Algorithms and software tools - C++
Lab 2

Exercise 1. *Extension of exercises 1 and 2 of Lab 1…*

1. Modify the class *Date* so that it has an additional attribute which is a *char* * that stores the day of the week ("Monday", "Tuesday",…). Create a copy constructor and a destructor for this class (use functions *strlen* and *strcpy*).

2. Modify the independent functions *before*, *difference* and *duration* so that they use pass-by-reference parameters (instead of pass-by-value).

3. Adapt the entire application accordingly* and compare the creation of objects in the case of pass-by-reference and in the case of pass-by-value.

Exercise 2.

1. Define a class *PPoint* that has as attributes two *pointers to integer x* and *y* (coordinates of the point). Create a constructor that takes as arguments two *integers* used to initialize the coordinates.

2. Create a copy constructor and a destructor for this class.

3. Create a method *add* that adds a PPoint *p* to the current point, and a method *print* that prints the point.

4. Define a class *Array_of_PPoint* that has as attributes: a dynamic array of *PPoint*, and an *unsigned integer* that represents the length of this array. Create a constructor that takes as arguments an unsigned integer *len* and that allocates memory for an array of length *len*. Do you need to modify something regarding the constructor(s) of class *PPoint*? What and why? **

5. Create a method *add* of *Array_of_PPoint* that adds a *PPoint p* to every element of the array, and a method *print_tab* to print the content of the array.

6. If the copy constructor and the destructor print messages on the screen (for instance "Calling copy constructor" and "Calling destructor" respectively), how many messages will be printed when executing the following main function, according to your definition of *add*?

```
int main(){
  Array_of_PPoint a(4);
  PPoint p(2,6);
  a.add(p);
  a.print_tab();
  return 0;
}
```

7. Start documenting your code using the conventions of the Doxgen tool (see http://www.doxygen.nl/).
Then add a target `doc` to your `Makefile` in order to generate the documentation in a `doc/` directory.

---

* We do not really care about the realism of the solution as to the real calendar, the objective of this exercise is just to "play" with a dynamic attribute. If you feel confortable, you can implement a realistic solution thanks to function *mktime* (see here : http://www.cplusplus.com/reference/ctime/mktime/)

** Note. To become familiar with rand(), you can use it to initialize points randomly, for instance in [[0,5]] x [[0,5]]

Exercise 3. *Extension of exercise 4 of Lab 1…*

1. Define a class *PiecewicePoly1* that has as attributes:
- an unsigned integer *npoints* which represents the number of points to be interpolated
- a dynamic array of floats *Xi* which represents an strictly ordered array of *x* coordinates
- a dynamic array of floats *Yi* which represents the values at point *Xi*: Yi = F(Xi)
- a dynamic array *polys* of *Poly1* (polynomials of degree 1) that will interpolate points (Xi, Yi)

2. Define a constructor:

*PiecewisePoly1(const float *Xi, const float *Yi, unsigned int npoints);*

that:
- checks that *npoints* is at least 2
- checks that *Xi* and *Yi* do not contain any NaN values
- checks that *Xi[i] < Xi[i+1]* for all *i* in [[0, *npoints*-2]]
- allocates and copy *Xi* (use *memcpy* defined in <cstring>)
- allocates and copy *Yi*
- allocates *polys*, an array of (*npoints*-1) *Poly1*
- initializes all polynomials such that they interpolate the points given as input:
  If Pi = polys[i], Xl = X[i], Xr = X[i+1], Yl = Y[i], Yr = Y[i+1]
  We have:
  Pi(Xl) = Yl
  Pi(Xr) = Yr
  Which is enough to determine the two coefficients of each polynomial.

3. Define a destructor.

4. Define *float xmin()* and *float xmax()* that return Xi[0] and Xi[npoints-1].

5. Define a method *float val(x)* that:
- returns Yi[0] if x < xmin()
- returns Yi[npoints-1] if x >= xmax()
- returns poly[i].val(x) if Xi[i] <= x < Xi[i+1]

6. Define a method *void dump(std::ostream& os, unsigned int N)* that evaluates the piecewise polynomial between *xmin()* and *xmax()* on *N* points.
This method should dump a point per line in the format: x val(x)

```
os << x << '\t' << val(x) << std::endl;
```

7. Define a *main* that declares and initializes two static arrays *Xi* and *Yi* of a given size *npoints*. Create a PiecewisePoly1(*Xi, Yi, npoints*) and use the *dump* method for 100 points (use *std::cout* as *ostream*).

8. Use the following command in a terminal to plot the result (where ./bin/ex3_poly is your executable):

```
./bin/ex3_poly | gnuplot -p -e "plot '<cat' with points"
```