**Lab 8**

|  |
| --- |
| 그림입니다. 원본 그림의 이름: YU_UI_RGB-10.png 원본 그림의 크기: 가로 2256pixel, 세로 3047pixel 프로그램 이름 : Adobe ImageReady |

|  |  |
| --- | --- |
| 과목명 | 객체지향프로그래밍과자료구조 |
| 교수님 | 김영탁 교수님 |
| 이 름 | 김주환 |
| 학 번 | 21812158 |
| 일 자 | 2021.10.29.금 |

|  |
| --- |
| /\* main.cpp \*/  /\* Description  \* Heap priority queue 구현  \* Programmed by J. H. Kim  \* Last updated : 2021-10-29 \*/  #include <iostream>  #include <fstream>  #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(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 << endl << "Current 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();  } |
| /\* Evemt.h \*/  #ifndef E\_H  #define E\_H  #include <iostream>  #include <string>  #include <fstream>  #include <iomanip>  using namespace std;  enum EventStatus { GENERATED, ENQUEUED, PROCESSED, UNDEFINED };  #define MAX\_EVENT\_PRIORITY 100  #define NUM\_EVENT\_GENERATORS 10  class Event {  friend ostream& operator<<(ostream& fout, const Event& e);  public:  Event();  Event(int event\_id, int event\_pri, string title, int gen\_addr);  void printEvent(ostream& fout);  void setEventHandlerAddr(int evtHndlerAddr) { event\_handler\_addr = evtHndlerAddr; }  void setEventGendAddr(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;  EventStatus eventStatus;  };  Event\* genRandEvent(int evt\_no);  #endif // !E\_H |
| /\* Event.cpp \*/  #include "Event.h"  Event::Event() {    }  Event::Event(int evt\_id, int evt\_pri, string title, int evtGenAddr) {  event\_no = evt\_id;  event\_gen\_addr = evtGenAddr;  event\_handler\_addr = -1; // event handler is not defined at this moment  event\_pri = evt\_pri; // event\_priority  event\_title = title;  eventStatus = GENERATED;  }  Event\* genRandEvent(int evt\_no)  {  Event\* pEv;  int evt\_prio;  string title = " ";  int evt\_generator\_id;  evt\_prio = rand() % MAX\_EVENT\_PRIORITY;  evt\_generator\_id = rand() % NUM\_EVENT\_GENERATORS;  pEv = (Event\*) new Event(evt\_no, evt\_prio, title, evt\_generator\_id);  return pEv;  }  void Event::printEvent(ostream& fout)  {  fout << "Event(no:" << setw(3) << event\_no << ", pri:" << setw(3) << event\_pri;  fout << ", gen:" << setw(3) << event\_gen\_addr << ", title:" << setw(3) << event\_title << ")";  }  ostream& operator<<(ostream& fout, const Event& evt)  {  fout << "Event(no:" << setw(3) << evt.event\_no << ", pri:" << setw(3) << evt.event\_pri;  fout << ", gen:" << setw(3) << evt.event\_gen\_addr << ", title:" << setw(3) << evt.event\_title << ")";  return fout;  } |
| /\* T\_Entry.h \*/  #ifndef T\_E\_H  #define T\_E\_H  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; }  ~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;  };  template<typename K, typename V>  T\_Entry<K, V>& T\_Entry<K, V>::operator=(T\_Entry<K, V>& right)  {  \_key = right.getKey();  \_value = right.getValue();  return \*this;  }  template<typename K, typename V>  void T\_Entry<K, V>::fprint(ostream& fout)  {  fout << "[Key:" << setw(2) << this->getKey() << ", " << \*(this->getValue()) << "]";  }  #endif // !T\_E\_H |
