FACULTY OF ENGINEERING
CHULALONGKORN UNIVERSITY
2110211 INTRODUCTIONS TO DATA STRUCTURE
Year II, First Semester, Final Examination, Dec 7, 2022 08:30-11:30

ชื่อ-นามสกุล.....เลขประจำตัว.....ตอนเรียนที่.....เลขที่ใน CR58.....

<u>หมายเหตุ</u>

1. ข้อสอบมีทั้งหมด 10 ข้อ ในกระดาษคำถามคำตอบ 10 หน้า
2. **ไม่อนุญาตให้นำตำราและเอกสารใดๆ เข้าในห้องสอบ**
3. **ไม่อนุญาตให้ใช้เครื่องคำนวณใดๆ**
4. ห้ามการหยิบยืมสิ่งใดๆ ทั้งสิ้น จากผู้สอบอื่นๆ เว้นแต่เจ้าหน้าที่ควบคุมการสอบจะหยิบยืมให้
5. ห้ามนำส่วนใดส่วนหนึ่งของข้อสอบและสมุดคำตอบออกจากห้องสอบ
6. ผู้เข้าสอบสามารถออกจากห้องสอบได้ หลังจากผ่านการสอบไปแล้ว 45 นาที
7. เมื่อหมดเวลาสอบ ผู้เข้าสอบต้องหยุดการเขียนใดๆ ทั้งสิ้น
8. **นิสิตกระทำผิดเกี่ยวกับการสอบ ตามข้อบังคับจุฬาลงกรณ์มหาวิทยาลัย มีโทษ คือ พ้นสภาพการเป็นนิสิต หรือ ได้รับสัญลักษณ์ F ในรายวิชาที่กระทำผิด และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้**

ห้ามนิสิตพกโทรศัพท์และอุปกรณ์สื่อสารไว้กับตัวระหว่างสอบ หากตรวจพบจะถือว่านิสิตกระทำผิดเกี่ยวกับการสอบ อาจต้องพ้นสภาพการเป็นนิสิต หรือ ให้ได้รับ F และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้

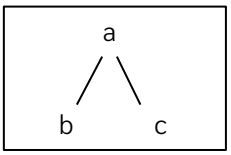
*** ร่วมรณรงค์การไม่กระทำผิดและไม่ทุจริตการสอบที่คณะวิศวกรรมศาสตร์ ***

ข้าพเจ้ายอมรับในข้อกำหนดที่กล่าวมานี้ ข้าพเจ้าเป็นผู้ทำข้อสอบนี้ด้วยตนเองโดยมิได้รับการช่วยเหลือ หรือให้ความช่วยเหลือ ในการทำข้อสอบนี้

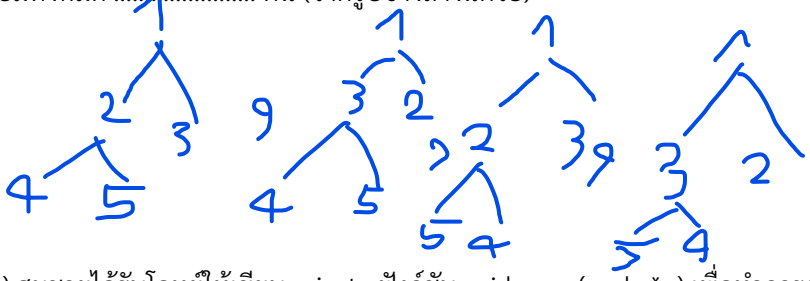
ลงชื่อนิสิต.....
วันที่.....

- ใช้ดินสอเขียนคำตอบได้
- ให้เขียนเลขที่ในใบเซ็นชื่อเข้าสอบทุกหน้า
- หากพื้นที่สำหรับเขียนคำตอบไม่เพียงพอ ให้เขียนไว้ด้านหลังของหน้านั้น ห้ามเขียนข้ามไปหน้าอื่น และให้ระบุไว้ในพื้นที่สำหรับเขียนคำตอบว่า “มีต่อด้านหลัง”

1. (5 คะแนน) กำหนดให้ CP::priority_queue<int,std::greater<int>> มีเก็บข้อมูล 5 ตัวได้แก่ตัวเลข 1 ถึง 5 อย่างละตัว จงวาด Binary Heap ที่เก็บข้อมูลของ CP::priority_queue นี้ที่เป็นไปได้ทั้งหมดที่แตกต่างกัน (ตัวอย่างด้านขวานี้เป็นตัวอย่างการวาด Binary Heap ขนาด 3 ปม โดยไม่ได้ระบุค่าไว้ นิสิตจะต้องตอบเป็น Binary Heap ขนาด 5 ปมโดยระบุค่าในปมต่าง ๆ ด้วย)



