

ชื่อ-นามสกุล _____ เลขประจำตัว

| | | | | | | | | | | | |
|--|--|--|--|--|--|--|--|--|--|---|---|
| | | | | | | | | | | 2 | 1 |
|--|--|--|--|--|--|--|--|--|--|---|---|

 CR58 _____

หมายเหตุ

1. ข้อสอบมีทั้งหมด 12 ข้อในกระดาษคำถามคำตอบจำนวน 10 แผ่น 10 หน้า คะแนนเต็ม 85 คะแนน
2. ไม่อนุญาตให้นำตำราและเครื่องคำนวณต่างๆ ใดๆ เข้าห้องสอบ
3. ควรเขียนตอบด้วยลายมือที่อ่านง่ายและชัดเจน สามารถใช้ดินสอเขียนคำตอบได้
4. ห้ามการหยิบยื่นสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่ผู้คุมสอบจะหยิบยื่นให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบออกจากห้องสอบ ข้อสอบเป็นทรัพย์สินของราชการซึ่งผู้ลักพาอาจมีโทษทางคดีอาญา
6. ผู้ที่ประสงค์จะออกจากห้องสอบก่อนหมดเวลาสอบ แต่ต้องไม่น้อยกว่า 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. ผู้ที่ปฏิบัติเข้าข่ายทุจริตในการสอบ ตามประกาศคณะกรรมการการศึกษาระดับปริญญาตรี

มีโทษ คือ ได้รับ สัญลักษณ์ F ในรายวิชาที่ทุจริต และพักการศึกษาอย่างน้อย 1 ภาคการศึกษา

รับทราบ

ลงชื่อนิสิต (.....)

0. ในวิชานี้ อัตราส่วนของคะแนนสอบย่อย (quiz) อยู่ที่ 20% และข้อสอบปลายภาคอยู่ที่ 40% นิสิตสามารถเลือกปรับน้ำหนักของคะแนนสอบย่อยได้ตั้งแต่ 0% ถึง 20% ค่าน้ำหนักในส่วนที่ขาดหายไปนั้นจะถูกนำไปคิดเป็นน้ำหนักของข้อสอบปลายภาคแทน จงระบุอัตราส่วนของคะแนนสอบย่อยที่ต้องการ โดยใส่เป็นตัวเลขจำนวนเต็มตั้งแต่ 0 ถึง 20 ลงในช่องด้านขวานี้ (ถ้าไม่ระบุหรือระบุค่าที่ไม่ถูกต้องจะถือว่าระบุเลข 20 ไว้) (ตัวอย่างการคิดคะแนน ถ้ากรอกเลข 14 หมายความว่าคะแนนสอบย่อยเป็น 14% และสอบปลายภาคเป็น 46%) ****

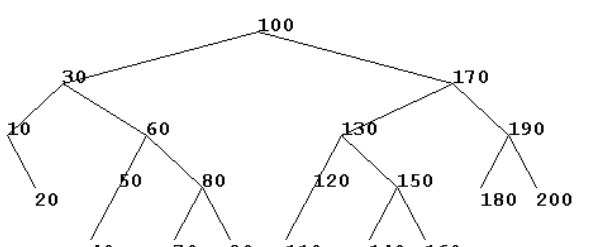
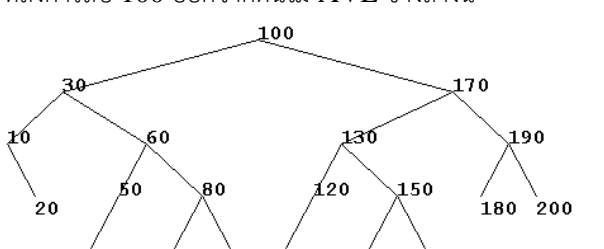
1. (5 คะแนน) จงวิเคราะห์ห่อักรากการเติบโตของส่วนของโปรแกรมต่อไปนี้ **ให้ตรงกับความเป็นจริงมากที่สุด** โดยให้เขียนอยู่ในรูปสัญกรณ์เชิงเส้นกำกับในตัวแปร n