| /\* TA\_Entry.h \*/  #ifndef TA\_E\_H  #define TA\_E\_H  #include "T\_Entry.h"  enum SortingDirection { INCREASING, DECREASING };  template<typename K, typename V>  class TA\_Entry {  public:  TA\_Entry(int n, string nm);  ~TA\_Entry();  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);  int binary\_search(T\_Entry<K, V> search\_key);  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;  };  template<typename K, typename V>  TA\_Entry<K, V>::TA\_Entry(int n, string nm) {  t\_GA = (T\_Entry<K, V>\*) new T\_Entry<K, V>[n];  if (t\_GA == NULL) {  cout << "Error in creation of dynamic array of size (" << n << ") !!" << endl;  exit;  }  capacity = n;  num\_elements = 0;  name = nm;  }  template<typename K, typename V>  TA\_Entry<K, V>::~TA\_Entry() {  if (t\_GA != NULL) delete[] t\_GA;  }  template<typename K, typename V>  void TA\_Entry<K, V>::reserve(int new\_capacity) {  if (capacity >= new\_capacity)  return; // already big enough  T\_Entry\* t\_newGA = (T\_Entry\*) new T\_Entry[new\_capacity];  if (t\_newGA == NULL)  {  cout << "Error in creation of dynamic array of size (" << new\_capacity << ") !!" << endl;  exit;  }  cout << this->getName() << " expands capacity to " << setw(3)  << new\_capacity << endl;  for (int i = 0; i < num\_elements; i++)  t\_newGA[i] = t\_GA[i];  delete[] t\_GA;  t\_GA = t\_newGA;  capacity = new\_capacity;  }  template<typename K, typename V>  void TA\_Entry<K, V>::insert(int i, T\_Entry<K, V> element) {  if (isValidIndex(i)) {  for (int j = num\_elements - 1; j >= i; j--)  t\_GA[j + 1] = t\_GA[j]; //shift up elements in one position  t\_GA[i] = element;  num\_elements++;  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::remove(int i) {  if (isValidIndex(i))  {  for (int j = i + 1; j < num\_elements; j++)  t\_GA[j - 1] = t\_GA[j]; //shift down elements in one position  num\_elements--;  }  if (num\_elements < (capacity / 2))  {  int new\_capacity = capacity / 2;  V\* t\_newGA = (V\*) new V[new\_capacity];  if (t\_newGA == NULL)  {  return; // new memory allocation failed.  // Just return without modification.  }  cout << this->getName()  << " reduces capacity to " << setw(3)  << new\_capacity << endl;  for (int i = 0; i < num\_elements; i++)  t\_newGA[i] = t\_GA[i];  delete[] t\_GA;  t\_GA = t\_newGA;  capacity = new\_capacity;  }  }  template<typename K, typename V>  T\_Entry<K, V>& TA\_Entry<K, V>::at(int i) {  if (isValidIndex(i))  {  return t\_GA[i];  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::set(int i, T\_Entry<K, V> element) {  if (isValidIndex(i))  {  t\_GA[i] = element;  }  }  template<typename K, typename V>  T\_Entry<K, V> TA\_Entry<K, V>::getMin(int begin, int end) {  V minValue;  int index\_min;  minValue = t\_GA[begin];  index\_min = begin;  for (int i = begin + 1; i <= end; i++)  {  if (t\_GA[i] < minValue) // T must provide operator<() overloading !!  {  minValue = t\_GA[i];  index\_min = i;  }  }  return t\_GA[index\_min];  }  template<typename K, typename V>  T\_Entry<K, V> TA\_Entry<K, V>::getMax(int begin, int end) {  V maxValue;  int index\_max;  maxValue = t\_GA[begin]; index\_max = begin; for (int i = begin + 1; i <= end; i++)  {  if (t\_GA[i] > maxValue) // T must provide operator>() overloading !!  {  maxValue = t\_GA[i];  index\_max = i;  }  }  return t\_GA[index\_max];  }  template<typename K, typename V>  void TA\_Entry<K, V>::shuffle() {  srand(time(0));  int index1, index2;  int rand\_1, rand\_2;  V temp;  for (int i = 0; i < num\_elements; i++)  {  rand\_1 = rand();  rand\_2 = rand();  index1 = ((rand\_1 << 15) | rand\_2) % num\_elements;  rand\_1 = rand();  rand\_2 = rand();  index2 = ((rand\_1 << 15) | rand\_2) % num\_elements;  temp = t\_GA[index1];  t\_GA[index1] = t\_GA[index2];  t\_GA[index2] = temp;  }  }  template<typename K, typename V>  int TA\_Entry<K, V>::sequential\_search(T\_Entry<K, V> search\_key) {  int index;  K key;  if (search\_key == "ST\_ID") { // student\_ID  for (int index = 0; index < num\_elements; index++) {  t\_GA[index].getKey(search\_key, &key);  if (key == search\_key)  return index;  }  }  return -1;  }  template<typename K, typename V>  int TA\_Entry<K, V>::binary\_search(T\_Entry<K, V> search\_key) {  K key;  int low, mid, high; int loop = 1;  low = 0; high = num\_elements - 1;  while (low <= high) {  cout << setw(2) << loop << "-th loop: current search range [" << setw(3) << low << ", " << setw(3) << high << "]" << endl;  mid = (low + high) / 2;  t\_GA[mid].getKey(search\_key, &key);  if (key == search\_key)  return mid;  else if (key > search\_key)  high = mid - 1;  else  low = mid + 1; loop++;  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::selection\_sort(SortingDirection sd) {  int index\_min, index\_max; // index of the element with minimum value  T\_Entry<K, V> tempElement;  K minKey, maxKey, key;  for (int i = 0; i < num\_elements - 1; i++) {  if (sd == INCREASING) { // sorting in increasing (non\_decreasing) order  index\_min = i;  key = t\_GA[i].getKey(); // 키 반환  minKey = (K)key;  for (int j = i + 1; j < num\_elements; j++) {  key = t\_GA[j].getKey();  if ((K)key < minKey) {  index\_min = j;  minKey = (K)key;  }  }  if (index\_min != i) { // if a smaller element is found, then swap  tempElement = t\_GA[index\_min];  t\_GA[index\_min] = t\_GA[i];  t\_GA[i] = tempElement;  }  }  else { // sorting in decreasing (non\_increasing) order  index\_max = i;  key = t\_GA[i].getKey();  maxKey = (K)key;  for (int j = i + 1; j < num\_elements; j++) {  key = t\_GA[j].getKey();  if ((K)key > maxKey) {  index\_max = j;  maxKey = (K)key;  }  }  if (index\_max != i) { // if a smaller element is found, then swap  tempElement = t\_GA[index\_max];  t\_GA[index\_max] = t\_GA[i];  t\_GA[i] = tempElement;  }  }  } // end for  }  template<typename T>  void \_quick\_sort(T\* array, int size, int left, int right,  SortingDirection sd = INCREASING) {  int pI, newPI; // pivot index  if (left >= right)  {  return;  }  else  {//select a pI (pivotIndex) in the range left ≤ pI ≤ right  pI = (left + right) / 2;  }  newPI = \_partition(array, size, left, right, pI, sd);  if (left < (newPI - 1)) {  \_quick\_sort(array, size, left, newPI - 1, sd);  // recursively sort elements on the left of pivotNewIndex  }  if ((newPI + 1) < right) {  \_quick\_sort(array, size, newPI + 1, right, sd);  // recursively sort elements on the right of pivotNewIndex  }  }  template<typename K, typename V>  void TA\_Entry<K, V>::quick\_sort(SortingDirection sd) {  \_quick\_sort(t\_GA, size, 0, num\_elements - 1, sd);  }  template<typename K, typename V>  void TA\_Entry<K, V>::fprint(ofstream& fout, int elements\_per\_line) {  int count = 0;  while (count < num\_elements)  {  for (int i = 0; i < elements\_per\_line; i++)  {  fout << t\_GA[count] << " ";  count++;  if (count % elements\_per\_line == 0)  