Lab 7. Stack의 구현

◎ 실험 실습 일시 : 2009. 4. 27.
◎ 담당 교수 : 정진우
◎ 담당 조교 : 곽문상
◎ 보고서 제출기한 : 2009. 5. 3.

학과 : _____
학번 : _____
성명 : _____

◎ 실습 과제 목적 : 이론시간에 배운 Stack을 실제로 구현할 수 있다.
◎ 실습 과제 내용 : 주어진 소스를 이용해 Stack의 각 함수를 구현한다.
◎ 실습순서

단계 1. 다음 프로그램은 원통형 CD보관함을 모델링한 것이다. 주어지는 코드와 제시된 함수 설계를 참고해 각 함수들을 완성하시오.

1) 스택을 초기화하는 함수 **void initStack(STACK *stack)**을 구현한다. 이 함수는 매개변수로 STACK 형의 구조체 포인터를 받고 반환 값은 없다. 매개변수 stack의 멤버인 stackSize, bottom, top을 각각 0과 NULL로 초기화시킨다.

2) 스택을 출력하는 함수 **void printStack(STACK *stack)**을 구현한다. 매개변수로 출력할 STACK형의 구조체 포인터를 받고 반환 값은 없다. 매개변수 stack의 bottom 노드부터 시작해서 모든 노드의 data를 출력한다.

3) 스택의 상태를 알아보는 **bool isEmpty(STACK *stack)**와 **bool isFull(STACK *stack)**를 구현한다. 이 두 함수는 각각 스택이 비어있는 지 가득 차있는 지를 검사하는 함수로 STACK형의 구조체 포인터를 매개변수로 받고 반환 값은 bool형의 true/false 이다.

4) 스택에 data를 하나 삽입하는 **void push(STACK *stack, STACKNODE *node)**를 구현한다. 매개변수로 STACK형의 구조체 포인터와 STACKNODE 형의 node를 받으며 반환 값은 없다. 이 함수는 insertCD() 함수에서 호출되는 함수로, 매개변수로 전달 받은 스택의 top에 node를 삽입하는 함수이다. 새로 만들어지는 노드는 동적메모리 할당을 하는 함수인 malloc()을 이용해서 만들며, 삽입이 불가능한 상황 (ex) 스택이 가득 차있는 경우)에는 그에 따른 에러 메시지를 출력해야 한다.

5) 스택에서 node를 하나 꺼내는 **void pop(STACK *stack, STACKNODE *removedNode)**을 구현한다. 매개변수로 STACK형의 구조체 포인터와 STACKNODE 형의 removedNode를 받으며 반환 값은 없다. 이 함수는 removeCD() 함수에서 호출되는 함수로, 매개변수로 전달 받은 stack의 top위치에 있는 노드를 꺼내는 함수이다. 꺼낸 노드의 정보는 두 번째 매개 변수인 removedNode에 저장이 되며, 삭제된 노드에 할당된 메모리 공간은 해제되어야 한다.

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

//스택의 최대 크기
#define MAX 10

//스택 노드 구조체
struct StackNode {
	char *title;			// cd 이름
	char *singer;			// 가수
	struct StackNode *next;		// 다음 노드의 포인터
};
typedef struct StackNode STACKNODE;

//스택 구조체
struct Stack{
	int stackSize;			// 스택의 크기
	STACKNODE *top;			// 제일 위쪽 노드의 포인터
	STACKNODE *bottom;		// 제일 아래쪽 노드의 포인터
};
typedef struct Stack STACK;


void initStack(STACK *stack);		//스택 초기화 함수
void printStack(STACK *stack);		//스택 출력 함수
bool isEmpty(STACK *stack);		//스택이 비어있는지 확인하는 함수
bool isFull(STACK *stack);		//스택이 가득 차 있는 지 확인하는 함수
void insertCD(STACK *stack);		//CD를 집어넣는 함수
void push(STACK *stack, STACKNODE *node);   //스택에 node를 삽입하는 함수
void removeCD(STACK *stack);		//CD를 꺼내는 함수
void pop(STACK *stack, STACKNODE *removedNode); //스택에서 node를 꺼내는 함수


void main()
{
	STACK *stack;
	char cmd;

	printf("*************** Choose Command!! *****************₩n");
	printf("+ : push, - : pop, F: full check, E: empty check ₩n");
	printf("Q: Quit₩n");
	printf("*****************************************************₩n");

