp extension.
- Files containing only declarations: These use the .h extension, which stands for "header." A .h file serves to group declarations shared by multiple .cpp files, ensuring proper compilation. To include the necessary .h files in a .cpp file, the #include directive is used.

For example, if the file to include is named untel.h, you would use: #include if it's a standard C++ library file, or #include "untel.h" if it's a file that we have written ourselves.

2- C ++ Compilor:

A compiler is a software tool that analyzes source code for syntax errors and then translates it from C++ into machine code (a sequence of ones and zeros) that the computer can process. The output file, generated after the compilation, retains the same name as the original source file but with a .obj extension. Several compilers are available, including Microsoft's Visual C++, Borland's C++ Builder, and GCC, a free and open-source option.

3- Linker

A linker is a software tool that combines object files generated by the compiler into a single executable program. After the compiler processes the source code and produces object files (with the .obj or .o extension), the linker resolves references between different object files and libraries, ensuring that the program can run correctly. It essentially "links" the compiled code with external libraries and functions, so that everything needed to run the program is included.

The linker performs several important tasks:

Symbol resolution: It matches function and variable references in the object files to their definitions.

Address assignment: It assigns memory addresses to variables and functions.

Library linking: It connects the program with necessary libraries (e.g., standard C++ libraries or third-party libraries).

Relocation: It adjusts code and data in the object files to work with the memory layout in the final executable.

Without a linker, the program would not be able to execute because the different pieces of compiled code wouldn't be properly combined.

Common linkers include:

GNU Linker (ld) for Linux.

MSVC Linker for Microsoft Visual C++.

Clang Linker for Clang.

Chapter 1: Basic Syntax in C++ Language

1.1 First program

The "Hello World" program is the first step in learning any programming language and one of the simplest programs we can learn. This program will display the message "Hello World" on the screen.

```
// Simple C++ program to display "Hello World":#
include<iostream>
using namespace std;
int main( )
 cout<< " Hello World ";
return 0 ;
display
"Hello World"
```

Let's examine each line of the program above:

1. // Simple C++ program to display "Hello World": This line is a comment. A comment is used to provide additional information about the program. It contains no programming logic. When the compiler encounters a comment, it simply ignores the line. Any line starting with // or enclosed in /*...*/ in C++ is considered a comment.
2. #include: In C++, any line starting with the hash symbol (#) is called a directive and is processed by the preprocessor, a program invoked by the compiler. The #include directive is typically placed at the beginning of the program. It allows the inclusion of certain objects, types, or functions into the program. The file name to include can either be inside angle brackets (< >) or enclosed in double quotes ("). Therefore, #include <iostream> tells the compiler to include the standard iostream file, which contains the declarations of all standard input/output functions (console/keyboard) for reading and displaying data.
3. int main(): This line is used to declare a main function named "main" that returns an integer value. Every C++ program must have a main function. The opening curly brace { indicates the beginning of the main function, and the closing curly

brace } marks the end of the main function. The instructions between these braces make up the body of the main function.

4. cout << "Hello World";: This line tells the compiler to display the message "Hello World" on the screen. This line is called a statement in C++. The semicolon at the end of the statement signifies the end of the instruction. Everything following the << symbol is displayed on the output device.
5. return 0;: This marks the end of the main function. This statement is primarily used in functions to return the results of operations performed by the function. In the case of main, returning 0 generally indicates that the program has executed successfully.

1.2 Instruction Comment

It is important to comment the code (program), meaning adding non-compiled information in the source code to better explain and remember what has been programmed. Two styles of comments are available in C++:

1. //: This allows you to comment out everything after it, up to the next line break.
2. /**...**/: This allows you to comment out multiple lines.

Example:

```
int b; // variable used to count grades
nb = 10; /* by default, we know
        the number of grades is:
        10... and all these lines are comments */
```

1.3 Keywords:

Keywords (or reserved words) are words recognized by the compiler that form the vocabulary of the C++ language. Table 2.1 contains the complete list of C++ language keywords.

Table 2.1: List of C++ Language Keywords

alignas	alignof	and	and_eq	asm	auto	
bitand	bitor	bool	break			
case	catch	char	char16_t	char32_t	class	compl
const	constexpr	const_cast	continue			
decltype	default	delete	do	double	dynamic_cast	
else	enum	explicit	export	extern		
false	float	for	friend			
goto						
if	inline	int				
long						
mutable						
namespace	new	noexcept	not	not_eq	nullptr	
operator	or	or_eq				
private	protected	public				
register	reinterpret_cast		return			
short	signed	sizeof	static	static_assert		static_cast
struct	switch					
template	this	thread_local		throw	true	try
typedef	typeid	typename				
union	unsigned	using				
virtual	void	volatile				
wchar_t	while					
xor	xor_eq					

1.4 types of variables

Like most programming languages, C++ uses the concept of variables. A variable can be seen as a memory location that holds a certain value.

Every variable in C++ must be declared: the declaration specifies the variable's identifier (its name) and its type.

Syntax:

type identifier;

1.4.1 Integers

Integers in C++ are expressed in three different types:

- **short**: A short integer (can be signed or unsigned).
- **int**: A standard integer (can be signed or unsigned).
- **char**: Used to manipulate characters, but can also be treated as integers.

1.4.2 Floating-Point Numbers

Floating-point numbers are expressed in three types:

- **float**: Single-precision floating-point type.
- **double**: Double-precision floating-point type.

- **long double**: Extended-precision floating-point type.

1.4.3 Booleans

Boolean variables can take one of two logical values: **true** or **false**.

Refer to Table 2.2 for a comprehensive list of fundamental data types used in C/C++ programming.

