FACULTY OF ENGINEERING
CHULALONGKORN UNIVERSITY
2110211 INTRODUCTIONS TO DATA STRUCTURE
Year II, First Semester, Final Examination, Dec 9, 2020 08:30-11:30

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

<u>หมายเหตุ</u>

1. ข้อสอบมีทั้งหมด 14 ข้อ ในกระดาษคำถามคำตอบ 8 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) ข้อความต่อไปนี้เป็นการพูดถึงประสิทธิภาพในการทำงานของฟังก์ชันต่าง ๆ จงระบุว่าข้อความแต่ละข้อความนั้น “จริง” หรือ “เท็จ” หากตอบว่า “เท็จ” ให้ระบุว่าข้อความดังกล่าวเป็น “เท็จ” เพราะเหตุใด (ในแต่ละข้อ หากตอบถูกต้องจะได้คะแนน 1 สำหรับข้อที่คำตอบที่ถูกต้องคือจริง การตอบ “เท็จ” มา จะได้คะแนน -0.5 แต่สำหรับข้อที่คำตอบที่ถูกต้องคือเท็จ การตอบ “จริง” มา จะได้คะแนน -1 หรือหากให้เหตุผลที่ผิด จะได้คะแนน 0.5 แต่ถ้าหากไม่ตอบเลยจะได้คะแนน 0)

   1.0 (ตัวอย่าง) คำสั่ง size() ในโครงสร้างข้อมูล CP::list ใช้เวลา O(n)
   เท็จ เนื่องจาก size() ใช้เวลา O(1)

   1.1 การเพิ่มข้อมูลเข้าไปใน Binary Heap แต่ละครั้งใช้เวลา $\Theta(\log n)$

   .....

   1.2 การหาข้อมูลใน AVL Tree ใช้เวลา O(log n)

   .....

   1.3 คำสั่ง operator++ ของ iterator ของ CP::map_bst ใช้เวลา O(log n)

   .....

   1.4 สำหรับโครงสร้างข้อมูล CP::list นั้น การเรียกดูข้อมูลลำดับที่ 10 จากต้น list (ถ้ามี) ใช้เวลา O(1)

   .....

   1.5 ค่า load factor ของ Hash Table แบบ open addressing นั้นมีค่าน้อยกว่า 1 เสมอ

   .....

2. (5 คะแนน) ในโครงสร้างข้อมูล Hash Table แบบ Separate Chaining ตามที่ใช้ในคลาส CP::unordered_map นั้น แต่ละช่องในตารางจะเป็นโครงสร้างข้อมูลแบบ vector<ValueT> ถ้าหากเราออกแบบ CP::unordered_map ใหม่โดยให้แต่ละช่องในตารางเป็นโครงสร้างข้อมูลแบบ set<ValueT> แทน จะมีข้อดีและข้อเสียในด้านประสิทธิภาพด้านพื้นที่และเวลาที่ใช้ต่างกันอย่างไร โดยใช้สมมติฐานคือ ข้อมูลที่ใส่เข้ามาใน Hash Table นั้นมีการกระจายตัวที่ดี

   .....
   .....
   .....
   .....

3. (5 คะแนน) จงระบุประสิทธิภาพเชิงเวลาของส่วนของโปรแกรมต่อไปนี้

`a1(int n) {`
`  stack<int> s;`
`  for (int i = 0;i < n;i++)`
`  s.push(10);`
`  for (int i = 0;i < n;i++)`
`  s.pop();`
`}`
	a1(n) ใช้เวลา เป็น Θ(?)

`a2(int n) {`
`  set<int> s;`
`  for (i=1; i<=n; i++) {`
`  for (j=i; j<=n; j++) {`
`  s.insert(99)`
`  }`
`  }`
`  return s`
`}`
	a2(n) ใช้เวลา เป็น Θ(?)

`a3(int n) {`
`  list<int> l;`
`  for (int i = 0;i < n;i++)`
`  l.insert(l.begin(),i);`
`  for (int i = 0;i < n;i++)`
`  find(l.begin(),l.end(),3);`
`}`
	a3(n) ใช้เวลา เป็น Θ(?)

`a4(int n) {`
`  priority_queue<int> pq;`
`  for (int i = 0;i < n;i++) {`
`  pq.push(i);`
`  }`
`  Vector<priority_queue<int>> v;`
`  for (int i = 0;i < n;i++) {`
`  v.push_back(pq);`
`  }`
`}`
	a4(n) ใช้เวลา เป็น Θ(?)

