Stack with Transactions

(1 sec, 512mb)

ก่อนที่จะเริ่มทำโจทย์ข้อนี้ ให้ทำความเข้าใจแนวคิดของ Transaction ซึ่งเป็นแนวคิดที่สำคัญในคอมพิวเตอร์ โดยเฉพาะในระบบฐานข้อมูล หรือระบบกระจายศูนย์ (Distributed System)

Transaction คือกลุ่มของการดำเนินการที่ถูกมองว่าเป็นหน่วยการทำงานเดียวที่ต้องสำเร็จทั้งหมด (All) หรือไม่ก็ล้มเหลวทั้งหมด (Nothing)

ลองนึกภาพการแก้ไขเอกสาร:

- คุณเปิดไฟล์ขึ้นมา (สถานะเริ่มต้น)
- คุณพิมพ์ข้อความเพิ่ม ลบข้อความบางส่วน และจัดรูปแบบ (กลุ่มของการดำเนินการ)
- เมื่อคุณพอใจแล้ว คุณกด **Save**. การเปลี่ยนแปลงทั้งหมดจะถูกบันทึกอย่างถาวร การกระทำนี้เทียบเท่ากับการ **commit**
- แต่ถ้าคุณเปลี่ยนใจและกด **Don't Save**. การเปลี่ยนแปลงทั้งหมดที่คุณทำมาจะถูกยกเลิก และไฟล์จะกลับไปสู่สถานะเดิมตอนที่คุณเปิดขึ้นมา การกระทำนี้เทียบเท่ากับการ **rollback**

จงเพิ่มบริการให้กับ CP::stack<T> เพื่อให้รองรับการทำงานในลักษณะของ Transaction ซึ่งเป็นกลุ่มของคำสั่งที่สามารถยืนยัน (commit) เพื่อให้การเปลี่ยนแปลงคงอยู่ หรือย้อนกลับ (rollback) เพื่อยกเลิกการเปลี่ยนแปลงทั้งหมดในกลุ่มได้ โดยให้เพิ่มบริการต่อไปนี้:

1. **void begin()**: เริ่มต้น transaction ใหม่ การเรียก push หรือ pop ใด ๆ หลังจากนี้จะถือว่าเป็นส่วนหนึ่งของ transaction ล่าสุด
2. **void commit()**: ยืนยันการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นใน transaction ล่าสุดที่ยังไม่สิ้นสุด ทำให้การเปลี่ยนแปลงนั้นกลายเป็นส่วนหนึ่งของ transaction ชั้นนอกหรือกลายเป็นสถานะถาวรของ stack หากไม่มี transaction ชั้นนอกอีกแล้ว
3. **void rollback()**: ยกเลิกการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นใน transaction ล่าสุดที่ยังไม่สิ้นสุด และคืนสถานะของ stack ให้เป็นเหมือนกับตอนก่อนที่จะเรียก begin() ของ transaction นั้น

การทำงานของบริการ

- รับประกันว่าจะไม่มีการเรียก begin() ซ้อนกัน
- การทำงาน size() และ top() ควรแสดงผลสถานะปัจจุบันของ stack เท่านั้น
- หากมีการเรียก commit หรือ rollback ในขณะที่ไม่มี transaction ที่กำลังทำงานอยู่ (ยังไม่เคยเรียก begin หรือ commit/rollback ไปครบแล้ว) ฟังก์ชันดังกล่าวไม่ต้องทำงานใด ๆ

- หากไม่ได้มีการเริ่ม transaction ให้ถือว่า stack ทำงานแบบปกติ

ตัวอย่าง: กำหนดให้ stack เริ่มต้นเป็น [] (ด้านขวาคือ top of stack)

การดำเนินการ	stack ที่ภายนอกเห็น	stack สถานะภายใน	คำอธิบาย
เริ่มต้น	[ ]	[ ]	
push(10)	[10]	[10]	ทำงานปกติ
push(20)	[10, 20]	[10, 20]	
begin()	[10, 20]	[10, 20]	เริ่ม transaction 1
push(30)	[10, 20]	[10, 20, 30]	push เกิดขึ้นใน transaction
pop()	[10, 20]	[10, 20]	pop เกิดขึ้นใน transaction
push(40)	[10, 20]	[10, 20, 40]	push เกิดขึ้นใน transaction
rollback()	[10, 20]	[10, 20]	ยกเลิก transaction 1
push(50)	[10, 20, 50]	[10, 20, 50]	ทำงานปกติ
begin()	[10, 20, 50]	[10, 20, 50]	เริ่ม transaction 2
pop()	[10, 20, 50]	[10, 20]	pop เกิดขึ้นใน transaction
push(60)	[10, 20, 50]	[10, 20, 60]	push เกิดขึ้นใน transaction
commit()	[10, 20, 60]	[10, 20, 60]	ยืนยัน transaction 2
rollback()	[10, 20, 60]	[10, 20, 60]	ไม่มี transaction จึงไม่มีผล

คำอธิบายฟังก์ชัน main

main() จะสร้าง CP::stack<int> ขึ้นมาและอ่านคำสั่งทีละบรรทัด โดยแต่ละบรรทัดจะระบุคำสั่งที่ต้องการทดสอบ มีรูปแบบดังนี้:

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

- **pop**: เรียก stk.pop()
- **begin**: เรียก stk.begin()
- **commit**: เรียก stk.commit()
- **rollback**: เรียก stk.rollback()
- **print**: พิมพ์ข้อมูลทั้งหมดใน stack ออกมา (เพื่อการตรวจสอบ)
- **exit**: สิ้นสุดการทำงาน

รับประกันว่าจะไม่มีการเรียก pop หรือ top ในขณะที่ stack ว่าง (ตามสถานะที่มองเห็นในขณะนั้น)

ข้อบังคับ

1. 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
2. ในไฟล์ student.h ดังกล่าวจะต้องไม่ทำการอ่านเขียนข้อมูลใด ๆ ไปยังหน้าจอหรือคีย์บอร์ดหรือไฟล์ใด ๆ
3. หากใช้ VS Code ให้ทำการ compile ที่ไฟล์ main.cpp
4. **main ที่ใช้จริงใน grader นั้นจะแตกต่างจาก main ที่ได้รับในไฟล์โปรเจ็กต์เริ่มต้นแต่จะทำการทดสอบในลักษณะเดียวกัน**