CENG 242

Homework #4

(Due: May 3rd 2009)

In this homework, you will implement the `QuadTree` class with C++ which was the subject of the third homework also. The `QuadTree` class will store the overloaded operators along with the diagonal corner points which define the location of the quadtree inside its parent.

As described in homework 3, a quadtree consists of 4 quadrants where each quadrant also defines a quadtree until reaching a pixel. A pixel is reached if the quadtree can not partitioned any more. As a data structure a nodes in a quadtree has exactly four children each representing one of the quadrants or no children if it is a pixel. The public methods of the `QuadTree` class can be implemented with the declarations below. For using points Point structure for throwing exceptions Error structures can be implemented.

```
struct Point {

int coordx;

int coordy;

Point (int a, int b) {coordx=a; coordy=b;}

};
```

```
struct Error {

const char *mess;

Error(const char *p) { mess=p;}

};
```

In the case of an operation between two quadtrees with different sizes an exception with `throw Error (“size mismatch”)` is thrown.

A QuadTree instance is constructed by its size value with (0,0) and (size-1, size-1) points being the diagonal corners. A quadtree instance can be created by the copy constructor also.

```
QuadTree (int size);

QuadTree (const QuadTree & rhs);

~QuadTree ();
```

The insertion and deletion of rectangle operations can be called by (p1,p2) being (top-right, bottom-left) or (bottom-left, top-right) points.

```
QuadTree & insertNode (Point p1, Point p2);

QuadTree & deleteNode (Point p1, Point p2);
```

The overloaded binary operators are “+, -, * and =”. For taking the union of a quadtree with another instance + operator, for taking the difference of a quadtree from another instance – operator, for taking the intersection of a quadtree with another instance * operator and lastly assigning a quadtree instance to the quadtree = operators are overloaded with following declarations. For negating a quadtree, the unary operator ~ overloaded. ~ operator does not change the content of the class itself but it returns the negated value of the quadtree.

```
QuadTree  operator- (const QuadTree &);

QuadTree  operator+ (const QuadTree &);

QuadTree  operator* (const QuadTree &);

QuadTree & operator= (const QuadTree &);

QuadTree  operator~ ();
```

For printing the values of the nodes of a quadtree “<<” operator can be overloaded.

```
friend ostream & operator<<(ostream & ost, const Quadtree &
q);
```

Specifications:

- All work should be done **individually**.
- Your codes should be written in C++ and you should tar your source files in **hw4.tar.gz**. The tar file should include **hw4.cpp**, **hw.h** and **Makefile**. In hw4.h, you should include the function prototypes and member variables of the class. In hw4.cpp, you should include the function definitions but **do not include the main function**, since a main function will be used for evaluating your codes.
- In evaluation, black box testing will be used, so be careful about the data type and function names.
- You will submit your code through **Cow** system.
- You should test your codes in **inek** machines by compiling with **g++** before submitting.