FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 Introduction to Data Structures

Final Examination, First Semester, 2017

ชื่อ-นามสกุล_____เลขประจำตัว CR60_____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 11 ข้อในกระดาษคำถามคำตอบจำนวน 10 แผ่น 10 หน้า คะแนนเต็ม 88 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้เข้าสอบสามารถออกจากห้องสองได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะวิศวกรรมศาสตร์
   มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

หมายเหตุเพิ่มเติม:

1. **หากที่ไม่พอเขียนคำตอบให้เขียนในด้านหลังได้แต่ต้องระบุไว้ให้ชัดเจน**
2. **ให้เขียนชื่อ-นามสกุล รหัสนิสิต และ CR ให้ครบทุกหน้า**
3. **ในข้อที่มีการเขียนโปรแกรมนั้น จะมีการพิจารณาประสิทธิภาพในการทำงานของโปรแกรมที่เขียนมาด้วย ให้พยายามเขียนโปรแกรมที่ทำงานได้เร็วที่สุด**

1. (5 คะแนน) จงระบุประสิทธิภาพในการทำงานของฟังก์ชันต่าง ๆ ของโครงสร้างข้อมูลดังต่อไปนี้ โดยให้เติมคำในช่องว่าง กำหนดให้ n คือจำนวนข้อมูลในโครงสร้างข้อมูลดังกล่าว ให้ระบุประสิทธิภาพในการทำงานโดยใช้สัญกรณ์เชิงเส้นกำกับบนตัวแปร n ให้ตรงกับความเป็นจริงมากที่สุด (การวิเคราะห์ไม่จำเป็นต้องระบุในกรณีเฉลี่ย ให้ใช้กรณีแย่ที่สุดได้เลย)

ฟังก์ชัน	ประสิทธิภาพในการทำงาน
insert ของ CP::list	
erase ของ CP::list	
insert ของ CP::vector	
push_back ของ CP::vector	
operator[ ] ของ CP::unordered_map	
operator[ ] ของ CP::map_bst	
operator++ ของ CP::map_bst::iterator	
begin ของ CP::map_bst	
top ของ CP::priority_queue	
pop ของ CP::priority_queue	

2. (5 คะแนน) จงวิเคราะห์อัตราการเติบโตของส่วนของโปรแกรมต่อไปนี้ ให้ตรงกับความเป็นจริงมากที่สุด โดยให้เขียนอยู่ในรูปสัญกรณ์เชิงเส้นกำกับในตัวแปร n

`a1(int n) {`
`  s = 0`
`  for (int i=1; i<=n; i++) s++;`
`  for (int i=n; i>=1; i--) s++;`
`  return s`
`}`
	a1(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a2(int n) {`
`  s = 0`
`  for (int i=1; i<=n; i++)`
`  for (int j=1; j<=n; j--)`
`  if (i==j)`
`  for (int k = 0;k < n;k++)`
`  s++;`
`  return s`
`}`
	a2(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a3(int n) {`
`  std::list<int> l;`
`  for (int i = 0;i < n;i++) {`
`  l.insert(l.end(),i);`
`  l.erase(l.begin());`
`  }`
`}`
	a3(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a4(int n) {`
`  std::priority_queue<int> pq;`
`  for (int i = n;i >= 0;i--)`
`  pq.push(i);`
`}`
	a4(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

`a5(int n) {`
`  std::map<int,int>m;`
`  for (int i = n;i >= 0;i--)`
`  m[i] = -i;`
 	
`}`
	a5(n) ใช้เวลา เป็น $\Theta(\ ?\ )$

************** สำคัญ!!! **************

ตั้งแต่ข้อ 3 เป็นต้นไปเป็นการเขียนโปรแกรม การเขียนโปรแกรมจะต้องไม่ผิด syntax ในส่วนที่สำคัญ โดยเฉพาะอย่างยิ่งการเรียก function และ argument ของฟังก์ชันจะต้องถูกต้อง นอกจากนี้คะแนนจะแปรผันตาม "ประสิทธิภาพ" ของโปรแกรมที่เขียนมา