| | |
|---|--|
| <pre>void a(int x) { if (x > 0) a(x/2); } a(n);</pre> | มีอัตราการเติบโต เป็น $\Theta(\text{_____})$ |
| <pre>map<int,int> m; for (i = 0;i < n;i++) m[42] = i;</pre> | มีอัตราการเติบโต เป็น $\Theta(\text{_____})$ |
| <pre>list<int> l; for (i = 0;i < n;i++) { l.push_back(i); l.remove(i); }</pre> | มีอัตราการเติบโต เป็น $\Theta(\text{_____})$ |
| <pre>unordered_map<int,float> m; for (i = 0;i < n;i++) m[42] = i;</pre> | มีอัตราการเติบโต เป็น $\Theta(\text{_____})$ |
| <pre>priority_queue<int> m; for (i = n;i > 0;i--) m.push(i);</pre> | มีอัตราการเติบโต เป็น $\Theta(\text{_____})$ |

2. (4 คะแนน) จงเติมค่าลงในช่องว่าง เพื่อให้ส่วนของโปรแกรมต่อไปนี้ มีอัตราการเติบโตของเวลาการทำงานเป็นไปตามที่กำหนดให้

| | |
|---|--|
| <pre>void a(int x) { if (x > 0) a(_____); } a(n);</pre> | มีอัตราการเติบโต เป็น $\Theta(n)$ |
| <pre>list<int> l; for (i = 0; i < n; i++) l.push_front(i); for (i = 0; i < n; i++) l.remove(_____)</pre> | มีอัตราการเติบโต เป็น $\Theta(n^2)$ |
| <pre>map<int,int> m; for (i = 0; i < n; i++) m.insert(make_pair(_____, _____))</pre> | มีอัตราการเติบโต เป็น $\Theta(n \log n)$ |
| <pre>// สมมติให้ m เป็น map<int,int> ซึ่งมีข้อมูลอยู่ n ตัวอยู่แล้ว // โดยสร้างมาจาก m[i] = i (ไม่นับเวลาในการสร้าง m ขึ้นมา) typedef map<int,int>::iterator Itr; for (Itr it = m.begin(); it!=m.end(); it++) { m._____ }</pre> | มีอัตราการเติบโต เป็น $\Theta(n \log n)$ |

3. (6 คะแนน) กำหนดให้ใช้วิธีการเพิ่มและลบข้อมูลในต้นไม้ AVL ตามโปรแกรมที่นำเสนอในชั้นเรียนของวิชานี้

| | |
|---|--|
| <p>จงวาดต้นไม้ AVL ในช่องทางขวานี้ หลังการเพิ่ม 75 ในต้นไม้ AVL ข้างล่างนี้</p>  | |
| <p>จงวาดต้นไม้ AVL ในช่องทางขวานี้ เริ่มจากต้นไม้ว่าง เพิ่มข้อมูลตามลำดับดังนี้ 1,2,3,4,5,6,7,8,9,10,11,12,13,14 และ 15</p> | |
| <p>จงวาดต้นไม้ AVL ในช่องทางขวานี้ หลังการลบ 100 ออกจากต้นไม้ AVL ข้างล่างนี้</p>  | |

