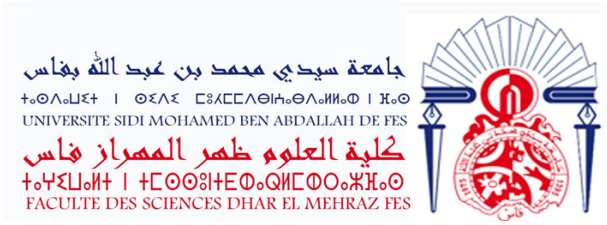

Université Sidi Mohamed Ben Abdellah
USMBA
Faculté de sciences de Fès
FSDM
Laboratoire Informatique, Imagerie et Analyse Numérique
LIIAN

Advanced cours completion in C++

May 16, 2020

Anselme Russel Affane Moundounga

May 2020

Contents

Chapter 1

Introduction

Programming is an essential tool in the field of scientific research. Indeed, it allows to verify a set of stated theory, to demonstrate postulates through numerical calculation. Traditionally, certain tasks carried out in the context of hypothesis testing are sometimes tedious and still not obvious. This is the case, for example, when it is necessary to perform an iterative and repetitive calculation. Programming tools provide enormous flexibility and save time on certain tasks.

In the field of computer science and programming, there is a multitude of programming languages, those considered low level, i.e. closer to the hardware and sometimes not very obvious like Assembler or C; or those closer to the natural language of the user like Python, R, and so on.

C++ is a programming language located at the border between high level languages and low level languages. Some characteristics of the language are (Nebra Mathieu, 2011):

- It is very widespread, it is one of the most widely used programming languages on the planet. So there is a lot of documentation on the Internet and help is easily available on forums.
- It is fast, very fast even, which makes it a language of choice for critical applications that need performance. This is particularly the case for video games, but also for financial tools or certain military programs that need to work in real time.
- It is portable, the same source code can theoretically be transformed without problem into executable under Windows, Mac OS and Linux. You will not need to rewrite your program for other platforms.

- There are many libraries for C++. Libraries are extensions for the language, a bit like plug-ins. Basically, C++ does not know do a lot, but by combining it with good libraries, you can create 3D, network, audio, windowed, etc. programs.
- It is multi-paradigm, which means you can program in C++ in different ways using the Object Oriented Programming (OOP) technique. It is a technique that allows to simplify the organization of the code in our programs and to make it easy to make certain pieces of reusable code.

To perform our future scholars and compare results by different way, numerical computing, solver resolution, manual calculation, we decided to use the famous library provide in the 3^{dr} Numerical Recipes Edition(William H. Press, 2007).

Chapter 2

C Family Syntax

This chapter is a part of (William H. Press, 2007). Not only C ++ , but also Java, C#, and (to varying degrees) other computer languages, share a lot of small-scale syntax with the older C language (Josuttis, 2012). By small scale, we mean operations on built-in types, simple expressions, control structures, and the like. In this section, we review some of the basics, give some hints on good programming, and mention some of our conventions and habits.

2.1 Operators

A first piece of advice might seem superfluous if it were not so often ignored: You should learn all the C operators and their precedence and associativity rules. You might not yourself want to write

```
n << 1 | 1;
```

as a synonym for 2*n+1 (for positive integer n), but you definitely do need to be able to see at a glance that

```
n << 1 + 1;
```

is not at all the same thing! Please study the table on the next page while you brush your teeth every night. While the occasional set of unnecessary parentheses, for clarity, is hardly a sin, code that is habitually overparenthesized is annoying and hard to read.

Associativity and Operator		Function	Use	See Page
L	::	global scope	::name	286
L	::	class scope	class::name	88
L	::	namespace scope	namespace::name	82
L	.	member selectors	object.member	23
L	->	member selectors	pointer->member	110
L	[]	subscript	expr [expr]	116
L	()	function call	name (expr_list)	23
L	()	type construction	type (expr_list)	164
R	++	postfix increment	lvalue++	147
R	--	postfix decrement	lvalue--	147
R	typeid	type ID	typeid(type)	826
R	typeid	run-time type ID	typeid(expr)	826
R	explicit cast	type conversion	*cast_name*<type>(expr)	162
R	++	prefix increment	++lvalue	147
R	--	prefix decrement	--lvalue	147
R	~	bitwise NOT	~expr	152
R	!	logical NOT	!expr	141
R	-	unary minus	-expr	140
R	+	unary plus	+expr	140
R	*	dereference	*expr	53
R	&	address-of	&lvalue	52
R	()	type conversion	(type) expr	164
R	sizeof	size of object	sizeof expr	156
R	sizeof	size of type	sizeof(type)	156
R	sizeof...	size of parameter pack	sizeof...(name)	700
R	new	allocate object	new type	458
R	new[]	allocate array	new type[size]	458
R	delete	deallocate object	delete expr	460
R	delete[]	deallocate array	delete[] expr	460
R	noexcept	can expr throw	noexcept (expr)	780