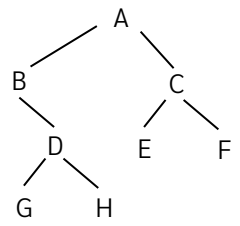
`a5(int n) {`
`  CP::map_bst<int,int> m;`
`  for (int i = 0;i < n;i++) {`
`  m[i] = 0;`
`  }`
`}`
	a5(n) ใช้เวลา เป็น Θ(?)

4. (4 คะแนน) เราต้องการใส่ข้อมูลตัวเลขจำนวน 8 ตัว ได้แก่ 1,2,3,4,5,6,7,8 เข้าไปใน AVL Tree ว่าง โดยไม่ต้องการให้การใส่ข้อมูลนั้นเกิดการ “หมุน” ใน AVL Tree เลย จงระบุลำดับการใส่ข้อมูลดังกล่าว

.....

.....

5. (4 คะแนน) จากรูปต้นไม้ Binary Tree นี้ จงแสดงผลการทำ Inorder Traversal และ Postorder Traversal



ผลการทำ Inorder Traversal

ผลการทำ Post Traversal

6. (4 คะแนน) จงวาดต้นไม้ Binary Heap (แบบตัวมากสุดอยู่บน) สุดท้ายที่เกิดจากการทำงานต่อไปนี้

ข้อย่อย	การทำงาน	รูป Binary Heap
(1)	เริ่มด้วย `Binary Heap` ว่าง ใส่ข้อมูล `6,5,4,3,2,1` ตามลำดับ แล้วทำการ `pop 1` ครั้ง	
(2)	เริ่มด้วย `Binary Heap` ว่าง ใส่ข้อมูล 1 ถึง 100 ตามลำดับ แล้วทำการ `pop 98` ครั้ง	

7. (5 คะแนน) ให้พิจารณาตาราง Hash แบบ open addressing ที่ใช้วิธีการแก้การชนกันแบบ Linear Probing โดยใช้ Hash Function เป็นดังส่วนของโปรแกรมนี้ if (x % 2 == 0) return 2; else return x % 13 จงเขียนข้อมูลในตาราง Hash ดังกล่าว ซึ่งเป็นผลของการดำเนินการต่อไปนี้ตามลำดับ (ช่องที่เคยมีข้อมูลให้เขียนเครื่องหมาย X สำหรับช่องที่ไม่มีข้อมูลให้ปล่อยว่างไว้)

การทำงาน	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]
(ตัวอย่าง) เพิ่ม 10			10										
(ตัวอย่าง) เพิ่ม 24			10								23		
เพิ่ม 1													
เพิ่ม 4													
เพิ่ม 3													
เพิ่ม 6													
ลบ 4													

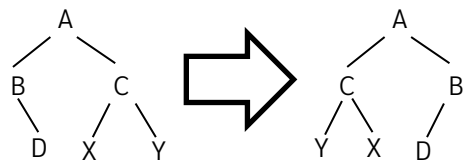
***** สำหรับข้อที่ 8 เป็นต้นไป *****
นิสิตสามารถเขียนฟังก์ชันอื่นเพิ่มเติมเพื่อช่วยในการใช้งานได้
แต่ต้องไม่ขัดกับข้อกำหนดที่ระบุในโจทย์

8. (10 คะแนน) ดร.โช ทัคเกอร์ นักสัตววิทยาผู้เชี่ยวชาญการผสมพันธุ์สัตว์ อยากจำลองการผสมพันธุ์ของสัตว์ โดยจำลองโครโมโซมของสัตว์ (ชนิดเดียวกัน) แต่ละตัวด้วยลิงค์ลิสต์ ดร. ทัคเกอร์ ให้คุณ ซึ่งเป็นผู้ช่วยวิจัย เขียนฟังก์ชั่นเพิ่มให้ คลาส CP::list ได้แก่ void crossover(CP::list<T> &other) ในการเรียกใช้ฟังก์ชันนี้ ให้ถือว่า other นั้นมีขนาดเท่ากับ list ที่เรียกใช้ โดยฟังก์ชันนี้จะทำให้ข้อมูลใน list ที่เรียก และ other นั้นสลับที่กันในตำแหน่งเลขคี่ทั้งหมด ตัวอย่างเช่นให้ l มีข้อมูลเป็น <A1, A2, A3, A4, A5> และ other มีข้อมูลเป็น <B1, B2, B3, B4, B5> การเรียก l.crossover(other) จะทำให้ l กลายเป็น <A1, B2, A3, B4, A5> และ other กลายเป็น <B1, A2, B3, A4, B5>