Table 2.2: Fundamental Types in C/C++

Type	Description	Typical Size	Value Range
char	Character (8-bit integer)	1 byte	-128 to 127 (signed) or 0 to 255 (unsigned)
short	Short integer	2 bytes	-32,768 to 32,767 (signed)
int	Standard integer	4 bytes	-2,147,483,648 to 2,147,483,647 (signed)
long	Long integer	4 or 8 bytes	Depends on the platform
long long	Very long integer	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	Floating-point number (single precision)	4 bytes	~3.4E-38 to ~3.4E+38
double	Floating-point number (double precision)	8 bytes	~1.7E-308 to ~1.7E+308
long double	Floating-point number (extended precision)	8 or 16 bytes	Depends on the platform
bool	Boolean (logical values)	1 byte	true or false
unsigned	Unsigned integer (used with other types)	Depends on the type	0 to the maximum of the specified type

Notes:

- The sizes and ranges may vary slightly depending on the compiler and system architecture.
- Unsigned types like *unsigned int* are used for non-negative values only.
- The *bool type* for logical values was introduced in C++ starting from the C++98 standard.

1.5 Variable Declaration in C++

The general syntax for declaring a variable in C++ is as follows:

```
1- Type variable ;
2- Type var 1 , var 2 , var 3 ;
```

Rules for Variable Names:

- Variable names must start with a letter or an underscore (_).
- Accents are not allowed in variable names.
- C++ is case-sensitive, meaning Var1 and var1 are considered different identifiers.

Example:

```
int     i  ,  j ,  k ;
float  valeur ;
```

Initializing Variables During Declaration: A variable can be declared and assigned a value (initialized) at the same time:

```
type   var1=0 ;
```
est équivalent à
```
{type  var 1 ;
var   1=0 ;
```

This is equivalent to:

```
type var1; var1 = 0;
```

Example:

```
int x = 10;
float y = 3.14;
```

Example

```
int a,b ;
a=4 ;
b=a ; // The value of a is copied into b: b is now 4.
```

1.6 Declaration of Constants

C++ allows two methods to define constants:

First Method

Declaring a variable whose value remains constant throughout the scope of the main function.

Example:

```
void main()
{
    const float PI = 3.14159;
    float perimeter, radius = 8.7;
    perimeter = 2 * radius * PI;
}
```

Second Method

Defining a symbol using the # define preprocessor directive.

Example:

```
#define PI 3.14159
void main()
{
    float perimeter, radius = 8.7;
    perimeter = 2 * radius * PI;
}
```

Note:

The *enum* keyword can be used to create custom types and define variables of those types.

Example:

```
enum color {green, red, black, white}; // Definition
of a custom type "color"
color c1, c2;
```

1.7 Input and Output Commands (cin, cout)

The cin command is used to input numeric values, single characters, and strings from the keyboard.

Syntax:

```
cin >> variable1 >> variable2;
```

Example:

```
int x;
cin >> x;
```

The cout command is used to display output on the screen (console output).

Syntax:

```
cout << variable1 << variable2;
```

with cout command we can:

1. Displaying the value of a variable:

```
int x, y;
char z;
cout << x << " " << y << " " << z;
```

2. Displaying a message:

```
cout << "This is a message.";
```

3. Combining a message with a variable value:

```
int x = 3, y = 5, z;
z = x + y;
cout << x << " + " << y << " = " << z;
```

4. Displaying the value of an expression:

```
cout << x << " + " << y << " = " << x + y;
```

Additional Formatting Options:

- Insert a tab using \t:

```
cout << x << "\t" << y << "\t" << z;
```

- Move to a new line using \n:

```
int x = 3, y = 5;
cout << "x = " << x << "\n" << "y = " << y;
```

- Using endl for a new line:

  Example:

```
cout << "Hello" << endl << "Goodbye";
```

1.8 Some Useful Functions in C++

- clrscr( ) : Clear the screen.

- `getch ( )` : Pause the execution of a program until a key is pressed.

- `getche( )` : Read a character variable

- `Caractère=getche( )` is equivalent to `cin >> character;`

- `gotoxy (a,b)` : Move the cursor to column a and row b before performing a read or write operation.

1.9 Operators

a- A unary operator: is an operator that has only one operand.

Example

```
     int  a ; // a declaration ,
&   a   ; // The "ampersand" (&) returns the memory address
// of a.
```

b- A binary operator: has two operands.

Example

```
     int   a , b ;
     a = b + 1 ;
```

1.9.1 Assignment Operator

The symbol "=" is used to assign the value of an expression to an identifier.

Example

x= 2*y+5 ; We say that x gets the result of the expression 2*y+5.

1.9.2 Arithmetic Operators

+ : Addition z = x + y
- : Subtraction z = x - y
* : Multiplication z = x * y
/ : Division z = x / y
++ : Increment i++

1.9.3 Relational Operators

> : Greater than x > y

<= : Greater than or equal to x <= y

< : Less than x < y

>= : Less than or equal to x >= y

== : Equal to x == y

!= : Not equal to x != y

1.9.4 Boolean Operators:

! : Logical NOT

&& : Logical AND

| (ALTGR+6) : Logical OR.

Chapter 2: Control Structures

Control structures in C++ allow the flow of execution to be controlled based on conditions or loops. They are categorized into:

2.1 Conditional Statements if, if-else, if-else if-else

This Structure is used to Perform a test, it then executes instructions depending on whether the condition is met or not.

Syntax

```
if (condition){
          Instruction block ;
                                    }
else
         {
          Instruction block;
                                    }
```

Example :

```
#include <iostream>
using namespace std ;
int main()
{
int x ;
cout<< "entrer un nombre x : ";
cin<<x ;
if (x<0)
cout<<"\n Negative integer ";
else
cout<<"\n  Positive or zero integer " ;
}
```

Note: If the choice is made between more than two conditions, we use:

```
If ….else if ….else
```

Example

```
if (x>0)
{
cout<< "\n positif"ve integer;
else if x=0
cout<<"\n zero integer";
else
cout<<"\n Negative integer  ";}
```

2.2 Loops

2.2.1 while Loop

Syntax

```
while (condition)
                  {
                   Instruction block  ;
                    }
```

The instruction block will continue to repeat as long as the condition is satisfied.