Figure 2.1: Operator Precedence and Associativity Rules in C and C ++

2.2 Control Structures

These should all be familiar to you.

2.2.1 Iteration

In C++ family languages simple iteration is performed with a for loop. In this, we have Initialization, condition and increment as below:

```
for (initialisation ; condition ; incrementation)
{
}
```

A concret example to display numbers from 0 to 9:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
{
    for (Int compteur(0) ; compteur < 10 ; compteur++)
    {
        cout << compteur << endl;
    }
    return 0;
}
```

2.2.2 Conditional

The conditional or if structure looks, in full generality, like this:

```
if (...) {
    ...
}
else if  (...){
    ...
}
else {
    ...
}
```

An example:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
{
    if (b > 3) {
        if (a > 3) b += 1;
    }
    else {
        b -= 1;
    }
    return 0;
}
```

As judged by the indentation used on successive lines, the intent of the writer of this code is the following: If b is greater than 3 and a is greater than 3, then increment b. If b is not greater than 3, then decrement b.

2.2.3 While iteration

Iternative to the for iteration is the while structure, for example:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
{
    while (n < 1000) {
        n *= 2;
        j += 1;
    }
    return 0;
}
```

The control clause (in this case n < 1000) is evaluated before each iteration. If the clause is not true, the enclosed statements will not be executed. In particular, if this code is encountered at a time when n is greater than or equal to 1000, the statements will not even be executed once.

2.2.4 Do-While iteration

Companion to the while iteration is a related control structure that tests its control clause at the end of each iteration:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
{
    do {
        n *= 2;
        j += 1;
    } while (n < 1000);
    return 0;
}
```

In this case, the enclosed statements will be executed at least once, independent of the initial value of n.

Your can read deeply (William H. Press, 2007) and (Nebra Mathieu, 2011) to get more in C++ programming language.

Chapter 3

Vector and Matrix Objects

The C ++ Standard Library (Josuttis, 2012) includes a perfectly good vector<> template class. About the only criticism that one can make of it is that it is so feature-rich that some compiler vendors neglect to squeeze the last little bit of performance out of its most elementary operations, for example returning an element by its subscript. That performance is extremely important in scientific applications (William H. Press, 2007)!

The result of this history is that C ++ , at least now, has a good (but not always reliably optimized) class for vectors and no dependable class at all for matrices or higher-dimensional arrays. What to do? We will adopt a strategy that emphasizes flexibility and assumes only a minimal set of properties for vectors and matrices. We will then provide our own, basic, classes for vectors and matrices(William H. Press, 2007).

For most compilers, these are at least as efficient as vector<> and other vector and matrix classes in common use. But if, for you, they're not, then it is easy to change to a different set of classes, as we will explain.

3.1 C++ Arrays

3.1.1 Static arrays

3.1.1.1 Example 1:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
```

```
{
    Int bestScore[5]; //Declare a table of 5 inters
    Doub anglesTriangle[3]; //Declare a table of  3 doubles

    return 0;
}
```

3.1.1.2 Example 2:

```
#include <iostream>
#include "nr3.h" //This our Numerical Recipses 3 main header
using namespace std;//To indicate in which feature set our source file will draw from

Int main()
{
    Int const tabDim(5); //Dimension of array
    Doub table[tabDim];

    //Supplying the table
    for (Int i(0); i < tabDim; i++){
        cout << "Enter the " << i+1 << "of the table : ";
        cin >> table[i] <<endl;
    }

    //Display the table
    for (Int i(0); i < tabDim; i++){
        cout << table[i] <<endl;
    }

    return 0;
}
```

3.1.1.3 Example 3: Compute the mean

```
#include <iostream>
#include "nr3.h"
using namespace std;