1.1 คำตอบมีทั้งหมด .....4..... ต้น (วาดรูปข้างล่างนี้ด้วย)



2. (5 คะแนน) สมชายได้รับโจทย์ให้เขียน private ฟังก์ชัน void erase(node *n) เพื่อทำการลบ node ที่ถูกชี้โดยตัวแปร n ออกจากโครงสร้างข้อมูลประเภท non-circular singly linked list with header โดยรับประกันว่า n จะชี้ไปยังตัวแปร node ของ linked list นี้เสมอ (โดยไม่เคยชี้ไปที่ mHeader เลยด้วย) อย่างไรก็ตาม ส่วนของโปรแกรมที่สมชายเขียนนี้มีจุดที่ผิดพลาดอยู่ จงระบุว่าจุดที่ผิดพลาดคือจุดใดและผิดพลาดอย่างไร

```
01: class list {
02:   class node { // คลาส node ของ non-circular singly linked list with header
03:     friend class list;
04:     public:
05:       T data;
06:       node *next; // this is singly linked list, there is no prev
07:   };
08:
09:   private:
10:     // คลาสนี้มีฟังก์ชันและตัวแปรตามที่ควรทำงานได้ของ non-circular singly linked list with header
11:     void erase(node *n) {
12:       node* tmp = n->next;
13:       n->data = tmp->data;
14:       n->next = tmp->next;
15:       delete tmp;
16:       mSize--;
17:     }
18: }
```

.....

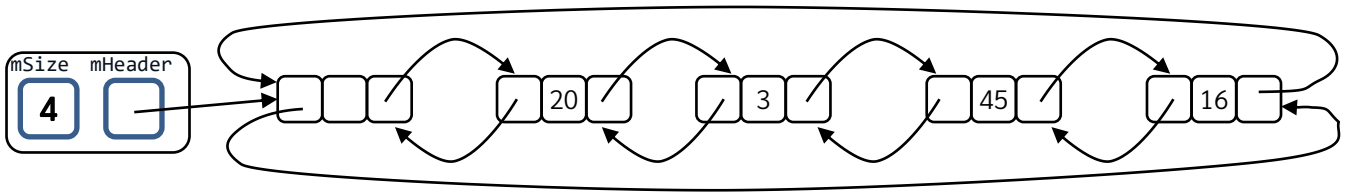
.....

.....

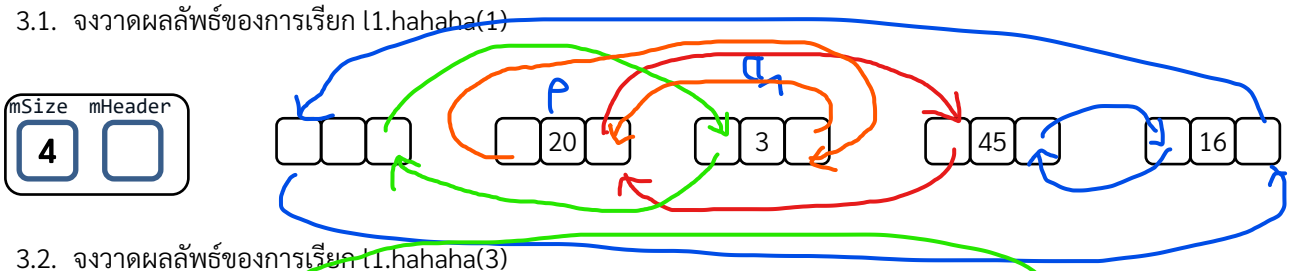
.....

3. (6 คะแนน) กำหนดให้มีฟังก์ชัน void hahaha(node *n) ซึ่งเป็นฟังก์ชันในคลาส CP::list<int> และกำหนดให้รูปต่อไปนี้แสดงถึงตัวแปร CP::list<int> l1 ที่ประกอบด้วยข้อมูล 4 ตัว จงวาดผลลัพธ์ที่เกิดขึ้นของการเรียก hahaha(n) โดยที่ n มีค่าเป็นตามข้อย่อยต่อไปนี้ (ให้คิดแต่ละข้อแยกกัน กล่าวคือ ให้ถือว่าแต่ละข้อเรียกโดยที่ l1 มีค่าเป็นดังรูป) โดยให้วาดผลลัพธ์โดยการลากเส้นเชื่อมของปมต่างๆ ใหม่ โดยห้ามแก้ไขค่าตัวเลขต่าง ๆ ในรูป

