FACULTY OF ENGINEERING

CHULALONGKORN UNIVERSITY

2110211 ENG MECHANICS I

Year 2rd , First Semester , Midterm Examination 5 Oct 2017 8:30-11:30

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

<u>หมายเหตุ</u>

1. ข้อสอบมีทั้งหมด.....ข้อ ในกระดาษคำถาม.....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 ได้รับ สัญลักษณ์ F ในรายวิชาที่กระทำผิด และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้**

ห้ามนิสิตพกโทรศัพท์และอุปกรณ์สื่อสารไว้กับตัวระหว่างสอบ หากตรวจพบจะถือว่านิสิตกระทำผิดเกี่ยวกับการสอบ อาจต้องพ้นสภาพการเป็นนิสิต หรือ ให้ได้รับ F และอาจพิจารณาให้ถอนรายวิชาอื่นทั้งหมดที่ลงทะเบียนไว้ในภาคการศึกษานี้

*** ร่วมรณรงค์การกระทำผิด หรือทุจริตการสอบเป็นศูนย์ที่คณะวิศวกรรมศาสตร์ ***

ข้าพเจ้ายอมรับในข้อกำหนดที่กล่าวมานี้ ข้าพเจ้าเป็นผู้ทำข้อสอบนี้ด้วยตนเองโดยมิได้รับการช่วยเหลือ หรือให้ความช่วยเหลือ ในการทำข้อสอบนี้

ลงชื่อนิสิต.....

วันที่.....

1. (5 คะแนน) ให้นิสิตเขียน ฟังก์ชั่นที่รับ vector v กับ ค่า x เข้ามา คืนค่าเป็นเวคเตอร์ตัวใหม่ ที่เกิดจากการเอา x ออกจาก v ทั้งหมด (นั่นคือ ในคำตอบ จะต้องไม่มี x เลย)

```
vector<int> removeAll(vector<int>& v1, int x) {




}
```

2. (10 คะแนน) ชมรมนักประดิษฐ์คณะวิศวกรรมนั้นมีกิจกรรมที่ต้องใช้อุปกรณ์มากมาย ทำให้ห้องชมรมรกอยู่ตลอดเวลา ดังนั้นสมาชิกในชมรมจึงตกลงกันว่าจะจัดเวรทำความสะอาด โดยสมาชิกทุกคนในชมรมจะผลัดเวรกันทำความสะอาดวันละ 1 คน หมุนเวียนกันไปเรื่อย ๆ ตัวอย่างเช่น สมมติให้สมาชิกชมรมคือ A, B, C ดังนั้น หน้าที่ทำความสะอาดในวันแรกคือ A, วันถัดมาคือ B วันถัดมาคือ C และในวันถัดไปก็วนกลับมาเป็น A อีก เป็นเช่นนี้ไปเรื่อย ๆ
อย่างไรก็ตาม เนื่องจากชมรมนักประดิษฐ์นั้นเป็นที่นิยมเป็นอย่างมาก ทำให้มีสมาชิกในชมรมนั้นเพิ่มขึ้นอยู่ตลอดเวลา เมื่อมีสมาชิกใหม่เข้ามาในชมรม ณ วันใด สมาชิกดังกล่าวจะต้องรับหน้าที่ทำความสะอาดทันทีที่ทำได้ ถ้าหากวันเดียวกันนั้นมีสมาชิกใหม่เข้ามาหลายคน สมาชิกเหล่าใหม่เหล่านั้นก็ต้องรับหน้าที่ทำความสะอาดเรียงต่อกันไปตามลำดับ ตัวอย่างเช่น สมมติให้ปัจจุบัน สมาชิกในชมรมคือ A, B, C เช่นเดียวกับตัวอย่างด้านบน และวันนี้ควรจะเป็นหน้าที่ของ B แต่วันนี้มีสมาชิกใหม่เข้ามาสองคนคือ X, Y จะทำให้หน้าที่ในการทำความสะอาดในวันนี้เป็น X, และวันถัดไปคือ Y และวันถัด ๆ ไปคือ B, C, A, X, Y, B, C, A, X, Y ไปเรื่อย ๆ