Doub moyenneNotes (Doub tabNotes[], Int dim){
  Doub moyenne(0);
  for(Int i(0); i < dim; i++){
    cout << "Entrer la note " << i+1 << " : ";
    cin >> tabNotes[i];

```

```
    moyenne += tabNotes[i];
  }
  //moyenne /= dim;
  cout << "La moyenne des notes est : " << endl;
  return (moyenne /= dim);
}

Int main(){
  //Doub listNotes[100];
  Int taille;
  //Doub moyenne(0);

  cout << "Saisir le nombre de notes : ";
  cin >> taille;
  Doub listNotes[taille];

  //moyenneNotes(listNotes, taille);

  cout << moyenneNotes(listNotes, taille) << endl;

  return 0;

}
```

3.1.2 C++ Dynamic arrays

Before to use a dynamic array in C++, we should add #include<vector> at the beginning of the program. A *vector* models a dynamic array.

3.1.2.1 Example 1 :

```
#include <iostream>
#include <vector> //Don-t forget!
#include "nr3.h"
using namespace std;

Int main(){
  vector<int> table(5);
  return 0;
}
```

3.1.2.2 Example 2 :

```
#include <iostream>
#include <vector> //Don't forget!
#include "nr3.h"
using namespace std;

Int main(){
  vector<Int> tableau(5, 3); //Creates an array of 5 integers all worth 3

    vector<string> listNames(12, "No name"); //Creates an array of 12 strings all
    worth No name

    vector<Doub> table2; //Creates an array of 0 decimal places

  return 0;
}
```

3.1.3 Abilities of Vectors

Operation	Effect
`vector<Elem> c`	Default constructor; creates an empty vector without any elements
`vector<Elem> c(c2)`	Copy constructor; creates a new vector as a copy of *c2* (all elements are copied)
`vector<Elem> c = c2`	Copy constructor; creates a new vector as a copy of *c2* (all elements are copied)
`vector<Elem> c(rv)`	Move constructor; creates a new vector, taking the contents of the rvalue *rv* (since C++11)
`vector<Elem> c = rv`	Move constructor; creates a new vector, taking the contents of the rvalue *rv* (since C++11)
`vector<Elem> c(n)`	Creates a vector with *n* elements created by the default constructor
`vector<Elem> c(n,elem)`	Creates a vector initialized with *n* copies of element *elem*
`vector<Elem> c(beg,end)`	Creates a vector initialized with the elements of the range [*beg*,*end*)
`vector<Elem> c(initlist)`	Creates a vector initialized with the elements of initializer list *initlist* (since C++11)
`vector<Elem> c = initlist`	Creates a vector initialized with the elements of initializer list *initlist* (since C++11)
`c.~vector()`	Destroys all elements and frees the memory

Figure 3.1: Constructors and Destructor of Vectors

3.1.4 Vector Operations

You can create vectors with and without elements for initialization. If you pass only the size, the elements are created with their default constructor. Note that an explicit call of the default constructor also initializes fundamental types, such as int, with zero (Josuttis, 2012).

Operation	Effect
`c.empty()`	Returns whether the container is empty (equivalent to `size()==0` but might be faster)
`c.size()`	Returns the current number of elements
`c.max_size()`	Returns the maximum number of elements possible
`c.capacity()`	Returns the maximum possible number of elements without reallocation
`c.reserve(num)`	Enlarges capacity, if not enough yet[6]
`c.shrink_to_fit()`	Request to reduce capacity to fit number of elements (since C++11)[6]
`c1 == c2`	Returns whether *c1* is equal to *c2* (calls == for the elements)
`c1 != c2`	Returns whether *c1* is not equal to *c2* (equivalent to `!(c1==c2)`)
`c1 < c2`	Returns whether *c1* is less than *c2*
`c1 > c2`	Returns whether *c1* is greater than *c2* (equivalent to *c2*<*c1*)
`c1 <= c2`	Returns whether *c1* is less than or equal to *c2* (equivalent to `!(c2<c1)`)
`c1 >= c2`	Returns whether *c1* is greater than or equal to *c2* (equivalent to `!(c1<c2)`)

Figure 3.2: Nonmodifying Operations of Vectors

3.1.5 Other example of Using Vectors

Explicit details into (Josuttis, 2012).