มีข้อจำกัดว่า **ฟังก์ชั่นนี้จะต้องเขียนด้วยการเปลี่ยนพอยต์เตอร์เท่านั้น ห้ามก๊อปปี้สมาชิกจากลิสต์หนึ่งลงไปอีกลิสต์หนึ่ง รวมถึงห้ามใช้คำสั่ง insert, erase ของ CP::list ด้วย**

```
template <typename T>
class list {
  protected:
    class node { // มีฟังก์ชันและข้อมูลอื่น ๆ ตามปรกติ แต่มิได้แสดงไว้เพื่อประหยัดพื้นที่
      public:
        T data;
        node *prev, *next;
    };
    node *mHeader; // pointer to a header node
    size_t mSize;
  public:  // มีฟังก์ชันและข้อมูลอื่น ๆ ตามปรกติ แต่มิได้แสดงไว้เพื่อประหยัดพื้นที่
    void crossover(list<T> &other) {









    }
};
```

9. (10 คะแนน) กำหนดให้มี Binary Tree โดยที่คลาส node ของ Binary Tree นี้เป็นดังข้อที่แล้ว จงเขียนฟังก์ชัน mirror(node* n) ซึ่งจะทำการสลับลูกซ้ายขวาทั้งหมดในปมต่างๆ ใน subtree ที่มี n เป็นราก (กล่าวคือ สลับลูกซ้ายขวาของทุก ๆ ปมที่เป็น descendant ของ n) รูปด้านขวามือนี้แสดงต้นไม้ตั้งต้น และผลลัพธ์จากการเรียก mirror ที่ปม A



Binary Tree ตั้งต้น หลังเรียก mirror(A)

```
void mirror(node* n) {









}
```

10. (10 คะแนน) สำหรับคลาส CP::map_bst นั้น การหาข้อมูลลำดับที่ K (ข้อมูลที่มีค่า Key มากสุดเป็นลำดับที่ K เมื่อให้ข้อมูลตัวแรกสุดคือลำดับที่ 0) สามารถทำได้โดยสร้าง iterator เริ่มที่ begin แล้วทำ operator++ เป็นจำนวน K-1 ครั้ง ซึ่งใช้เวลารวมเป็น O(K) เราต้องการเพิ่มบริการให้กับ CP::map_bst คือ iterator get_kth(int K) ซึ่งคืนค่า iterator ไปยังข้อมูลลำดับที่ K ดังกล่าว (ให้ถือว่า K จะมีค่าตั้งแต่ 0 ถึง size()-1 เสมอ) **อย่างไรก็ตาม ฟังก์ชันนี้จะต้องใช้เวลาเป็น O(h) เมื่อ h คือความสูงของต้นไม้**

เพื่อการดังกล่าว กำหนดให้คลาส node ของ CP::map_bst นั้นมี member ใหม่คือ TreeSize ซึ่งเก็บข้อมูลขนาดของ subtree ที่มี node ดังกล่าวเป็นราก ให้ถือว่าฟังก์ชันอื่น ๆ ของ CP::map_bst ทำการปรับปรุงค่า TreeSize ให้อย่างถูกต้องแล้วและสามารถใช้ค่าดังกล่าวได้เลย

```
template <typename KeyT, typename MappedT, typename CompareT = std::less<KeyT> >
class map_bst {
  protected:
    class node { // มีฟังก์ชันและข้อมูลอื่น ๆ ตามปรกติ แต่มิได้แสดงไว้เพื่อประหยัดพื้นที่
      public:
        ValueT data;
        node  *left,*right,*parent;
        int TreeSize; // ขนาดของต้นไม้ย่อยที่มีปมนี้เป็นราก เรียกใช้ได้เลย
    };
  public: // มีฟังก์ชันและข้อมูลอื่น ๆ ตามปรกติ แต่มิได้แสดงไว้เพื่อประหยัดพื้นที่
    iterator get_kth(int k) {









    }
};
```