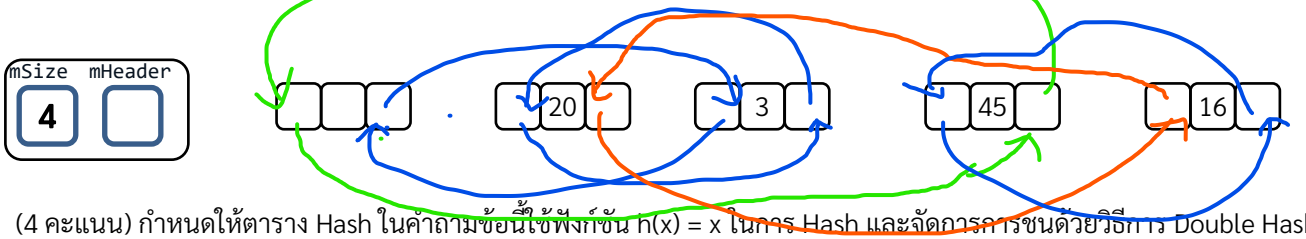
```
void hahaha(size_t count) {
  if (mSize <= 1) return;
  node *p = mHeader->next;
  while (count > 0) {
    count--;
    if (p->next->data < p->data) {
      node *a1 =  p->next;
      a1->next->prev = p;
      a1->prev->next = a1->next;
      a1->prev = a1->prev->prev;
      a1->prev->next = a1;
      a1->next = a1->next->prev;
      a1->next->prev = a1;
    } else {
      p =  p->next;
    }
  }
}
```



3.1. จงวาดผลลัพธ์ของการเรียก l1.hahaha(1)



3.2. จงวาดผลลัพธ์ของการเรียก l1.hahaha(3)



4. (4 คะแนน) กำหนดให้ตาราง Hash ในคำถามข้อนี้ใช้ฟังก์ชัน h(x) = x ในการ Hash และจัดการการชนด้วยวิธีการ Double Hashing โดยที่ฟังก์ชัน Hash อันที่สองที่ใช้คือที่ใช้คือเอาตัวเลขแต่ละหลักมาบวกกัน เช่น h2(7184)=7+1+8+4 เป็นต้น จงเติมตารางข้างล่างที่ระบุถึง หมายเลขช่องแรกที่พยายามใส่ข้อมูล (Home Slot) และ ลำดับของหมายเลขช่องที่ถูกตรวจ (Probe Sequence) นอกเหนือจาก Home Slot เมื่อมีการใส่ข้อมูล**ตามลำดับ**ดังต่อไปนี้ในตาราง Hash ขนาด 11 ช่องพร้อมทั้งระบุข้อมูลในตาราง Hash เมื่อใส่ข้อมูลทั้งหมดเรียบร้อยแล้วด้วย

Key	Home Slot	Probe Sequence
45		
24		
16		
3		
34		
33		
22		
52		
31		
86		

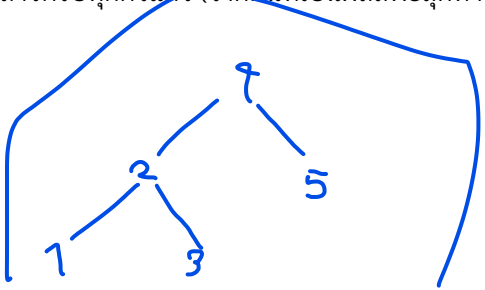
ข้อมูลในตารางหลังใส่ข้อมูลทั้งหมดแล้ว

หมายเลขช่อง	ค่าที่เก็บ
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

5. (4 คะแนน) สมมติให้เรามีต้นไม้ AVL ว่างอยู่ 1 ต้น และเราต้องการใส่ข้อมูลตัวเลข 1 – 5 ทีละตัว ลงไปในต้นไม้นี้ จงหาลำดับการใส่ข้อมูลดังกล่าว พร้อมทั้งวาดต้นไม้ AVL ที่เกิดจากลำดับการใส่ข้อมูลดังกล่าว ที่ทำให้ต้นไม้นั้นมีค่า balance ที่ปมรากไม่ใช่ 0 และการใส่ข้อมูลตามลำดับนั้นไม่ทำให้เกิดการหมุนของปมใด ๆ เลย