4. (4 คะแนน) ในข้อนี้เราจะพิจารณาถึงวิธีการแก้ปัญหาการชนกันของ hash แบบ open addressing แบบใหม่วิธีการหนึ่ง ซึ่งมีชื่อว่า Cuckoo hashing โดยวิธีการนี้จะใช้ hash function สองตัว คือ $h(x)$ และ $g(x)$ การทำงานของ Cuckoo hashing เป็นดังนี้
- สำหรับข้อมูล x ใด ๆ เราจะมีกฎเหล็กก็คือ **x จะต้องอยู่ ณ ช่อง $h(x)$ หรือไม่ก็ $g(x)$ ช่องใดช่องหนึ่งเท่านั้น** ดังนั้น ข้อมูลใด ๆ ก็ตามจะมีที่อยู่ที่เป็นไปได้เพียงสองช่องเท่านั้น
 - การตรวจสอบว่ามี x อยู่ใน hash table หรือไม่ สามารถทำได้ในเวลา $O(1)$ แน่ๆ โดยตรวจสอบว่า ช่อง $h(x)$ มี x อยู่หรือไม่ ถ้าไม่มีให้ไปตรวจสอบว่าช่อง $g(x)$ มี x อยู่หรือไม่ ถ้าไม่มีอีก เราก็สรุปได้ทันทีว่าไม่มี x อยู่
 - การเพิ่มข้อมูล ทำดังนี้ สมมติว่า ต้องการเพิ่ม x เราจะตรวจสอบว่าช่อง $h(x)$ นั้นมีข้อมูลอื่นอยู่หรือไม่ ถ้าไม่มี เราก็ใส่ x ลงไปในช่อง $h(x)$ ได้เลย แต่ถ้ามีข้อมูลอื่นอยู่ (สมมติว่าเป็น y) เราจะ "ดัน" y ที่อยู่ในช่อง $h(x)$ "ออกไป" โดยดันออกไปอยู่ในช่องที่เป็นไปได้อีกช่องหนึ่งของ y การที่ y อยู่ในช่อง $h(x)$ แสดงว่า $h(x)$ นั้นมีค่าเท่ากับ $h(y)$ หรือไม่ก็เท่ากับ $g(y)$ ถ้า $h(x)$ เท่ากับ $h(y)$ เราก็ดัน y ที่อยู่ในช่อง $h(y)$ ออกไปอยู่ในช่อง $g(y)$ แต่ถ้า $h(x)$ เท่ากับ $g(y)$ เราก็ดัน y ที่อยู่ในช่อง $g(y)$ ออกไปอยู่ในช่อง $h(y)$ ถ้าผลการดัน y ออกไป ทำให้ y ต้องไปอยู่ในช่องที่มีข้อมูลอื่นอยู่ y ก็จะเข้าไปแทนที่ข้อมูลดังกล่าว และดันข้อมูลนั้นออกไป ต่อไปเรื่อย ๆ การ "ดันออกไป" เรื่อย ๆ ใน Cuckoo Hashing อาจทำให้เกิดการดันเป็นวงวน (เช่น x ดัน y , y ดัน z และ z ย้อนกลับมาดัน x) เมื่อเกิดการดันแบบวงวนนี้ เราจะขยายขนาดตาราง และ rehash ซึ่งสามารถพิสูจน์ได้ว่า ถ้าเราควบคุมให้ load factor น้อยกว่า 0.5 แล้ว การใส่ข้อมูล (รวมการ rehash ถ้ามี) จะใช้เวลาโดยเฉลี่ยเป็น $O(1)$

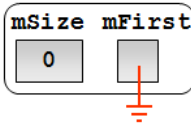
หมายเหตุ: วิธีการนี้ถูกเรียกว่า Cuckoo Hashing เพราะว่ามันทำงานเหมือนนกคูกู (นกคูกูเห่า หรือ กากาเห่า) ซึ่งจะวางไข่ในรังของนกตัวอื่น โดยเตะไข่ของตัวเองออกไป

คำถาม: เริ่มด้วย hash table ว่าง ๆ จงเติมตัวเลขลงใน hash table แบบ Cuckoo Hashing ด้านล่างนี้ หลังจากการใส่ข้อมูล 20, 50, 53, 75 และ 100 ตามลำดับจากซ้ายไปขวา กำหนดให้ $h(x)$ และ $g(x)$ ของข้อมูล เป็นดังแสดงในตารางด้านขวานี้ (รับประกันว่าข้อมูลชุดนี้จะไม่ทำให้เกิดการดันเป็นวงวน)

| x | $h(x)$ | $g(x)$ |
|-----|--------|--------|
| 20 | 9 | 1 |
| 50 | 6 | 3 |
| 53 | 9 | 4 |
| 75 | 9 | 6 |
| 100 | 1 | 9 |
| 67 | 1 | 6 |
| 105 | 6 | 9 |
| 3 | 3 | 0 |
| 36 | 3 | 3 |
| 39 | 6 | 3 |

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|-----------|---|---|---|---|---|---|---|---|---|---|----|
| เพิ่ม 20 | | | | | | | | | | | |
| เพิ่ม 50 | | | | | | | | | | | |
| เพิ่ม 53 | | | | | | | | | | | |
| เพิ่ม 75 | | | | | | | | | | | |
| เพิ่ม 100 | | | | | | | | | | | |

