FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

YEAR II, Second Semester, Midterm Examination, February 9, 2015, Time 13:00 – 16:00

ชื่อ-นามสกุล_____เลขประจำตัว | | | | | | | | | 2 | 1 | CR58_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 9 ข้อในกระดาษคำถามคำตอบจำนวน 8 แผ่น 8 หน้า คะแนนเต็ม 87 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้ที่ประสงค์จะออกจากห้องสอบก่อนหมดเวลาสอบ แต่ต้องไม่น้อยกว่า 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์
   มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

---

1. (8 คะแนน) จงตั้งชื่อฟังก์ชัน (ที่ได้ความหมายเดียวกับที่ทำงาน) พร้อมทั้งเขียนอธิบายเพิ่มเติมอีกเล็กน้อยได้ในช่องว่างด้านขวา

```
bool removeMiddleElement (vector<int>& v ) {
  v.remove( v.begin() + v.size()/2 );
}
```

ตัวอย่าง

ลบข้อมูลตัวตรงตำแหน่งกลาง ๆ ของ vector v ออกไป

```
set<int> _____(map<int,string>& m) {
    map<int,string>::iterator i = m.begin();
    set<int> s;
    while(i!=m.end()) {
      s.insert(i->first);
      i++;
    }
    return s;
}
```

.....

```
int _____(set<int>& s) {
    set<int>::iterator i = s.begin();
    for (size_t k=0; k<s.size(); k+=2) i++;
    return *i;
}
```

.....

```
vector<int> _____ (vector<int> v) {
    set<int> s(v.begin(),v.end());
    vector<int> newv(s.begin(),s.end());
    return newv;
}
```

.....

```
int _____(vector<int> v) {
    vector<int> t(v);
    sort(t.begin(),t.end());
    int c = 1, x = t[0];
    for (size_t i=1; i<t.size(); i++)
        if (x != t[i]) { c++; x = t[i]; }
    return c;
}
```

.....

ชื่อ_____หมายเลขประจำตัว_____เลขที่ใน CR58_____

2. (16 คะแนน) จงเขียนรายละเอียดการทำงานของเมท็อดต่อไปนี้ให้ตรงตามที่เขียนใน comment

```
void remove_2nd_max( priority_queue<int>& pq ) {
// ลบตัวที่มีค่ามากที่สุดเป็นอันดับสองใน pq ออกไป (ให้ถือว่า pq.size() มีค่าไม่น้อยกว่า 2 แน่ๆ)









}
```

```
void mafiaPush( queue<int>& q, int element ) {
// เพิ่ม element เข้าไปในแถวคอย q แต่ว่านี่เป็นการเพิ่มไปที่ "หัวคิว" (ไม่ใช่ต่อท้ายตามปกติทั่วไป









}
```

```
bool less( pair<int,double>& a, pair<int,double>& b ) {
// ตรวจสอบว่า a "น้อยกว่า" b หรือไม่ โดยการเปรียบเทียบน้อยกว่า ให้เปรียบเทียบตัวขวาของ pair เป็นสำคัญก่อน
// หากตัวขวาเท่ากันจึงเปรียบเทียบตัวทางซ้ายใน pair









}
```

```
int swap_top_and_bottom(stack<int>& s) {
// สลับตัวล่างสุด (ที่ก้นสแตก) กับตัวบนสุดของสแตก









}
```

ชื่อ_____หมายเลขประจำตัว_____เลขที่ใน CR58_____

3. (6 คะแนน) ตอบคำถามต่อไปนี้สั้น ๆ ว่า แต่ละปัญหาต้องมีที่เก็บข้อมูลประเภทใด

0. ต้องการเก็บรายชื่อนิสิตคณะวิศวฯทุก ๆ รุ่น แต่ละรุ่นมีหมายเลขรุ่นกำกับ เพื่อเขียนเมท็อด **`int getClassID(string name)`** ที่คืนหมายเลขรุ่นของคนชื่อ **`name`**

   ตอบ: `map<string,int>` *key* คือชื่อ *mapped value* คือหมายเลขรุ่น (ข้อนี้เป็นตัวอย่าง)

1. ต้องการที่เก็บข้อมูลเพื่ออำนวยความสะดวกในการเขียนเมท็อด **`string getFMStationName(double freq)`** เพื่อขอชื่อสถานีวิทยุจากคลื่นความถี่

   .....

