데이터 구조 Assn3

20180551 이준석

POVIS ID : ljs9904ljs

Problem 1 (R-5.6)

```
typedef struct node {
        int data;
        struct node* next;
}NODE;

typedef struct stack {
        NODE* top;
}STACK;

void popAllStack(STACK* s) { //모든 element들을 pop하는 함수를 재귀로 구현
        NODE* temp;
        if (s->top->next == NULL) {
                free(s->top);
                return;
        }
        else {
                temp = s->top;
                s->top = temp->next;
                free(temp);
                popAllStack(s);
        }
}
```

마지막 하나를 제거할 때를 종료 조건으로 한다. 그 외에는 pop을 수행하고 다시 한 번 popAllStack 함수를 호출한다.

Problem 2 (R-5.10)

function call	Output
insertFront(3)	3 //제일 앞에 추가 후 deque의 목록
insertBack(8)	3, 8 //제일 뒤에 추가 후 deque의 목록
insertBack(9)	3, 8, 9 //제일 뒤에 추가 후 deque의 목록
insertFront(5)	5, 3, 8, 9 //제일 앞에 추가 후 deque의 목록
removeFront()	3, 8, 9 //제일 앞 삭제 후 deque의 목록
eraseBack()	3, 8 //제일 뒤 삭제 후 deque의 목록
first()	3 //deque의 목록 중 제일 앞 출력
insertBack(7)	3, 8, 7 //제일 뒤에 추가 후 deque의 목록
removeFront()	8, 7 //제일 앞 삭제 후 deque의 목록
last()	7 //deque의 목록 중 제일 뒤 출력
eraseBack()	8 //제일 뒤 삭제 후 deque의 목록

Problem 3 (C-5.1)

```
typedef struct node {
        int data;
        struct node* next;
}NODE;
typedef struct stack {
        NODE* top;
        int size;
}STACK;
typedef struct deque {
        STACK* front;
        STACK* back;
}DEQUE;

void pushFront(DEQUE* d, int data) {
        push(d->front, data);
}

void pushBack(DEQUE* d, int data) {
        push(d->back, data);
}

void popFront(DEQUE* d, int* data) {
        int tempData;

        if (d->front == NULL) { //front stack이 비어있을 때
                while (d->back != NULL) {
                        pop(d->back, &tempData);
                        push(d->front, tempData);
                } //back stack의 모든 node를 front stack으로 옮긴다.
                pop(d->front, data); //원래의 target을 pop한다.

                while (d->front != NULL) {
                        pop(d->front, &tempData);
                        push(d->back, tempData);
                } //임시로 옮겨둔 것을 다시 back stack으로 옮긴다.
        }
        else //front stack이 비어있지 않으면
                pop(d->front, data); //front stack에서 pop을 한다.
}
```

```
void popBack(DEQUE* d, int* data) {
	int tempData;

	if (d->back == NULL) {
		while (d->front != NULL) {
			pop(d->front, &tempData);
			push(d->back, tempData);
		}
		pop(d->back, data);

		while (d->back != NULL) {
			pop(d->back, &tempData);
			push(d->front, tempData);
		}
	}
	else
		pop(d->back, data);
}

void front(DEQUE* d, int* data) {
	int tempData;

	if (d->front == NULL) {
		while (d->back != NULL) {
			pop(d->back, &tempData);
			push(d->front, tempData);
		}
		top(d->front, data);

		while (d->front != NULL) {
			pop(d->front, &tempData);
			push(d->back, tempData);
		}
	}
	else
		top(d->front, data);
}
```

```
void back(DEQUE* d, int* value) {
	int tempData;

	if (d->back == NULL) {
		while (d->front != NULL) {
			pop(d->front, &tempData);
			push(d->back, tempData);
		}
		top(d->back, data);

		while (d->back != NULL) {
			pop(d->back, &tempData);
			push(d->front, tempData);
		}
	}
	else
		top(d->back, data);
}

int empty(DEQUE* d) { // 비어있으면 1, 아니면 0 리턴
	if (d->front->top == NULL && d->back->top == NULL)
		return 1;
	else
		return 0;
}

int size(DEQUE* d) {
	return (d->front->size + d->back->size);
}
```

스택 2개를 이용해 deque를 구현하였다. 스택 2개의 bottom들을 서로 붙인다는 이미지로 만들었다. 여기서 문제가 하나 발생한다. front stack이 전부 다 비어있을 때 popFront를 하면 back stack의 bottom node를 pop해야하는데 접근할 수가 없는 것이다. 그래서 back stack의 모든 node를 front stack으로 옮겨놓고 back stack의 bottom node를 pop한 뒤, front stack에 임시로 옮겨둔 것을 다시 back stack으로 옮기는 방식을 사용하였다. popBack, front, back 함수에서도 마찬가지의 방식을 활용하여 문제를 해결하였다.

