Compilation

TP 0.1 : A C++ primer

C. Alias & S. Filip

Disaster! The compiler is in C++ and not in Caml. What an idea! In fact, we will see that a lot of processing is imperative in nature and that C++ is quite well suited for this style of programming. What follows is a short introduction to C++ which will prove to be sufficient for most of this class.

Exercise 1. *Object?*

When we program, we have the tendency to place inside one file all the functions which work with one data structure. Here an example, a file called list.h:

```
typedef struct {...} list_t;

list_t* empty_list();
list_t* add(list_t*, int);

int get_nth(list_t, int);
int length(list_t*);
```

The object oriented paradigm offers a syntactic way to make explicit such a grouping. The functions (methods) are syntactically attached to a variable (object) of a certain type (class). Instead of writing add(list, 2), we will use list.add(2)...
The type (the class) of an object is declared in the following way:

```
Fichier List.h:

class List
{
 public:
  int is_empty;
  int head;
  List* tail;

  // the list declaration

  List();          // Constructors
  List(List*,int); //

  void add(int e);
  int get_nth(int);
  int length();
};
```

Inside List.cc, we implement the methods:

```
#include "List.h"

List::List()
{ this->is_empty = true; }

List::List(List* t, int h)
{ is_empty = true; tail = t; head = h; }
```

These 2 methods are *constructors*. They allow for the construction of an object of type (class) List in memory. For example:

```
  List* l1 = new List(); //[], appel au premier constructeur
  List* l2 = new List(new List(),1); //[1]
```

Note that 'this' is a pointer to the object which is being created (returned by the constructor). It can be implicit, as in the second constructor. Inside the methods, 'this' serves as a pointer to the object being modified. Inside l1->add(1), 'this' corresponds to l1.

Try it yourself.

- Copy the code above inside the files List.h and List.cc respectively. Finish implementing the methods.

Exercise 2. *Input/Output*

Copy, compile, test:

```
#include <iostream>

using namespace std;

int main(void)
{
  cout << "Hello" << " world" << endl;
}
```

We should in fact write ((cout « "Hello") « " world") « endl; :

- (cout « "Hello") adds the string "Hello" to the standard output stream (screen) and returns the resulting stream.
- (...) « "world" adds world to the resulting stream. Etc, etc...

In C++, we can define operators (+, -, * ,«, etc) on objects. Let us write a « for our List... Add:

```
inside List.h:
ostream& operator<< (ostream& sout, List& l);

inside List.cc:
ostream& operator<< (ostream& sout, List& l)
{
  if(l.is_empty) return sout;
  sout << l.head << " " << *(l.tail);
  return sout;
}
```

We can thus write `cout « *ma_liste « endl;`... But we can just as well replace cout with a file stream (see the fstream class), and write the list inside a file.

Try it yourself.

- Using your favorite search engine, modify/complete the program to write a list inside a file.

Exercise 3. *STL*

The STL (Standard Template Library) is a comprehensive C++ library which implements a vast majority of useful data structures (like lists and sets). For most day to day programming you will do, using `vector` (contiguous array of elements) and `map` (think of associations lists in Caml) will prove to be sufficient.

Here are some good to know idioms (not necessarily understanding all the details):

```
#include <vector>
#include <map>

using namespace std;

int main(void)
{
  vector<int> vec; //array
  vec.push_back(7); //[7]
  vec.push_back(5); //[7,5]
  cout << vec.size << endl //2
       << vec[0] << endl   //7
       << vec[1] << endl;  //5

  map<string,int> age;
  age["toto"] = 25;
  age["titi"] = 8;
  if(age.find("toto") != age.end())
    //does toto exist ?
    cout << "toto is " << age["toto"] << "years old" << endl;

  for(map<string,int>::const_iterator it = age.begin();
      it != age.end(); ++it)
    //For each pair (name,age) inside the map...
    cout << (*it).first << " -> " << (*it).second << endl;
```

age.begin() returns a pointer (called an iterator) to the first element. age.end() points just after the last element of the map. You can probably guess what the if and for do...

Try it yourself.

- Copy the code and experiment.