5. (7 คะแนน) จงวาดรูปของโครงสร้างข้อมูลหลังการ new เพื่อสร้างอ็อบเจกต์แล้วได้ทีเก็บข้อมูลว่างๆ ดังต่อไปนี้

| | |
|--|---|
| ตัวอย่าง: สร้าง singly linked list without header ว่างๆ  | สร้าง binary heap ว่างๆ (กำหนดให้ จองอาร์เรย์ขนาด 5 ช่องตอนเริ่มต้น) |
| สร้าง non-circular doubly linked list with header ว่างๆ | สร้าง circular doubly linked list with header ว่างๆ |

| | |
|---|---|
| สร้าง binary search tree ว่างๆ | สร้าง AVL tree ว่างๆ |
| สร้าง hash table with separate chaining ว่างๆ (กำหนดให้จอง table ขนาด 5 ช่องตอนเริ่มต้น) | สร้าง hash table with quadratic probing ว่างๆ (กำหนดให้จอง table ขนาด 5 ช่องตอนเริ่มต้น) |

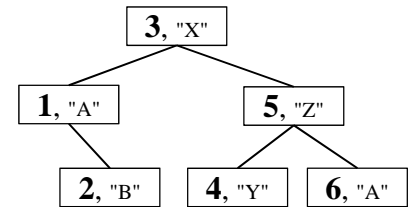
6. (4 คะแนน) ฟังก์ชัน `operator++` ข้างล่างนี้เป็นของ iterator ของ AVL tree ฟังก์ชันนี้มีที่ผิดอยู่ 2 บรรทัด

| | |
|---|---|
| <pre>class tree_iterator { protected: node* ptr; public: tree_iterator& operator++() { if (ptr->right != NULL) { ptr = ptr->right; while (ptr != NULL) { ptr = ptr->left; } } else { node *parent = ptr->parent; while (parent->right == ptr) { ptr = parent; parent = ptr->parent; } ptr = parent; } return (*this); } }</pre> | <p>จงระบุบรรทัดที่ผิด และแก้ไขให้ถูกต้อง (เขียนตอบในช่องว่างทางขวา)</p> |
|---|---|

7. (5 คะแนน) จงเขียนฟังก์ชัน `operator[]` ให้กับคลาส `list` (ที่นำเสนอในห้องแบบ circular doubly linked list with header) เพื่อให้เราสามารถ เขียน `x[k]` (ได้เหมือน vector) เพื่ออ้างอิงถึงข้อมูลตัวที่ index `k` ของ `list x` เนื่องจากเป็น linked list ฟังก์ชันนี้คงต้องใช้เวลา $O(n)$ แต่นิสิตต้องพยายามใช้โครงสร้างข้อมูลเท่าที่มี เพื่อให้ทำงานเร็วที่สุดเท่าที่จะทำได้

| |
|---|
| <pre>template <typename T> class list { // คลาสนี้ทำงานได้ตามปกติทุกอย่าง ชื่อ function และ class ย่อยเขียนมาเพื่อเพียงเป็นข้อมูลอ้างอิงเท่านั้น protected: class node { public: T data; node *prev; node *next; }; node *mHeader; size_t mSize; public: T& operator[](int k) { } };</pre> |
|---|

8. (10 คะแนน) จงเขียนฟังก์ชัน `toBalancedBST(vector<pair<KeyT, MappedT>> v)` เพื่อสร้าง binary search tree ที่ได้ดุล (คือต้นไม้ที่เตี้ยสุด) จาก vector of `pair<KeyT, MappedT>` โดยข้อมูลใน vector นี้ได้ถูกเรียงลำดับจากน้อยไปมากตามค่าของแรก (`first`) ของ `pair` เรียบร้อยแล้ว ตัวอย่างเช่น `v` เป็น vector ที่มีข้อมูลเป็น { (1,"A"), (2,"B"), (3,"X"), (4,"Y"), (5,"Z"), (6,"A") } `toBalancedBST(v)` จะ



ได้ต้นไม้ข้างขวานี้ (นิสิตสามารถสร้างฟังก์ชันอื่น ๆ เพิ่มเติมได้ตามต้องการ และให้ถือว่า `map_bst` มีฟังก์ชันให้เรียกใช้ได้เหมือน `std::map` ทุกประการ

```
map_bst<KeyT, MappedT> toBalancedBST(vector<pair<KeyT, MappedT>> v) {
    map_bst<KeyT, MappedT> t;          // create a new empty map_bst
```

