This chapter focuses on introducing the C++ programming language. We first describe how C++ was born as an extension of the C programming language. Then we give details on how to generate C++ programs and describe two new features of C++ called reference and overloading. Eventually, we provide a quick outlook of the features of C++ that will be introduced later in class.

1. Connection between C and C++

1.1. Quick history of C++. The C++ programming language has been developed in 1979 by Bjarne Stroustrup in AT&T Bell Labs. The language was originally named "C with classes" till 1983 when it was changed to C++. It became public on 1985 with the publication of "The C++ programming language" by B. Stroustrup. In 1989, a new norm of C++ was released (C++ 2.0). It has been followed by the release of four C++ standard respectively called C++98 (1998), C++03 (2003), C++11 (2011) and C++14 (2014).

1.2. Relation with C. The C++ programming language was developed as an extension of the C programming language that would be more efficient and flexible as it is object oriented. As a consequence, C++ inherits most of C's syntax. For instance, a simple instruction ends with a semi colon ";" or a program starts by executing the instructions of a function called main. Most of C programs are compatible with C++, meaning that a compiler can consider them to be a C++ program and generate an executable that does the same tasks. Note that the compatibility is not 100% as new keywords have been added to C++. Moreover C++ provides extra safety that does not allow the conversion of certain type of variables (like void* to other pointer type).

One of the most well known new feature of C++, as its original name suggests, is the addition of classes. A class defines a new type of data, like a structure in C, that also allows to create new operators involving elements of the class. The C++ programming language brought a lot more new features like references, the overloading of function/operator, the creation of template, new ways to do dynamical memory allocation or handle input/outputs. Note that the previous list is non exhaustive. Some features of C++ have been added to C. We can cite for instance the one line comment with a double slash and the declaration of variables in the middle of a function or in the instructions of a loop for.

2. How to start a C++ program

2.1. Structure and compilation. A C++ program is composed of two kind of files that are:
   - source files with the extension ".cc"
   - header files with the extension ".hh"

Similarly of a C Program (which uses the extension .c and .h), the sources files are used to define the main function and the instructions associated to other functions. As in C, there is only one
function "main" which is executed first by the program. The header files are used to declare function prototypes, structures and classes. They should present header guard to not be included multiple times in the compilation of a source file.

To generate an executable, one can use the compiler g++ as follows:

```
g++ main.cc -o a.exe
```

where we use the option -o so the generated executable is named "a.exe".

2.2. **A first example of C++ program.** The example ex25_C++_hello.cc, available on CAAM 519’s webpage, shows how to print hello world in a terminal with the standard libraries stdio.h and iostream. Here are the instructions:

```c++
#include <cstdio>
#include <iostream>

int main (void)
{
    printf("Hello world with printf from a C++ program\n");
    std::cout << "Hello world" << std::endl;
    return 0;
}
```

Note that the name of C++ libraries does not have extension ".h" or ".hh". The name of a C library can be modified as above by removing the .h and adding c at the beginning. The instructions cout/endl can also be used to print the value of variables without referring to their types with %d or else as it is required by the function printf.

### 3. New features to pass arguments and declare functions

This section introduces two new features of C++ that are connected to the use of functions. Firstly, we introduce a new kind of variable called reference. They are parameters that share similarities with pointers. Then we introduce the notion of overloading functions.

3.1. **References: a new kind of variable.** The C++ programming language introduces the notion of references. A reference is used to create an alias to an existing variable by adding the attribute & during its definition. It is done as follows:

```c++
int i;
int & ref_i = i;
i=5; // ref_i is now 5
ref_i=10; // i is now 10
```

The reference ref_i acts like a pointer that contains the memory address where the variable i is stored. As a consequence, modifying the value of ref_i is equivalent to modify the value of i (and reciprocally). However references are not pointers. Firstly, to access the value stored in a pointer, the programmer has to use the key "*". The value referred by a reference is accessible by typing the name of the reference. Like pointers, it is possible to add the attribute const such that the reference can not be used to modify the value of the original variable. It is done as follows:
C++ PROGRAMMING LANGUAGE: INTRODUCTION.

```
int i=10;
const int & ref_i = i; // ref_i is 10
i=5; // ref_i is now 5
// ref_i=10; // Do not work
```

Unlike pointer, a reference has to be initialized when created. Moreover, it is not possible to change
the memory address associated to a reference. Eventually a reference has to refer to a memory
address that exists (in the Stack or the Heap). Note that a pointer does not need to satisfy the
previous three conditions. Indeed, one can create uninitialized pointer, change address of
pointer and define NULL pointers.

References can be used as a function’s arguments and output. Like pointers, they allow to modify
the value of the given references and the variable they refer to. It can be done with the following
instructions:

```
int i=10;
int & ref_i =i;
add(ref_i);
```

where the function add is defined as follows.

```
void add(int & a){
    a+=1;
}
```

An advantage of reference is that the programmer can not work on the memory associated to the ref-
erence. It avoids problems like memory overread and overwrite. Moreover, the access to a reference
value does not involve dereferencing with the key "*". We refer to the example ex26.references.cc
for more details on the use of reference with array, pointers and functions.

3.2. Overloading: a new feature of C++. C++ allows to define functions that have the same
name. Such feature is call overload and can only be apply to functions that present different:

- number of arguments.
- type of arguments.
- sequence of arguments.

For instance a developer can write a source files with the following functions:

```
void fct_overload(int, double);
void fct_overload(double, int);
void fct_overload(int, int, int);
void fct_overload(int, double, double);
void fct_overload(int*, double*);
```

This feature is possible thanks to how C++ manages name mangling in object files. Unlike C where
only the name of the function is stored, C++ stores more information. First you have the number
of characters of the function’s name, then comes the name of the function. Eventually information
about the first input, then the second input, and so on. Inputs as referred as i for an integer, d for
a double, Pi for a pointer of integer and Ri for a reference of integer. No information on the output
is provided, so it is not possible to differentiate function with regard to the type of their output.
As a consequence, the output’s type of the function in the above example is irrelevant (and could be different for each functions).

We draw your attention on overloading functions that may uses references as arguments. For instance an integer and a reference of integer are two different type of arguments. So one may implement the following prototypes:

```c
void fct_overload(int);
void fct_overload(int &);
```

and see its compiler generates an executable. However if the function is used in the code as follows:

```c
int i=10;
fct_overload(i);
```

we will get a compilation error. Indeed the call is ambiguous to the compiler as the variable ”i” can be interpreted as an integer or as a reference of integer. We refer to the example ex_27a_overloading.cc and ex_27b_overloading.cc for more information.

**Remark:** The overloading feature in C++ can also be used to overload operator. For instance, one can ”redefine” the operator addition ”+” so it can now also works with variables that are not of integer or double types. This feature can be very useful to add or multiply vectors/matrices that are defined with a structure or class.

4. **Outlook of class’s content**

In the following chapters, we aim to introduce the following features of C++.

- Classes. Derived classes.
- Inheritance. Virtual functions.
- Namespaces.
- Polymorphism. Overloading (operators).
- Dynamical memory allocation with new/delete.
- Management of input and output.
- Templates.
- Use of Fortran libraries.
- Management of exception (try/throw/catch).
- Description of some standard libraries.

We also plan to describe how to use the debugger gbd and valgrind.