จงเขียนคลาส CleaningDuty ซึ่งช่วยคำนวณเวรทำความสะอาดในแต่ละวัน โดยคลาสดังกล่าวต้องมีฟังก์ชันดังนี้

- string getTodayCleaner() ซึ่งจะคืนชื่อของคนที่มีหน้าที่ทำความสะอาดในวันนี้ ฟังก์ชันนี้จะถูกเรียกใช้ตอนปิดชมรมในตอนเย็นของแต่ละวัน วันละ 1 ครั้งเท่านั้น
- void addNewMember(string name) ซึ่งทำหน้าที่ลงทะเบียนสมาชิกใหม่เข้าสู่ชมรม ฟังก์ชันนี้จะอาจจะถูกเรียกได้หลายครั้งในแต่ละวัน แต่จะถูกเรียกใช้ก่อน getTodayCleaner ในแต่ละวันแน่นอน

ในคลาสนี้จะมีการกำหนด data member ให้เป็น queue<string> จำนวน 2 ตัว ห้ามทำการเพิ่ม data member อื่นใดอีก

```
class CleaningDuty {
  protected:
    queue<string> q1,q2;
   // ห้ามทำการเพิ่ม data member อื่นใดอีก
  public:
    // เขียนฟังก์ชันเพิ่มเติม ในส่วนนี้ สามารถเพิ่มฟังก์ชันอื่น ๆ ได้ตามต้องการ









};
```

3. จงพิจารณาฟังก์ชั่นต่อไปนี้ ซึ่งรับข้อมูลเป็น vector<bool> ซึ่งกำหนดลำดับการทำงานการนำข้อมูลเข้า/ออกจาก stack โดยการนำข้อมูลเข้านั้นจะใส่ข้อมูลเป็นตัวเลขลงไปใน stack เรียงไปตั้งแต่ 1, 2, 3,... ไปเรื่อย ๆ ส่วนการนำข้อมูลออกนั้นจะนำข้อมูลออจาก stack ไปใส่ใน vector

```
vector<int> process( vector<bool> input ) {
  int number = 1;
  vector<int> result;
  stack<int> s;
  for( size_t i = 0;i < input.size(); i++) {
    if (input[i]) {
      s.push(number);
      number++;
    } else {
      result.push_back(s.top());
      s.pop();
    }
  }
  while (s.empty() == false) {
      result.push_back(s.top());
      s.pop();
  }
  return result;
}
```

3.1 (3 คะแนน) จงเขียนผลลัพธ์ที่คืนค่ามาจากฟังก์ชัน process เมื่อกำหนดให้ input เป็นดังต่อไปนี้

a) ให้ input = {T,T,T,F,F,F} ผลลัพธ์คือ _____

b) ให้ input = {T, F, T, F, T, T} ผลลัพธ์คือ_____

c) ให้ input = {T,T,T,F,T,T,F,T,T,T,F} ผลลัพธ์คือ_____

3.2 (7 คะแนน) จงเขียนฟังก์ชัน vector<bool> inverse(vector<int> desired_result) ที่คำนวณค่า input ซึ่งเมื่อเรานำเอา input ดังกล่าวนั้นไปใช้เป็นข้อมูลนำเข้าของฟังก์ชัน process แล้ว จะได้ผลลัพธ์เป็น desired_result ตัวอย่างเช่น การเรียก inverse({4, 3, 2, 1}) นั้นจะต้องคืนค่าเป็น {T, T, T, T, F, F, F, F} เป็นต้น

```
vector<bool> inverse( vector<int> desired_result ) {
    vector<bool> input;
    // write your code here









    return input;
}
```

