Middle East Technical University | Department of Computer Engineering

CENG 242

Programming Language Concepts

Spring 2016-2017

Homework 3 - Pokemon Training

Due date: 9 May 2017, Tuesday, 23:55

1 Objective

This homework aims to help you get familiar with the fundamental C++ programming concepts.

2 Problem Definition

In this homework you are going to implement a 'pokemon training' scenario in which you make the trainers ready for Big Pokemon Tournament. In the homework, you are given a big 3D region, say Pokemon Field, which is a rectangular solid consisting of unit cells in some of which there exists a pokemon. Each pokemon trainer wants to catch as many pokemons as s/he can. For that reason, they travel in Pokemon Field and catch the pokemons which they find. Also, trainers sometimes put their pokemons into challenge for them to gain experience of battling.

Here the details of pokemon training are: Each trainer can travel only in some part of Pokemon Field which is allowed him/her to scan. In other words, a trainer can catch only the pokemons existing in his/her field of view. Note that trainers may have intersecting fields of view each. In that case, the trainer who scan the intersecting region before the others catches all the pokemons living in that common area, and the other trainers can not find any pokemon in that area since there does not remain any.

The information related to Pokemon Field is provided by Professor Oak. He gives the coordinates of two crossing cells of Pokemon Field (the corner consisting of the minimum coordinates, and the corner consisting of the maximum coordinates). Next, he informs about which pokemon exists in which cell. From now on, each trainer should build the map of Pokemon Field as nested bounding volumes. The bounding volume data structure is explained in Section 2.1.

The information related to pokemons living in Pokemon Field contains name of the pokemon, type of the pokemon and the evolution level of the pokemon. The last one, the experience level, is the minimum required number of battles in which the pokemon participates to gain experience and evolve. After finishing the pokemon hunting, the trainers put their pokemons into challenge for them to gain experience. After gaining sufficient amount of experience, pokemons may evolve into its next version if wished (when we call evoluation operator inside main).

Pokemon challenges are resulted depending on types of the pokemons. The detailed information related to pokemon types are given in Section 3. Types of the pokemons are regarded only when they are put into a challenge. When a trainer wants to catch a pokemon, its type does not matter, just that the trainer's passing through the cell where the pokemon lives is enough.