2. ต้องการที่เก็บข้อมูล เพื่อให้บริการค้นอัลบั้มเพลงต่าง ๆ ของนักร้องต่าง ๆ โดยค้นด้วยชื่อของนักร้อง ภายในอัลบั้มจะเก็บชื่ออัลบั้มพร้อมกับรายชื่อของเพลงและความยาวของเพลงเป็นวินาทีกำกับทุกเพลงด้วย

   .....

3. ต้องการที่เก็บข้อมูล เพื่อเก็บเมทริกซ์ของจำนวนจริงที่มีขนาดใหญ่ (เช่น 500×500) เพื่อการคำนวณทางวิศวกรรม โดยสังเกตว่าข้อมูลภายในเมทริกซ์ มีค่าไม่ค่อยเหมือนกันเลย

   .....

4. (5 คะแนน) จงเขียนรายละเอียดของโครงสร้างข้อมูลที่ชื่อว่า CP::triplet<T1,T2,T3> ซึ่งมีลักษณะคล้าย CP::pair แต่ว่าเป็นโครงสร้างข้อมูลที่เก็บข้อมูลจำนวน 3 สิ่งที่สามารถแตกต่างกันได้ โดยให้เขียน function ต่อไปนี้

- triplet(T1 _first,T2 _second, T3 _third) เป็น constructor ที่กำหนดค่าเริ่มต้นของ Object ตาม parameter
- bool operator<(const triplet<T1,T2,T3>& other) เป็น function ที่ override operator < เพื่อใช้ในการเปรียบเทียบ โดยใช้วิธีเปรียบเทียบลักษณะเดียวกับ pair<T1,T2> แต่จะคำนึงถึงสมาชิก third ด้วย

```
template <typename T1, typename T2, typename T3>
class triplet<T1,T2,T3> {
public:
   T1 first; T2 second; T3 third;
   Triplet(T1 _first, T2 _second, T3 _third) {




   }

   bool operator<(const triplet<T1,T2,T3>& other) {




   }
};
```

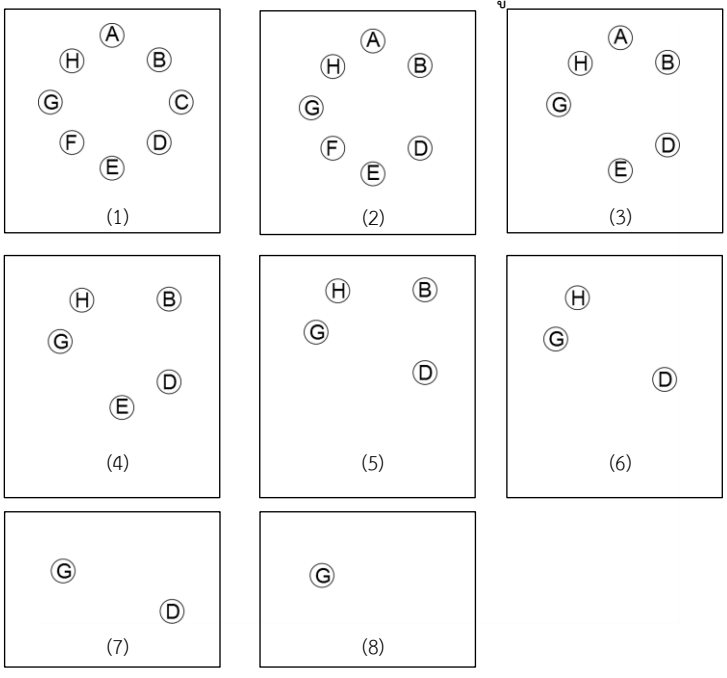
5. (5 คะแนน) กำหนดให้เรามี std::priority_queue<int> pq และ std::queue<int> q2 อยู่ โดยที่ทั้ง pq และ q2 นั้นมีจำนวนข้อมูลเท่ากัน จงเขียนฟังก์ชันที่รับพารามิเตอร์เป็นข้อมูลทั้งสองนี้ แล้วตรวจสอบว่า ถ้าเราดึงข้อมูลออกจาก pq และออกจาก q2 นั้น ลำดับของข้อมูลที่จะดึงออกมาจาก pq นั้น มีลำดับตรงข้ามกับข้อมูลที่จะดึงออกมาจาก q2 (ตัวอย่าง เช่น ถ้าของที่ดึงออกมาจาก pq เป็น 50,40,30,20,10 ตามลำดับ ข้อมูลที่ดึงออกมาจาก q2 จะต้องเป็น 10,20,30,40,50 ตามลำดับ เช่นกัน) ในฟังก์ชันนี้ นิสิตสามารถเรียกใช้โครงสร้างข้อมูลอื่น ๆ เพิ่มได้ และสามารถใช้ฟังก์ชันจากไลบรารี่ใน c++ ช่วยได้เช่นกัน อย่างไรก็ตาม หลังจากจบการทำงานของฟังก์ชันนี้แล้ว ข้อมูลใน pq และ q2 จะต้องเหมือนเดิม (หมายความว่า ข้อมูลใดที่เคยอยู่ใน pq หรือ q2 ก็ต้องมีอยู่เหมือนเดิมหลังจากการทำงาน และข้อมูลใน q2 จะต้องเรียงลำดับตามเดิมด้วยเช่นกัน)

```
bool check_reverse(priority_queue<int> &pq, queue<int>& q2) {




}
```