	//스택에 동적으로 메모리 할당
	stack = (STACK *)malloc(sizeof(STACK));

	//스택 초기화
	initStack(stack);
```

```
	do {
		// 커맨드 입력
		printf("Command: ");
		cmd = getch();
		putch(cmd);
		printf("₩n");
		// 커맨드는 무조건 대문자로 입력
		cmd = toupper(cmd);

		switch (cmd)
		{
			case '+' :  // push
				insertCD(stack);
				break;
			case '-' :  // pop
				removeCD(stack);
				break;
			case 'E' :  // check empty
				if (isEmpty(stack)) printf ("Stack is empty!₩n");
				else printf ("Stack isn't empty₩n");
				break;
			case 'F' :  // check full
				if (isFull(stack)) printf ("Stack is full!₩n");
				else printf ("Stack isn't full!₩n");
				break;
			case 'Q' :  // quit
				break;
			default :   // wrong command
				printf("₩nWrong command! Retry!₩n");
		}
		// 스택 출력
		printStack(stack);
	} while ( cmd !='Q' );

}

// 스택 출력
void printStack(STACK *stack)
{
	int nSerial=1;
	// 임시 사용 포인터 선언
	STACKNODE *tempCursor, *cursor;
	// 스택의 첫 노드를 가리킴
	tempCursor = stack->bottom;
	cursor = stack->top;


	if (isEmpty(stack))  // 리스트가 비어있는 경우
	{
		printf("Stack is empty!₩n");
	}
```

```
		else
		{
			printf("CD list₩n");
			printf ("=================================================₩n");

			while(tempCursor != NULL)
			{
				printf("%d : %s (%s)₩n",nSerial, tempCursor->title,tempCursor->singer);
				tempCursor = tempCursor->next;
				nSerial++;
			}

			printf("₩n");
		}
}
// 스택 초기화 함수
void initStack(STACK *stack)
{

}
// 스택이 비어있는지 확인
bool isEmpty(STACK *stack)
{

}
// 스택이 가득찼는지 확인
bool isFull(STACK *stack)
{

}
//CD를 집어넣는 함수
void insertCD(STACK *stack)
{
		// 새로운 노드 선언
		STACKNODE *tempNode;
		// 새로운 노드에 동적 메모리 할당 후 data 저장
		tempNode = (STACKNODE *)malloc(sizeof(STACKNODE));
		tempNode->next = NULL;
		tempNode->title = (char *)malloc(sizeof(char)*20);
		tempNode->singer = (char *)malloc(sizeof(char)*20);

		printf ("=================================================₩n");
		printf ("CD Title : ");
		gets(tempNode->title);
		printf ("Singer : ");
		gets(tempNode->singer);
		printf("₩n");
		printf ("=================================================₩n");

		push(stack,tempNode);
}
```

```
//Push
void push(STACK *stack, STACKNODE *node)
{

}

//CD를 꺼내는 함수
void removeCD(STACK *stack)
{
	// 새로운 노드 선언
	STACKNODE *deletedNode;
	// 새로운 노드에 동적 메모리 할당 후 data 저장
	deletedNode = (STACKNODE *)malloc(sizeof(STACKNODE));
	deletedNode->next = NULL;
	deletedNode->title = NULL;
	deletedNode->singer = NULL;

	pop(stack, deletedNode);

	printf ("₩nRemoved CD : %s (%s)₩n",deletedNode->title,deletedNode->singer);
}
//Pop
void pop(STACK *stack, STACKNODE *removedNode)
{

}
```

단계 2. 단계 1에서 완성된 프로그램을 실행시켜 각 함수들을 테스트해보고 그 결과 화면을 캡쳐하시오.

Extra 문제 (+2점). 두 개의 스택을 하나로 합치는 **void mergeStack(STACK *stack1, STACK *stack2)**를 만드시오. 이 함수는 매개 변수로 STACK형의 구조체 stack1과 stack2를 받으며 반환 값은 없다. 두 개의 스택을 합칠 때는 스택의 연산자인 push()와 pop()만을 이용해야 하고, 결과적으로 만들어지는 스택은 첫 번째 매개 변수인 stack1에 저장해야 한다. (즉, stack1위에 stack2가 새로 쌓이는 형태임)

◎ 과제 수행 결과 (결과 화면 및 소감)