5.1. ลำดับของข้อมูลที่ใส่คือ.....4 2 5 1 3..... (หากมีหลายลำดับที่ตรงตามเงื่อนไข ตอบอันเดียวพอ)

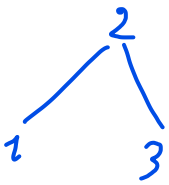
5.2. จงวาดต้นไม้ที่เป็นผลลัพธ์ของการใส่ข้อมูลดังกล่าวครบทุกตัวแล้ว (วาดต้นที่เป็นผลลัพธ์สุดท้ายต้นเดียวพอ)



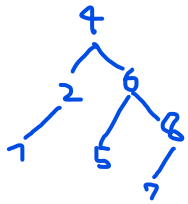
6. (6 คะแนน) ฟังก์ชัน void recursive(node *n, size_t level) ในข้อนี้เป็นฟังก์ชันของคลาส CP::map_bst โดยฟังก์ชันนี้จะทำการพิมพ์ค่าบางอย่างออกทางหน้าจอ ในแต่ละข้อย่อยต่อไปนี้มีผลลัพธ์ของการเรียกใช้ฟังก์ชัน recursive(mRoot,0) ของต้นไม้ CP::map_bst<int, bool> อยู่ จงวาดต้นไม้ Binary Search Tree ที่ทำให้การเรียก recursive(mRoot,0) ได้ผลตามที่ระบุ โดยวาดเฉพาะ key ของแต่ละปม

```
void recursive(node *n, size_t level) {
  if (n != nullptr) {
    cout << n->data.first << " ";
    recursive(n->right, level + 1);
    cout << "x" << level << " ";
    recursive(n->left, level + 1);
  }
}
```

6.1. 2 3 x1 x0 1 x1



6.2. 4 6 8 x2 7 x3 x1 5 x2 x0 2 x1 1 x2



7. (10 คะแนน) ในข้อย่อยต่อไปนี้ให้ตอบโดยเลือกคำตอบที่ถูกต้องที่สุด แต่ละข้อย่อยมีคะแนน 1 คะแนน หากไม่ตอบในข้อย่อยใด จะได้คะแนน 0 แต่ถ้าหากตอบผิดในข้อย่อยใด จะได้คะแนน -0.5 ต่อข้อ อย่างไรก็ตาม คะแนนที่น้อยที่สุดที่เป็นไปได้ของข้อนี้คือ 0 (กล่าวคือ ถึงแม้จะตอบผิดจนได้คะแนนรวมติดลบ ก็จะถือว่าได้คะแนนเป็น 0) ให้ตอบโดยการเขียนตัวเลือกที่ต้องการลงในตารางข้างล่างนี้เท่านั้น

ข้อ 7.1	ข้อ 7.2	ข้อ 7.3	ข้อ 7.4	ข้อ 7.5	ข้อ 7.6	ข้อ 7.7	ข้อ 7.8	ข้อ 7.9	ข้อ 7.10

** การทำเครื่องหมายบนตัวเลือกข้างล่างจะไม่นับเป็นการตอบ จะยึดการตอบจากการเติมตัวเลือกลงในตารางข้างบนนี้เท่านั้น **

7.1. ข้อใดคือเวลาการทำงานที่แย่ที่สุดในการเพิ่มข้อมูลให้แก่ Doubly Linked List ณ หน้าข้อมูลลำดับที่ k เมื่อ 0 <= k < n และ n คือจำนวนข้อมูลใน Double Linked List

ก. O(n log n)
ข. O(log n)
ค. O(n) → ผิด Node K
ง. O(1)

7.2. ส่วนของโปรแกรมทางขวานี้ทำงานอะไร? (เมื่อ header เป็นปมหัวของ Circular Doubly Linked List with Header) จงเลือกข้อที่เหมาะสมที่สุด

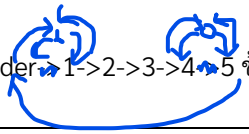
ก. พิมพ์ success ถ้าหา x ไม่เจอ
ข. พิมพ์ fail ถ้าหา x ไม่เจอ
ค. พิมพ์ success ถ้ามีข้อมูลที่มีค่าเท่ากับ 1
ง. พิมพ์ fail ถ้ารายการว่าง

```
// ส่วนของโปรแกรมสำหรับข้อ 7.2
void func(int x) {
  int flag = 0;
  if (header != null)    {
    node<int>* temp = header->next;
    while ( (temp != header) && (!(temp->data == x)) ) {
      temp = temp->next;
      flag = 1;
      break;
    }
  }
  if (flag) cout << "success" << endl;
  else cout << "fail" << end;
}
```

7.3. เหตุใดเราจึงอยากให้ Binary Search Tree มีความสูงสมดุล?