6. (10 คะแนน) ในข้อนี้ให้นิสิตเขียนฟังก์ชั่นเพื่อคำนวณลำดับของคนที่จะถูกเลือกตามกฎดังต่อไปนี้

   - ให้คน n คนยืนเป็นวงกลม นับจากคนแรกไป k คนแล้วเลือกคนนั้นและให้ออกจากวงกลมไป จากนั้นก็นับไปอีก k คนแล้วเลือกคนนั้นเป็นคนต่อไปซึ่งคนนั้นก็จะออกจากวงกลมไปเช่นเดียวกัน โดยขั้นตอนการเลือกแบบนี้จะทำไปเรื่อยๆ จนกว่าทุกๆคนจะถูกเลือก โดยระหว่างนับจะข้ามคนที่ถูกเลือกไปแล้ว

เช่น หากมีคน n=8 คน คือ A B C D E F G H ตามลำดับ และ k = 3 ลำดับของคนที่ถูกเลือกจะเป็น C F A E B H D G ตามรูปข้างล่าง



ในข้อนี้ ข้อมูลนำเข้าจะเป็น circular queue ของ STL ชื่อว่า queue<Person> persons โดยคนแรกจะอยู่ที่ persons.front() และคนถัดๆไปตามลำดับของวงกลมก็จะอยู่ในลำดับตาม queue นี้ และ n=persons.size() ข้อมูลส่งออกจะเป็น STL vector ซึ่งจะต้องเก็บลำดับของคนที่ถูกเลือกเอาไว้

```
vector<Person> GetCircularSelectionOrder(queue<Person> persons, int k) {
	int n = persons.size();
	vector<Person> output;









	return output;
}
```

7. (10 คะแนน) ในข้อนี้ให้นิสิตเขียนคำสั่งในการย้ายข้อมูลจำนวน k ตัวภายใน vector v จากตำแหน่งที่ระบุโดย iterator from ถึง from+k-1 ไปยังข้างหน้าของตำแหน่งที่ระบุโดย iterator to โดยนิสิตสามารถมั่นใจได้ว่า from และ to เป็น iterator ที่ถูกต้องของ v , to ไม่ได้อ้างอิงถึงข้อมูลระหว่างตัวที่ from ถึง from + k - 1 และ ข้อมูลตัวที่ from ถึง from + k - 1อยู่ใน v จริง

ตัวอย่าง 1: หาก v คือ <3,9,2,1,4,5,6,8,7> หลังเรียก moveData(v, v.begin()+1, v.begin()+6, 3) v จะเป็น <3,4,5,9,2,1,6,8,7>

ตัวอย่าง 2: หาก v คือ <3,9,2,1,4,5,6,8,7> หลังเรียก moveData(v, v.begin()+7, v.begin()+5, 2) v จะเป็น <3,9,2,1,4,8,7,5,6>

```
void moveData(vector<int>& v, vector<int>::iterator from,
		vector<int>::iterator to, int k)
{









}
```