Example

```
#include <iostream>
int main()
{
int i=0;
while (i<=9)
{
cout<<  "\n i="<<i ;
++i ;// i=i+1} }
```

2.2.2 do_while Loop

Syntax :

```
do
{ Instruction block;}
while (condition);
```

Example

```
#include <iostream>
int main()
{
int i=0
do{
cout<<"\n i="<<i ;
++i ;
}
while (i<=9);
}
```

2.2.3 for Loop

Syntax

```
for (initial value; condition; step)
{
Instruction block ;
}
```

Example

```
#include <iostream>
int main()
{
int i ;
for (i=0 ; i<=0 ; ++1)
cout<< "\n i="<< i;
     }
```

2.2.4 Nested Loops

Loops can be nested within each other.

Example

```
#include< iostream>
using namespace std ;
int main ()
```

```
{
int i,j,s;
for ( i=0; i<=3; ++i)
for(j=0; j<2; ++j)
{
S=i+j;
cout<< "\n i="<<i"\t j="<<
     }}
```

Note

To terminate a loop or exit a selection statement, the BREAK instruction is used.

Example

```
If (condition)
Break ;
```

2.2.4 The switch-case Statement

This statement allows selecting a specific group of instructions among multiple instructions.

Syntax

```
Switch (value or expression)
{
Case1….. ;
Break ;
Case2 ……. ;
Break ;
……
Default…….. ;
}
```

Example

```
#include< iostream>
using namespace std ;
int main ()
```

```
{
float x,y;
int type ;
cout<< "\n Enter a number : ";
cin>>x ;
cout<<"\n enter operation type :1 to divide by 3, 2 to
multiply by 3 ";
cin>> type ;
switch (type)
{ case 1 ;
y=x/3;
cout<<"\n x/3 ="<<y;
break;
case 2 ;
             y=x*3 ;
             cout<<"\n x*3 ="<<y ;
            break ;
           default :
           cout<<"\n Invalid operation type........ ";
} // switch end.
```

Chapter 3: Pointers and Arrays

3.1 Pointers

A pointer is used to access a variable in memory,it contains the address of the memory location where the variable is stored. The pointer type is an unsigned integer, and its size depends on the processor type and the current mode.

Example

```
 float *X ;
Int *a,*b ;
```

X is a pointer that will store the address of a real number, while a and b are pointers that will each store the address of an integer.

Notation

`int u,*v=&u ;` we can also write `int u,*v ; v=&u ;`

We declare an integer u and a pointer v, which is initialized with =&u. This notation is read as "the address of the variable u."

Example

```
#include <iostream>
using namespace std;
int main()
{    int x=4,y=3,*t,*z;
     t=&x;// The address of x is assigned to t.
z=&y;// The address of x is assigned to t.
cout<<"\n x="<<x<<" &x="<< &x<<"t= "<<t<<"*t="<<*t;
cout<<"\n y="<<y<<"&y="<<&y<<"z= " <<z<<"*z="<<*z;}
```

```
x=4 &x=0x22fe94t= 0x22fe94*t=4
y=3&y=0x22fe90z= 0x22fe90*z=3
--------------------------------
```

3.2 Array

An array is a collection of data of the same type, the elements of an array can be manipulated using index positions.

3.2.1 One-Dimensional Array

Declaration

```
<Type> <NomTableau> [<T>];
 Declaration of an Array Initialized with Elements
<Type> <NomTableau> [<T>] =
{{<Variable1,...,VariableN>};
```

So, NomTableau is an array of size T, which must be a constant value.

The array size is fixed once and cannot be modified during execution; this is known as a static array.

Each element is identified (initialized or accessed) using an index that specifies its position in the array.
In C++, the first element is located at position 0 and is identified by index 0, while the last element is at index (T-1).

To manipulate array elements (initialization, display, operations), the for loop is commonly used.

Example

```
#include <iostream>
using namespace std;
int main()
{
  int i;
int tableau[5]; // Declaration of an Array with 5 Elements
  for (int i(0); i<4; i++ )
cin >> tableau[i];//Reading Elements from the Keyboard

  for (int i(0); i<4; i++ )
```

```
        cout<<"the array ["<<i<<"] Contains the Value
    "<<tableau[i]<<endl;
        return 0;
```

Program Execution

```
2
5
8
7
the array [0] Contains the Value 2
the array [1] Contains the Value 5
the array [2] Contains the Value 8
the array [3] Contains the Value 7
```

A dynamic array is an array whose number of elements can vary during program execution. It allows adjusting the array size based on the programmer's needs. To work with this type of array, the following operators are used:

a- New operator

Syntax

```
pointeur=new type[size];
```

The new operator allocates a memory block capable of storing size elements of type type and returns the address of this memory block.

The parameter size is an integer that can be a variable, a constant, or an expression. The new operator returns a pointer to the specified type. Therefore, the pointer variable is of type type*.

The array elements are indexed from 0 to size-1 and are accessed like a static array. If there is not enough available memory, new returns the NULL pointer.

b- delete[] operator

Syntax

```
delete[] pointeur;
```

The delete[] operator is used to delete an array that was previously allocated using new. Make sure to include the square brackets ([]) right after delete; otherwise, the array will not be properly deallocated.

In C++, the programmer is responsible for properly managing the destruction of dynamically allocated arrays.

Example

```
#include <iostream>
using namespace std;
int main()
{
    int i, size;
    cout << "Enter the size  : ";
    cin >> size;
    int *t;
    t = new int[size];
    for (i = 0; i < size; i++)
        t[i] = i * i;
    for (i = 0; i < size; i++)
       cout << t[i] << endl;
    delete[] t;
    return 0;}
```

Execution 1 :

```
 Enter the size  : 3
0
1
4
```

Execution 2 :

```
 Enter the size  : 6
0
1
4
9
16
25
```

3.2.2 Two-Dimensional Array

Declaration

```
<Type> <NomTableau> [Dimension1][Dimension2];
```

Where Dimension1 is the number of rows and Dimension2 is the number of columns. Two indices are used to identify the elements of the two-dimensional array.