4. (10 คะแนน) ในข้อนี้มี vector<string> name เก็บชื่อของคน N คนโดยรับประกันว่าชื่อไม่ซ้ำกัน และมี vector<pair<string,string>> father โดยที่ father[j].first เป็นพ่อของ father[j].second จงเขียนฟังก์ชัน vector<string> getCousin(vector<string> name, vector<pair<string,string>> father, string query) ที่คำนวณหา "ลูกพี่ลูกน้อง" ของ query ซึ่งเป็นชื่อที่กำหนดให้มา โดยนิยามของ "ลูกพี่ลูกน้อง" คือคนที่มีปู่เหมือนกับเราแต่มีพ่อไม่เหมือนเรา

ตัวอย่างเช่น name = {"John", "Jack", "Leo", "Eric", "Robert", "Adam", "Mike", "Manny"} father= {{"Leo","John"}, {"John","Eric"}, {"John","Robert"}, {"Leo","Adam"}, {"Adam","Mike"}, {"Adam", "Manny"} }

การเรียก getCousin(name, father, "Eric") จะได้ผลเป็น { "Mike", "Manny" }

การเรียก getCousin(name, father, "John") จะได้ผลเป็น { }

```
vector<string> getCousin(const vector<string> &name,
                         const vector<pair<string,string>> &father,
                         string query) {
  vector<string> result;
  // write your code here









  return result;
}
```

5. (5 คะแนน) ในข้อนี้ให้นิสิตเขียนฟังก์ชั่น removeMin(priority_queue<int>& pq) ที่เอาค่าที่น้อยที่สุด ออกจาก priority queue ที่เรียงข้อมูลจากมากไปน้อยตามปกติ โดยหลังจบฟังก์ชั่นข้อมูลตัวอื่นๆยังต้องคงอยู่เหมือนเดิม ตัวอย่างเช่น หากตอนเริ่ม priority queue มี 8 7 5 3 1 อยู่ หลังเรียกฟังก์ชั่นนี้จะมีข้อมูลเป็น 8 7 5 3 (เนื่องจาก 1 เป็น ค่าที่น้อยที่สุด จึงถูกเอาออกไป) โดยหาก priority queue ไม่มีข้อมูล ให้ไม่ต้องทำอะไรโดยสมมุติว่าข้อมูลใน priority queue นั้นต่างกันทุกตัว

```
void removeMin(priority_queue<int>& pq) {









}
```

6. ข้อนี้ จะให้ออกแบบสร้าง priority_queue ขึ้นมาเอง

   a. (5 คะแนน) จงเขียนบรรยาย ว่า นิสิต จะทำการสร้าง priority queue ขึ้นมาเองได้อย่างไร ใช้ data structure และ library อะไรบ้าง โดยต้องบรรยายการทำงานของ pop, top และ push **ทั้งนี้ ไม่ต้องสนใจประสิทธิภาพการทำงาน**

   b. (10 คะแนน) ในข้อนี้ให้เขียนคลาส MyPriorityQ จากการออกแบบในข้อด้านบน โดยต้องเติม include , ตัวแปร และฟังก์ชั่น top, pop, push ให้ถูกต้อง ถ้าต้องการแจ้ง error ให้ใช้คำสั่ง throw error(ข้อความ) ได้เลย

```
//เติม include ที่จำเป็นให้ถูกต้อง



template <typename T>
class MyPriorityQ{
public:
  // เติมตัวแปรให้ครบถ้วน



  T top(){ //เติมโค้ด




  }

  void pop(){//เติมโค้ด




  }

  void push(T x){ //เติมโค้ด




  }
};
```

7. (15 คะแนน) ในข้อนี้จะให้เขียน ฟังก์ชั่นเพื่อการ sort การ์ด จากเกม magic the gathering (อย่างง่าย) การ์ดในเกมนี้นั้น มีลักษณะดังรูปด้านล่าง ข้อมูลสำหรับการ sort นั้นจะอยู่ที่แถบด้านบนของการ์ดเท่านั้น

