CS 7B - Spring 2014 - Final Exam 5/20/14

Write responses on separate paper.

1. Define each of the following terms in the context of this course. If it helps your definition, give an example.
   (a) *pointer*

   (b) *dereferencing a pointer*

   (c) *dangling pointer*

   (d) *dynamic array*

   (e) *abstract data type*

   (f) *template function*

   (g) *template class*

   (h) *standard template library*

   (i) *contiguous memory*

   (j) *overloading*

   (k) *object*

   (l) *instance of an class*

   (m) *data hiding*

   (n) *initialization list*

   (o) *friend*

   (p) *protected*

   (q) *inheritance*

   (r) *polymorphism*

   (s) *virtual function*

   (t) *abstract base class*

2. Carefully analyze the following mystery program. What is output to the console?

```
#include <iostream>
using namespace std;
void  mystery(int v) {
    cout << " " << v;
    if(v == 4) return;
    if(v < 4) mystery(7-v);
    else mystery(9-v);
    cout << " " << 10*v;
}
int main() {
    mystery(1);
    return 0;
}
```

3. The running time of particular sorting algorithm for different sizes of arrays of randomly ordered integers is approximated by the formula (N is the number of integers to sort) time (milliseconds) $= 0.12+0.03N+0.24N^2$ Circle each of the following sorting algorithms that are consistent with the above formula.
   (a) merge sort
   (b) insertion sort
   (c) bubble sort

4. Which one of the following must be true if the statement "`this->foo++;`" is part of a valid C++ program?
   (a) `foo` must be defined as part of a class.
   (b) The statement must be inside a class definition.
   (c) The class being defined contains a method named "`foo`".
   (d) The above code must be inside an class method.
   (e) The local variable named "foo" is a `const`.

5. **Fast Flood Simulation**
   Consider the complete program below which uses recursion simulate fast flooding of a geographic area. The terrain is represented as an evenly-spaced square grid of $\_N$ by $\_N$ miles ($\_N$ is odd), each grid square represents 1 square mile.

   Flooding starts at the upper left region and will recursively attempt to flood the four nearest neighbors (North, South East and West). Water will flow into a neighbor if it is lower and there's water in the source square.

```
#include <ctime>
#include <cstdlib>
#include <iostream>
#include <vector>
#define N 5

using namespace std;

class Cell {
    friend ostream& operator<<(ostream& ostr, const Cell& x) {
        ostr << '(' << x._elev << ", " << x._waterLevel << ')';
        return ostr;
    }
public:
    Cell(unsigned elev, unsigned waterLevel) :
        _elev(elev), _waterLevel(waterLevel) {}
private:
    unsigned _elev; // elevation of cell
    unsigned _waterLevel; // water level
```

```
friend class Grid;
};

class Grid {
public:
    Grid(int n) : _N(n) {
        createGrid();
    }
    void createGrid() {
        for(int i = 0; i != _N*_N; ++i) {
            if(i/_N==0 || i/_N==_N-1 ||
               i%_N==0 || i%_N==_N-1) {
                   Cell c(0,0);
                   _grid.push_back(c);
               }
            else {
                Cell c(40+_N*(rand()%(_N+1)),1);
                _grid.push_back(c);
            }
        }
    }
    void updateGrid(int x) {
        //int row = x/_N, col = x%_N;
        // look N
        if(_grid[x-_N]._elev != 0 && _grid[x]._elev > _grid[x-_N]._elev && _grid[x].
    _waterLevel) {
            _grid[x]._waterLevel-=1;
            _grid[x-_N]._waterLevel+=1;
            updateGrid(x-_N);
        }
        // look E
        if(_grid[x+1]._elev != 0 && _grid[x]._elev>_grid[x+1]._elev && _grid[x]._waterLevel) {
            _grid[x]._waterLevel-=1;
            _grid[x+1]._waterLevel+=1;
            updateGrid(x+1);
        }
        // look S
        if(_grid[x+_N]._elev != 0 && _grid[x]._elev>_grid[x+_N]._elev && _grid[x]._waterLevel)
     {
            _grid[x]._waterLevel-=1;
            _grid[x+_N]._waterLevel+=1;
            updateGrid(x+_N);
        }
        // look W
        if(_grid[x-1]._elev != 0 && _grid[x]._elev>_grid[x-1]._elev && _grid[x]._waterLevel) {
            _grid[x]._waterLevel-=1;
            _grid[x-1]._waterLevel+=1;
            updateGrid(x-1);
        }
    }

    void updateGrid() {
        for(vector<Cell>::iterator i = ++_grid.begin(); i != --_grid.end(); ++i)
            ++(i->_waterLevel);
        updateGrid(_N*_N/2);
    }
    void printGrid() {
        for(vector<Cell>::iterator i = _grid.begin(); i != _grid.end(); ++i) {
            cout << *i << " ";
            if((i - _grid.begin())%_N==_N-1) cout << endl;
        }
    }
private:
    int _N; // grid is _N by _N square
    vector<Cell> _grid; //each cell has an elevation and a water level
```

```
};

int main()
{
    srand(unsigned(time(0)));
    bool keepGoing = 1;
    Grid g(5);
    while(keepGoing) {
        g.printGrid();
        g.updateGrid();
        cin >> keepGoing;
    }

    return 0;
}
```

(a) Give a detailed description for how the constructor for `Grid` works.

(b) On what line of the code is a Cell instantiated?

(c) How is `Grid` made to be a friend of `Cell`? Why is that necessary?

(d) How is `updateGrid()` overloaded? Why is that necessary?

(e) Suppose the original `g` looks like this:

```
(0, 0)  (0, 0)  (0, 0)  (0, 0) (0, 0)
(0, 0) (45, 1) (40, 1) (45, 1) (0, 0)
(0, 0) (65, 1) (55, 1) (60, 1) (0, 0)
(0, 0) (65, 1) (60, 1) (50, 1) (0, 0)
(0, 0)  (0, 0)  (0, 0)  (0, 0) (0, 0)
```

Then after one `updateGrid()` `g` looks like this:

```
(0, 0)  (0, 1)  (0, 1)  (0, 1) (0, 1)
(0, 1) (45, 2) (40, 3) (45, 2) (0, 1)
(0, 1) (65, 2) (55, 1) (60, 2) (0, 1)
(0, 1) (65, 2) (60, 2) (50, 2) (0, 1)
(0, 1)  (0, 1)  (0, 1)  (0, 1) (0, 0)
```

Give a detailed explanation of how `updateGrid()` produces this change.

(f) How might you improve this model for fast flooding?