Example

C++ Program to initialize a 3x2 array with user-entered values and display It

```
#include <iostream>
using namespace std;
int main()
{
    int i, j; //Location indices
    int tab[3][2]; /* Declaration of an integer array
with 3 rows and 2 columns.*/
    // Array initialization
for (i=0;i<3;i++)
     for (j=0;j<2;j++)
{cout<<"\n Enter an integer: \t";cin>>tab[i][j];}
//array display
for (i=0;i<3;i++)
     for (j=0;j<2;j++)
cout<<"\n tab["<<i<<"]["<<j<<"]="<<tab[i][j];
return 0;
```

Execution

```
Enter an integer:      2
Enter an integer:      5
Enter an integer:      9
Enter an integer:      10
Enter an integer:      12
Enter an integer:      3

tab[0][0]=2
tab[0][1]=5
tab[1][0]=9
tab[1][1]=10
tab[2][0]=12
tab[2][1]=3
```

It is possible to declare arrays with more than two dimensions.

Example :

```
int tab1[3][10][5] ;
char tab1[4][8][12] ;
```

The same principle used for two-dimensional arrays is applied to manipulate multi-dimensional arrays.

Chapter 4: Functions

4.1 Functions

A function is a self-contained piece of a program (a subprogram) used by another program to perform a specific operation. Functions allow:

1. Creating modules, each accomplishing a specific task.
2. Reusing parts of a program to avoid repetition.
3. Making program writing, reading, and debugging easier.

Declaration

Every function used in a program must be declared before main(), and the declaration ends with a semicolon. The function declaration is called its **prototype**.

```
Type_retour    nom_fonction(type1,   Parameter1,type2,
Parameter 2……)
{
Function body
}
```

 Return Type: This is the type of value returned by the function. If the function does not return any value, the void type is used instead.

 Parameter List: Each parameter is defined by its type followed by its name. If the function has no parameters, its name is followed by an empty pair of parentheses ().

 Function Body: This is the set of instructions that make up the function.

4.2 Calling a function

A function is called within main() or another function by specifying its name followed by its parameters inside parentheses. If the function has no parameters, the parentheses remain empty.

```
X= function_name (parameter 1,…, parameter N)
```

Example

```
//Main program
#include<iostream>
Using namespace std;
//fnct cube
double cube (double x)
{
Return x*x*x;
}
//fnct Hello
Void Hello()
{
Cout<<"hello !"<<endl ;
//main program
int main()
{
Hello() ;//calling of hello() function
Cout<<"\n enter number ";
Cin>>a ;
Cout<<"the cube of "<<a<<" is :"<<cube(a) // calling
of cube() function.
}
```

4.3 Local Variables

Variables declared inside main() are local to main(), and variables declared inside a function are local to that function.

Example

```
#include<iostream>
void fnct() // Void function with no parameters;
int main()
{
Int i=5;
Cout<<"\n Beginning of the call to fnct () : i="<<i ;
```

```
fnct()
Cout<<"\n After the call to fnct() : i="<<i ;
}
// fnct function definition
Void fnct()
{
Int i=3 ;
i=i+5 ;
Cout<<”\n Inside fnct () : i=”<<i;
}
```

The program displays the following:

Before calling fnct(): i = 5

Inside fnct(): i = 8

After calling fnct(): i = 5

Note :

In the main program main(), i is 5, while in fnct(), i is 8. This proves that they are not the same i. The compiler allocates two different memory locations—one for the i declared in main() and another for the i declared in fnct().

4.4 Global Variables

A variable is considered global if it is declared before main().It is accessible to all functions compiled within the same source program.

Example:

```
#include<iostream>
using namespace std;
void fnct();
int i ; //global variable
int main()
{
i=5;
cout<<    "\n Bifore calling fnct(t) : i="<<i;
fnct();
cout<<"\n After calling  fnct()  : i="<<i ;
```

```
}
void fnct()
{
i=i+5 ;
cout << "\n Inside fnct(): i="<<i ;
}
Bifore calling fnct(t) : i=5
Inside fnct() : i=10
After calling  fnct() : i=10
```

4.5 Recursive Functions

A function is considered recursive when it continuously calls itself until a specific condition is satisfied.

Example

```
#include<iostream>
int factorielle (int);
int main()
int n, result;
cout<<”\n enter n value”;
cin>>n ;
Result=factorielle(n) ;
cout<< "\n n !="<<result ;
}
int factorielle (int n)
{
if (n==0//n==1) return(1) ;
else return (n*factorielle (n-1)) ;
}
```

Chapter 5: Files

5.1 Files

In C++, generic streams allow access to and manipulation of various types of content, including files. However, there are limitations to the generality of streams due to different access types (read-only, write-only, etc.) and file modes (binary or text).

5.2 Text Mode

Text mode refers to storing data in its ASCII form. For example, the double value pi = 3.141592 will be stored as a sequence of characters: '3' '.' '1' '4' '1' '5' '9' '2' '\0'.

Files in text mode can be opened and viewed using any text editor (e.g., Vim, Notepad, etc.). Their size depends on the displayed precision and is often significantly larger than the actual memory required to store the values.

To work with file streams, the <fstream> library must be included. This allows access to files in read, write, or read/write mode.

