Do Stack

(1 sec, 512mb)

ข้อนี้เป็นการปรับปรุง CP::stack<T> โดยเราต้องการให้ stack ของเรานั้นสามารถ undo และ redo การทำงานล่าสุดได้ รวมทั้งสามารถเรียก undo และ redo ต่อเนื่องกันได้ โดยให้เพิ่มและแก้ไขบริการต่อไปนี้

- void push(const T& value) เป็นการเพิ่มข้อมูลไปที่ด้านบนสุดของ stack
- void pop() เป็นการลบข้อมูลออกจากด้านบนสุดของ stack
- void undo() เป็นการเลิกทำ push/pop ครั้งล่าสุด โดยให้คืน state ของ stack กลับไปที่ state เดิม ทั้งนี้ถ้ามีการ expand เกิดขึ้น ไม่จำเป็นต้องย้าย address ของ mData กลับไปที่เดิม หากยังไม่มีการเปลี่ยนแปลงกับ stack บริการนี้จะต้องไม่ทำอะไร
- void redo() เป็นการทำซ้ำจากการ undo ครั้งล่าสุด นั่นคือหากมีการเรียกฟังก์ชันใดๆ ต่อจากการ redo() ต้องทำงานเหมือนกันก่อนการเรียก undo() ทุกประการ หากล่าสุดไม่มีการ undo บริการนี้จะต้องไม่ทำอะไร

ตัวอย่างเช่น หากเรา push(4), push(5) และ push(10) ใส่ stack จะทำให้เรามีข้อมูลเป็น [4, 5, 10] (ด้านบนสุดของ stack อยู่ทางขวา) และถ้าเราทำการ undo() จะเป็นการย้อนกลับไปที่ [4, 5] และหากเรียก undo อีกครั้งจะเป็นการย้อนกลับไปที่ [4] หลังจากนั้นถ้าเรียก redo(), redo() จะกลับไปที่ [4, 5] และ [4, 5, 10] ตามลำดับ

อีกหนึ่งตัวอย่างเช่น หากเรา push(6), push(13), push(1) ใส่ stack จะทำให้เรามีข้อมูลเป็น [6, 13, 1] (ด้านบนสุดของ stack อยู่ทางขวา) หากทำการ undo() จะเป็นการย้อนกลับไปที่ [6, 13] แต่หลังจากนั้น pop() ทำให้กลายเป็น [6] หากเรียก redo() จะไม่เกิดอะไรขึ้น (เพราะ action ล่าสุดไม่ใช่การ undo) หากเรียก undo() ก็จะย้อนกลับมาเป็น [6, 13]

ตัวอย่างสุดท้าย หากเรา push(1), push(2), push(3) ใส่ stack จะทำให้เรามีข้อมูลเป็น [1, 2, 3] (ด้านบนสุดของ stack อยู่ทางขวา) หากเรามีการทำงานตามลำดับต่อไปนี้ undo() -> redo() -> undo() -> redo() -> undo() -> redo() จะต้องได้ข้อมูลเดิม แต่อย่างไรก็ดีหากหลังจากลำดับก่อนหน้ามีการสั่งคำสั่งตามลำดับนี้ undo() -> undo() -> push(10) -> redo() จะต้องได้ข้อมูลเป็น [1, 10] โดยที่ redo() ตัวสุดท้ายต้องไม่มีการทำคำสั่งใดๆ

ในข้อนี้ CP::stack<T> จะมีโครงสร้างข้อมูลที่เป็น member เพิ่มมาทั้งหมด 2 ตัวนั่นคือ

- std::stack<std::pair<int, T>> aux_stack_1
- std::stack<std::pair<int, T>> aux_stack_2

ให้นิสิตลองพิจารณาการปรับปรุงและเขียนฟังก์ชันต่าง ๆ ของ CP::stack นี้ให้สามารถทำงานได้ดังที่ควรเป็น (โดยนิสิตสามารถเลือกที่จะใช้หรือไม่ใช้ aux member อย่างไรก็ได้)

(มีต่อหน้าถัดไป)

ข้อบังคับ

- โจทย์ข้อนี้จะมีไฟล์ตั้งต้นมาให้ ซึ่งประกอบด้วยไฟล์ stack.h, main.cpp และ student.h อยู่ ให้นิสิตเขียน code เพิ่มเติมลงในไฟล์ student.h เท่านั้น และการส่งไฟล์เข้าสู่ระบบ grader ให้ส่งเฉพาะไฟล์ student.h เท่านั้น
- ไฟล์ student.h จะต้องไม่ทำการอ่านเขียนข้อมูลใด ๆ ไปยังหน้าจอหรือคีย์บอร์ดหรือไฟล์ใด ๆ
- หากใช้ VS Code ให้ทำการ compile ที่ไฟล์ main.cpp

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

คำอธิบายฟังก์ชัน main

main() จะเป็นการทดลองใช้งาน stack โดย main จะอ่านข้อมูลนำเข้าดังนี้

- บรรทัดแรก เป็นจำนวนเต็ม T ระบุจำนวนรอบของการทดสอบ สำหรับแต่ละรอบการทดสอบ

จากนั้น main ทำการสร้าง CP::stack<int> stk ขึ้นมา และจะรับข้อมูลนำเข้าแบบต่อเนื่องจนกว่าจะเจอคำสั่ง "q" โดยมีรูปแบบคำสั่งที่เป็นไปได้ทั้งหมดดังต่อไปนี้

- si หมายถึงให้พิมพ์ค่าของ stk.size() ออกทางจอ
- to หมายถึงให้พิมพ์ค่าของ stk.top() ออกทางจอ
- pu X หมายถึงให้เรียก stk.push(X)
- po หมายถึงให้เรียก stk.pop()
- un หมายถึงให้เรียก stk.undo()
- re หมายถึงให้เรียก stk.redo()
- q หมายถึงให้ main หยุดทางทำงาน

เมื่อออกจากการทำงานในแต่ละรอบของการทดสอบ โปรแกรมจะแสดง จำนวนข้อมูล, capacity และข้อมูลทุกตัวใน stk ตามลำดับ

ชุดข้อมูลทดสอบ

รับประกันว่าจำนวนครั้งที่มีการเรียก operation ต่าง ๆ ใน stack จะรวมกันในทุกรอบการทดสอบแล้วไม่เกิน 500,000 ครั้ง นอกจากนี้จะไม่มีการเรียก top หรือ pop เมื่อ state ของ stack ที่ถูกต้องนั้นว่างอยู่

- 10% ไม่มีการเรียก redo หรือ undo เลย
- 15% stack เก็บข้อมูลเป็น int และไม่มีการเรียก redo
- 10% stack เก็บข้อมูลเป็น int
- 20% ไม่มีการเรียก redo
- 10% มีการเรียก operation ของ stack รวมกันในทุกรอบการทดสอบแล้วไม่เกิน 5,000 ครั้ง
- 10% จะไม่มีการเรียก undo และ redo โดยที่ทำงานไม่ได้
- 25% ไม่มีเงื่อนไขอื่นใด

ตัวอย่างการทำงานของ main

ข้อมูลนำเข้า	ข้อมูลส่งออก
1 pu 4 pu 5 pu 6 si to un un to re to un to re q	3 6 4 5 4 Stack Size: 2 Stack Capacity: 4 Data: 4 5 ----------
2 pu 3 pu 4 pu 5 po un si un po re q pu 10 pu 11 pu 12 pu 13 po po si un to pu 14 re q	3 Stack Size: 1 Stack Capacity: 4 Data: 3 ---------- 2 12 Stack Size: 4 Stack Capacity: 4 Data: 10 11 12 14 ----------