ก. เพื่อประหยัดหน่วยความจำ
ข. เพื่อให้เขียนอ่านหน่วยความจำได้เร็วขึ้น
ค. เพื่อให้ง่ายในการจัดเก็บ
ง. เพื่อให้เข้าถึงข้อมูลได้อย่างรวดเร็ว

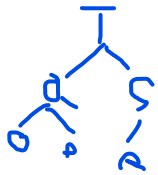
7.4. จงพิจารณา Circular Doubly Linked List With Header ที่มีปม ดังนี้ mHeader->1->2->3->4->5 ข้อใดคือผลลัพธ์หลังจากการทำงานของโปรแกรมดังต่อไปนี้



```
node<int>* tmp = new node<int> (6, header, header->next);
node<int>* tmp1 = new node<int> (0, header->prev->prev, header->prev);
header->next = tmp;
tmp->next->prev = tmp;
header->prev->prev = tmp1;
tmp1->prev->next = tmp1;
```

ก. header->0->1->2->3->4->5->6
ข. header->1->2->3->4->5->6
ค. header->6->1->2->3->4->0->5
ง. header->0->1->2->3->4->5

7.5. ถ้าผลของการทำ Post-Order Traversal ของ Binary Tree ต้นหนึ่งคือ O P Q R S T แล้ว ผลลัพธ์ของการทำ Pre-Order Traversal ที่เป็นไปได้ของต้นไม้ต้นนี้คือข้อใดต่อไปนี้



ก. T Q O P S R
ข. T Q R S O P
ค. T Q O S P R
ง. T O Q R P S

7.6. ขนาด (จำนวนปม) ของต้นไม้ที่มีมากที่สุดที่เป็นไปได้ที่ทำให้ลำดับของปมเมื่อทำ Pre-Order Traversal เท่ากับลำดับของปมเมื่อทำ Post-Order Traversal เสมอคือเท่าไร เมื่อต้นไม้ไม่มีข้อมูลซ้ำกันเลย

ก. 3
ข. 1
ค. 2
ง. ∞

7.7. จงพิจารณาส่วนของโปรแกรมดังต่อไปนี้สำหรับ การหมุนลูกทางซ้ายขึ้นมา สำหรับต้นไม้ AVL โดยแต่ละปมมีลูกทางซ้าย (left) ลูกทางขวา (right) และความสูง (height) อยู่ ค่าที่ถูกต้องของ **A** และ **B** คืออะไร

```
node* rotate_left(node* x) {
      node* w = x->left;
      x->set_left(w->right);
      w->set_right(x);
      x->height = std::max(x->left->height,
                           x->right->height)+1;
      w->height = std::max(A,B)+1;
      return w;
}
```

ก. w->right->height, w->height
ข. w->right->height, x->height
ค. w->left->height, w->height
ง. w->left->height, x->height

7.8. ปัญหาใดต่อไปนี้เป็นปัญหาหลักที่เกิดขึ้นระหว่างการทำ Linear Probing ของ Open Addressing Hash Table

ก. Primary collision ข. Secondary collision ค. Separate chaining ง. Rehashing

7.9. หาก Hash Table ของเราต้องการเก็บข้อมูลประเภท int แล้ว และเรามีสมมติฐานว่าค่าใด ๆ ที่เป็นไปได้ของ int มีโอกาสที่จะถูกใส่เข้ามาใน Hash ของเราด้วยความน่าจะเป็นเท่า ๆ กันฟังก์ชัน Hash ใดต่อไปนี้มีโอกาสทำให้เกิดการชนกันมากที่สุด โดยกำหนดให้ค่าที่ต้องการ hash คือ x และฟังก์ชันนั้นจะคืนค่าผลลัพธ์ของการหาค่า hash ของ x

ก. return x - 64;
ข. return x & 64;
ค. return x ^ 64;
ง. return x >> 64;

7.10. ข้อใดต่อไปนี้ไม่ใช่วิธีการในการหาช่องว่างเมื่อเกิดการชนกันของข้อมูลใน Open Addressing Hash Table

