CS302

Assignment 5: Resident Evil Mercenaries

Description

Mercenary mode from Resident Evil 3: Nemesis (of course Capcom did not include that in the remake but whatever). Mercenary mode from resident evil is an onslaught gameplay where you need to fight off zombies and other monsters throughout Raccoon City. Each enemy defeated or civilian rescued adds points to the player. For each monster defeated or civilian rescued a multiplier is applied and summed up, the multipliers are

- Zombies × 1.0
- Spiders × 1.1
- Drain Deimos × 1.2
- Zombie Dogs × 1.3
- Hunters × 1.4
- Nemesis × 1.5
- Civilians Rescued × 2.0

These values are summed up to compute the mercenary score. You will have to maintain a leaderboard for several mercenaries (the mercenary with the highest score is in the lead) using a priority Queue data structure (can be a min or max heap, it is up to you)

```
template <class t1, class t2>
class priorityQ
{
public:
  priorityQ();
  void push_back(const t1&, const t2&);
  void pop_front();
  void increaseKey(const t1&, const t2&);
  t2 get_front_priority() const;
  t1 get_front_key() const;
  bool isEmpty() const;
private:
  struct priorityType
  {
    t1 key;
```

```
    t2 priority;
  };

  void bubbleUp(std::size_t);
  void bubbleDown(std::size_t);

  std::vector<priorityType> heapArray;
  std::unordered_map<t1, std::size_t> itemToPQ;
};
```

Each member contains/performs the following

- `struct prioityType` - each element has a key id and a value
- `std::vector<prioityType> heapArray` - the list of elements that maintains the priority queue
- `std::unordered_map<t1, std::size_t> itemToPQ;` - a map that maps a key to the index in the `heapArray`
- `priorityQ<t1, t2>::priorityQ()` - sets up an empty priority queue, if you wish to have the root start from index 1, then just have `heapArray.resize(1)` in the constructor's body otherwise the constructor's body would be empty
- `void priorityQ<t1, t2>::push_back(const t1& key, const t2& value)` - inserts a new `priotityType` object to the back of `heapArray`, maps this key to the last index of the `heapArray` and then calls `bubbleUp( heapArray.size() - 1 )`
- `void priorityQ<t1, t2>::pop_front()` - assigns the last element of `heapArray` and assigns it to the root position, updates the `itemToPQ` map accordingly, then call `bubbleDown(1)` or `bubbleDown(0)` (depending on whether you designate 1 or 0 as the root)
- `void priorityQ<t1, t2>::increaseKey(const t1& key, const t2& value)` - using the `itemToPQ` map, you update the value field of the element in the `heapArray` and then call `bubbleUp` using the index of where `key` parameter is in `heapArray` (this is a clue that a max heap is most likely the best option to implement this priority queue)
- `t2 priorityQ<t1, t2>::get_front_priority() const` - returns the priority field of the root element
- `t1 priorityQ<t1, t2>::get_front_key() const` - returns the key field of the root element
- `bool priorityQ<t1, t2>::isEmpty() const` - returns `true` if there is nothing in the priority queue and `false` otherwise
- `void priorityQ<t1, t2>::bubbleUp(std::size_t index)` - function that performs the standard bubble up procedure, you compute the parent index (using the index parameter) and compare the parent and child's value fields, and swap if necessary, and continue up the tree until you cannot bubble up any further
- `void priorityQ<t1, t2>::bubbleDown(std::size_t index)` - function that performs the standard bubble down procedure, using the parameter index, compute the left and right children indices and compare their value fields and swap the parent with the correct child and continue until you cannot bubble down any further

Main

You will need to declare the following custom type

```
struct mercType
{
```

```
  //tracks how many of each monsters is
  //defeated and civilian saved for a
  //mercenary
  int zombies;
  int spiders;
  int drainDeimos;
  int zombieDogs;
  int hunters;
  int nemesis;
  int civilians;

  //default constructor, optional but can
  //be helpful
  mercType()
  {

  }

  //do not actual try to overload the
  //question mark operator, you can overload
  //any comparison operator you like and it will
  //be used in your bubbleUp and bubbleDown
  bool operator?(const mercType& rhs) const
  {

  }

  //You might need to overload + or +=
  //operators, would be needed in the
  //increaseKey function
};
```