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

In the main function, you should declare a stream variable of type fstream for the file.

```
int main(int argc, char *argv[])
{
fstream fileT;
```

Next, open the file using the open function. This function can only be executed from an fstream object (a variable of type fstream).

```
fileT. open ("test. txt", fstream:: in | fstream:: out
| fstream:: trunc);
if( fileT. fail() )
{
```

```
 cout << "Text File cannot be opened/created ! " <<
endl;
 return -1; // fin avec un retour d'erreur
 }
```

Writing some information:

```
 fileT << "Hello_SansEspace" << " " << 3.141592654 <<
" " << 2.718281 <<
endl;
 fileT << 6.022141 << " " << 1.380650 << endl;
Then, reading some information. First, move back to
the beginning of the file.
 //read Something
 fileT. clear(); // mise a 0 des flags
 fileT. seekg(0, fstream:: beg); // Return to the
beginning of the file.
Finally, read the information, ensuring the correct
types and order of data!
 string stT;
 double piT, eT, aT, bT;
fileT >> stT >> piT >> eT >> aT >> bT; // lecture

cout << stT << endl;
 cout << "PI : "<< piT << " E : " << eT << endl;
 cout << "Avogadro : " << aT << " Boltzmann : " << bT
<< endl;
 fileT. close(); // close file
 return 0;
 }
```

5.3 Binary mode

Binary mode refers to the raw storage of data in binary format (as it is stored in memory). For example, the double value pi = 3.141592 will be stored in its double (IEEE 754 floating-point) format. The content of such files is unreadable by text editing software. Accessing binary files is quite similar to accessing text files, with some key differences.

To read or write in binary mode, the fstream::binary flag must be added when opening the file using the open function.

Writing to a file opened in write mode (fstream::out) is done using the write function. This function saves the contents of a variable byte by byte. Therefore, the variable must be converted into a character array (char*) using the reinterpret_cast<char*>(&variable) function. Additionally, the write function must be provided with the exact number of bytes to write, which can be determined using sizeof(variable_type).

Reading data from a binary file (opened in read mode: fstream::in) is done using the read function. It works similarly to the write function. There are no separators between different variables since a fixed number of bytes is read at each read operation.

```
 #include <iostream>
 #include <fstream>
 using namespace std;
int main(int argc, char *argv[])
 {
 fstream fileB;
 fileB. open ("test. bin", fstream:: binary |
fstream:: in | fstream:: out |
fstream:: trunc );
 if( fileB. fail() )
 {
 cout << "Binary File cannot be opened/created ! " <<
endl;
 return -1; // end
 }
 // write something
 double pi = 3.1415192;
 double e = 2.718281;
 int c = 299999;
 fileB. write(reinterpret_cast<char *>(&pi),
sizeof(double) );
```

```
 fileB. write(reinterpret_cast<char *>(&e),
sizeof(double) );
 fileB. write(reinterpret_cast<char *>(&c),
sizeof(int) );
 //read Something
 fileB. clear();
 fileB. seekg(0, fstream:: beg);
 double piB, eB;
 int cB;
 fileB. read( reinterpret_cast<char *>(&piB),
sizeof(double) );
 fileB. read( reinterpret_cast<char *>(&eB) ,
sizeof(double) );
 fileB. read( reinterpret_cast<char *>(&cB) ,
sizeof(int) );
 cout << " PiB : " << piB << " EB : " << eB << " cB :
" << cB << endl;
 fileB. close(); // close file

 return 0;
 }
```

Using structures or classes to store organized data is highly recommended for binary mode.

However, you may sometimes need to work with files in C rather than C++. In C, stream objects (fstream) are not available. Instead, you must use the FILE* type and the following functions to access files:

- fopen – Opens a file (in binary or text mode, for reading or writing).
- fclose – Closes an open file.
- fseek – Moves within a file.
- fread – Reads a specified amount of data from a file (unit: byte or char).
- fwrite – Writes data to a file (unit: byte or char).

Below is a generic C function that reads any file in binary mode. To obtain the desired data type, you must cast the message variable accordingly.

For complex data formats, using structures is recommended. The message variable represents a data array, which is dynamically allocated in C by the LireFichier function. The function's message type could be defined as void to allow flexibility.

```
 #include <stdio.h>
 #include <stdlib.h>
 int LireFichier( const char fichier[], unsigned char
**message, int *taille,
int offset)
 {
 int nb_lu;
FILE* fich = fopen(fichier, "rb");
 if(! fich) return 1;
 // Lecture de la taille du fichier
 fseek( fich, 0, SEEK_END); // Positioning the pointer
//at the end of the file
 *taille = ftell(fich) - offset; // Reading the
pointer position (=
taille) fseek( fich, offset, SEEK_SET); //
Repositioning the pointer at the beginning
// of the file (offset)
 *message = (char*) malloc (*taille);
 if( *message == NULL ) return 3; // PB allocation
 nb_lu = fread( *message, sizeof(char), *taille,
fich); // Lecture
 if ( nb_lu != *taille ) return 1;
 fclose(fich);
 return 0; // no probleme
}
```

The generic function for writing to a file in binary mode is as follows. If the message is the string version of the data (fprintf), a readable text file will be created instead.

```
 #include <stdio.h>
 #include <stdlib.h>
int EcrireFichier( const char fichier[], const
unsigned char message[], int
```

```
taille)
 {
 FILE *f_dest;
 f_dest = fopen(fichier, "wb");
 if(! f_dest) return 1; //Write access problem
 fwrite( message, sizeof(char), taille, f_dest);
 fclose(f_dest);
return 0; //no problem ;
 }
```

Chapter 6: Introduction to Object-Oriented Programming in C++

6.1 Introduction

Object-Oriented Programming (OOP), or programming by objects, is a computer programming paradigm developed by Norwegian researchers Ole-Johan Dahl and Kristen Nygaard in the early 1960s and later expanded by American computer scientist Alan Kay in the 1970s.

- An object represents a concept, idea, or any physical entity, such as a car, a person, or a book page. It has an internal structure, a specific behavior, and the ability to interact with other objects. OOP focuses on representing these objects and their relationships. By leveraging interactions between objects, developers can design and implement the required functionalities more effectively, making problem-solving more efficient.