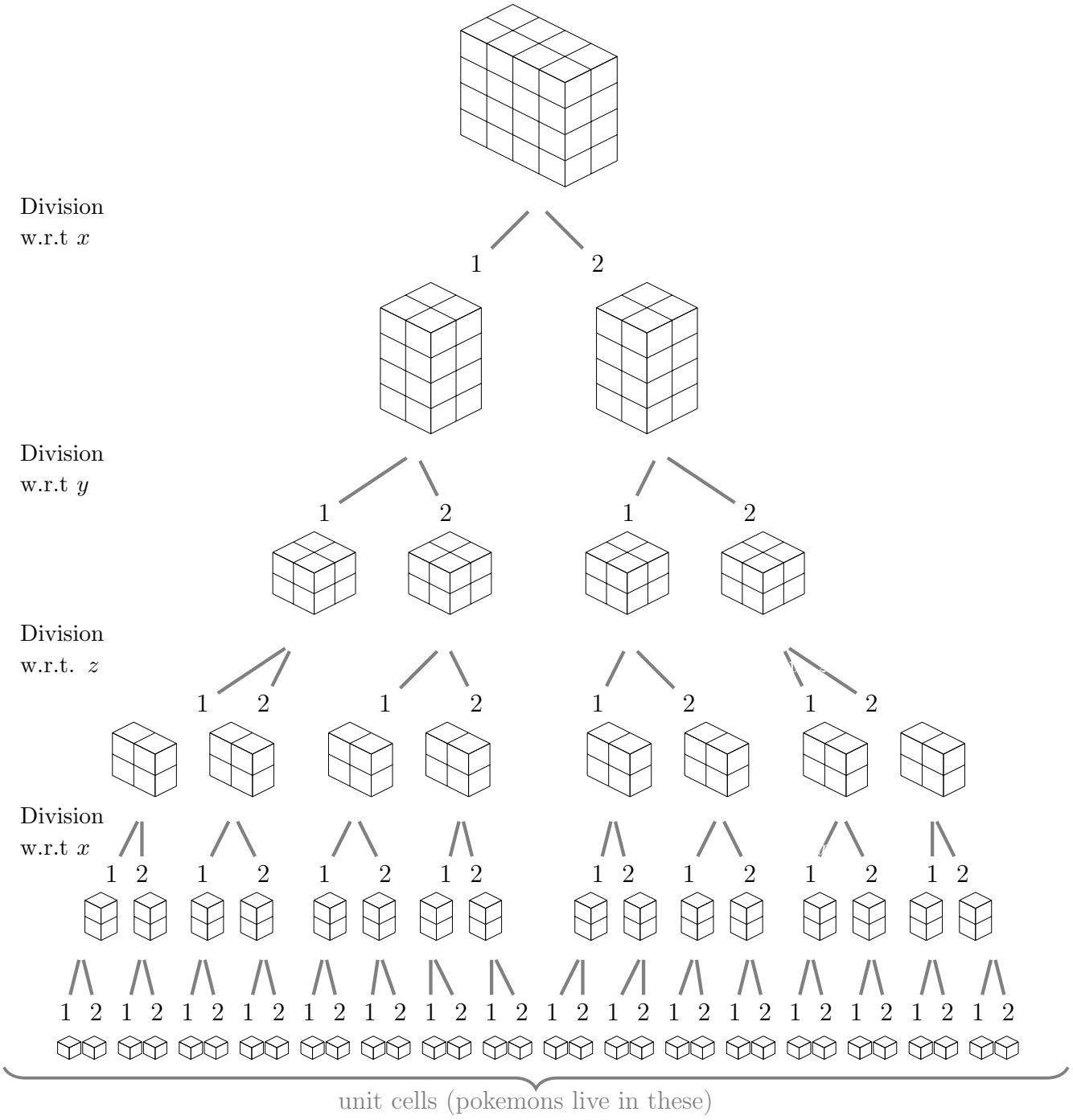

Figure 1: Bounding Volume Hierarchy of the Pokemon Field

2.1 Bounding Volume Hierarchy

1. Pokemon Field itself becomes the initial bounding volume, say B_0.

2. This initial bounding volume B_0 is divided into 2 halves by the x-coordinate, as the "first part" and the "second part", say $B_{1,1}$ and $B_{1,2}$, respectively.

3. Later, these subvolumes $B_{1,1}$ and $B_{1,2}$ are also divided into 2 halves, but with respect to y-coordinate this time. There occurs 4 new bounding volumes: $B_{2,1}$ and $B_{2,2}$ from the division of $B_{1,1}$, and $B_{2,3}$ and $B_{2,4}$ from the division of $B_{1,2}$.

4. The division operation continues for $B_{2,1}$, $B_{2,2}$, $B_{2,3}$ and $B_{2,4}$ with respect to z-coordinate this time. Hence, there occurs 8 new bounding volumes: $B_{3,1}$ and $B_{3,2}$ from the division of $B_{2,1}$, $B_{3,3}$ and $B_{3,4}$ from the division of $B_{2,2}$, $B_{3,5}$ and $B_{3,6}$ from the division of $B_{2,3}$, and $B_{3,7}$ and $B_{3,8}$ from the division of $B_{2,4}$.

5. The division operation continues in the same manner, dividing by x-coordinate again, next by y-coordinate, next by z-coordinate, and so on...

6. The division operation stops when a unit cell is arrived. In other words, the unit cells are not divided into 2. You can think the whole division procedure as a binary tree where each node is a bounding volume and the leaves of tree are the smallest bounding volumes which are the unit cells.

