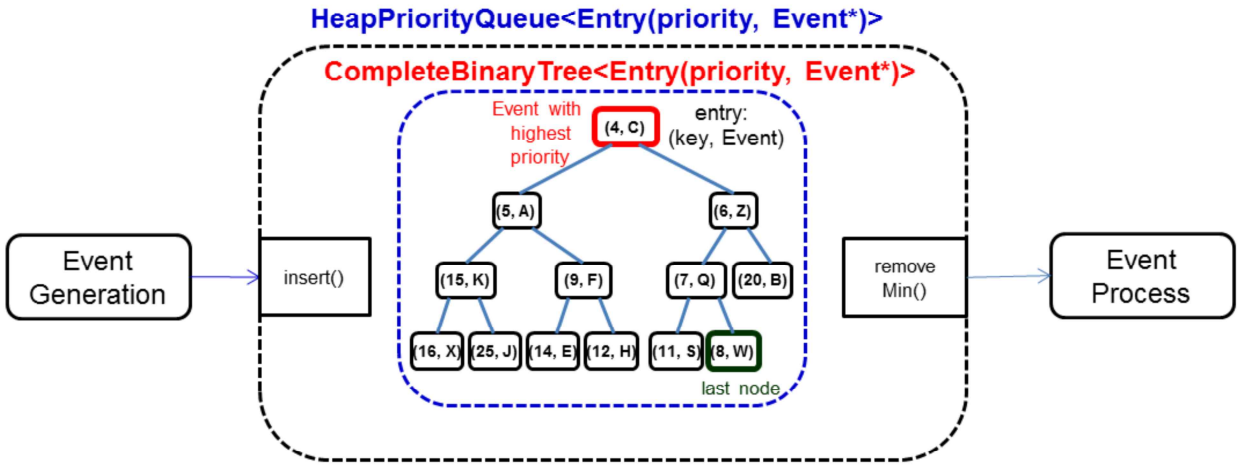
O-O Programming & Data Structure Lab. 8

8.1 Heap Priority Queue for Priority-based Event Handling



(1) class Event

```
enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };

class Event
{
        friend ostream& operator<<(ostream& fout, const Event& e);
public:
        Event(); // default constructor
        Event(int event_id, int event_pri, string title, int gen_addr); //constructor
        void printEvent(ostream& fout);
        void setEventHandlerAddr(int evtHndlerAddr) { event_handler_addr = evtHndlerAddr; }
        void setEventGenAddr(int genAddr) { event_gen_addr = genAddr; }
        void setEventNo(int evtNo) { event_no = evtNo; }
        void setEventPri(int pri) { event_pri = pri; }
        void setEventStatus(EventStatus evtStatus) { eventStatus = evtStatus; }
        int getEventPri() { return event_pri; }
        int getEventNo() { return event_no; }
        bool operator>(Event& e) { return (event_pri > e.event_pri); }
        bool operator<(Event& e) { return (event_pri < e.event_pri); }
private:
        int event_no;
        string event_title;
        int event_gen_addr;
        int event_handler_addr;
        int event_pri;  // event_priority
        EventStatus eventStatus;
};
```

(2) class T_Entry<K, V>

```
template<typename K, typename V>
class T_Entry
{
        friend ostream& operator<<(ostream& fout, T_Entry<K, V>& entry)
        {
                fout << "[" << entry.getKey() << ", " << *(entry.getValue()) << "]";
                return fout;
        }
public:
        T_Entry(K key, V value) { _key = key; _value = value; }
        T_Entry() { _key = 999; } // default constructor
        ~T_Entry() {}
        void setKey(const K& key) { _key = key; }
        void setValue(const V& value) { _value = value; }
        K getKey() const { return _key; }
```

```
        V getValue() const { return _value; }
        bool operator>(const T_Entry& right) { return (_key > right.getKey()); }
        bool operator>=(const T_Entry& right) { return (_key >= right.getKey()); }
        bool operator<(const T_Entry& right) { return (_key < right.getKey()); }
        bool operator<=(const T_Entry& right) { return (_key <= right.getKey()); }
        bool operator==(const T_Entry& right) { return ((_key == right.getKey()) && (_value ==
right.getValue())); }
        T_Entry& operator=(T_Entry& right);
        void fprint(ostream& fout);
private:
        K _key;
        V _value;
}
```