- Thus, modeling plays a crucial role in OOP, as it allows real-world elements to be transcribed into virtual forms. While it is possible to design an object-oriented application without specialized software tools, these tools greatly enhance design, maintenance, and productivity.

- Currently, there are two main categories of object-oriented programming languages:

- Class-based languages, which can be functional (e.g., Common Lisp Object System), imperative (e.g., C++, Java), or both (e.g., Python, OCaml).

- Prototype-based languages, such as JavaScript.

6.2 From Classical Programming to Object-Oriented Programmin

Classical programming, as studied through languages like C, Pascal, defines a program as a set of data on which procedures and functions operate. In this model:

- Data represents the passive part of the program.
- Procedures and functions represent the active part.
- Programming in this context involves:
- Defining a certain number of variables (structures, arrays, etc.).

- Writing procedures to manipulate these variables without explicitly associating them with one another.
- Executing a program, then, simply means calling these procedures in a specific order, as dictated by the sequence of instructions, while providing them with the necessary data to accomplish their tasks.

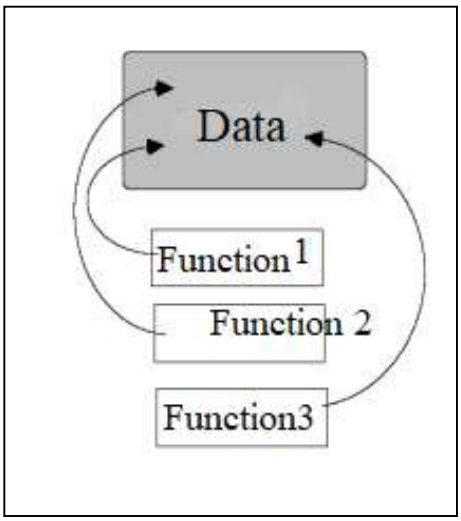

Figure 6.1 Data and Procedure Approach

In this approach, data and procedures are treated independently of each other, without considering the close relationships that exist between them. This raises several questions:

1. Is this separation (data, procedures) useful?
2. Why prioritize procedures over data? (What is the main objective?)
3. Why not consider programs primarily as sets of software objects characterized by the operations they perform?

Object-oriented languages emerged to address these questions. They are based on a single category of software entities: objects. An object integrates both static (data) and dynamic (behavior) aspects within the same concept.

With objects, data becomes the central focus. The first question to answer in this paradigm is: “What are we talking about?”

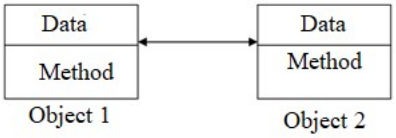

Figure 6.2 Object-Oriented Approach

6.3 Object and class

An object is a software entity that:

- Has an identity.
- Can store a state, meaning a set of information in internal variables.
- Responds to specific messages by triggering appropriate internal actions that change the object's state.

These operations are called methods. Methods are functions associated with objects that define their behavior.

6.4 Definitions

6.4.1 Attributes

The attributes of an object are the set of information, represented as variables, that define the state of the object.

6.4.2 Message

A message is a request to activate a method sent to an object.

6.4.3 Methods

Method is a function or procedure associated with an object that is triggered upon receiving a specific message. The method executed corresponds exactly to the received message.

The list of methods defined within an object forms its interface for the user. These are the messages the object can understand when received, and their reception triggers the corresponding methods.

6.4.5 Signature

The signature of a method specifies its name, the type of its arguments, and the return type of the data.

6.5 class

A class is a collection of variables of the same or different types, combined with a set of functions. A class is declared as follows:

```
Classe <nom de classe >
{// Member Variables
Type variable_1 ;
Type variable_2 ;
…… ;
Type variable_n ;
// member Fonctions
Type function_1() ;
Type function_2() ;
…… ;
Type fonction_k() ;
} ;
```

Note : Member variables are called attributes, and member functions are called methods.

Example

```
class client{
int no_client ;   //variable membre
 void initialiser (int) ; // fonction membre
},
```

6.6 Object

An object is a variable of a class type, and its declaration is done as follows:

```
<nom_class> variable_1, variable_2,… , variable_n ;
// variable_1, variable_2,… , variable_n sont des
objets // de type nom_class.
```

Example

`Client C1,C2 ;` // means that C1 and C2 are two variables or objects of type Client.

6.7 Accessing to Class Members

Once the class and its objects are declared, access to the class members (attributes and methods) is done as follows:

```
<nom_objet>.<nom_variable> ( ) ;
Ou
<nom_objet>.<nom_fonction> ( ) ;
```

Example

```
C1.no_client=111 ; // Initialize no_client of c1 with
111.
C1.no_client=111 ;
```

6.8 Private and Public Members of a Class

The member variables of a class are either private, declared using the private keyword, or public, declared using the public keyword. By default, all member variables of a class are private.

Example

```
# include <iostream>
using namespace std ;
//class declaration
class rectangle{
//  Declaration of the member variables width and
length.
double largeur , longueur ;
// Declaration of a member function surface
double surface ( )
{return largeur*longueur ;}
} ;
int main ( )
{
rectangle rect ; // Declaration of the object rect of
type Rectangle
```

```
cin>> rect.largeur ; // Reading the width of the
object rect
cin>> rect.longueur ; // Reading the length of the
object rect;
cout << "surface de rect = "<< rect.surface( )<<endl ;
return 0 ;}
```

The compiler will generate an error because all attributes (member variables and functions) are private by default. As a result, writing rect.largeur in the main function is not allowed. To make an attribute accessible from outside the class, it must be declared as public.

```
.    .    .
class rectangle{
public ;
double largeur , longueur ;
double surface ( )
{return largeur*longueur ;}
} ;
.    .    .
```

6.9 Class Constructor

A constructor is a member function of a class that has the same name as the class. Its role is to initialize objects of that class.