ก. Linear probing ข. Quadratic probing ค. Double hashing ง. Rehashing

ในข้อต่าง ๆ ต่อไปนี้เป็นการเขียน code คะแนนที่ได้จะแปรผันตามความถูกต้อง และ ประสิทธิภาพในการทำงาน โดยเฉพาะการเลือก data structure ในการใช้งานที่ถูกต้อง

8. (10 คะแนน) ในข้อนี้เราจะพิจารณาถึงคลาส CP::map_avl ที่ถูกปรับแก้ให้แต่ละปม มีการเพิ่มตัวแปร count ให้เก็บจำนวนของปมทั้งหมดของ sub-tree ที่ปมนี้เป็นรากอยู่ด้วย โดยให้นิสิตถือว่าค่าของ count นั้นถูกต้องอยู่แล้ว จงเพิ่มบริการ size_t countLessThan(KeyT a) ให้แก่ map_avl เพื่อหาจำนวนข้อมูลใน map นี้ที่มี key อยู่ก่อนค่า a

```
template <typename KeyT,typename MappedT,typename CompareT = std::less<KeyT> >
class map_avl {
protected:
  node *mRoot; CompareT mLess; size_t mSize;
  class node {  friend class map_bst; …
      protected: ValueT data; node *left;   node *right;   node *parent;   int height;  int count;  …
  }
  // มีฟังก์ชันอื่น ๆ ตามปรกติ แต่มิได้เขียนไว้เพื่อประหยัดพื้นที่
public:
```

```
        // เติม code ตรงนี้

      size_t countLessThan(KeyT a) {
        // เติม code ตรงนี้

      }
}
```

9. (10 คะแนน) Hash table แบบ separate chaining ตามที่เรียนในชั้นเรียนนั้น แต่ละช่องของตาราง Hash นั้นเก็บข้อมูลเป็น vector ของ pair ของ KeyT และ MappedT ในโจทย์ข้อนี้เราจะพิจารณา Hash Table แบบ Separate chaining รูปแบบใหม่ ที่ในแต่ละช่องของตาราง Hash นั้นเก็บข้อมูลประเภท Hash Table แบบ separate chaining แทน vector ดังกล่าว กล่าวคือ Hash รูปแบบใหม่นี้ประกอบด้วย Hash Table ซ้อนด้วย Hash Table อีกครั้งหนึ่ง เราเรียก Hash แบบใหม่นี้ว่า Two Level Hash Table โดยเรียกตาราง Hash หลักว่า “ตาราง Hash ด้านนอก” และเรียกแต่ละช่องในตาราง Hash หลักซึ่งเป็น Hash table อีกตัวว่า “ตาราง Hash ด้านใน” ในโจทย์ข้อนี้นิสิตจะต้องเขียนคลาส two_level_hash ซึ่งเป็น Two Level Hash Table ที่มีหลักการทำงานดังที่ได้กล่าวมาแล้ว

จากหลักการข้างต้น two_level_hash ต้องการใช้ Hash function 2 ตัวที่แตกต่างกัน กำหนดให้มี template parameter OutsideHashT และ HashT เป็นตัวระบุคลาสของ Hash function สำหรับตาราง Hash ด้านนอก และ ตาราง Hash ด้านใน ตามลำดับ โดยเราจะให้ตาราง Hash ด้านในแต่ละตัวนั้นใช้ Hash function เดียวกันทั้งหมดคือ HashT()

ในข้อนี้ กำหนดให้ two_level_hash มี data member ดังต่อไปนี้เท่านั้น

- size_t mSize
- std::vector<std::unordered_map<KeyT,MappedT,HashT,EqualT>> mBuckets
- OutsideHashT mHasher
- EqualT mEqual
- float maxLoadFactor;

นิสิตจะต้องเขียนฟังก์ชันต่อไปนี้ให้กับ two_level_hash

- MappedT& operator[](const KeyT& key)
    - รีเทิร์นค่าที่เก็บคู่กับ key ออกมา แต่ถ้าไม่มี key ให้สร้างด้วยค่า default แล้วรีเทิร์นค่านั้น
- std::pair<outside_iterator,bool> insert(const ValueT& val)
    - ใส่ค่า val ลงไปใน hash ถ้าหากยังไม่มี key ของ val อยู่ในข้อมูลของเรา แต่ถ้ามีข้อมูลอยู่แล้ว การเรียกฟังก์ชันนี้จะไม่เปลี่ยนแปลงข้อมูลใด ๆ
    - สมมุติว่ามีคนเขียนโค้ดของ iterator ของ two_level_hash ไว้ให้แล้ว โดยมีชื่อคลาสว่า outside_iterator