(3) class TA_Entry<K, V>

```
enum SortingDirection { INCREASING, DECREASING };

template<typename K, typename V>
class TA_Entry
{
public:
        TA_Entry(int n, string nm); // constructor
        ~TA_Entry(); // destructor
        int size() { return num_elements; }
        bool empty() { return num_elements == 0; }
        string getName() { return name; }
        void reserve(int new_capacity);
        void insert(int i, T_Entry<K, V> element);
        void remove(int i);
        T_Entry<K, V>& at(int i);
        void set(int i, T_Entry<K, V>& element);
        T_Entry<K, V> getMin(int begin, int end);
        T_Entry<K, V> getMax(int begin, int end);
        void shuffle();
        int sequential_search(T_Entry<K, V> search_key); // search and return the index; -1 if
not found
        int binary_search(T_Entry<K, V> search_key); // search and return the index; -1 if not
found
        void selection_sort(SortingDirection sd);
        void quick_sort(SortingDirection sd);
        void fprint(ofstream &fout, int elements_per_line);
        void fprintSample(ofstream &fout, int elements_per_line, int num_sample_lines);
        bool isValidIndex(int i);
        T_Entry<K, V>& operator[](int index) { return t_GA[index]; }
protected:
        T_Entry<K, V> *t_GA;
        int num_elements;
        int capacity;
        string name;
};
```

(4) class CompleteBinaryTree<K, V>

```
template<typename K, typename V>
class CompleteBinaryTree : public TA_Entry<K, V>
{
public:
        CompleteBinaryTree(int capa, string nm);

        int add_at_end(T_Entry<K, V>& elem);
        T_Entry<K, V>& getEndElement() { return t_GA[end]; }
        T_Entry<K, V>& getRootElement() { return t_GA[CBT_ROOT]; }
        int getEndIndex() { return end; }
        void removeCBTEnd();
        void fprintCBT(ofstream &fout);
        void fprintCBT_byLevel(ofstream &fout);
protected:
        void _printCBT_byLevel(ofstream &fout, int p, int level);
```

```
        int parentIndex(int index) { return index / 2; }
        int leftChildIndex(int index) { return index * 2; }
        int rightChildIndex(int index) { return (index * 2 + 1); }
        bool hasLeftChild(int index) { return ((index * 2) <= end); }
        bool hasRightChild(int index) { return ((index * 2 + 1) <= end); }
        int end;
};
```

(5) class HeapPrioQ<K, V>

```
template<typename K, typename V>
class HeapPrioQueue : public CompleteBinaryTree<K, V>
{
public:
        HeapPrioQueue(int capa, string nm);
        ~HeapPrioQueue();
        bool isEmpty() { return size() == 0; }
        bool isFull() { return size() == capacity; }
        int insert(T_Entry<K, V>& elem);
        T_Entry<K, V>* removeHeapMin();
        T_Entry<K, V>* getHeapMin();
        void fprint(ofstream &fout);
        int size() {return end; }
private:
};
```