fout << endl;  }  }  cout << endl;  }  template<typename K, typename V>  void TA\_Entry<K, V>::fprintSample(ofstream& fout, int elements\_per\_line, int num\_sample\_lines) {  string T\_type;  int last\_block\_start;  int count = 0;  T\_type = typeid(T\_Entry<K, V>).name();  for (int i = 0; i < num\_sample\_lines; i++)  {  for (int j = 0; j < elements\_per\_line; j++)  {  if (count >= num\_elements)  {  fout << endl;  return;  }  if ((T\_type == string("int")) || (T\_type == string("double")) ||  (T\_type == string("class std::basic\_string<char,struct std::char\_traits<char>,class std::allocator<char> > ")))  fout << setw(10) << t\_GA[count];  else  fout << t\_GA[count] << " ";  count++;  } fout << endl;  }  if (count < (num\_elements - elements\_per\_line \* num\_sample\_lines))  count = num\_elements - elements\_per\_line \* num\_sample\_lines;  fout << " . . . . . " << endl;  for (int i = 0; i < num\_sample\_lines; i++)  {  for (int j = 0; j < elements\_per\_line; j++)  {  if (count >= num\_elements)  {  fout << endl;  return;  }  if ((T\_type == string("int")) || (T\_type == string("double")) ||  (T\_type == string("class std::basic\_string<char, struct std::char\_traits<char>, class std::allocator<char> > ")))  fout << setw(10) << t\_GA[count];  else  fout << t\_GA[count] << " ";  count++;  } fout << endl;  } fout << endl;  }  template<typename K, typename V>  bool TA\_Entry<K, V>::isValidIndex(int i) {  if ((i < 0) || (i > num\_elements))  return false;  else  return true;  }  #endif // !TA\_E\_H |
| /\* CompleteBinaryTree.h \*/  #ifndef CBT\_H  #define CBT\_H  #include "TA\_Entry.h"  #include "T\_Entry.h"  #define CBT\_ROOT 1  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 this->t\_GA[end]; }  T\_Entry<K, V>& getRootElement() { return this->t\_GA[CBT\_ROOT]; }  int getEndIndex() { return end; }  void removeCBTEnd();  void fprintCBT(ofstream& fout);  void fprintCBT\_byLevel(ofstream& fout);  protected:  void \_fprintCBT\_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;  };  template<typename K, typename V>  CompleteBinaryTree<K, V>::CompleteBinaryTree(int capa, string nm)  :TA\_Entry<K, V>(capa, nm)  {  end = 0; // reset to empty  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::fprintCBT(ofstream& fout)  {  if (end <= 0)  {  fout << this->getName() << " is empty now !!" << endl;  return;  }  int count = 0;  for (int i = 1; i <= end; i++)  {  fout << setw(3) << this->t\_GA[i] << endl;  //if ((((count + 1) % 10) == 0) && (i != end))  //fout << endl;  count++;  }  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::\_fprintCBT\_byLevel(ofstream& fout, int index, int level)  {  int index\_child;  if (this->hasRightChild(index))  {  index\_child = this->rightChildIndex(index);  this->\_fprintCBT\_byLevel(fout, index\_child, level + 1);  }  for (int i = 0; i < level; i++)  fout << " ";  this->t\_GA[index].fprint(fout);  fout << endl;  if (this->hasLeftChild(index))  {  index\_child = this->leftChildIndex(index);  this->\_fprintCBT\_byLevel(fout, index\_child, level + 1);  }  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::fprintCBT\_byLevel(ofstream& fout)  {  if (end <= 0)  {  fout << "CBT is EMPTY now !!" << endl;  return;  }  \_fprintCBT\_byLevel(fout, CBT\_ROOT, 0);  }  template<typename K, typename V>  int CompleteBinaryTree<K, V>::add\_at\_end(T\_Entry<K, V>& elem)  {  if (end >= this->capacity)  {  cout << this->getName() << " is FULL now !!" << endl;  return end;  }  end++;  this->t\_GA[end] = elem;  return end;  }  template<typename K, typename V>  void CompleteBinaryTree<K, V>::removeCBTEnd()  {  end--;  this->num\_elements--;  }  #endif // !CBT\_H |