11. (10 คะแนน) จงเพิ่มบริการ iterator upper_bound(const KeyT &k) สำหรับโครงสร้างข้อมูล CP::map_avl ซึ่งจะคืนค่า iterator ของข้อมูลตัวแรกที่มีค่ามากกว่า k (หรือคืน end() หาก k มีค่ามากกว่าหรือเท่ากับข้อมูลทุกตัวใน map_avl) **ฟังก์ชันนี้ควรจะใช้เวลาในการทำงานเป็น O(log n)** แต่ถ้าหากทำไม่ได้ ขอให้พยายามทำให้ได้ในเวลา O(n)

```
template <typename KeyT, typename MappedT, typename CompareT = std::less<KeyT> >
class map_avl {
  protected:
```

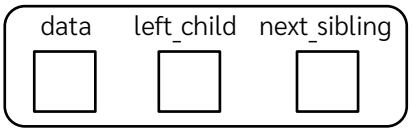
```
    class node {
      friend class map_avl;
      protected:
        ValueT data;  node  *left, *right, *parent;        int    height;
    };
    class tree_iterator {
      protected:
        node* ptr;
    };
    node     *mRoot;     CompareT  mLess;      size_t    mSize;
  public:
    iterator upper_bound(const KeyT &k) {









    }
 };
```

12. (10 คะแนน) จงเพิ่มบริการ void get_at_least(vector<T> &v, const &T k) ให้กับคลาส CP::priority_queue โดยที่ฟังก์ชันดังกล่าวจะแก้ไข v โดยการเพิ่มข้อมูลทั้งหมดใน priority_queue ที่มีค่ามากกว่าหรือเท่ากับ k เข้าไปใน v (ไม่จำเป็นต้องเรียงตามลำดับใด ๆ) ฟังก์ชันนี้ควรจะต้องใช้เวลาในการทำงานให้เร็วที่สุดเท่าที่ทำได้

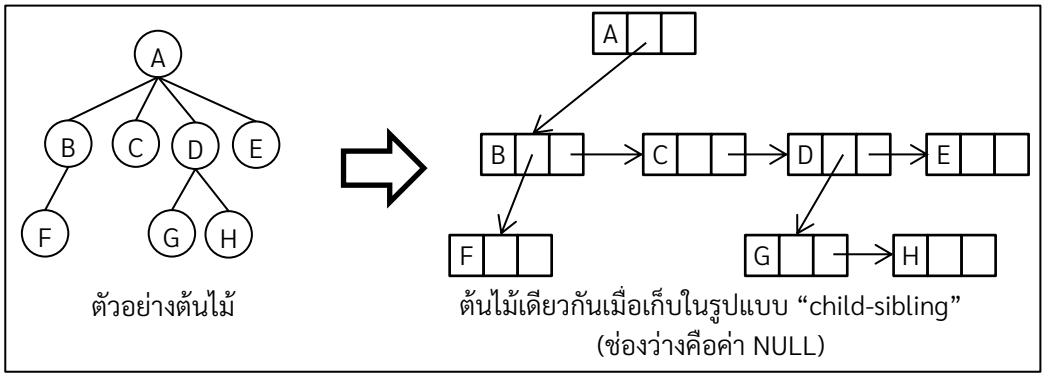
```
template <typename T,typename Comp = std::less<T> >
class priority_queue { // มีฟังก์ชันและข้อมูลอื่น ๆ ตามปรกติ แต่มิได้แสดงไว้เพื่อประหยัดพื้นที่
  protected:
    T *mData;
    size_t mCap;
    size_t mSize;
    Comp mLess;
  public:
    void get_at_least(vector<T> &v, const &T k) {









    }
};
```

13. (5 คะแนน) โครงสร้างข้อมูลประเภทต้นไม้ในวิชานี้เป็นต้นไม้แบบ Binary กล่าวคือ แต่ละปมมีลูกได้ไม่เกิน 2 ลูก อย่างไรก็ตาม ต้นไม้ทั่วสามารถมีลูกมากกว่านั้นได้ วิธีการหนึ่งในการสร้างปมที่มีลูกหลาย ๆ ลูกคือใช้ vector<node*> เพื่อเก็บลูกต่าง ๆ จากซ้ายไปขวา การใช้ vector นั้นมีข้อเสียคือเราจะเปลืองพื้นที่ในการเก็บข้อมูล เราสามารถปรับเปลี่ยนคลาส node สำหรับเก็บปมที่มีลูกหลาย ๆ ลูกโดยการให้ node นั้นเก็บลูกคนแรก (ลูกซ้ายสุด) และเก็บพี่น้องคนถัดมา (เรียกว่า node แบบ child-sibling) โดยมีโค้ดสำหรับคลาส node พร้อมรูปแสดงตัวอย่างดังด้านขวานี้