```
#include <vector>
#include <iostream>
#include <string>
#include <algorithm>
#include <iterator>
using namespace std;

int main()
{
    // create empty vector for strings
    vector<string> sentence;

    // reserve memory for five elements to avoid reallocation
```

```
14    sentence.reserve(5);
15
16    // append some elements
17    sentence.push_back("Hello,");
18    sentence.insert(sentence.end(),{"how","are","you","?"});
19
20    // print elements separated with spaces
21    copy (sentence.cbegin(), sentence.cend(),
22          ostream_iterator<string>(cout," "));
23    cout << endl;
24
25    // print "technical data"
26    cout << "  max_size(): " << sentence.max_size() << endl;
27    cout << "  size():     " << sentence.size()     << endl;
28    cout << "  capacity(): " << sentence.capacity() << endl;
29
30    // swap second and fourth element
31    swap (sentence[1], sentence[3]);
32
33    // insert element "always" before element "?"
34    sentence.insert (find(sentence.begin(),sentence.end(),"?"),
35                     "always");
36
37    // assign "!" to the last element
38    sentence.back() = "!";
39
40    // print elements separated with spaces
41    copy (sentence.cbegin(), sentence.cend(),
42          ostream_iterator<string>(cout," "));
43    cout << endl;
44
45    // print some "technical data" again
46    cout << "  size():     " << sentence.size()     << endl;
47    cout << "  capacity(): " << sentence.capacity() << endl;
48
49    // delete last two elements
50    sentence.pop_back();
51    sentence.pop_back();
52    // shrink capacity (since C++11)
53    sentence.shrink_to_fit();
54
55    // print some "technical data" again
```

```
    cout << "  size():     " << sentence.size()     << endl;
    cout << "  capacity(): " << sentence.capacity() << endl;
}
```

3.2 NR3 Typedefs

With numerical recipes, the flexibility is achieved by having several layers of typedef type-indirection, resolved at compile time so that there is no run-time performance penalty. The first level of type-indirection, not just for vectors and matrices but for virtually all vari- ables, is that we use user-defined type names instead of C ++ fundamental types. These are defined in **nr3.h**. The complete list of such definitions is below, 3.3 (William H. Press, 2007).

NR Type	*Usual Definition*	*Intent*
`Char`	`char`	8-bit signed integer
`Uchar`	`unsigned char`	8-bit unsigned integer
`Int`	`int`	32-bit signed integer
`Uint`	`unsigned int`	32-bit unsigned integer
`Llong`	`long long int`	64-bit signed integer
`Ullong`	`unsigned long long int`	64-bit unsigned integer
`Doub`	`double`	64-bit floating point
`Ldoub`	`long double`	[reserved for future use]
`Complex`	`complex<double>`	$2 \times$ 64-bit floating complex
`Bool`	`bool`	`true` or `false`

Figure 3.3: Numerical Recipes Typedef

3.2.1 NR3 Typedef: Int, Doub, Char types

3.2.1.1 EXAMPLE 1 : Square of n.

```
#include <iostream>
#include "nr3.h"
using namespace std;

```

```
Int leCarre(Int x){
  /*Int carre(0);
  carre = x * x;
  return carre;*/
  return x * x;
}

Int main(){
  Int x;
  cout << "Entrer un nombre (x) : ";
  cin >> x;
  cout << "x^2 = " << leCarre(x) << endl;
  return 0;
}
}
```

3.2.1.2 EXAMPLE 2 : Summation of a and b.

```
#include <iostream>
#include "nr3.h"
using namespace std;

Int addition(Int a, Int b){
    return a + b;
}

Int main(){
    Doub nbr1(0), nbr2(0), nbr3(0);
    cout << "Saisir le nombre 1 : ";
    cin >> nbr1;
    cout << "Saisir le nombre 2 : ";
    cin >> nbr2;
    cout << "Saisir le nombre 3 : ";
    cin >> nbr3;
    cout << " Le produit de " << nbr1 << " par " << nbr2 << " et par " << nbr3 << " =
     " << multiplication(nbr1, nbr2, nbr3) << endl;

    return 0;
}
```

3.2.1.3 EXAMPLE 3 : Multiplication a by b.