Problem 4 (C-5.2)

```
int findValue(STACK* s, QUEUE* q, int x) { //스택 s와 큐 q를 이용해 스택에 x가 있는지
                                           //찾는다. x가 있으면 1 리턴, 없으면 0 리턴
        int tempValue; // 스택에서 pop할 때 나온 값을 받을 변수
        int& temp=tempValue;
        int thereIsX = 0; // 찾는 값 x가 있는지 없는지. 있으면 1, 없으면 0

        while (s->top != NULL) { //스택 s에서 pop을 반복하며 x가 있는지 확인한다.
                pop(s, temp); // 스택 s에서 pop한다. pop할 때 나온 값을 임시로 저장
                enqueue(q, tempValue); // 스택 s에서 나온 값을 큐에 저장
                if (tempValue == x) { // pop할 때 나온 값이 찾는 값 x와 같은지 확인
                        thereIsX = 1; // x가 stack 안에 존재한다.
                }
        }

        /* 아래는 스택을 원상복구하기 위한 과정이다. 큐에는 원래 스택에 저장되어 있던 순
서의 역순으로 저장되어 있으므로 그 순서를 바꾸기 위함이다. */

        while (q->head != NULL) { // 큐에 저장되어 있는 값을 다시 스택으로 옮겨 저장
                dequeue(q, temp); // 큐의 값을 뽑아낸다.
                push(s, tempValue); // 큐에서 뽑아낸 값을 스택에 저장한다.
        }
        while (s->top != NULL) { // 스택에 저장되어 있는 값을 큐에 옮겨 저장한다.
                pop(s, temp);
                enqueue(q, tempValue);
        }
        while (q->top != NULL) { // 큐에 저장되어 있는 값을 스택에 옮겨 저장한다.
                dequeue(q, temp);
                push(s, tempValue);
        }

        return thereIsX; // 찾는 값 x가 있는지 없는지를 리턴해준다.}
```

스택에서 큐로 큐에서 스택으로 가는 과정을 총 2번 거쳐야하는 이유는 다음과 같다.

스택에 (top)1->2->3->4가 저장되어 있다고 하자.
pop해서 enqueue하면 큐에 (head)1->2->3->4(tail) 순으로 저장된다.
다시 dequeue해서 push하면 스택에 (top)4->3->2->1 순으로 저장된다.

즉, 한 번 스택의 값을 전부 큐로 옮겼다가 큐의 값을 전부 스택으로 옮기면 스택의 값이 원래의 역순으로 저장된다. 따라서 두 번 같은 과정을 거쳐야 원래의 순서로 다시 저장할 수 있는 것이다.

Problem 5 (C-5.5)

```
void push(QUEUE* qMain, QUEUE* qSub, int data) {
	int tempData;

	while (qMain->head != NULL) { // main queue의 데이터들을 sub queue로 옮긴다.
		dequeue(qMain, &tempData);
		enqueue(qSub, tempData);
	}
	enqueue(qMain, data); //입력받은 값을 main queue에 넣는다.

	while (qSub->head != NULL) { // sub queue의 데이터들을 main queue로 옮긴다.
		dequeue(qSub, &tempData);
		enqueue(qMain, tempData);
	}
}
void pop(QUEUE* qMain, int* data) {
	dequeue(qMain, data);
}
void top(QUEUE* qMain, int* data) {
	front(qMain);
}
void stackSize(QUEUE* qMain) {
	queueSize(qMain);
}
void stackIsEmpty(QUEUE* qMain) {
	queueIsEmpty(qMain);
}
```

처음에 큐에 값을 저장할 때부터 역순으로 저장해놓는다. 그러면 가장 앞에 있는 부분부터 값이 꺼내지더라도 스택의 Last In First Out처럼 기능하게 된다.

역순으로 저장하기 위해서 push를 구현할 때 두 개의 큐를 이용한다. 스택처럼 작동할 큐를 main queue라 하고 임시 저장 용도로 사용할 큐를 sub queue라 하자.

1. main queue에 저장되어 있는 값 전부를 sub queue에 저장한다.
2. 저장하고자 하는 값을 main queue에 enqueue한다.
3. sub queue에 저장되어 있는 값 전부를 main queue에 저장한다.

위 1, 2, 3번의 과정을 거치면 역순으로 저장된다.

push는 O(n), pop은 O(1)의 running time을 갖는다.