******** ให้สังเกตถึงข้อกำหนดต่าง ๆ ในโจทย์ให้รอบคอบ <u>การไม่ทำตามข้อกำหนดจะมีผลต่อคะแนนเป็นอย่างมาก</u> ********

3. (5 คะแนน) คลาส CP::unordered_map ซึ่งเป็น hash แบบ separated chaining มี implementation ดังแสดงในช่องข้างล่างนี้ จงเขียนฟังก์ชัน void remove_colliding_key(KeyT m) สำหรับคลาสดังกล่าว โดยฟังก์ชันนี้จะลบข้อมูลทุกตัวที่จะต้องเก็บใน bucket เดียวกับ bucket ที่จะเก็บข้อมูลที่มี key เป็น m หรือกล่าวอีกนัยหนึ่งคือ ฟังก์ชันนี้จะทำให้ใน unordered_map นั้นไม่มีข้อมูลที่ชนกับ m นั่นเอง ทำให้เมื่อเราต้องการจะใส่ key m เข้าไปนั้น การใส่ดังกล่าวจะไม่ชนกับใครเลย

```
template <typename KeyT,
          typename MappedT,
          typename HasherT = std::hash<KeyT>,
          typename EqualT = std::equal_to<KeyT> >
class unordered_map
{
  protected:
    std::vector<BucketT> mBuckets;
    size_t               mSize;
    HasherT              mHasher;
    EqualT               mEqual;
    float                mMaxLoadFactor;
  public:
    void remove_colliding_key(const KeyT& m) {









    }
};
```

4. (10 คะแนน) คลาส CP::unordered_map ซึ่งเป็น hash แบบ separated chaining มี implementation ดังแสดงในช่องข้างล่างนี้ จงเขียนฟังก์ชัน size_t longest_chain_size( ) สำหรับคลาสดังกล่าว โดยฟังก์ชันนี้จะคำนวณว่า bucket ซึ่งมีข้อมูลมากที่สุดใน unordered_map นั้นมีความยาวเท่าไร

```
template <typename KeyT,
          typename MappedT,
          typename HasherT = std::hash<KeyT>,
          typename EqualT = std::equal_to<KeyT> >
class unordered_map
{
  protected:
    std::vector<BucketT> mBuckets;
    size_t               mSize;
    HasherT              mHasher;
    EqualT               mEqual;
    float                mMaxLoadFactor;
  public:
    size_t longest_chain_size() {









    }
};
```

5. (10 คะแนน) การเก็บข้อมูลในคอมพิวเตอร์นั้นมีวิธีการเก็บข้อมูลแบบที่เรียกว่า cache คือการเก็บข้อมูลบางส่วนไว้ในหน่วยความจำที่เร็วมาก ๆ เพื่อให้ CPU สามารถเข้าถึงได้อย่างรวดเร็ว อย่างไรก็ตาม cache นั้นมักจะมีขนาดเล็ก เนื่องจากหน่วยความจำที่เร็วมาก ๆ นั้นมีราคาสูง เราจึงต้องเลือกว่าจะให้ข้อมูลใดบ้างเก็บไว้ใน cache

หลักการหนึ่งในการเลือกข้อมูลใส่ไว้ใน cache คือหลักการที่เรียกว่า LRU หรือ Least Recently Used ซึ่งมีหลักการทำงานดังนี้ กำหนดให้ ระบบคอมพิวเตอร์มี RAM ขนาด 4GB ให้ตำแหน่งต่าง ๆ ของ RAM นั้นระบุได้ด้วยตัวแปรประเภท size_t และข้อมูลในแต่ละตำแหน่งเป็นประเภท char ในการทำงานนั้น CPU จะร้องขอ RAM โดยระบุตำแหน่งต่าง ๆ มาให้ และต้องการข้อมูล char กลับไป

ให้ cache เริ่มต้นด้วย cache ว่างที่ไม่มีข้อมูลใดเลย cache สามารถเก็บข้อมูลได้ไม่เกิน n ตำแหน่ง เมื่อ CPU ต้องการข้อมูล ณ ตำแหน่ง addr นั้น cache จะทำการตรวจสอบว่ามีข้อมูลตำแหน่งดังกล่าวเก็บไว้ใน cache หรือไม่ ถ้ามี ก็จะคืนค่าข้อมูล char ที่เก็บไว้ให้กับ CPU แต่ถ้า cache ไม่มีข้อมูลตำแหน่งดังกล่าวอยู่ cache จะต้องเรียกฟังก์ชัน char get_memory(size_t addr) ซึ่งจะอ่านค่าจาก RAM ณ ตำแหน่ง addr แล้วนำผลลัพธ์ดังกล่าวคืนให้ CPU พร้อมกับเก็บค่าดังกล่าวไว้ในตัวเอง