Example

```
#include < iostream>
class chat { int age, poids, // variable privées
public;
chat ( ); // constructeur pour initialiser les
variables privées.
void afficher ( ); // pour afficher le contenu des
variable privées
}
chat :: chat ( )
```

```
{ cout << " \n age et poids " ;
cin >> age >>poids ;
}
Int main( )
{
// déclaration et création  de 2 objets
chat Minou, Minouche ;
//affichage des deux objets
Cout <<   " \n information sur minou "
Minou.afficher () ;
Cout <<   " \n information sur minouche "
Minouche.afficher () ;
}
```

6.10 Destructor

A destructor is automatically called at the end of the block where the object was declared. It is used to release the memory allocated in the constructor.

Example

```
class Voiture
{
    private:
        int longueur;

    public:
        // Constructeur
        Voiture(int long);

        // Destructeur
        ~Voiture();
};
```

Chapter 7 : Summary Exercises

7.1 Basic Concept

Exercise 1

Write a program that asks the user to enter the width and length of a field and then displays its perimeter and area.

Exercise 2

Write a program that asks the user to enter five integers and then displays their average. The program should use only two variables.

This exercise aims to verify the following technical points:

- The concept of variables and their declaration
- Calculation of the average
- Use of int and double types
- Use of cin and cout
- Assignment

Exercise 3

Write a program that asks the user to enter two integers, A and B, swaps the values of the variables A and B, and then displays them.

This exercise aims to verify the following technical points:

- The concept of variables and their declaration
- Use of cin and cout
- Assignment

- A basic "algorithm": swapping the values of two variables

Exercise 4

Write a program that asks the user to enter:

The pre-tax price per kilogram of tomatoes

The number of kilograms of tomatoes purchased

The VAT rate (e.g., 10%, 20%, etc.)

The program should then display the total price including tax.

This exercise aims to verify the following technical points:

- The concept of variables and their declaration
- Choosing relevant and explicit variable names
- Use of cin and cout
- Assignment
- Modeling an "economic" problem

7.2 Control Structures

Exercise 1

Write a program that asks the user to enter an integer and displays "WIN" if the integer is between 56 and 78 (inclusive), otherwise, it displays "LOSE".
This exercise aims to verify the following technical points:

- The concept of variables and their declaration;
- The use of cin and cout;
- Choosing a suitable control structure for the problem.

Exercise 2

Write a program that displays all integers from 8 to 23 (inclusive) using a for loop. This exercise aims to verify the following technical point:

- Basic use of a for loop.

Exercise 3

Same as Exercise 2, but using a while loop. This exercise aims to verify the following technical point:

- Basic use of a while loop.

Exercise 4

Write a program that asks the user to enter 10 integers and then displays their sum. This exercise aims to verify the following technical points:

- Use of a for loop;
- Study of a common algorithm: sum calculation.

Exercise 5

Write a program that asks the user to enter 10 integers and then displays the smallest of these integers.

Exercise 6

Write a program that asks the user to enter an integer N and calculates the sum of the cubes from 5^3 to N^3. This exercise aims to verify the following technical points:

- Simple use of a for loop;
- Study of a common algorithm: sum calculation;
- Modeling a simple problem from mathematics.

Exercise 7

Write a program that asks the user to enter an integer N and calculates u(N) defined by:

- u(0) = 3
- u(n+1) = 3 * u(n) + 4

This exercise aims to verify the following technical points:

- Simple use of a for loop;
- Study of a common algorithm: calculating terms of a recurrence sequence;
- Modeling a problem from mathematics.

Exercise 8

Write a program that asks the user to enter an integer N and calculates u(N) defined by:

- u(0) = 1
- u(1) = 1
- u(n+1) = u(n) + u(n-1)

This exercise aims to verify the following technical points:

- Simple use of a for loop;

- Study of a common algorithm: computing a recurrence sequence;
- Modeling a simple problem from mathematics.

Exercise 9

Write a program that performs operations on an integer (initial value 0). The program displays the integer's value and then shows the following menu:

1. Add 1
2. Multiply by 2
3. Subtract 4
4. Quit

The program then asks the user to enter an integer between 1 and 4. If the user enters a value between 1 and 3, the operation is performed, the new integer value is displayed, and the menu is shown again. This process continues until the user enters 4, at which point the program terminates.

This exercise aims to verify the following technical points:

- Use of a while loop;
- Use of a switch statement;
- Managing a program using a menu;
- Modeling a simple problem in a computer program.

Exercise 10

Write a program that asks the user to enter 10 integers and displays the number of occurrences of the highest value entered. This exercise aims to verify the following technical points:

- Use of a for loop;
- Study of a moderately difficult algorithm: counting occurrences of a value.

Exercise 11

Write a program that asks the user to enter an integer and determines whether it is prime or not. This exercise aims to verify the following technical points:

- Use of a moderately difficult while loop;
- Study of a fairly difficult algorithm: determining the primality of an integer;
- Modeling a problem from mathematics.

7.3 Array

Exercise 1

Write a program that asks the user to enter an integer and displays "WIN" if the integer is between 56 and 78 (inclusive), otherwise, it displays "LOSE".

This exercise aims to verify the following technical points:

- The concept of variables and their declaration
- Use of cin and cout
- Choosing an appropriate control structure for the problem

Exercise 2

Write a program that displays all integers from 8 to 23 (inclusive) using a for loop.

This exercise aims to verify the following technical point:

- Basic usage of a for loop

Exercise 3

Rewrite the previous exercise using a while loop.

This exercise aims to verify the following technical point:

- Basic usage of a while loop

Exercise 4

Write a program that asks the user to enter 10 integers and then displays their sum.

This exercise aims to verify the following technical points:

- Use of a for loop
- Understanding a common algorithm: sum calculation

Exercise 5

Write a program that asks the user to enter 10 integers and then displays the smallest of these integers.