```
#include <iostream>
#include "nr3.h"
```

```
using namespace std;

Doub multiplication(Doub a, Doub b, Doub c){
  return a * b * c;
}

Int main(){
    Doub nbr1(0), nbr2(0), nbr3(0);
    cout << "Saisir le nombre 1 : ";
    cin >> nbr1;
    cout << "Saisir le nombre 2 : ";
    cin >> nbr2;
    cout << "Saisir le nombre 3 : ";
    cin >> nbr3;
    cout << " Le produit de " << nbr1 << " par " << nbr2 << " et par " << nbr3 << " =
     " << multiplication(nbr1, nbr2, nbr3) << endl;

    return 0;
}

\subsubsection{EXAMPLE 4 : Area of the rectangle.}

\begin{lstlisting}[style=CStyle]
#include <iostream>
#include "nr3.h"
using namespace std;

Int surfaceRectange(Int L, Int l){
  Int s(0);
  if(L >= 0 && l >= 0)
    s = L*l;

  return s;
}

Int main(){
  Int longueur, largeur;

  cout << "Saisir la longueur du rectange : ";
  cin >> longueur;
  cout << "Saisir la hauteur du rectange : ";
  cin >> largeur;
```

```
44
45   cout << "La surface du rectangle (s = L * l) est : " << surfaceRectange(longueur,
       largeur) << " m^2 " << endl;
46   //surfaceRectange(longueur, largeur);
47
48   return 0;
49
50 }
```

3.2.1.4 EXAMPLE 5 : Add 2 to n.

```
1  #include <iostream>
2  #include "nr3.h"
3  using namespace std;
4
5  //Int ajoutDeux(Int a){
6  Int ajoutDeux(Int& a){ //passage par reference
7    a += 2; // ==> a = a + 2;
8
9    return a;
10 }
11
12 Int main(){
13     Int nbr(0), resultat(0);
14     cout << "Saisir un nombre : ";
15     cin >> nbr;
16
17     resultat = ajoutDeux(nbr);
18
19     cout <<"Le nombre fourni est : " << nbr << endl;
20     cout <<"Le resultat vaut : " << resultat << endl;
21
22     return 0;
23 }
```

3.2.2 NR3 Typedef: VecInt, VecDoub, VecChar types

The second level of type-indirection returns us to **the discussion of vectors and matrices**. The vector and matrix types that appear in Numerical Recipes source code are as follows.

- **Vectors**: VecInt, VecUint, VecChar, VecUchar, VecCharp, VecLlong, VecUllong, VecDoub, VecDoubp, VecComplex, and VecBool.

- **Matrices**: MatInt, MatUint, MatChar, MatUchar, MatLlong, MatUllong, MatDoub, MatComplex, and MatBool.

NB: These should all be understandable, semantically, as vectors and matrices whose elements are the corresponding user-defined types, above. Those ending in a "p" have elements that are pointers, e.g., VecCharp is a vector of pointers to char, that is, char*.

3.2.2.1 EXAMPLE 1 : Vectors.

```
#include <iostream>
#include "nr3.h"
using namespace std;

Int main(){
  VecDoub a(10);

  for(Int i(0); i < a.size(); i++){
    cout << "Saisir la valeur " << i+1 << " du tableau ";
    cin >> a[i];
  }
  cout << "V(i) = " ;
  cout << "{ ";
  for(Int i(0); i < a.size(); i++){
    cout << a[i];
    if(i < 9)
      cout << ", ";
  }
  cout << "} " << endl;
  return 0;
}
```

3.2.2.2 EXAMPLE 2 : Matrices.

```
#include <iostream>
#include "nr3.h"
using namespace std;

Int main(){
  Int l(3), c(4);
  MatDoub matrix(l, c);

  for(Int i(0); i < l; i++){
```

```
10      for (Int j(0); j < c; j++){
11        cout << "Saisir l'element [" << i << "][" << j << "] de la matrice ";
12        cin >> matrix[i][j];
13      }
14    }
15    cout << "Matrix[l][c] = " << "{ "<<  endl;
16    for(Int i(0); i < l; i++){
17      for (Int j(0); j < c; j++){
18        cout << matrix[i][j]<<  ", ";
19      }
20      cout <<  endl;
21    }
22    cout << "} " << endl;
23    return 0;
24  }
```

References

Josuttis, N. M. (2012). *The c++ standard library second edition.* -AddisonWesle.

Nebra Mathieu, M. S. (2011). *Programmez avec le langage c++.* -AddisonWesle.

William H. Press, W. T. V. B. P. F., Saul A. Teukolsky. (2007). *Numerical recipes the art of scientific computing (third edition).* CAMBRIDGE UNIVERSITY PRESS.