- outside_iterator มีคอนสตรัคเตอร์ที่เรียกได้คือ outside_iterator(a,b,c,d) โดยที่ a คือ iterator ที่ชี้ตำแหน่งใน mBuckets, b คือ iterator ชี้ตำแหน่งแรกสุดของ mBuckets, c คือ iterator ชี้ตำแหน่งหลังตำแหน่งท้ายสุดของ mBuckets และ d คือ iterator ที่ชี้ตำแหน่งใน hash table ย่อย ทั้งนี้การชี้ตำแหน่งในตารางนอกสุดนั้นขึ้นกับการเขียนคลาสของนิสิต
- เมธอด insert ต้องคืนค่า pair ของ outside_iterator และ boolean โดย outside_iterator ระบุตำแหน่งที่ val อยู่ ส่วน boolean จะมีค่าเป็นจริงก็ต่อเมื่อ key ของ val นั้นไม่เคยมีอยู่ใน hash มาก่อน

- size_t erase(const KeyT &key)
  - ลบข้อมูลที่มี คีย์มีค่าเท่ากับ key ออกจาก hash table
  - คืนค่า 0 ถ้าไม่มีข้อมูลดังกล่าว หรือ คืนค่า 1 ถ้ามีข้อมูลดังกล่าวอยู่ใน two_level_hash

เพื่อความสะดวก ส่วนของโปรแกรมต่อไปนี้แสดงถึงตัวอย่างของ hash table แบบ separate chaining ตามที่ใช้ในวิชานี้ โดยแสดงเฉพาะส่วนที่สำคัญ

```
template <typename KeyT,
          typename MappedT,
          typename HasherT = std::hash<KeyT>,
          typename EqualT = std::equal_to<KeyT> >
class unordered_map {
  protected:
    typedef std::pair<KeyT,MappedT>                 ValueT;
    typedef std::vector<ValueT>                     BucketT;
    typedef typename BucketT::iterator              ValueIterator;
    typedef typename std::vector<BucketT>::iterator BucketIterator;

    std::vector<BucketT> mBuckets;
    size_t               mSize;
    HasherT              mHasher;
    EqualT               mEqual;
    float                mMaxLoadFactor;

public:
   typedef hashtable_iterator iterator;

   MappedT& operator[](const KeyT& key) {
      size_t bucketIdx = hash_to_bucket(key);
      ValueIterator it = find_in_bucket(mBuckets[bucketIdx],key);
      if (it == mBuckets[bucketIdx].end()) { // if not found, insert the new one
        it = insert_to_bucket(std::make_pair(key, MappedT()),bucketIdx);
      }
      return it->second;
    }
    std::pair<iterator,bool> insert(const ValueT& val) {
      std::pair<iterator,bool> result;
      size_t bucketIdx = hash_to_bucket(val.first);
      ValueIterator it = find_in_bucket(mBuckets[bucketIdx], val.first);
      result.second = false;
      if (it == mBuckets[bucketIdx].end()) {
        it = insert_to_bucket(val, bucketIdx );
        result.second = true;
      }
      result.first = iterator(it,
                              mBuckets.begin()+bucketIdx,
                              mBuckets.end());
      return result;
    }
    size_t erase(const KeyT &key) {
      size_t bucketIdx = hash_to_bucket(key);
      ValueIterator it = find_in_bucket(mBuckets[bucketIdx],key);
      if (it == mBuckets[bucketIdx].end()) {
        return 0;
      } else {
        mBuckets[bucketIdx].erase(it);
        mSize--;
        return 1;  // erase 1 element
      }
    }
};
```

จงเขียนคลาส two_level_hash ในที่ว่างด้านล่างนี้ จัดที่ให้ดี **ถ้าที่ไม่พอให้เขียนที่ด้านหลังหน้า 8 เท่านั้น**

10. (10 คะแนน) โครงสร้างข้อมูลแบบ CP::priority_queue นั้นมีฟังก์ชันให้ใช้งานไม่มากนัก เราต้องการเพิ่มความสามารถให้กับคลาสดังกล่าวโดยต้องการให้เราสามารถ “ค้นหา” และ “เปลี่ยนแปลง” ค่าภายใน priority_queue นี้ได้ จงแก้ไขคลาส CP::priority_queue ให้มีฟังก์ชัน bool contain(const T &value) ซึ่งจะคืนค่า true ก็ต่อเมื่อมีค่า value อยู่ใน priority_queue ของเรา และ ฟังก์ชัน void change(const T& old_value, const T& new_value) ซึ่งจะเปลี่ยนค่าใด ๆ ใน priority_queue ของเราที่มีค่าเป็น old_value ให้มีค่าเป็น new_value โดยที่ยังทำให้ priority_queue ของเรานั้นยังคงโครงสร้างแบบ binary heap อยู่