8.2 main() function

The main() function should contain following procedure to manage student array.

```
/* main() for Heap Priority Queue based on Complete Binary Tree */
#include <iostream>
#include <fstream>
#include "Task.h"
#include "Event.h"
#include "HeapPrioQ.h"
#include <string>
#include <stdlib.h>
using namespace std;

#define INITIAL_CBT_CAPA 100
#define TEST_HEAP_PRIO_Q_EVENT
#define NUM_EVENTS 15

void main()
{
        ofstream fout;
        string tName = "";
        char tmp[10];
        int priority = -1;
        int current_top_priority;
        int duration = 0;
        int size;
        int *pE;

        fout.open("output.txt");
        if (fout.fail())
        {
                cout << "Fail to open output.txt file for results !!" << endl;
                exit;
        }
        Event events[NUM_EVENTS] =
        {
                //Event(int evt_no, int evt_pri, string title, int gen_addr)
                Event(0, 14, "evt_00", 0), Event(1, 13, "evt_01", 1), Event(2, 12, "evt_02", 2),
                Event(3, 11, "evt_03", 3), Event(4, 10, "evt_04", 4), Event(5, 9, "evt_05", 5),
                Event(6, 8, "evt_06", 6), Event(7, 7, "evt_07", 7), Event(8, 6, "evt_08", 8),
                Event(9, 5, "evt_09", 9), Event(10, 4, "evt_10", 10), Event(11, 3, "evt_11", 11),
                Event(12, 2, "evt_12", 12), Event(13, 1, "evt_13", 13), Event(14, 0, "evt_14",
```

```
14)
        };

        HeapPrioQueue<int, Event*> HeapPriQ_Event(INITIAL_CBT_CAPA,
           string("Event_Heap_Priority_Queue"));
        Event *pEv;
        T_Entry<int, Event*> entry_event, *pEntry_Event;

        for (int i = 0; i<NUM_EVENTS; i++)
        {
                entry_event.setKey(events[i].getEventPri());
                entry_event.setValue(&events[i]);
                HeapPriQ_Event.insert(entry_event);
                fout << "Insert " << events[i];
                fout << " ==> Size of Heap Priority Queue : " << setw(3)
                    << HeapPriQ_Event.size() << endl;
        }

        fout << "Final status of insertions : " << endl;
        HeapPriQ_Event.fprintCBT_byLevel(fout);

        for (int i = 0; i< NUM_EVENTS; i++)
        {
                fout << "\nCurrent top priority in Heap Priority Queue : ";
                pEntry_Event = HeapPriQ_Event.getHeapMin();
                fout <<  *pEntry_Event  << endl;
                pEntry_Event = HeapPriQ_Event.removeHeapMin();
                fout << "Remove " << *pEntry_Event;
                fout << " ==> " << HeapPriQ_Event.size() << " elements remains." << endl;
                HeapPriQ_Event.fprintCBT_byLevel(fout);
                fout << endl;
        }
        fout.close();
} // end main();
```

