<자료구조및실습 과제> - 스택 설계

문제: 심층문제 6-1 (스택 설계)를 확장한 문제

프로그램 요구사항

1) 스택을 배열 또는 연결리스트 가운데 어느 것으로 구현해도 좋다.

2) 주함수(즉, **main** 함수)는 아래에 보인 사용자의 명령 코드를 입력 받을 때마다 해당 부함수를 호출하여 명령을 수행한다.

- **p e** : 스택에 원소 **e**를 삽입 (push)
- **P** : 스택에 **100**만개 원소를 삽입 (pushMillion)
- **o** : 스택으로부터 원소를 삭제 (pop)
- **O** : 스택으로부터 **100**만개 원소를 삭제 (popMillion)
- **f** : 스택 원소 중 최소값 출력 (findMin)
- **q** : 수행 종료 (quit)

3) **p**, **o**, **f** 명령이 입력되면 별도로 작성된 **push**, **pop**, **findMin** 부함수를 각각 호출하여 수행한다.

4) **P** 명령이 입력되면 주함수는 부함수 **pushMillion**을 호출한다. **pushMillion** 함수는 난수발생함수를 **100**만번 호출하여 **1000** ~ **9999** 사이의 난수 정수 **100**만개를 차례로 스택에 삽입한 후 반환한다. 이때 스택 원소 삽입은 반드시 위 **3**항에서 호출한 **push** 함수를 **100**만번 호출하여 처리해야 한다.

5) **O** 명령이 입력되면 주함수는 부함수 **popMillion**을 호출한다. **popMillion** 함수는 **100**만번의 스택 삭제를 수행한 후 반환한다. 이때 스택 원소 삭제는 반드시 위 **3**항에서 호출한 **pop** 함수를 **100**만번 호출하여 처리해야 한다.

6) **q** 명령이 입력되면 주함수가 종료된다 - 현재 스택을 깨끗이 비우고 종료하는 것을 추천하지만 그렇게 하지 않더라도 감점은 없음.

7) 주함수는 **q**를 제외한 모든 명령코드에 대한 해당 부함수의 수행시간을 측정하여 출력해야 한다.

8) 프로그램이 완성되면 정확히 작동하는지 확인하기 위해 소규모로 테스트할 때 교재 **p.183**의 입출력 실행예를 사용할 것을 추천한다.

9) 소규모 입력에 대해 정확히 작동하는지 확인이 끝나면 아래 예시와 같은 대규모 실행을 수행한다.

입출력 예시

입력	출력
p 119	push 119 (1), cputime = XX.XXXXXXXXX
P	pushMillion (1000001), cputime = XX.XXXXXXXXX
f	min 119 (1000001), cputime = XX.XXXXXXXXX
p 33	push 33 (1000002), cputime = XX.XXXXXXXXX
p 119	push 119 (1000003), cputime = XX.XXXXXXXXX
f	min 33 (1000003) , cputime = XX.XXXXXXXXX
p 33	push 33 (1000004), cputime = XX.XXXXXXXXX

o pop 33 (1000003), cputime = XX.XXXXXXXXX
f min 33 (1000003), cputime = XX.XXXXXXXXX
o pop 119 (1000002), cputime = XX.XXXXXXXXX
O popMillion (2), cputime = XX.XXXXXXXXX
f min 119 (2), cputime = XX.XXXXXXXXX
q (프로그램 종료)

※ **참고:**

1. 위 실행예에 보인 출력형식을 준수해야 한다.
2. 실행예에서 괄호 속의 수는 해당 명령 수행직후의 스택 사이즈를 나타낸다.
3. cputime X.XXXXXXXX는 측정된 cputime in milliseconds이다. 이 실행시간은 각 부함수에 대한 호출과 반환 사이에 소요된 cputime을 표시해야 한다. 이 과제의 경우, 상수 시간을 나타내는 cputime은 거의 **0**초 혹은 아예 **0**초로 표시되더라도 무방하다.
4. 채점에 방해가 되므로, 위에 보인 것과 같이, 수많은 스택원소들을 출력하는 일은 절대 없어야 한다.
5. 위 실행예 수행에서, 자신의 PC 성능 문제(너무 느림)로 인해 불가피하다면 **100**만 대신 **10**만을 사용해서 작성해도 좋다.
6. 위 실행예는 예시일 뿐, 다른 실행예를 스스로 만들어 추가로 수행해보는 것도 좋다.

주의

1) 심층문제 **6-1**을 확장한 과제이므로 원래 심층문제의 요구사항들 역시 모두 준수해야 한다.

2) **pushMillion**, **popMillion** 함수는 선형시간에 수행하지만(위 실행예 일련번호 **2**번과 **11**번 작업), **push**, **pop**, **findMin** 세 개의 함수 모두 상수시간에 작동해야 만점을 받을 수 있다(위 **2**번과 **11**번을 제외한 모든 작업들) - 만약 **p**, **o**, **f** 명령 수행에 소요된 cputime이 상수시간(즉, **0**초 혹은 거의 **0**초)이라고 볼 수 없는 큰 수로 나타난다면 해당 함수가 상수시간이 아닌, 선형시간에 작동한다는 의미다. 이 경우 해당 함수를 상수시간에 작동하도록 수정해야 할 것이다.

3) 명령 에코, 괄호 속의 현재 스택 사이즈, cputime 등에 대한 계산 및 출력은 모두 (각 부함수가 아닌) 주함수에서 수행해야 한다. 이러한 계산 또는 출력 명령이 각 부함수에 포함되면 감점.

4) 연결스택 구현에서는 스택 사이즈를 유지하는 전역변수(예를 들어 **n**)를 사용해도 좋다. 참고로 순차스택 구현에서는 **top** 값을 이용해서 스택 사이즈를 구할 수 있으므로 이런 전역변수가 불필요하다.

5) 메모리 사용이 과다할 경우 만점을 기대할 수 없으므로 불필요한 메모리 사용을 줄이도록 설계하라. 참고로, 할당 크기 **100**만 정도의 정수 원소 스택 **1**개를 사용하는 것까지는 필수적이겠으나 만약 이 정도 크기의 메모리를 추가로 사용(즉, 결과적으로 필수 스택 크기의 약 **2**배수의 총메모리를 사용)하여 이 문제를 풀어낸다면 메모리 과다 사용으로 판정되어 감점될 것임. 따라서 이보다는 현저히 적은 추가 메모리를 사용해서 풀어낼 방안을 생각해내야 한다.

6) 데이터구조 및 알고리즘 면에서 간결하게 해결할 수 있는 것을 불필요하게 복잡하게 해결하면 (입출력이 정확하더라도) 감점.

7) 프로그램 작성 시에 모듈화를 고려하여 설계할 것. 모듈화가 잘 되어 있지 않은 프로그램은 **10%** 감점. 모듈화란 각 부함수가 논리적, 기능적인 면에서 역할이 분명하며 독립적임을 의미한다.

8) 속임수에 의한 프로그램을 작성한 경우 부정행위로 처리함. 속임수 예를 한 개 들면, **findMin** 작업시 **pushMillion**에 의해 삽입된 **100**만개의 원소들을 제외한 나머지 스택 원소들만 비교하여 최소 원소를 찾는다면 상수시간에 작동하는 것처럼 "연기하는" **findMin**을 얻을 수는 있으나 이는 속임수에 해당함.

9) 출력 화면을 캡쳐하여 제출하는 과제가 아님. 주어진 실행예 및 스스로 만든 추가 실행예를 각자 수행해보고 잘 수행하는지 확인한 후 소스프로그램만을 **OJ**에 재출할 것.