**** รับประกันว่า ค่าใน priority_queue แตกต่างกันทั้งหมด และ change จะไม่ทำให้ค่าใน priority_queue ซ้ำกันแน่นอน****

ในโจทย์ข้อนี้ นิสิตสามารถเพิ่มเติม data member อื่นใดเข้าไปใน priority_queue นี้ได้ และนิสิตสามารถแก้ไขฟังก์ชันใด ๆ ที่มีอยู่แล้วของ priority_queue นี้ได้เช่นกัน เพื่อความสะดวก มีส่วนของโปรแกรมของคลาส CP::priority_queue ให้แล้วในด้านล่างนี้ นิสิตสามารถเขียนคำตอบลงไปในคลาสดังกล่าวได้เลย หากต้องการแก้ไขเพิ่มเติมฟังก์ชันใด ๆ สามารถใช้วิธีเขียนแทรกหรือเขียนใหม่ไว้ด้านข้างของฟังก์ชันดังกล่าวได้เลย

เงื่อนไขจำเป็น (แต่อาจจะไม่พอเพียง) ต่อการได้คะแนนเต็มในข้อนี้คือฟังก์ชัน push, pop นั้นจะต้องใช้เวลา O(log n) ในขณะที่ contain, change ต้องใช้เวลาเป็น O(log(n) * log(n)) และ top ควรจะใช้เวลาเป็น O(1)

```
template <typename T,typename Comp = std::less<T> >
class priority_queue {
  protected:
    T *mData; size_t mCap; size_t mSize; Comp mLess; // เพิ่ม data member ได้

    // ให้ถือว่าฟังก์ชัน expand() ของคลาสนี้ทำงานได้ถูกต้องเสมอ นิสิตไม่จำเป็นต้องแก้ไขฟังก์ชันดังกล่าว
    void fixUp(size_t idx) {
      T tmp = mData[idx];
      while (idx > 0) {
        size_t p = (idx - 1) / 2;
        if ( mLess(tmp,mData[p]) ) break;
        mData[idx] = mData[p];
        idx = p;
      }
      mData[idx] = tmp;
    }

    void fixDown(size_t idx) {
      T tmp = mData[idx];
      size_t c;
      while ((c = 2 * idx + 1) < mSize) {
        if (c + 1 < mSize && mLess(mData[c],mData[c + 1]) ) c++;
        if ( mLess(mData[c],tmp) ) break;
        mData[idx] = mData[c];
        idx = c;
      }
      mData[idx] = tmp;
    }

  public:
    // ให้ถือว่า constructor, destructor, empty(), size() ของคลาสนี้ทำงานได้ถูกต้องเสมอ นิสิตไม่จำเป็นต้องแก้ไขฟังก์ชันดังกล่าว
    const T& top() { return mData[0]; }

    void push(const T& element) {
      if (mSize + 1 > mCap)
        expand(mCap * 2);
      mData[mSize] = element;
      mSize++;
      fixUp(mSize-1);
    }

    void pop() {
      mData[0] = mData[mSize-1];
      mSize--;
      fixDown(0);
    }
    // เติม code ตรงนี้ ถ้าที่ไม่พอให้เขียนไว้ด้านหลังของหน้า 9 เท่านั้น

};
```

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

vector<T> และ list<T>

Element Access สำหรับ vector	T& operator[] (size_t n); T& at(int dx);

Modifier ที่ใช้ได้ทั้ง list และ vector	void push_back(const T& val);
	void pop_back();
	iterator insert(iterator position, const T& 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 ที่ใช้ได้เฉพาะlist	void push_front(const T& val);
	void pop_front;
	void remove(const T& val);

set<T>

Operation	iterator find (const T& val); size_t count (const T& val);
Modifier	pair<iterator,bool> insert (const T& val); void insert (InputIterator first, InputIterator last); iterator erase (iterator position); iterator erase (iterator first, iterator last); size_t erase (const T& val);

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 T& val); // add the element
	void pop(); // remove the element

	queue<T>	stack<T> and priority_queue<T, ContainerT = vector<T>, CompareT = less<T> >
Element Access	T front(); T back();	T top();

Useful functions

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