Priority Stack

(1s, 512mb)

ข้อนี้เป็นการปรับปรุง CP::stack<T> ให้มีลักษณะการทำงานเหมือน **Priority Queue** โดยที่ข้อมูลที่อยู่ "ด้านบนสุด" ของ stack (ที่เข้าถึงได้ผ่าน top()) จะต้องเป็นข้อมูลที่มีค่า **มากที่สุด** เสมอ

กล่าวคือ CP::stack ในข้อนี้จะไม่ได้ทำงานแบบ LIFO (Last-In, First-Out) อีกต่อไป แต่จะทำงานแบบ "Max-Out" แทน

จงแก้ไขและเพิ่มบริการต่อไปนี้:

- **void push(const T& element)**: เพิ่ม element เข้าไปใน stack โดยต้องรักษาคุณสมบัติที่ว่าข้อมูลภายใน stack จะต้องเรียงลำดับจาก **มากไปน้อย** (โดยตัวมากที่สุดอยู่บนสุด)
- **void pop()**: ลบข้อมูลตัวที่อยู่บนสุด (ตัวที่มีค่ามากที่สุด) ออกจาก stack
- **const T& top()**: คืนค่าข้อมูลที่อยู่บนสุด (ตัวที่มีค่ามากที่สุด)
- **size_t size()** และ **bool empty()**: คืนค่าขนาดและสถานะของ stack ตามปกติ

ตัวอย่างการทำงาน

กำหนดให้ stack มีข้อมูล (จากบนลงล่าง) เป็น [50, 30, 10]

1. **เรียก top()**: จะได้ค่า 50
2. **เรียก push(20)**:
    - สถานะสุดท้าย คือ [50, 30, 20, 10]
3. **เรียก push(70)**:
    - สถานะสุดท้าย คือ [70, 50, 30, 20, 10]
4. **เรียก pop()**:
    - ลบ 70 ออก
    - สถานะสุดท้าย คือ [50, 30, 20, 10]

โจทย์ข้อนี้จะมีไฟล์โปรเจ็คของ Code::Blocks ให้ ซึ่งในไฟล์โปรเจ็คดังกล่าวจะมีไฟล์ stack.h, main.cpp และ student.h อยู่ **ให้นิสิตเขียน code เพิ่มเติมลงในไฟล์ student.h เท่านั้น** และการส่งไฟล์เข้าสู่ระบบ grader ให้ส่งเฉพาะไฟล์ student.h เท่านั้น

ในไฟล์ student.h ดังกล่าวจะต้องไม่ทำการอ่านเขียนข้อมูลใด ๆ ไปยังหน้าจอหรือคีย์บอร์ดหรือไฟล์ใด ๆ

คำอธิบายฟังก์ชัน main

main() จะสร้าง CP::stack<int> ขึ้นมา และอ่านคำสั่งทีละบรรทัด โดยแต่ละบรรทัดจะมีรูปแบบดังนี้:

- **push X**: เรียก stk.push(X) โดย X เป็นจำนวนเต็ม

- **pop**: เรียก stk.pop()
- **top**: พิมพ์ค่าของ stk.top()
- **size**: พิมพ์ค่าของ stk.size()

รับประกันว่าจะไม่มีการเรียก pop หรือ top ในขณะที่ stack ว่าง