7. The pokemons are hold only in the leaves of the tree, namely in the unit cells.

8. The size of each dimension of Pokemon Field is given as a positive power of 2. In any step of the division procedure, there may occur subvolumes whose size of dimension along x-coordinate resulted in 1 whereas the size of dimensions along y- or z-coordinate is not 1 yet (or similarly, subvolumes whose size of dimension along y-coordinate is 1 whereas the size of dimension along z-coordinate is not 1). In such a case, the division steps are applied in the same order just by skipping the step where it is required to divide the volume by the 1-unit-size coordinate.

9. Only the volumes whose at least one unit cell includes a pokemon are divided. In other words, the resulting tree may not be balanced.

To illustrate, a sample bounding volume hierarchy is shown in Figure 1.

When a trainer needs to scan Pokemon Field, he should follow the bounding volume hierarchy explained above. He goes on the following way: He starts from the initial bounding volume. If the subvolumes include any pokemon (namely in the unit cells of the subvolume), then he checks whether there is an intersection between his own field of view and the subvolumes (first and second parts of the root volume). If his own region intersects with a subvolume B including some number of pokemons, then he should check subvolumes of B with the same method. In this way, he reaches the unit cells by just checking the bounding volumes intersecting with his field of view and containing at least one pokemon in one of its cells.

3 Specifications

You are going to implement three C++ classes: `Pokemon`, `Region` and `Trainer`. For each class, we are providing you with a header file including the declarations for public methods and constructors. You are going to include these header files in your `.cpp` files and provide your constructor, destructor and method bodies in the `.cpp` files.

You are free to define as many private members, methods and constructors as you like in the header files as long as you conform with the specifications below. However, do **NOT** add, remove or modify the given public methods and constructors (doing so will result in severe grade deduction).

	Fire	**Water**	**Grass**	**Electric**	**Flying**
Fire	-	Water	Fire	Fire	Flying
Water	Water	-	Grass	Electric	Water
Grass	Fire	Grass	-	Grass	Flying
Electric	Fire	Electric	Grass	-	Electric
Flying	Flying	Water	Flying	Electric	-

Table 1: Challenge Results Between Pokemons Types

3.1 Pokemon Class

The `Pokemon` class is used for representing the pokemons. You are provided with the following `pokemon.h` header file:

```
#ifndef POKEMON_H
#define POKEMON_H

#include <string>

using namespace std;

class Pokemon {
  private:
    // Add private members, methods and constructors here as you need
  public:
    // Do NOT make any modifications below
    Pokemon(const string&, const string&, int);
    Pokemon(const Pokemon&);
    ~Pokemon();
    const string& getName() const;
    bool operator>>(const Pokemon&);
    friend Pokemon operator&(Pokemon&, Pokemon&);
    Pokemon& operator=(const Pokemon&);
};

#endif
```

Name	Description
`Pokemon::Pokemon(const string&, const string&, int)`	Constructor. The first parameter is the name, the second parameter is the type and the third parameter is the minimum number of battle experiences required to evolve.
`Pokemon::Pokemon(const Pokemon&)`	Copy Constructor.
`Pokemon:: Pokemon()`	Destructor.
`const string& Pokemon::getName() const`	Returns the name of Pokemon object.
`bool Pokemon::operator>>(const Pokemon&)`	Overloaded >> operator. If the Pokemon object has gathered sufficient amount of experience by getting into challenges, then it can be evolved into its next version when this operator is used. Evolution happens by changing the name, type and the evolution level of Pokemon object with those of the one given in the parameter. If it can be evolved, true is returned, otherwise false is returned. Some pokemons may not be evolved into a higher version. For those, the evolution level is given -1.
`friend Pokemon operator&(Pokemon&, Pokemon&)`	Overloaded & operator. It gets 2 pokemons given in the parameter into challenge. The pokemons which get into challenge gain 1 unit of experience. Returns the winner pokemon.
`Pokemon& Pokemon::operator=(const Pokemon&)`	Overloaded assignment operator.