9. จงเขียนรายละเอียดของคลาส `priority_queue_using_map` มีไว้สร้าง priority queue โดยอาศัย `std::map` ในการจัดเก็บข้อมูล (แทนที่จะใช้อาเรย์ตามที่ได้นำเสนอในชั้นเรียน) ภายในมีข้อมูล `mSize` แบบ `size_t` ไว้เก็บจำนวนข้อมูล และ `m` เป็น `std::map`
- (1 คะแนน) จงกำหนดว่า `map` ที่ใช้เป็น `map` ที่มี `key` และ `mapped value` เป็นประเภทใด (เขียนในตัวโปรแกรมข้างล่างนี้)
 - (9 คะแนน) จงเขียนฟังก์ชัน `push()`, `pop()` และ `top()` ที่ทำงานตามที่ `priority_queue` ควรจะทำ (ในเวลา $O(\log n)$)

```
template <typename T> // ให้ถือว่า T เป็นประเภทข้อมูลที่ต้องมี operator < แล้ว
class priority_queue_using_map {
protected:
    size_t mSize;
    std::map< ____, ____> m; // ตอบข้อ 1) ตรงที่ขีดเส้นใต้
public:
    // ตอบข้อ 2) โดยเขียนฟังก์ชันต่อไปนี้
    const T& top() {

    }

    void push(const T& val) {

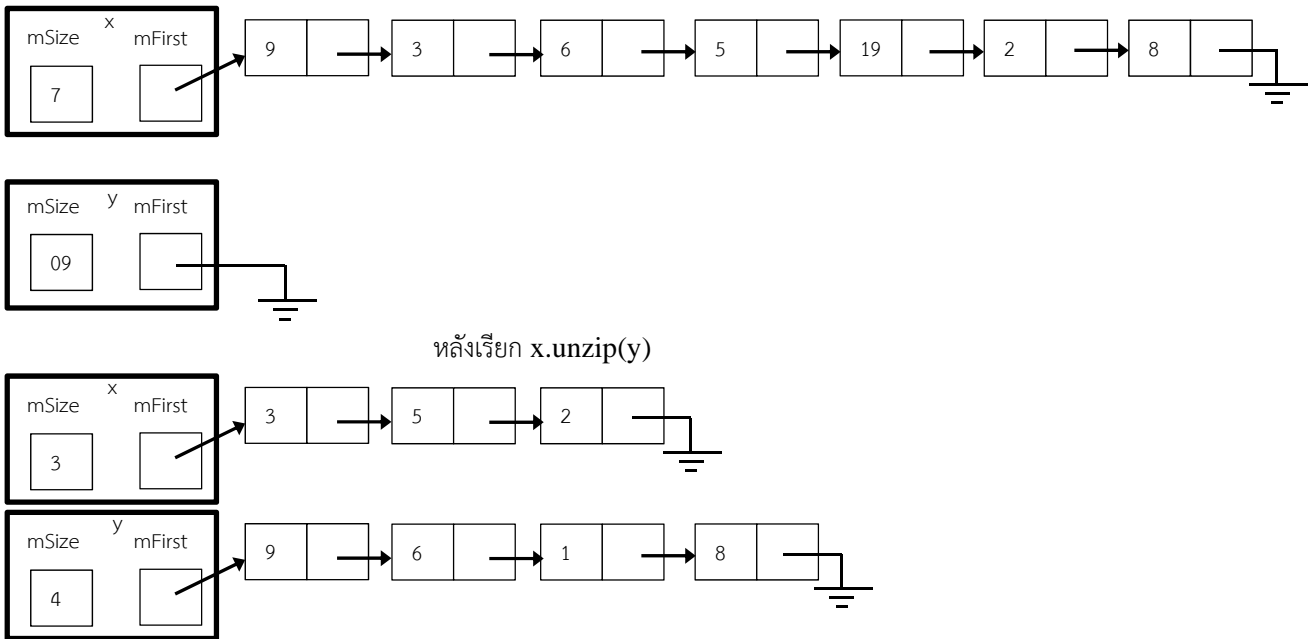
    }

    void pop() {

    }

};
```

