Double Stack

(1 sec, 512mb)

ข้อนี้เป็นการปรับปรุง CP::stack<T> โดยเราต้องการให้ Stack ของเรานั้นสามารถ push, pop ตรง “ด้านล่างสุด” ของ stack ได้ รวมถึงขอดูข้อมูลตรงด้านล่างสุดด้วย โดยให้เพิ่มและแก้ไขบริการต่อไปนี้

- const T& top() ซึ่งจะคืนข้อมูลที่อยู่ด้านบนสุดของ stack
- void pop() เป็นการลบข้อมูลออกจากด้านบนสุดของ stack
- const T& bottom() ซึ่งจะคืนข้อมูลที่อยู่ด้านล่างสุดของ stack
- void push_bottom(const T& element) เป็นการ push ข้อมูลไปไว้ด้านล่างสุดของ stack
- void pop_bottom() เป็นการลบข้อมูลออกจากด้านล่างสุดของ stack

ตัวอย่างเช่น หากเรา push(1), push(2) และ push(3) ใส่ stack จะทำให้ stack เรามีข้อมูลเป็น [1,2,3] (ให้ด้านขวาของ stack เป็น top of stack) และถ้าหลังจากนั้น เรา push_bottom ด้วย 10, 20 และ 30 ตามลำดับ stack เราจะมีข้อมูลเป็น [30, 20, 10, 1, 2, 3] เป็นต้น

อย่างไรก็ตาม การ push_bottom หรือ pop_bottom นั้น หากเราทำการเพิ่มบริการดังกล่าวให้กับ stack โดยยังคงใช้ mData เหมือน stack ปรกติ จะใช้เวลาพอสมควร เพื่อเป็นทางเลือกในการแก้ปัญหาดังกล่าว คลาส CP::stack ในข้อนี้จะมี data member เป็น std::stack<T> จำนวน 2 ตัวคือ stack_a และ stack_b **ให้ใช้เท่านั้น** และมีคำสั่งเพิ่มเติมดังนี้

- void push(const T& element) จะทำการ push ข้อมูลไปไว้ที่บน stack_a
- size_t size() จะคืนค่า stack_a.size() + stack_b.size()
- bool empty() จะคืนค่า stack_a.empty() && stack_b.empty()

ให้นิสิตลองพิจารณาการปรับปรุงและเขียนฟังก์ชันต่าง ๆ ของ CP::stack นี้ให้สามารถทำงานได้ดังที่ควรเป็น (นิสิตสามารถเลือกที่จะใช้ stack_a, stack_b อย่างไรก็ได้ จะใช้แค่ตัวเดียวก็ได้)

คำอธิบายฟังก์ชัน main

main() จะเป็นการทดลองใช้งาน stack ด้วยคำสั่งต่าง ๆ โดย main จะ สร้าง CP::stack<int> หรือ CP::stack<std::string> ชื่อ stk และ stk2 มาแล้วอ่านคำสั่งทีละบรรทัด โดยบรรทัดแรกเป็นการระบุประเภทข้อมูล (1 คือ int, 2 คือ std::string) ซึ่งแต่ละบรรทัดหลังจากนั้นจะเริ่มต้นด้วย string ที่ระบุคำสั่ง และอาจจะตามด้วยค่าต่าง ๆ ที่จำเป็นสำหรับคำสั่งนั้น โดยมีรูปแบบของแต่ละคำสั่งดังต่อไปนี้ (ให้ X หมายถึงค่าประเภท int ใด ๆ)

- s หมายถึงให้ main พิมพ์ค่าของ stk.size() ออกทางหน้าจอ
- tp X หมายถึงให้ main เรียก stk.push(X)
- to หมายถึงให้ main เรียก stk.pop()
- tt หมายถึงให้ main พิมพ์ค่าของ stk.top()
- bp X หมายถึงให้ main เรียก stk.push_bottom(X)
- bo หมายถึงให้ main เรียก stk.pop_bottom()
- bt หมายถึงให้ main พิมพ์ค่าของ stk.bottom()
- q หมายถึงให้ main หยุดการทำงาน

ชุดข้อมูลทดสอบ

รับประกันว่าจำนวนคำสั่งที่กระทำต่อ stack จะไม่เกิน 1,000,000 คำสั่งแน่นอน และไม่มีคำสั่ง to, tt, bo, bt เมื่อ stack ขนาด 0

- 10% stack เก็บข้อมูล int และ ไม่มีการเรียกคำสั่ง bp, bo หรือ bt แน่นอน
- 15% stack เก็บข้อมูล int และ มีการเรียกคำสั่ง bp, bo หรือ bt คำสั่งละไม่เกิน 1 ครั้ง
- 25% stack เก็บข้อมูล int และ ขนาดของ stack ไม่เคยเกิน 100
- 10% stack เก็บข้อมูล string และ ขนาดของ stack ไม่เคยเกิน 100
- 20% ให้ A คือคำสั่ง bo, หรือ bt และ B คือคำสั่ง to หรือ tt เราจะรับประกันว่าไม่มีชุดคำสั่งในรูปแบบ A B A B A (หรือ B A B A B) อย่างแน่นอน (แต่อาจจะมี A B A หรือ B A B หรือ A B A B ก็ได้)
- 20% ไม่มีข้อจำกัดอื่นใด

ข้อบังคับ

- 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
- หากใช้ VS Code ให้ทำการ compile ที่ไฟล์ main.cpp

**** main ที่ใช้จริงใน grader นั้นจะแตกต่างจาก main ที่ได้รับในไฟล์โปรเจ็กต์เริ่มต้นแต่จะทำการทดสอบในลักษณะเดียวกัน ****

คำแนะนำ

คะแนน 20% สุดท้ายของข้อนี้ จำเป็นต้องคิดวิธีการใช้งาน stack_a และ stack_b อย่างเหมาะสม หากทำไม่ได้ ให้ข้ามไปทำข้ออื่น ๆ ก่อน

(ตัวอย่างข้อมูลทดสอบอยู่หน้าถัดไป)

ตัวอย่างการทำงานของ main

ข้อมูลนำเข้า	ข้อมูลส่งออก
1 tp 1 tp 10 tp 3 tp 4 bt bo bt tt s to bp 100 bt s q	1 10 4 3 100 3
2 tp A tp Bb tp CcC tp Dae bt bo bt tt s to bp E bt s q	A Bb Dae 3 E 3