HW2 REPORT

O. Kürşat Karayılan

150140011

In this homework we were asked to create an event-scheduler using MIN-HEAP. Since root of min-heap always minimum one, it is good to use min-heap for event-scheduling problem. When an event comes, we can simply extract root node.

I created a struct first like below. It has name, val for firing time and time variable for stating whether starttime or endtime. If a node is starttime then time equals 0.

```
struct event
{
    string name;
    int val;
    int time;       //0 for starttime and 1 for endtime

};
```

Then I created these two global variables for the simplicity. An array with event type and size to keep heap size.

```
event heap[1000];
int size=0;
```

I created two nodes for each event. One for starttime and one for endtime. Below picture I read event information from events.txt and create nodes according to them.

```
while(!inFile.eof()){
    inFile >> name;
    inFile >> starttime;
    inFile >> endtime;
    insert(name, stoi(starttime), 0);
    insert(name, stoi(endtime), 1);
}
inFile.close();
```

Insert function inserts them to end of the heap. Then while loop replaces it to proper location in heap.

```
void insert(string name, int val, int time){
    int i = size;
    heap[i].name = name;
    heap[i].val = val;
    heap[i].time = time;
    size++;

    while(i != 0 && heap[parent(i)].val > heap[i].val){
        swap(i, parent(i));
        i = parent(i);
    }
}
```

Then I created virtual clock with for loop. When size becomes zero we break the loop. Since root of the heap is next possible event, we compare it with time and if it does not match this means no event. If it matches we have two option. If time is zero this means event started. If time is one it ends. We extract root node from the heap with removeMin function if an event starts or ends.

```
for(int i=1; i<1000; i++){      //virtual clock
    if(size==0){
        cout << "TIME " << i << ": NO MORE EVENTS, SCHEDULER EXITS" << endl;
        break;
    }
    if(heap[0].val != i){
        cout << "TIME " << i << ": NO EVENT" << endl;
    }
    while(heap[0].val == i){
        if(size==0) break;
        if(heap[0].time == 0){
            cout << "TIME " << i << ": " << heap[0].name << " STARTED" << endl;
            removeMin();
        }
        else if(heap[0].time == 1){
            cout << "TIME " << i << ": " << heap[0].name << " ENDED" << endl;
            removeMin();
        }
    }
    if(size==0){
        cout << "TIME " << i << ": NO MORE EVENTS, SCHEDULER EXITS" << endl;
        break;
    }
}
```

In removemin function, If size is one we simply decrease size by one and it becomes empty. Otherwise we make last element of heap first, then heapify it. Heapify function maintain the heap-min property.

```
void removeMin(){
    if(size==1){
        size--;
    }else{
        heap[0] = heap[size - 1];
        size--;
        heapify(0);
    }
}
```

Heapify function looks that certain node and makes sure that it is smallest element between its child. It recursively checks and ensure min-heap property.

```
void heapify(int i){
    int l = left(i);
    int r = right(i);
    int min = i;

    if((l < size) && (heap[l].val < heap[min].val)){
        min = l;
    }
    if((r < size) && (heap[r].val < heap[min].val)){
        min = r;
    }

    if(min != i){
        swap(i, min);
        heapify(min);
    }
}
```