| /\* HeapPrioQ.h \*/  #ifndef HPQ\_H  #define HPQ\_H  #include "CompleteBinaryTree.h"  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() == this->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 this->end; }  private:  };  template<typename K, typename V>  HeapPrioQueue<K, V>::HeapPrioQueue(int capa, string nm)  :CompleteBinaryTree<K, V>(capa, nm)  { }  template<typename K, typename V>  HeapPrioQueue<K, V>::~HeapPrioQueue()  { }  template<typename K, typename V>  void HeapPrioQueue<K, V>::fprint(ofstream& fout)  {  if (size() <= 0)  {  fout << "HeapPriorityQueue is Empty !!" << endl;  return;  }  else  CompleteBinaryTree::printCBT(fout);  }  template<typename K, typename V>  int HeapPrioQueue<K, V>::insert(T\_Entry<K, V>& elem)  {  int index, parent\_index;  T\_Entry<K, V> temp;  if (isFull())  {  cout << this->getName() << " is Full !!" << endl;  return size();  }  index = this->add\_at\_end(elem);  /\* up-heap bubbling \*/  while (index != CBT\_ROOT)  {  parent\_index = this->parentIndex(index);  if (this->t\_GA[index].getKey() > this->t\_GA[parent\_index].getKey())  break;  else  {  temp = this->t\_GA[index];  this->t\_GA[index] = this->t\_GA[parent\_index];  this->t\_GA[parent\_index] = temp;  index = parent\_index;  }  }  return size();  }  template<typename K, typename V>  T\_Entry<K, V>\* HeapPrioQueue<K, V>::getHeapMin()  {  T\_Entry<K, V>\* pMinElem;  if (size() <= 0)  {  return NULL;  }  pMinElem = (T\_Entry<K, V>\*) new T\_Entry<K, V>;  \*pMinElem = this->getRootElement();  return pMinElem;  }  template<typename K, typename V>  T\_Entry<K, V>\* HeapPrioQueue<K, V>::removeHeapMin()  {  int index\_p, index\_c, index\_rc;  T\_Entry<K, V>\* pMinElem;  T\_Entry<K, V> temp, t\_p, t\_c;  int HPQ\_size = size();  if (HPQ\_size <= 0)  {  return NULL;  }  pMinElem = (T\_Entry<K, V>\*) new T\_Entry<K, V>;  \*pMinElem = this->getRootElement();  if (HPQ\_size == 1)  this->removeCBTEnd();  else {  index\_p = CBT\_ROOT;  this->t\_GA[CBT\_ROOT] = this->t\_GA[this->end];  this->end--;  /\* down-heap bubbling \*/  while (this->hasLeftChild(index\_p))  {  index\_c = this->leftChildIndex(index\_p);  index\_rc = this->rightChildIndex(index\_p);  if (this->hasRightChild(index\_p) && (this->t\_GA[index\_c] > this->t\_GA[index\_rc]))  index\_c = index\_rc;  t\_p = this->t\_GA[index\_p];  t\_c = this->t\_GA[index\_c];  if (t\_p > t\_c)  {  //swap(index\_u, index\_c);  temp = this->t\_GA[index\_p];  this->t\_GA[index\_p] = this->t\_GA[index\_c];  this->t\_GA[index\_c] = temp;  index\_p = index\_c;  }  else  break;  } // end while  }  return pMinElem;  }  #endif // !HPQ\_H |
|  |