อย่างไรก็ตาม เนื่องจาก cache เก็บข้อมูลได้จำกัด เมื่อ cache จะต้องเก็บข้อมูลตำแหน่ง addr เพิ่ม โดยที่ cache เก็บข้อมูลไว้ครบ n ตัวแล้ว และข้อมูล n ตัวนั้นไม่มีตำแหน่ง addr อยู่ cache จะต้องทิ้งข้อมูลที่เก็บไว้ไป 1 ตัวเพื่อให้มีที่สำหรับเก็บข้อมูลใหม่ cache จะทิ้งข้อมูลตัวที่ “ถูกเรียกใช้ในอดีตอันไกลที่สุด” หรือ Least recently used ทิ้งไปนั่นเอง

จงเขียนคลาส LRUCache ซึ่งจำลองการทำงานของ cache ดังที่ได้อธิบายข้างต้น โดย LRUCache นั้นจะต้องมีฟังก์ชันดังต่อไปนี้

- LRUCache(size_t n) เป็น constructor ซึ่งรับจำนวนข้อมูล n ที่ cache สามารถเก็บได้
- char request(size_t addr) เป็นฟังก์ชันที่ cpu จะเรียกใช้ cache เพื่อถามถึงข้อมูลที่เก็บไว้ใน RAM ณ ตำแหน่ง addr โดยฟังก์ชันดังกล่าวจะต้องทำงานตามที่ได้ระบุไว้ ฟังก์ชันนี้สามารถเรียกใช้ฟังก์ชัน get_memory ได้

**** ขอให้พยายามเขียน request ให้ใช้เวลาในการทำงานเป็น O(1) แต่ถ้าทำไม่ได้ ก็ขอให้เขียนให้เร็วที่สุดเท่าที่จะทำได ****

a. จงอธิบาย data member ต่าง ๆ ที่ใช้ในคลาสที่ออกแบบขึ้น ว่ามี member อะไรบ้าง และ แต่ละตัวทำหน้าที่อะไร ทำไมถึงเลือกใช้ประเภทตัวแปรดังกล่าว

b. จงอธิบายว่า request นั้นทำงานอย่างไรโดยสังเขป พร้อมทั้งวิเคราะห์ประสิทธิภาพในการทำงาน

c. จงเขียน class LRUCache ดังกล่าว

6. (15 คะแนน) มีโค้ดของคลาสที่ใช้สร้างลิงค์ลิสต์ให้มาดังนี้

```
template <typename T>
class node {
   public:
     T     data;
     node *prev;
     node *next;
     node() :
        data( T() ), prev(this), next(this) { }
     node(const T& data, node* prev, node* next) :
        data( T(data) ), prev(prev), next(next) { }
 };

class list_iterator {
      public:
          node* ptr;

        list_iterator() : ptr( NULL ) { }

        list_iterator(node *a) : ptr(a) { }

        list_iterator& operator++() {
          ptr = ptr->next;
          return (*this);
        }
```

```
            list_iterator& operator--() {
              ptr = ptr->prev;
              return (*this);
            }

            T& operator*() { return ptr->data; }
            T* operator->() { return &(ptr->data); }
            bool operator==(const list_iterator& other) {
                           return other.ptr == ptr; }
            bool operator!=(const list_iterator& other) {
                            return other.ptr != ptr; }
};

class list {
    node   *mHeader; // pointer to a header node
    int       mSize;
};
```

a. (5 คะแนน) จงเขียนฟังก์ชัน **list_iterator insert(list_iterator it, const T& element)** ของคลาส List ซึ่งทำการเติมแทรกข้อมูล element เข้าไปที่ตำแหน่ง it ในลิสต์

b. (10 คะแนน) สมมุติว่ามีฟังก์ชัน erase(list_iterator it) ของคลาส List ซึ่งทำการลบข้อมูลที่ระบุโดย it ออกจากลิสต์ไป จงเขียนฟังก์ชัน **list_Iterator moveTo(T data, int position)** ฟังก์ชันนี้ทำการย้าย data ไปยังตำแหน่งที่ระบุ (ตำแหน่งนั้นเป็นตำแหน่งก่อนที่ data จะโดนย้าย และของที่เก็บอยู่หลัง header พอดีถือว่าอยู่ที่ตำแหน่งเบอร์ 0) ฟังก์ชันนี้รีเทิร์นอิเทอเรเตอร์ระบุตำแหน่งที่ data ย้ายไป