10. (10 คะแนน) จงเขียนฟังก์ชัน `unzip(slist<T>& y)` ให้กับคลาส `slist` ที่เป็น รายการแบบโยงเดี่ยวไม่วนที่ไม่มีปมหัว (**non-circular singly linked list without header**) โดยฟังก์ชัน `unzip` นี้จะทำการย้ายข้อมูลตัวแรก, ตัวที่สาม, ตัวที่ 5, ... ไปยัง `y` ตัวอย่างเช่น หากเรามี `slist<int> x` ที่เก็บ 9,3,6,5,1,2,8 อยู่ตามลำดับ หลังจากเราเรียก `x.unzip(y)` แล้ว `x` จะเก็บ 3,5,2 ส่วน `y` จะเก็บ 9,6,1,8



ข้อกำหนด: ห้ามนิสิตเรียกใช้บริการใดๆของ `slist` หรือ `STL` แต่สามารถใช้บริการของ `node` ได้, เวลาการทำงานของ `unzip` จะต้องไม่เกิน $O(n)$ และ เรายอมรับกันว่า `y` ที่ให้มาจะเป็น `slist` ว่าง ที่ไม่มีข้อมูลเก็บอยู่)

```

template <typename T>
class slist {
protected:
    class node {
    public: T mData; node* mNext;
        node() {...}; node(T data, node* next) {...}
    };
    node* mFirst; int mSize;
public:
    void unzip(slist<T>& y) { // เดิมโค้ดที่นี่
    }
}

```

11. (10 คะแนน) โดยปกติแล้ว **min binary heap** ทั่วไปจะมีบริการในการลบข้อมูลตัวที่น้อยที่สุด แต่ไม่สามารถลบข้อมูลตัวอื่นๆ หรือแก้ไขข้อมูลได้ ซึ่งในการใช้งานบางประเภทนั้น เรามีความจำเป็นที่จะต้องลบหรือแก้ไขข้อมูลตัวใดๆ ใน **heap** ก็ได้ในเวลาอันรวดเร็ว เราจะยอมให้ผู้ใช้งานอ้างอิงถึงข้อมูลใดๆผ่านทางสิ่งที่เรียกว่า **Handle** ได้ โครงสร้างข้อมูลมีชื่อว่า **Addressable Heap (AHeap)** ซึ่งต้องมีบริการดังต่อไปนี้

- **AHeap()**; สร้าง **heap** ว่าง
- **AHandle push(const T x)**; เพิ่ม **x** เข้าไปใน **heap** แล้วคืน **AHandle** ที่ใช้สำหรับอ้างอิงถึง **x** ในอนาคต
- **T top()**; คืนข้อมูลที่มีค่าน้อยสุดใน **heap**
- **void pop()**; ลบข้อมูลที่มีค่าน้อยสุดใน **heap**
- **void deleteAt(const AHandle h)**; ลบข้อมูลที่ **h** อ้างอิงถึงออกจาก **heap**
- **void changeKey(const AHandle h, const T y)**; เปลี่ยนค่าของข้อมูลที่ **h** อ้างอิงถึงไปเป็น **y**
- **bool empty()**; คืนค่าว่า **heap** ว่างหรือไม่

ตัวอย่างการใช้งาน **AHeap** ที่ต้องการ

```
AHeap<int> aheap;
AHeap<int>::AHandle h_a = aheap.push(9);
AHeap<int>::AHandle h_b = aheap.push(2);
AHeap<int>::AHandle h_c = aheap.push(1);
AHeap<int>::AHandle h_d = aheap.push(8);
cout<<ahheap.top()<<endl; // แสดง 1 ออกมา
ahheap.pop();
ahheap.changeKey(h_a, 3);
cout<<ahheap.top()<<endl; // แสดง 2 ออกมา
ahheap.pop();
cout<<ahheap.top()<<endl; // แสดง 3 ออกมา
ahheap.pop();
AHeap<int>::AHandle h_e = aheap.push (7);
```

```
ahheap.changeKey(h_d, 6);
cout<<ahheap.top()<<endl; // แสดง 6 ออกมา
ahheap.pop();
AHeap<int>::AHandle h_f = aheap.push (12);
AHeap<int>::AHandle h_g = aheap.push (5);
ahheap.deleteAt(h_e);
ahheap.changeKey(h_f, 4);
cout<<ahheap.top()<<endl; // แสดง 4 ออกมา
ahheap.pop();
cout<<ahheap.top()<<endl; // แสดง 5 ออกมา
ahheap.pop();
// aheap ว่าง
```