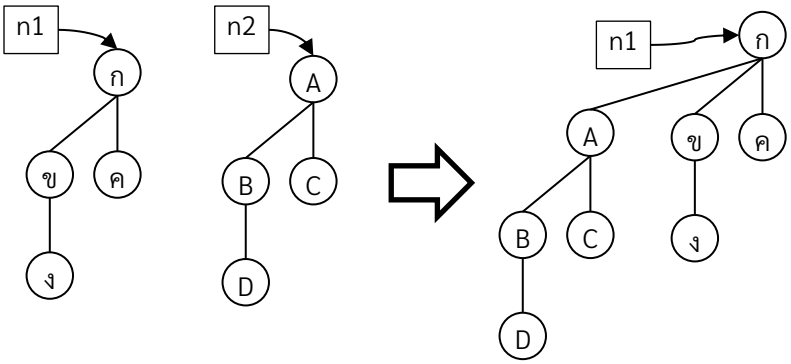
```
class cs_node {
  T data;
  cs_node *left_child;
  cs_node *next_sibling;
};
```



จงเขียนฟังก์ชัน void merge(cs_node *n1, cs_node *n2) โดยที่ n1 และ n2 เป็นปมรากของต้นไม้สองต้น ฟังก์ชันนี้จะทำการรวมต้นไม้ทั้งสองเข้าด้วยกัน โดยมีหลักการคือ นำเอา n2 มาเพิ่มเป็นลูกคนแรกของ n1 (โดยที่ลูกคนอื่น ๆ ของ n1 ยังคงมีอยู่ตามปรกติ แต่กลายเป็นลูกลำดับถัด ๆ ไปแทน) ตัวอย่างในรูปด้านล่างนี้แสดงถึงต้นไม้ n1 และ n2 ที่มีต้นละ 4 ปม และผลลัพธ์ของการเรียก merge(n1, n2)




```
Void merge(cs_node *n1, cs_node *n2) {

}
```



14. (5 คะแนน) Binomial Tree เป็นต้นไม้ประเภทหนึ่งซึ่งใช้ในการสร้างโครงสร้างข้อมูลชื่อ Binomial Heap โดยต้นไม้ Binomial Tree นั้นมีลำดับกำกับอยู่ กำหนดให้ B(i) หมายถึง Binomial Tree ลำดับที่ i เราสามารถนิยาม B(i) สำหรับค่า i ใด ๆ ได้ดังนี้ 1) กำหนดให้ B(0) คือต้นไม้ที่ประกอบด้วยปม 1ปมเท่านั้น และ 2) กำหนดให้ B(i) คือต้นไม้ที่เกิดจากผลของการ merge(n1, n2) เมื่อ n1 และ n2 ต่างก็เป็นรากของต้นไม้ B(i-1) (กล่าวคือ B(i) เกิดจากการนำเอา B(i-1) จำนวน 2 ต้นมา merge กันนั่นเอง) โดยให้ merge นั้นทำงานเช่นเดียวกับที่ระบุไว้ในข้อ 13 จงเขียนฟังก์ชัน cs_node* generate(int n) ซึ่งจะทำการสร้าง Binomial Tree ลำดับที่ n โดยจะคืนค่าปมรากของ B(n) โดยให้ถือว่ามีฟังก์ชัน merge และคลาส cs_node ตามที่กำหนดไว้ในข้อ 13 ที่ทำงานถูกต้องให้เรียกใช้ได้อยู่แล้ว

```
cs_node* generate(int n) {

}
```

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<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 ที่ใช้ได้เฉพาะlist	void push_front(const ValueT& val);
	void pop_font;
	void remove(const ValueT& val);

set<T, CompareT = less<T>>, unordered_set<T, HashT = hash<T>, EqualT = equal_to<T>>

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);

map<KeyT, MappedT, Compare = less<KeyT>>,
unordered_map<KeyT, MappedT, HashT = hash<KeyT>, EqualT = equal_to<KeyT>>

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

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