8. (10 คะแนน) จากคลาส CP::vector<T> ที่นิสิตได้เรียนรายละเอียดของโครงสร้างในวิชานี้ จงเขียนฟังก์ชัน vector<T> intersect(const vector<T> &a) ที่รับเวกเตอร์ a เพื่อประมวลผลและคืนค่ามาเป็นเวกเตอร์ใหม่ที่เป็นผลของการหา intersection ระหว่าง a กับเวกเตอร์ของเรา โดยจะพิจารณาสมาชิกที่ซ้ำกันด้วย กล่าวคือ สำหรับข้อมูลที่มีค่า X ที่ปรากฏใน a เป็นจำนวน n ครั้ง และปรากฏในเวกเตอร์ของเรา m ครั้ง เราจะต้องคืนค่าเวกเตอร์ที่มีค่า X ปรากฏขึ้นเป็นจำนวน min(n,m) ครั้ง ตัวอย่างเช่น ถ้า v1 = {1,3,1,3,5} และ v2 = {9,1,1,3,5,5} ผลลัพธ์ของการเรียก v1.intersect(v2) จะได้เป็น {1,1,3,5} **<u>ห้ามใช้ฟังก์ชันของ algorithm ช่วยในการเขียน และห้ามทำให้ข้อมูลใน a และในเวกเตอร์ของเรามีการเปลี่ยนแปลง</u>**

```
template <typename T>
class vector {
protected:
   T      *mData;   size_t  mCap;   size_t  mSize;

public:
   // คลาสนี้ทำงานได้ตามปรกติทุกอย่าง ให้นิสิตเขียนบริการ intersect เพิ่มเติม ตามข้อกำหนดด้านบน
   vector<T> intersect(const vector<T>& a) {









   }
};
```

9. (15 คะแนน) จงออกแบบโครงสร้างข้อมูล สำหรับการใช้งานบัตรโดยสารรถไฟฟ้าแบบเติมเงิน กำหนดให้บัตรโดยสารรถไฟฟ้าแต่ละใบมีหมายเลขเป็น string ความยาว 16 ตัวอักษรซึ่งไม่ซ้ำกันเลย และบัตรโดยสารแต่ละใบจะมีเงินจำนวนหนึ่งอยู่ การใช้งานคือผู้ใช้จะต้องนำบัตรมาแสดงเพื่อใช้เข้าและออกจากระบบรถไฟฟ้า ณ สถานี ต่าง ๆ สมมติให้มีสถานีทั้งหมด 100 สถานี และแต่ละสถานีมีหมายเลขกำกับอยู่ตั้งแต่หมายเลข 0 ถึง 99 โครงสร้างข้อมูลที่จะต้องสร้างขึ้นนี้จะใช้ในการหักค่าบริการจากบัตรโดยสาร นอกจากนี้ เพื่อป้องกันการทุจริต ระบบจะต้องตรวจสอบข้อบังคับของการใช้บริการตามกฎต่อไปนี้

   1) ถ้าจำนวนเงินในบัตรมีน้อยกว่าหรือเท่ากับ 0 บัตรนั้นจะไม่สามารถนำมาใช้เข้าระบบได้
   2) ถ้าการใช้งานบัตรครั้งล่าสุดเป็นการเข้าสู่ระบบ บัตรนั้นจะไม่สามารถใช้เข้าสู่ระบบได้ จนกว่าบัตรนั้นจะถูกใช้ออกจากระบบ
   3) ถ้าการใช้งานบัตรครั้งล่าสุดเป็นการออกจากระบบ บัตรนั้นจะไม่สามารถเข้าสู่ระบบได้ จนกว่าบัตรนั้นจะถูกใช้ออกจากระบบ
   4) การออกจากระบบจะทำได้ก็ต่อเมื่อเงินที่เหลือในบัตรมีค่าไม่น้อยกว่าค่าโดยสาร กำหนดให้มีฟังก์ชัน int get_fare(int fsid, int tsid) ซึ่งจะคืนค่าโดยสารเมื่อเราขึ้นจากสถานี fsid และลงที่สถานี csid