Table 2: Public Constructors and Methods of Pokemon Class

Pokemons are either free in a cell of Pokemon Field, or owned by a trainer. A pokemon can not be owned by more than one trainer. However, there can be many samples of the same pokemon, of course, and each sample can be owned by different trainers. Also, a trainer can have more than one sample of the same pokemon.

A pokemon belongs to exactly one of 5 following types: Fire, Water, Grass, Electric and Flying. While a trainer tries to catch a pokemon, s/he just throws the pokeball on it and catches. The types of pokemons matter only when the trainers want to get them into challenge. In the Table 1, you can see which one of two types surpasses the other. You should implement the `challenge()` method with respect to the rule given that table. Putting the same types of two pokemons into challenge is not a valid battle. Such a case will not be tested.

3.2 Region Class

The `Region` class is used for holding the bounding volumes in a hierarchic way explained in the problem definition part. Each region object may consist of two constructed sub-regions, or one constructed and one null subregion, or two null subregions. The sub-regions which do not include any pokemon in any of its unit cells must not be constructed (null). Note that regions of a unit cell are also null. You are provided with the following `region.h` header file:

```
#ifndef REGION_H
#define REGION_H

#include "pokemon.h"
#include <iostream>
#include <exception>

using namespace std;

class Region {
  private:
    // Add private members, methods and constructors here as you need
  public:
    // Do NOT make any modifications below
    Region(const int[3], const int[3]);
    Region(const Region&);
    ~Region();
    int getMinBorder(char) const;
    int getMaxBorder(char) const;
    void placePokemon(const Pokemon&, int, int, int);
    Pokemon& operator()(int, int, int);
    int getPokemonCount(const int[3], const int[3]) const;
    Region crop(const int[3], const int[3]) const;
    void patch(Region);
    Region& operator=(const Region&);
};

class pokemonException: public exception {
  virtual const char* what() const throw() {
        return "There does not exist any pokemon in the given coordinates!";
  }
};

#endif
```

Name	Description
`Region::Region(const int[3], const int[3])`	Constructor. The first parameter is the minimum x,y,z coordinates of region. The second parameter is the maximum x,y,z coordinates of region.
`Region::Region(const Region&)`	Copy Constructor.
`Region:: Region()`	Destructor.
`int Region::getMinBorder(char) const`	Returns minimum x, minimum y or minimum z coordinate of Region object if the parameter is 'x', 'y' or 'z', repectively.
`int Region::getMaxBorder(char) const`	Returns maximum x, maximum y or maximum z coordinate of Region object if the parameter is 'x', 'y' or 'z', repectively.
`void Region::placePokemon(const Pokemon&, int, int, int)`	Places the pokemon given in the parameter into the unit cell whose x, y, z coordinates are given in the second, third and fourth parameters, respectively. To find the related unit cell, the bounding volume hierarchy must be followed starting from the root.
`Pokemon& Region::operator()(int, int, int);`	Overloaded () operator. Returns the pokemon living in the unit cell whose x, y, z coordinates are given in the parameter. If there is not any pokemon in the specified cell, then a pokemonException is thrown.
`int Region::getPokemonCount(const int[3], const int[3]) const`	Returns the number of pokemons in the sub-region of region object where the minimum and maximum x, y, z coordinates of the sub-region is given in the first and second paremeters, respectively.
`Region Region::crop(const int[3], const int[3]) const`	Returns a copy of the sub-region of region object where the minimum and maximum x, y z coordinates of the sub-region is given in the first and second paremeters, respectively.
`void Region::patch(Region)`	Patches the region given in the parameter as a subregion onto region object by regarding that the corresponding coordinates match.
`Region& Region::operator=(const Region&)`	Overloaded assignment operator.

