Object Oriented Programming with C++
Labwork 5 – *Inheritance*

Exercise 1:

What are the following program results?

```
#include <iostream>
using namespace std;
// ================================================================
class A {
  int na;
 public:
  A( int nn=1 ) : na( nn ) { cout << "$$creates␣object␣A␣" << na << endl; }
};
// ================================================================
class B : public A {
  float xb ;
 public:
  B( float xx=0.f ) : xb( xx ) { cout << "$$creates␣object␣B␣" << xb << endl; }
};
// ================================================================
class C : public A {
  int nc;
public:
  C( int nn=2 ) : A( 2*nn+1 ), nc( nn ) { cout << "$$creates␣object␣C␣" << nc << endl; }
};
// ================================================================
class D : public B, public C {
  int nd;
 public:
  D(int n1, int n2, float x) : B(x), C(n1), nd(n2) { cout<<"$$creates␣object␣D␣"<<nd<<endl; }
};
// ================================================================
int main() {
  D d( 10, 20, 5.f );
  return 0;
}
```

Modify this program such that any D-object contains only once the members of A (in fact, the member `na`). Do it so that the constructor of A is called with value `2*nn+1`, in which `nn` is the argument sent to the constructor of C.

Exercise 2:

Write a class allowing to represent a generic linked list, meaning that data contained into cells are generic pointers `void *`. An element (a cell) must be implemented via a structure containing two members (`next` and `contents`). Class "list" will contains at least the following members:

```
class List {
  ...                 // needs at least to define the class Element
  Element *m_first;
public:
  class Iterator {    // local but public class callable using List::Iterator
    ... // needs some data members
  public:
    const void*head() const; // returns the current head of list
    void next();             // jump to the next element
    bool ended() const;      // true if current scan is ended
  };
  void add(void*);           // adds an element in head of list
  Iterator getIterator() const; // returns a new iterator
};
```

The idea is to be able to write something like:

```
void printList( const List& l ) {
  for( List::Iterator i=l.getIterator(); !i.ended(); i.next() ) { // scan all the elements
    const void*const data = i.head(); // get the current data
    std::cout <<"data␣at␣"<<data<<std::endl;
  }
}
```

Notice that such a list cannot be modified excepting by adding new element at the beginning. Moreover, an iterator can survive to the class, which may lead to strange behavior!

1. Which function members should be constant? Why?
2. Complete this class with missing members (if any), and write their implementations.

Exercise 3:

Let be the following class:

```
class Point {
  int m_x, m_y;
public:
  Point( int x=0, int y=0 ) : m_x(x), m_y(y) {}
  void print() const {
    std::cout<<"Point("<<m_x<<",␣"<<m_y<<")"<<std::endl;
  }
};
```

Write class `PointList` allowing to handle lists of points. In particular, your class will propose the following member:

- A dedicated iterator, that manipulate instance of `Point` (no pointer in its prototype).
- A method to add a `Point` in head of the list.

Take care about memory allocations. In order to verify that no memory leak occurs, you can monitor your classes as in Lecture 4, slide 24.

Exercise 4: Sheeps

The goal is to draw some ASCII pictures. To do that, you have to use the class `Drawing`, whose main method prints some graphical objects. Its main constructor receives its width and height, which are constant member values (so, you need to set them using *initialization list*).

The `Drawing` class knows the objects to draw thanks to a list of primitives. Such primitives are built upon a base class, that should be derived to obtain real forms like rectangle, disc, ellipse ... This "mother" class is named `Graphic`. Its main data members are a background "color" (ASCII mode, so a letter, e.g. 'X', or 'o', *etc.*), a base position, and a very important method saying if the graphic must to be printed for a given position: `bool toPrint( unsigned x, unsigned y)`).

You have to add some derived classes (`Rectangle`, etc.). Each one adds its own specific functionalities, for instance to manipulate rectangle's size.

In the main function, some drawings are proposed. You may add new ones, be creative! You can play with colors, characters, and so on.

Your implementation must work. If it is not the case, may be it is because of the *slicing* problem?