7. (13 คะแนน) ให้คลาสที่ใช้เขียน priority queue ของจำนวนเต็ม ด้วยฮีป มีโค้ดดังนี้

```
class priority_queue {
  protected:
    int*   mData;
    size_t mCap;
    size_t mSize;
};
```

a. (10 คะแนน) จงเขียนฟังก์ชัน **void pop()** ซึ่งทำการเอาข้อมูลที่มีค่ามากที่สุดออกจากฮีป (ฮีปต้องยังคงเป็นฮีปหลังรันฟังก์ชันไป)

b. (3 คะแนน) สมมุติให้มีฟังก์ชัน

- void fixDown(int position) ซึ่งทำการย้ายข้อมูลจากตำแหน่ง position ขึ้นต้นไม้ฮีปไปจนกว่าต้นไม้จะเป็นฮีปอีกครั้ง
- void fixUp(int position) ซึ่งทำการย้ายข้อมูลจากตำแหน่ง position ลงต้นไม้ฮีปไปจนกว่าต้นไม้จะเป็นฮีปอีกครั้ง

จงเขียนฟังก์ชัน **void changeData(int p, int data)**
ซึ่งจะทำการเปลี่ยนข้อมูลที่ตำแหน่ง p ของ mData ให้เป็น data (ซึ่งพอทำเสร็จแล้ว โครงสร้างข้อมูลเรายังคงต้องเป็นฮีปอยู่)

8. (5 คะแนน) ให้นิสิตวาด Binary Search Tree ที่เกิดจากการใส่ข้อมูลดังต่อไปนี้ตามลำดับ (หรือระบุว่าปมใดเป็นราก และ แต่ละปมมีลูกซ้าย/ขวาเป็นปมอะไรบ้าง)

   7 9 10 1 11 3 6 8 2

9. (5 คะแนน) เริ่มต้นด้วย Binary Search Tree ที่มีลักษณะดังต่อไปนี้

   รากเป็น ปม 12

ปม	ลูกซ้าย	ลูกขวา
12	6	20
6	4	8
20	17	22
4	-	-
8	-	-
17	15	18
15	13	-
18	-	-
22	-	-
13	-	-

   ให้นิสิตวาด Binary Search ที่เกิดจากการลบข้อมูล 12 17 6 13 ตามลำดับ (หรือระบุว่าปมใดเป็นราก และ แต่ละปมมีลูกซ้าย/ขวาเป็นปมอะไรบ้าง)

10. (5 คะแนน) จงเขียนฟังก์ชั่นที่ใช้นับจำนวนปมในต้นไม้ทวิภาคโดยเติมจากโค๊ดข้างล่างนี้

```
class Tree{
    class Node{
        public: friend class Tree;
            Node() {data = -1; left = NULL; right = NULL;}
            Node(const int x, Node* left, Node* right) : data(x), left(left), right(right) {}
        protected: int data; Node* left; Node* right;};
public:
    Tree() {…}
    ~Tree() {….}
    void clear(Node*& r) {…}
    void insert(int x) {…}
    int countNodes() { return countNodes(mRoot); }
    int countNodes(Node* r) {
          // เติมค็ดตรงนี้
    }
protected:
    Node* mRoot;
};
```

11. (10 คะแนน) ให้นิสิตวาด AVL Tree ที่เกิดจากการใส่ข้อมูลดังต่อไปนี้ตามลำดับ (หรือระบุว่าปมใดเป็นราก และ แต่ละปมมีลูกซ้าย/ขวาเป็นปมอะไรบ้าง)

7 9 10 1 11 3 6 8 2

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> **และ** list<ValueT>

Element Access สำหรับ vector	ValueT& operator[] (size_t n); ValueT& at(inti dx);

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);
	void clear();
	void resize(size_t n);

Modifier ที่ใช้ได้เฉพาะ list	void push_front(const ValueT& val);
	void pop_font;
	void remove(const ValueT& val);

Class set<ValueT>

Operation	iterator find (const ValueT& val); size_t 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_t erase (const ValueT& val);

Class map<KeyT, MappedT>

Element Access	MappedT& operator[] (const KeyT& k);
Operation	iterator find (const KeyT& k); size_t 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_t 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 functions

```
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);
```