เรามีคลาส Card ให้ แต่เขียนมาไม่หมด สิ่งที่นิสิตต้องทำคือ เขียนฟังก์ชัน < เพื่อเอาไว้เทียบการ์ดสองใบ โดยให้เรียงการ์ดด้วย total casting cost จากน้อยไปมาก (สามารถเพิ่มฟังก์ชันอื่นได้ถ้าจำเป็น) แต่ถ้า total cating cost เท่ากัน ให้เรียงตามชื่อการ์ด (เรียงตามพจนานุกรม)

ทั้งนี้ total casting cost = ค่าร่ายทั่วไป + จำนวนหน่วยทั้งหมดของค่าร่ายเฉพาะธาตุ

จากการ์ดตัวอย่างด้านบน total casting cost จะเท่ากับ 4+1+2 = 7

ส่วนจากการ์ดตัวอย่างอีกใบด้านล่างนี้ total casting cost จะเป็น 2+2+2+2+2 = 10 (ไม่มีค่าร่ายทั่วไป มีแต่ค่าร่ายจากธาตุ)

```
//ไม่ต้องสน include นะ ให้ถือว่า include ทุกอย่างที่จำเป็นมาแล้ว
class MTGCard{
public:
  // ห้ามแก้หรือเพิ่ม-ลด ตัวแปร
  string name;
  int genericCost;
  vector<pair<string,int> > elementCost; // แต่ละ pair จะบอกว่าธาตุอะไร มีกี่หน่วย


  //เขียนฟังก์ชั่นที่จำเป็นกับการ sort (ถ้ามี)









  bool operator<(const Card &other){ //เติมโค้ด

  }
};
```

8. (10 คะแนน) ในข้อนี้จะให้เขียนบริการของคลาส CP::vector<T> ที่ชื่อว่า duplicateElements(const vector<vector<T>::iterator>& its) เพื่อการทำการทำซ้ำข้อมูลตัวที่ระบุโดย iterator its[0], its[1], … itrs[its.size()-1] เอาไว้ติดกันโดยเรารับประกันว่า its[0] จะอ้างอิงถึงของมูลที่มาก่อนหน้า its[1], its[1] จะอ้างอิงถึงของมูลที่มาก่อนหน้า its[2], …, its[its.size()-2] จะอ้างอิงถึงของมูลที่มาก่อนหน้า its[its.size()-1] เสมอ

ตัวอย่าง

หาก vector<int> v = {9,2,6,8,1,7} และ vector<vector<int>::iterator> its = {v.begin(), v.begin()+3, v.begin()+5}

หลังจากเรียก v.duplicateElements(its) แล้ว v จะเก็บค่า {9,9,2,6,8,8,1,7,7}

โดยนิสิตสามารถใช้บริการใดๆที่มีใน CP::vector ได้โดยไม่มีข้อจำกัด

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

    void rangeCheck(int n) {…}
    void expand(size_t capacity) {…}
    void ensureCapacity(size_t capacity) {…}
  public:
    vector(const vector<T>& a) {…}
    vector() {…}
    vector(size_t cap) {…}
    vector<T>& operator=(vector<T> other) {…}
    ~vector() {…}
    bool empty() const {…}
    size_t size() const {…}
    size_t capacity() const {…}
    void resize(size_t n) {…}
    iterator begin() {…}
    iterator end() {…}
    T& at(int index) {…}
    T& at(int index) const {…}
    T& operator[](int index) {…}
    T& operator[](int index) const {…}
    void push_back(const T& element) {…}
    void pop_back() {…}
    iterator insert(iterator it,const T& element) {…}
    void erase(iterator it) {…}
    void clear() {…}
    void duplicateElements(const vector<vector<T>::iterator>& its) {
        // เติมโค้ด