You will need to declare the following

```
priorityQ<std::string, mercType> mercenaries;
```

Each element of the `priotityQ` contains a mercenary name (`std::string`) and their priority (`mercType` and the comparison operator is used to maintain the leader), you will be given two input files, the first input files contains a list of mercenaries names (one name per line and each line is terminated by an end of line characer) and each mercenary is inserted into the `priorityQ` using its name as the first argument and `mercType()` (a default initial object) as the second argument.

The second input file is a simulation file, each line of the file contains

```
MercenaryName MonsterName/Civilian Amount
```

Each component is separated by a space and each line is terminated by an end of line character. For each line read, you need to update the mercenary's monster/civilian counter (by passing in the appropriate arguments into the increaseKey function), you might need to create a temporary `mercType` and pass that into the second argument of increaseKey and this temporary object is added to the priority field of a mercenary, and of course the mercenary name read on that particular line would be the first argument. And you repeat this for every line of this simulation file.

After each line of the simulation file is processed, if there is a new leader, output the new leader's name and score. Once the entire file is read, output the rank from 1 to the amount of mercenaries along with the mercenary's name with their score.

Specifications

- Use your own custom `priorityQ` object, and use this to maintain all the racers
- Do not modify the `priorityQ` class by adding new members
- Comment your code thoroughly
- Global variables are not permitted
- Do not use the dot operator within the `prioittyQ` class since it's a generic class, use any operator overloaded in the `mercType` instead

Sample Run

```
% g++ main.cpp
% ./a.out

Enter mercenaries file: mercenaries.txt
Enter simulation file: simulation04.txt

We have a new leader: Carlos Score: 4.2

We have a new leader: Ethan Score: 15.0

We have a new leader: Claire Score: 32.0

We have a new leader: Carlos Score: 41.8

We have a new leader: Jill Score: 59.4

We have a new leader: Ada Score: 62.6

We have a new leader: Krauser Score: 102.8

We have a new leader: Ada Score: 114.5

We have a new leader: Krauser Score: 140.6

We have a new leader: Ada Score: 148.6

We have a new leader: Carlos Score: 254.3

We have a new leader: Krauser Score: 260.2

We have a new leader: Wesker Score: 363.6

We have a new leader: Krauser Score: 368.6

We have a new leader: Leon Score: 476.3

We have a new leader: Krauser Score: 479.2

We have a new leader: Leon Score: 490.4

We have a new leader: Ada Score: 504.7
```

```
We have a new leader: Krauser Score: 549.2

We have a new leader: Leon Score: 572.8

We have a new leader: Krauser Score: 583.6

We have a new leader: Leon Score: 584.3

Rank 1
Name: Leon Score: 615.2

Rank 2
Name: Krauser Score: 592.2

Rank 3
Name: Wesker Score: 584.5

Rank 4
Name: Ada Score: 580.8

Rank 5
Name: Mikhail Score: 578.4

Rank 6
Name: Heisenberg Score: 562.0

Rank 7
Name: Nicholai Score: 550.9

Rank 8
Name: Ethan Score: 522.6

Rank 9
Name: Dimitrescu Score: 520.7

Rank 10
Name: Carlos Score: 480.7

Rank 11
Name: Luis Score: 456.8

Rank 12
Name: Jill Score: 451.3

Rank 13
Name: Claire Score: 440.1

Rank 14
Name: Chris Score: 378.4
```

Submission

Submit the source file to code grade by the deadline

References

- Supplemental Video https://youtu.be/ryLy6vPdwIo
- Link to the top image can be found at https://en.wikipedia.org/wiki/Resident_Evil_3:_Nemesis#/media/File:Resident_E