จงเติมโค้ดของคำสั่ง **top**, **deleteAt**, **changeKey**, **fixDown**, **Swap** ในช่องว่างข้างล่างต่อไปนี้ให้สมบูรณ์ โดยนิสิตสามารถเพิ่มตัวแปร, เพิ่มฟังก์ชัน หรือใช้ **STL** ใดๆก็ได้ โดยคำสั่ง **deleteAt**, **changeKey**, **fixDown** จะต้องใช้เวลา $O(\log n)$ และ คำสั่ง **top** และ **Swap** จะต้องทำงานในเวลา $O(1)$ และ **handle** จะไม่ต้องเปลี่ยนอีกเลยหลังจากที่ **return** โดย **insert** ไม่ว่าข้อมูลนั้นๆจะถูกเปลี่ยนกี่ครั้งก็ตาม นิสิตไม่ต้องตรวจสอบความถูกต้องของ **handle** ในการ **deleteAt**, **changeKey**, ไม่ต้องกังวลว่า **heap** ว่างตอน **deleteAt**

```
template <typename T>
class AHeap {
public:
    typedef int AHandle;
protected:
    vector<T> data;
    vector<AHandle> hAti; // Handle at index i
    vector<int> iOfh; // Index of handle i
    vector<int> freeHandles; // Handles indices that can be reused
public:
    AHeap(){}
    AHandle push(const T x) {
        Handle h = getFreeHandle();
        data.push_back(x);
        iOfh[h] = data.size()-1;
        hAti.push_back(h);
        fixUp(data.size()-1);
        return h;
    }
    void pop() {
        Swap(0,data.size()-1);
        freeHandles.push_back(hAti.back());
        data.pop_back();
        hAti.pop_back();
        fixDown(0);
    }
}
```

```
T top() {

}

void deleteAt(AHandle h) {

}

void changeKey(AHandle h, T x) {

}

bool empty() {
    return data.size() == 0;
}

void fixUp(int index) {
    while (index > 0) {
        int p = (index - 1)/2;
        if (data[p] < data[index]) break;
        Swap(p, index);
        index = p;
    }
}

void fixDown(int index) {

}

void Swap(int i, int j) {

}

int getFreeHandle() {
    if (freeHandles.size() > 0) {
        int t = freeHandles.back();
        freeHandles.pop_back();
        return t;
    }
    iOfh.push_back(-1);
    return iOfh.size()-1;
}

};
```


12. (10 คะแนน) จากโครงสร้างข้อมูล hash table ประเภท Cuckoo Hashing ที่ได้กล่าวถึงในข้อที่ 4 ให้นักศึกษาเขียนโครงสร้างข้อมูลดังกล่าว สำหรับเก็บข้อมูลประเภท int เพื่อความง่าย กำหนดให้โครงสร้างข้อมูลนี้ไม่จำเป็นต้องทำการ rehash และข้อมูลที่ใส่เข้าไปนั้นมีเฉพาะจำนวนเต็มบวกเท่านั้น (รับประกันว่าไม่มีค่า 0 หรือ ติดลบ) และ กำหนดให้มีฟังก์ชัน `size_t h(int x)` และ `size_t g(int x)` ซึ่งเป็น hash function ที่จะคืนค่าอยู่ในช่วง 0 ถึง $2^{32}-1$ กลับมาให้ ให้นักศึกษาเขียนโครงสร้างข้อมูลที่มีฟังก์ชันต่อไปนี้
- `cuckoo(size_t s)` เป็น constructor ที่สร้าง cuckoo hashing ซึ่งมีขนาดตารางเริ่มต้นเป็น s
 - `~cuckoo()` เป็น destructor ของคลาสนี้
 - `bool find(int val)` หาดูว่าใน hash table นี้มีข้อมูล val อยู่หรือไม่ ให้ return true เมื่อพบข้อมูลเท่านั้น
 - `void remove(int val)` ทำการลบข้อมูล val ออกจาก hash table นี้
 - `bool insert(int val)` ทำหน้าที่ใส่ข้อมูล val ลงไปใน hash table นี้ โดยจะคืนค่า true ถ้าสามารถใส่ข้อมูลเข้าไปได้ กล่าวคือ ไม่ได้มีข้อมูล val อยู่ใน hash table นี้มาก่อน และ การใส่ val ไม่ทำให้เกิดการโยนค่าเป็นวงวน

```
class cuckoo {
protected: // ประกาศตัวแปรที่ต้องใช้ตรงนี้

    size_t h(int val) { ... } // มี hash function h(x) ให้ใช้อยู่แล้ว (คืนค่า 0 ถึง  $2^{32}-1$ )
    size_t g(int val) { ... } // มี hash function g(x) ให้ใช้อยู่แล้ว (คืนค่า 0 ถึง  $2^{32}-1$ )
public:
    cuckoo(size_t s) {

    }
    ~cuckoo() {

    }
    bool find(int val) {

    }
    void remove(int val) {

    }

    }
    bool insert(int val) {

    }
};
```