โดยให้เขียนโครงสร้างข้อมูลที่มีบริการดังต่อไปนี้

   1) bool register(string CID, int value) เป็นการลงทะเบียนบัตรใหม่ที่มีหมายเลขเป็น CID และมีเงินเริ่มต้นเป็น value โดยจะคืนค่า true ก็ต่อเมื่อบัตรดังกล่าวยังไม่เคยลงทะเบียนในระบบเท่านั้น และคืนค่า false ในกรณีอื่น ๆ
   2) bool addValue(string CID, int value) เป็นการเติมเงินลงไปในบัตรหมายเลข CID ด้วยมูลค่า value จะคืนค่า true ก็ต่อเมื่อบัตรดังกล่าวได้ลงทะเบียนในระบบแล้วเท่านั้น และคืนค่า false ในกรณีอื่น ๆ
   3) bool enter(string CID, int fsid) เป็นการเข้าสู่ระบบของบัตรหมายเลข CID โดยขึ้นที่สถานี fsid โดยจะคืนค่า true ก็ต่อเมื่อการเข้าสู่ระบบไม่ผิดกฎข้างต้นเท่านั้น และคืนค่า false ในกรณีอื่น ๆ
   4) bool leave(string CID, int tsid) เป็นการออกจากระบบของบัตรหมายเลข CID โดยออกที่สถานี tsid และทำการหักเงินจากบัตรตามค่าโดยสารที่ควรเป็น โดยจะคืนค่า true ก็ต่อเมื่อการออกจากระบบไม่ผิดกฎข้างต้น และคืนค่า false ในกรณีอื่น ๆ

ชื่อ_____หมายเลขประจำตัว_____เลขที่ใน CR58_____

ขอให้พยายามทำให้ได้ประสิทธิภาพในการทำงานที่เร็วที่สุด โดยระลึกว่าในแต่ละวันมีจำนวนครั้งที่คนเข้าสู่ระบบและออกจากระบบมากกว่า 10 ล้านครั้ง **นิสิตสามารถใช้ class และ function ใด ๆ ใน stl ได้ตามปรกติ**

```
class BTS {
protected: // ประกาศสมาชิกที่ต้องใช้ตรงนี้



public:
  BTS() {

  }

  ~BTS() {

  }

  bool register_card(string CID,int value) {









  }

  bool add_value(string CID,int value) {









  }

  bool enter(string CID,int fsid) {









  }

  bool leave(string CID,int tsid) {









  }
};
```

STL Reference

Common

All classes support these two capacity functions;

Capacity	`size_t size(); // return the number of items in the structure`
	`bool empty(); // return true only when size() == 0`

Container Class

All classes in this category support these two iterator functions.

Iterator	`iterator begin(); // an iterator referring to the first element`
	`iterator end(); // an iterator referring to the` *`past-the-end`* `element`

Class vector<ValueT>

Element Access	`operator[] (size_t n);`

Modifier ที่ใช้ได้ทั้ง list และ vector	`void push_back(const ValueT& val);`
	`void pop_back();`
	`iterator insert(iterator position, const ValueT& val);`
	`iterator insert(iterator position, InputIterator first, InputIterator last);`
	`iterator erase(iterator position);`
	`iterator erase(iterator first, iterator last);`

Class set<ValueT>

Operation	`iterator find (const ValueT& val);` `size_type count (const ValueT& val);`

Modifier	`pair<iterator,bool> insert (const ValueT& val);`
	`void insert (InputIterator first, InputIterator last);`
	`iterator erase (iterator position);`
	`iterator erase (iterator first, iterator last);`
	`size_type erase (const ValueT& val);`

Class map<KeyT, MappedT>

Element Access	`MappedT& operator[] (const KeyT& k);`
Operation	`iterator find (const KeyT& k);` `size_type count (const KeyT& k);`

Modifier	`pair<iterator,bool> insert (const pair<KeyT,MappedT>& val);`
	`void insert (InputIterator first, InputIterator last);`
	`iterator erase (iterator position);`
	`iterator erase (iterator first, iterator last);`
	`size_type erase (const KeyT& k);`

Container Adapter

These three data structures support the same data modifiers but each has different strategy. These data structures do not support iterator.

Modifier	`void push (const ValueT& val); // add the element`
	`void pop();  // remove the element`

Class queue<ValueT>

Element Access	`ValueT front();` `ValueT back();`

Class stack<ValueT>

Element Access	`ValueT top();`

Class priority_queue<ValueT, ContainerT = vector<ValueT>, CompareT = less<ValueT> >

Element Access	`ValueT top();`

Useful function

```
iterator find (iterator first, iterator last, const T& val);
void sort (iterator first, iterator last, Compare comp);
pair<T1,T2> make_pair (T1 x, T2 y);
```