Exercise 6

Write a program that asks the user to enter an integer N and calculates the sum of the cubes from 5^3 to N^3.

This exercise aims to verify the following technical points:

- Simple usage of a for loop
- Understanding a common algorithm: sum calculation
- Modeling a simple mathematical problem

Exercise 7

Write a program that asks the user to enter an integer N and calculates u(N), defined as follows:

- u(0) = 3
- u(n+1) = 3 * u(n) + 4

This exercise aims to verify the following technical points:

- Simple usage of a for loop
- Understanding an algorithm for computing terms of a recursive sequence
- Modeling a mathematical problem

Exercise 8

Write a program that asks the user to enter an integer N and calculates u(N), defined as follows:

- u(0) = 1
- u(1) = 1
- u(n+1) = u(n) + u(n-1)

This exercise aims to verify the following technical points:

- Simple usage of a for loop
- Understanding an algorithm for computing a recursive sequence
- Modeling a simple mathematical problem

Exercise 9

Write a program that allows performing operations on an integer (initially set to 0). The program displays the current value of the integer and then shows the following menu:

1. Add 1
2. Multiply by 2

3. Subtract 4
4. Quit

The program then asks the user to enter a number between 1 and 4. If the user enters a value between 1 and 3, the corresponding operation is performed, the new value of the integer is displayed, and the menu is shown again. This process continues until the user enters 4, at which point the program terminates.

This exercise aims to verify the following technical points:

- Use of a while loop
- Use of a switch statement
- Managing a program using a menu system
- Modeling a simple problem into a computer program

Exercise 10

Write a program that asks the user to enter 10 integers and then displays the number of times the highest number appears.

This exercise aims to verify the following technical points:

- Use of a for loop
- Understanding a moderately difficult algorithm: counting occurrences of a value

---

Exercise 11

Write a program that asks the user to enter an integer and determines whether it is a prime number.

This exercise aims to verify the following technical points:

- Use of a while loop (moderate difficulty)
- Understanding a relatively difficult algorithm: checking primality of an integer

- Modeling a mathematical problem

7.4 Functions

Exercise 1

Write a function distance that takes four double parameters xa, ya, xb, yb, representing the coordinates of two points A and B, and returns the distance AB. Test this function.

This exercise aims to verify the following technical points:

- Creating simple functions
- Passing parameters by value
- Using return
- Calling a function

Exercise 2

Write a function f that takes a double x and a bool OK as parameters and returns a double.

The function should return sqrt((x - 1) * (2 - x)) if it is defined at x.

The function should modify OK to true if the function is defined at x, otherwise, it should set OK to false.

Test this function.

This exercise aims to verify the following technical points:

- Creating simple functions
- Passing parameters by value and by reference
- Using return
- Handling both input and output parameters in a function

- Calling a function
- Testing a function

Exercise 3

Write a function f that takes an integer as a parameter and returns a bool:

true if the integer is prime,

false otherwise.

Test this function.

This exercise aims to verify the following technical points:

- Creating simple functions
- Calling a function
- Validating data before calling a function
- A function returning a boolean

Exercise 4

Write a function swap that takes two integer parameters a and b and swaps their values. Test this function.

This exercise aims to verify the following technical points:

- Creating simple functions
- Calling a function
- Passing parameters by reference

Exercise 5

Write a function f that takes an array t of any size and an integer n indicating the size of the array. The function should return a bool b indicating whether there is a value between 0 and 10 in the first n elements of the array. Test this function.

This exercise aims to verify the following technical points:

- Writing a function that takes an array of any size as a parameter
- Searching for an element in an array that meets a condition
- Using return.

7.8 Files

Exercise 1

Write a program that writes the following text into a file named example.txt:

```
Hello word:
Here is a program illustrating writing to a file
```

This exercise aims to verify the following technical points:

- Opening a file for writing
- Checking if a file is open (especially verifying write permissions)
- Closing the file after writing

Exercise 2

Write a program that reads the example.txt file created in the previous exercise and displays its content. You should obtain:

```
Hello word:
Here is a program illustrating writing to a file
```

This exercise aims to verify the following technical points:

- Opening a file for reading
- Checking if a file is open (especially verifying read permissions)
- Reading an entire file

- Closing the file after reading

Exercise 3

Write a program that writes a string in binary format, followed by a list of integers from 0 to 1000.

```
Liste des entiers de 1 à 1000
0
1
2
...
1000
```

This exercise aims to verify the following technical points:

- Opening a file for writing in binary mode
- Checking if a file is open (especially verifying write permissions)
- Writing data in binary format
- Closing the file after writing

Exercise 4

Write a program that reads the binary file created in the previous exercise and displays its content.

You should obtain:

```
Liste des entiers de 1 à 1000
0
1
2
...
1000
```

This exercise aims to verify the following technical points:

- Opening a file for reading in binary mode
- Checking if a file is open (especially verifying read permissions)
- Reading data written in binary format
- Closing the file after reading

Reference List

1. Bjarne Stroustrup, Marie-Cécile Baland, Emmanuelle Burr, Christine Eberhardt, « Programmation:

Principes et pratique avec C++ », Edition Pearson, 2012.

2. Jean-Cédric Chappelier, Florian Seydoux, « C++ par la pratique. Recueil d'exercices corrigés et aide-mémoire », PPUR Édition : 3e édition, 2012.

3. Jean-Michel Léry, Frédéric Jacquenot, » Algorithmique, applications aux langages C, C++ en Java », Edition Pearson, 2013.

4. Frédéric DROUILLON, « Du C au C++ - De la programmation procédurale à l'objet », Eni; Édition : 2e édition, 2014.

5. Claude Delannoy, « Programmer en langage C++ », Edition Eyrolles, 2000.

6. Kris Jamsa, Lars Klander, « C++ La bible du Programmeur », Edition Eyrolles, 2000.

7. Bjarne Stroustrup, « Le Langage C++ », Édition Addison-Wesley, 2000.