Common

All classes support these two capacity functions;

| | |
|----------|---|
| Capacity | <code>size_t size(); // return the number of items in the structure</code> <code>bool empty(); // return true only when size() == 0</code> |
|----------|---|

Container Class

All classes in this category support these two iterator functions.

| | |
|----------|--|
| Iterator | <code>iterator begin(); // an iterator referring to the first element</code> <code>iterator end(); // an iterator referring to the <i>past-the-end</i> element</code> |
|----------|--|

Class vector<ValueT>, list<ValueT>

| | |
|--|--|
| Element Access | <code>operator[] (size_t n);</code> |
| Modifier ที่ใช้ได้ทั้ง list และ vector | <code>void push_back(const ValueT& val);</code> <code>void pop_back();</code> <code>iterator insert(iterator position, const ValueT& val);</code> <code>iterator insert(iterator position, InputIterator first, InputIterator last);</code> <code>iterator erase(iterator position);</code> <code>iterator erase(iterator first, iterator last);</code> |
| Modifier ที่ใช้ได้เฉพาะ list | <code>void push_front(const ValueT& val);</code> <code>void pop_front();</code> <code>void remove(const ValueT& val);</code> |

Class set<ValueT, CompareT = less<ValueT> >

`unordered_set<ValueT, HashT = hash< ValueT >, EqualT = equal_to< ValueT > >`

| | |
|-----------|--|
| Operation | <code>iterator find (const ValueT& val);</code> <code>size_type count (const ValueT& val);</code> |
| Modifier | <code>pair<iterator,bool> insert (const ValueT& val);</code> <code>void insert (InputIterator first, InputIterator last);</code> <code>iterator erase (iterator position);</code> <code>iterator erase (iterator first, iterator last);</code> <code>size_type erase (const ValueT& val);</code> |

Class map<KeyT, MappedT, CompareT = less<KeyT> >

`unordered_map<KeyT, MappedT, HashT = hash<KeyT>, EqualT = equal_to<KeyT> >`

| | |
|----------------|--|
| Element Access | <code>MappedT& operator[] (const KeyT& k);</code> |
| Operation | <code>iterator find (const KeyT& k);</code> <code>size_type count (const KeyT& k);</code> |
| Modifier | <code>pair<iterator,bool> insert (const pair<KeyT,MappedT>& val);</code> <code>void insert (InputIterator first, InputIterator last);</code> <code>iterator erase (iterator position);</code> <code>iterator erase (iterator first, iterator last);</code> <code>size_type erase (const KeyT& k);</code> |

Container Adapter

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

| | |
|----------|--|
| Modifier | <code>void push (const ValueT& val); // add the element</code> <code>void pop(); // remove the element</code> |
|----------|--|

Class queue<ValueT>

| | |
|----------------|---|
| Element Access | <code>ValueT front();</code> <code>ValueT back();</code> |
|----------------|---|

Class stack<ValueT>

| | |
|----------------|----------------------------|
| Element Access | <code>ValueT top();</code> |
|----------------|----------------------------|

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

| | |
|----------------|----------------------------|
| Element Access | <code>ValueT top();</code> |
|----------------|----------------------------|

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);`