**2. 2021-2 객체지향형 프로그래밍과 자료구조 실습 Oral Test**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 학번 | 21812158 | 성명 | 김주환 | 점수 |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (1) 완전 이진 트리와 이진 탐색 트리의 차이점에 대하여 세부 항목별 대조표를 만들어 설명하라.   |  |  |  | | --- | --- | --- | |  | 완전 이진 트리 | 이진 탐색 트리 | | 동작 원리 | 마지막 level의 좌측 node에 삽입하거나 추출하는 원리로 동작한다. | 키 값을 기준으로 조건부 정렬  왼쪽 서브 트리 < Key < 오른쪽 서브 트리 | | 생성 | 마지막 node 다음에 추가하고 level의 마지막이라면 다음 level의 가장 왼쪽 node부터 채운다. | 원하는 event 삽입  root부터 차례대로 비교해가면서 조건에 부합하는 위치를 찾아서 삽입한다. | | 추출 | 마지막 node 제거 | 원하는 event를 추출  추출하고 아래 부분을 조건에 부합하도록 정렬 | | 복잡도 | O(log n) | O(n) | | 특징 | level이 2^n으로 구성된다.  각 부모node는 2개의 자식 node로 구성된다. | 일반적인 배열보다 빠른 탐색이 가능하다.  하지만 크기를 기준으로 정렬하기 때문에 운이 안좋은 경우 일반적인 배열과 같이 트리가 구성될 수 있다. 이 경우 탐색 속도가 느려진다. | | 탐색 | 빠르다  정렬이 된 상태기 때문이다. | 느리다  기준에 따라 원하는 node를 찾을 경우 높이만큼 시간이 걸리기 때문이다. | | 삽입/제거 | 느리다  마지막 node에 대해 삽입 제거를 실행하면서 정렬도 같이하기 때문이다. | 빠르다  기준에 따라 위치를 찾아서 삽입 제거하기 때문이다. | |
| (2) 힙 우선 순위 큐에 새로운 항목이 추가하기 위한 insert() 함수의 세부 동작을 pseudo code로 표현하고, 상세하게 설명하라.   |  | | --- | | **Procedure insert(T\_Entry)**   1. // declare arguments : index, parent\_index 2. if (isFull()) return; 3. index = add\_at\_end(elem); 4. // up-heap bubbling 5. while (index != 1) { 6. parent\_index = index / 2; // use func parentIndex(); 7. if (Key of index > Key of parent\_index) break; 8. else swap(); // swap between parent and child 9. } 10. return;   **End of Procedure insert(T\_Entry)** |   up-heap bubbling   1. 노드 생성 2. 생성된 노드와 부모 노드의 우선 순위 비교 3. 새로운 노드의 우선순위가 더 높은 경우 swap 4. 2 ~ 3의 과정을 반복 5. 새로운 노드가 root에 도달한 경우 break   3.2 새로운 노드의 우선순위가 더 낮은 경우 break |
| (3) 힙 우선 순위 큐의 포함된 항목 중 가장 우선 순위가 높은 항목을 추출하는 removeMin() 함수의 세부 동작을 pseudo code로 표현하고, 상세하게 설명하라.   |  | | --- | | **Procedure removeHeapMin(void)**   1. // declare arguments : index\_p, index\_c, index\_rc 2. MinElem, size 3. if (size <= 0) return; 4. MinElem = rootElem; 5. if (size == 1) removeEnd(); 6. else { 7. arr[Root] = arr[end]; 8. end--; 9. // down-heap bubbling 10. while (hasLeftChild(index\_p)) { 11. if (hasRightChild(index\_p) &&(arr[index\_c] > arr[index\_rc])) 12. index\_c = index\_rc; 13. if (arr[index\_p] > arr[index\_c]) swap(); 14. else break; 15. } 16. } 17. return;   **End of Procedure removeHeapMin(void)** |   down-heap bubbling   1. root값 추출 2. 기존 트리의 마지막 node를 root로 지정 3. 부모 노드와 child의 우선순위 비교 4. 부모의 우선순위가 더 높을 경우 break   4.2 child의 우선순위가 더 높은 경우, 높은 우선순위를 가지는 child와 swap  5. 3 ~ 4 반복  6. 최초 root로 지정해준 node가 last node에 위치한다면 break |
| (4) STL에서 제공되는 iterator는 무엇이며, Circular Queue를 위한 Iterator인 CirQ\_Iterator는 어떻게 구현할 수 있는가에 대하여 pseudo code를 사용하여 설명하라.   |  |  | | --- | --- | | iterator | begin(), end(), prev(), next(), advance(), distance()  algorithm이 container로 순차적으로 접근하게 해준다. (포인터와 유사)  순차적인 접근을 위해 증감연산자와 \*을 통해 값을 읽어올 수도 있다. |  |  | | --- | | **Procedure class CirItor**   1. // declare arguments : T\* pE, T\* pBegin, T\* pEnd 2. class CirItor { 3. CirItor(); // Constructor 4. T& operator\*() { return \*pE } 5. CirItor operator++() { 6. if (pE == pEnd) pE = pBegin; 7. else ++pE; 8. return (\*this); 9. } 10. CirItor operator—() { 11. if (pE == pBegin) pE = pEnd; 12. else –pE; 13. return (\*this); 14. } 15. bool operator==(const CirItor& p) { return (pE == p.pE); } 16. bool operator!=(const CirItor& p) { return (pE != p.pE); } 17. };   **End of Procedure class CirItor** |     생성자   1. 주어진 초기값으로 초기화한다. 2. 초기값이 없다면 default constructor를 실행한다.   포인터   1. 해당 주소의 값을 반환한다.   증/감 연산자(++, --)   1. 현재 마지막/최초 위치라면 최초/마지막 위치로 변경 2. 나머지 경우 다음/이전 위치로 변경   비교 연산자(==, !=)   1. this의 위치와 전달받은 위치가 같은/다른지 판단 2. 같다/다르다면 True 반환 |