8.3 Example of Output

```
Insert Event(no:  0; pri: 14; gen:  0, title: evt_00) ==> Size of Heap Priority Queue :   1
Insert Event(no:  1; pri: 13; gen:  1, title: evt_01) ==> Size of Heap Priority Queue :   2
Insert Event(no:  2; pri: 12; gen:  2, title: evt_02) ==> Size of Heap Priority Queue :   3
Insert Event(no:  3; pri: 11; gen:  3, title: evt_03) ==> Size of Heap Priority Queue :   4
Insert Event(no:  4; pri: 10; gen:  4, title: evt_04) ==> Size of Heap Priority Queue :   5
Insert Event(no:  5; pri:  9; gen:  5, title: evt_05) ==> Size of Heap Priority Queue :   6
Insert Event(no:  6; pri:  8; gen:  6, title: evt_06) ==> Size of Heap Priority Queue :   7
Insert Event(no:  7; pri:  7; gen:  7, title: evt_07) ==> Size of Heap Priority Queue :   8
Insert Event(no:  8; pri:  6; gen:  8, title: evt_08) ==> Size of Heap Priority Queue :   9
Insert Event(no:  9; pri:  5; gen:  9, title: evt_09) ==> Size of Heap Priority Queue :  10
Insert Event(no: 10; pri:  4; gen: 10, title: evt_10) ==> Size of Heap Priority Queue :  11
Insert Event(no: 11; pri:  3; gen: 11, title: evt_11) ==> Size of Heap Priority Queue :  12
Insert Event(no: 12; pri:  2; gen: 12, title: evt_12) ==> Size of Heap Priority Queue :  13
Insert Event(no: 13; pri:  1; gen: 13, title: evt_13) ==> Size of Heap Priority Queue :  14
Insert Event(no: 14; pri:  0; gen: 14, title: evt_14) ==> Size of Heap Priority Queue :  15
Final status of insertions :
              [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
          [Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
              [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
     [Key: 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
              [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
          [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
              [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
[Key: 0, Event(no: 14; pri:  0; gen: 14, title: evt_14)]
              [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
          [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
              [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
              [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
          [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
              [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]
```

```
 Current top priority in Heap Priority Queue : [Key: 0, Event(no: 14; pri:  0; gen: 14, title: evt_14)]
 Remove [0, Event(no: 14; pri:  0; gen: 14, title: evt_14)] ==> 14 elements remains.
          [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
               [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
     [Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
               [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
          [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
               [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
 [Key: 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
               [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
          [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
               [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
               [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
          [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
               [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


 Current top priority in Heap Priority Queue : [Key: 1, Event(no: 13; pri:  1; gen: 13, title: evt_13)]
 Remove [1, Event(no: 13; pri:  1; gen: 13, title: evt_13)] ==> 13 elements remains.
          [Key:10, Event(no:  4; pri: 10; gen:  4, title: evt_04)]
     [Key: 3, Event(no: 11; pri:  3; gen: 11, title: evt_11)]
               [Key: 9, Event(no:  5; pri:  9; gen:  5, title: evt_05)]
          [Key: 4, Event(no: 10; pri:  4; gen: 10, title: evt_10)]
               [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
 [Key: 2, Event(no: 12; pri:  2; gen: 12, title: evt_12)]
               [Key: 7, Event(no:  7; pri:  7; gen:  7, title: evt_07)]
          [Key: 6, Event(no:  8; pri:  6; gen:  8, title: evt_08)]
               [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
     [Key: 5, Event(no:  9; pri:  5; gen:  9, title: evt_09)]
               [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
          [Key: 8, Event(no:  6; pri:  8; gen:  6, title: evt_06)]
               [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


. . . . .


 Current top priority in Heap Priority Queue : [Key:11, Event(no:  3; pri: 11; gen:  3, title: evt_03)]
 Remove [11, Event(no:  3; pri: 11; gen:  3, title: evt_03)] ==> 3 elements remains.
      [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
 [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
      [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


 Current top priority in Heap Priority Queue : [Key:12, Event(no:  2; pri: 12; gen:  2, title: evt_02)]
 Remove [12, Event(no:  2; pri: 12; gen:  2, title: evt_02)] ==> 2 elements remains.
 [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
      [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


 Current top priority in Heap Priority Queue : [Key:13, Event(no:  1; pri: 13; gen:  1, title: evt_01)]
 Remove [13, Event(no:  1; pri: 13; gen:  1, title: evt_01)] ==> 1 elements remains.
 [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]


 Current top priority in Heap Priority Queue : [Key:14, Event(no:  0; pri: 14; gen:  0, title: evt_00)]
 Remove [14, Event(no:  0; pri: 14; gen:  0, title: evt_00)] ==> 0 elements remains.
 CBT is EMPTY now !!
```

8.4 Oral Test

(1) 완전 이진 트리 (complete binary tree)와 이진 탐색 트리의 차이점에 대하여 세부 항목별 대조표를 만들어 설명하라.

(2) 힙 우선 순위 큐 (heap priority queue)에 새로운 항목이 추가하기 위한 insert() 함수의 세부 동작을 pseudo code 으로 표현하고, 상세하게 설명하라.

(3) 힙 우선 순위 큐 (heap priority queue)의 포함된 항목 중 가장 우선 순위가 높은 항목을 추출하는 removeMin() 함수의 세부 동작을 pseudo code 으로 표현하고, 상세하게 설명하라.

(4) STL (Standard Template Library)에서 제공되는 Iterator 는 무엇이며, Circular Queue 를 위한 Iterator 인 class CirQ_Iterator 는 어떻게 구현할 수 있는가에 대하여 pseudo code 를 사용하여 설명하라.