Table 3: Public Constructors and Methods of Region Class

3.3 Trainer Class

You are provided with the following `trainer.h` header file:

```
#ifndef TRAINER_H
#define TRAINER_H

#include "pokemon.h"
#include "region.h"
#include <string>
#include <vector>
#include <iostream>

using namespace std;

class Trainer {
  private:
    // Add private members, methods and constructors here as you need
```

```
    vector<const Pokemon*> pokemons;
  public:
    // Do NOT make any modifications below
    Trainer(const string&, const Pokemon&, const int[3], const int[3]);
    Trainer(const Trainer&);
    ~Trainer();
    void catchPokemon(const Pokemon& newPokemon) {pokemons.push_back(&newPokemon)↩
        ;};
    void scanRegion(Region&);
    friend ostream& showPokemons(ostream&, const Trainer&);
};

#endif
```

Name	Description
`Trainer::Trainer(const string&, const Pokemon&, const int[3], const int[3])`	Constructor. The first parameter is the name of Trainer, the second parameter is the first pokemon of Trainer. The third and fourth parameters are the minimum and maximum x, y, z coordinates of field of view of the trainer (i.e. the boundaries of the part of Pokemon Field in which trainer is permitted to hunt.
`Trainer::Trainer(const Trainer&)`	Copy Constructor.
`Trainer:: Trainer()`	Destructor.
`void Trainer::catchPokemon(const Pokemon&)`	Trainer catches the pokemon given in the parameter.
`void Trainer::scanRegion(Region&)`	Trainer scans the bounding volume hierarchy of the Region object given in the parameter and catches the pokemons that he finds in the parts which are in his field of view (i.e. in the parts of region which he is permitted to hunt).
`friend ostream& showPokemons(ostream&, const Trainer&)`	This method is used to print the names of the pokemons that a trainer has.

Table 4: Public Constructors and Methods of Trainer Class

4 Regulations

- **Memory-leak:** The class destructors must free all of the used heap memory. Any heap block, which is not freed at the end of the program will result in grade deduction. Please check your codes using `valgrind --leak-check=full` for memory-leaks.
- **Allowed Libraries:** You may include and use C++ Standard Library. Use of any other library (especially the external libraries found on the internet) is forbidden.
- **Programming Language:** You must code your program in C++. Your submission will be compiled with `g++` on department lab machines. You are expected to make sure your code compiles successfully with `g++` using the flags `-ansi -pedantic`.
- **Late Submission:** You have a total of 7 days for late submission. You can spend this credit for any of the assignments or distribute it for all. If total of late submissions exceeds the limit, a penalty of $5 * day * day$ is applied.
- **Cheating:** In case of cheating, the university regulations will be applied.

- **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.
- **Grading:** This homework will be graded out of 100. Please take attention to the followings:
  - The **most important part** of the Homework is implementing class contructors/destructors/-copy constructors correctly. For that reason, a substantial amount of points are given to results of these concepts. Also, note that it is very possible that the other methods may not work correctly if your class contructors, copy constructors and = operator are not well-implemented.
  - You **MUST** use bounding volume data structure in the Homework. Using other type of data structures are both forbidden and insufficient for the test cases that we use in evaluation.
  - Tricky cases like patching a region A onto a region B where A's size in any dimension is larger than B's size in the corresponding dimension will no be tested.
  - You **MUST NOT** build any sub-region which does not include any pokemon in the bounding volume hierarchy. It is a data structure where only the parts whose at least one unit cell includes a pokemon. The empty parts (which does not have any unit cell including some pokemon) of the bounding volume is NULL.

5 Submission

Submission will be done via Ceng Class. Do **NOT** write a `main` function in any one of the files. You are going to submit the following files:

- pokemon.h
- pokemon.cpp
- region.h
- region.cpp
- trainer.h
- trainer.cpp

Your codes must successfully compile and run using the command sequence below:

```
$ g++ -ansi -pedantic pokemon.cpp region.cpp trainer.cpp main.cpp -o hw3
$ ./hw3
```

You are not going to submit a `main.cpp` file. We will use our own `main.cpp` during grading. There is a single `main.cpp` provided above. You may use it while developing and debugging you codes, if you like.