1.8 PREDEFINED C++ FUNCTIONS

A function is a **set of instructions** that processes a set of input data, called parameters or arguments, and obtains a result. When we use functions, they appear in expressions as operands, the value of the operand being actually the result of the function, obtained after processing the current values of the parameters.

For example, in C++ there is no operation to calculate the square root of a real number, for example $\sqrt{5}$. This can be done using the **sqrt function**, by calling *sqrt(5);* this must be done in an expression, for example a display:

main.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main ()
5  {
6      cout<<sqrt(5);
7  }
```

The form of a function

About the sqrt function (and in fact about any functions), some specific information needs to be known in order to know how and when we can use it:

- the name of the function
- the number of parameters
- the type of parameters
- result type

This information is specified through a function **declaration** mechanism called a prototype. For example the sqrt function determines the square root of a real (non-negative) number and its result is also a real number. Its prototype is:

main.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int main ()
5  {
6      double sqrt(double);
7  }
```

Prototypes of functions in the same category are grouped together in a **header file**. It must be included in our program, via the *#include* directive. For example, if we use read/write

operations we will include the iostream header, and if we use math functions we will include the *cmath* header.

Mathematical functions

Name	Header	Prototype	Result
abs	**cstdlib**	**int abs(int x)**	**The absolute value of the argument, | x|, integer**
abs, fabs	**cmath**	**double abs(double x), double fabs(double x)**	**The absolute value of the argument, | x|, real number**
sqrt	**cmath**	**double sqrt(double x)**	**The square root of the argument, √x**
pow	**cmath**	**double pow(double x, double y)**	**Raising to power, x^y**
sin	**cmath**	**double sin(double x)**	**Trigonometric function sine, sinx**
cos	**cmath**	**double cos(double x)**	**Trigonometric function cosine, cosx**
tan	**cmath**	**double tan(double x)**	**The tangent trigonometric function,tanx**
floor	**cmath**	**double floor(double x)**	**Largest integer less than or equal to x**
ceil	**cmath**	**double ceil(double x)**	**Smallest integer greater than or equal to x**

Functions for pseudorandom numbers

Numbers generated in a programming language, by means of an algorithm, are **not random numbers**. They are called **pseudorandom** (falsely random, apparently random).

The C++ language provides mechanisms for generating such numbers:

main.cpp

```
1  #include <iostream>
2  #include <cmath>
3  using namespace std;
4  int rand ()
```

Returns a number in the closed range **[0,RAND_MAX]**, where **RAND_MAX** depends on the compiler, but by default is not less than 32767 (2^{15}-1).

The function is declared in the *cstdlib* file, so the line must be included:

main.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3  using namespace std;
4  int rand ()
```

The following program generates and displays three pseudorandom numbers:

main.cpp

```
1  #include <iostream>
2  #include <cstdlib>
3
4  using namespace std;
5
6  int main()
7  {
8      int n;
9
10     n = rand();
11     cout << n << endl;
12
13     n = rand();
14     cout << n << endl;
15
16     n = rand();
17     cout << n << endl;
18
19     return 0;
20 }
```

Unfortunately, every time the program will generate and display the **same numbers**. Why? Each time we call (use) the *rand()* function, the generated value is calculated based on a previous value. At each call, along with the generation of the result, the value to be used for the next call is also prepared. The value used in the first call of the *rand()* function – called seed, can be specified through the void *srand(int s);* function, where s is the seed value.

The following program will generate three numbers. They will be different from those generated by the previous program, but since the same seed is used, they will be the same every time:

main.cpp

```cpp
#include <iostream>
#include <cstdlib>

using namespace std;

int main()
{
    smile(10); // initialize seed

    int n;

    n = row();
    cout << n << endl;

    n = row();
    cout << n << endl;

    n = row();
    cout << n << endl;

    return 0;
}
```

To solve the problem of repeating generated values, we need to use a different seed each time the program runs. This can be the current computer time, obtained as follows*: time(0)*, the time() function being declared in the *ctime* file:

main.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int main()
{
    srand(time(0)); // initialize seed valid

    int n;

    n = rand();
    cout << n << endl;

    n = rand();
    cout << n << endl;

    n = rand();
    cout << n << endl;

    return 0;
}
```

Generating numbers from a given range

We aim to generate numbers from the range **[0,a)**. A simple solution is to generate some number, then determine the remainder of its division by a, which is sure to be in the given interval.

- *n = rand() % a;*

To generate numbers from the interval **[a,b)**, we will generate a number from the interval **[0,b-a), then we will add the value of a to it**:

- *n = rand() % (b-a) + a;*