   }
}
```

9. (10 คะแนน) จงเพิ่มบริการให้กับคลาส CP::stack โดยให้เพิ่มฟังก์ชัน void remove_all (const T& data) ซึ่งจะทำการลบข้อมูลใด ๆ ที่อยู่ใน stack ที่มีค่าเท่ากับ data ออกไปทั้งหมด โดยยังคงรักษาลำดับของข้อมูลใน stack ของตัวอื่น ๆ ให้เป็นตามเดิม (ตัวอย่างเช่น ใน stack มีข้อมูลเป็น {1, 2, 1, 3, 1, 4, 5, 7} โดยกำหนดให้ตัวซ้ายสุดในรายการดังกล่าวเป็น top of stack เมื่อผู้ใช้สั่ง remove_all(1) แล้ว stack จะเหลือข้อมูลเป็น {2, 3, 4, 5, 7} เป็นต้น) ฟังก์ชันนี้ควรจะทำงานให้เร็วที่สุดเท่าที่จะเป็นไปได้ กล่าวคือ เวลาที่ใช้ในการทำงานควรจะแปรผันตรงกับจำนวนข้อมูลที่อยู่ใน stack

```
template <typename T>
class stack {
  protected:
    T *mData; size_t mCap, mSize  // ห้ามประกาศ data member เพิ่มเติม
  public:
    // คลาสนี้ทำงานได้ตามปรกติทุกอย่าง โดยฟังก์ชันอื่น ๆ มิได้ระบุไว้เพื่อประหยัดหน้ากระดาษ ให้นิสิตเขียนบริการเพิ่มเติม ตามข้อกำหนดด้านบน
    // ให้เขียนเติมฟังก์ชันด้านล่างนี้ ให้ทำงานให้ถูกต้อง
    Void remove_all(const T& data) {
    // เติมโค๊ด









    }
};
```

10. (10 คะแนน) ในข้อนี้จะให้เขียนบริการของคลาส CP::queue<T> ที่ชื่อว่า vector<queue<T>> split_queue(int k) เพื่อทำการแบ่ง แถวคอยปัจจุบันเป็น k แถวคอยโดยข้อมูลหน้าสุดจะไปอยู่แถวคอยที่ 0 ข้อมูลตัวถัดไปไปอยู่แถวคอยที่ 1 ... ข้อมูลตัวที่ k-1 อยู่แถวคอยที่ k-1 ข้อมูลตัวที่ k อยู่แถวคอยที่ 0 ..... ไปเรื่อยๆ และจะทำการนำข้อมูลทั้งหมดออกจากแถวคอยปัจจุบันด้วย ตัวอย่างเช่น หาก queue<int> q เก็บข้อมูล {3,9,8,6,7,2,1,3,8,4} ตามลำดับจากหน้าไปหลังอยู่ หากเราเรียก vector<queue<int>> qs = q.split_queue(3)

ผลที่ได้คือ qs[0] จะเป็นแถวคอยที่เก็บ {3,6,1,4}

qs[1] จะเป็นแถวคอยที่เก็บ {9,7,3}

qs[2] จะเป็นแถวคอยที่เก็บ {8,2,8}

และ q จะไม่มีข้อมูลอยู่

โดยนิสิตสามารถใช้บริการใดๆที่มีใน CP::queue ได้โดยไม่มีข้อจำกัด

```
template <typename T>
class queue {
  protected:
    T *mData;
    size_t mCap;
    size_t mSize;
    size_t mFront;
    void expand(size_t capacity) {..}
    void ensureCapacity(size_t capacity) {...}
  public:
    queue(const queue<T>& a) {..}
    queue() {...}
    queue<T>& operator=(queue<T> other) {...}
    ~queue() {...}
    bool empty() const {..}
    size_t size() const {…}
    const T& front() const {…}
    const T& back() const {…}
    void push(const T& element) {…}
    void pop() {…}
     vector<queue<T>> split_queue(int k) {
```

```
        // เติมโค้ด
    }
}
```

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	`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_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);
```