Assignment 3 보고서

소프트웨어학과 202220757
김재학

목차

1. 코드 간략한 설명

개발환경: VisualStudio 2022, SDL검사 사용안함.

1.1. Array를 사용한 Stack

우선 다음과 같이 stack을 구성하는 구조체와 함수를 정의하였다.

```
//array stack 구현
typedef struct {
        int data[MAX_STACK_SIZE];
        int top;
}STACK;

STACK* creatStack();
bool isFull(STACK* stack);
bool isEmpty(STACK* stack);
void Push(STACK* stack, int item);
int Pop(STACK* stack);
void deleteStack(STACK* stack);
```

STACK* creatStack();함수를 사용하면 MAX_STACK_SIZE 만큼의 배열을 포함한 구조체를 동적 할당하여 이 구조체를 가리키는 포인터를 반환한다.

```
STACK* arrStack = creatStack();
```

따라서 위와 같은 형태로 Array Stack을 Heap영역에 생성할 수 있다.

또한, void deleteStack(STACK* stack); 함수를 사용하여 Heap영역에 생성한 스택을 해제할 수 있다.

isFull(), isEmpty()의 경고문구는 Push()와 Pop()을 실행할 때 나오게 된다.

이외의 함수 기능에 대한 설명은 일반적인 Stack의 기능과 동일하므로 생략한다.

1.2. Linked list를 사용한 Stack

```
//linked stack 구현
typedef struct NODE{
        int data;
        struct NODE* link;
}NODE;

typedef struct {
        NODE* top;
        int count;
}StackList;

StackList* newStackList();
void initList(StackList* stack);
bool isEmptyList(StackList* stack);
bool isFullList(StackList* stack);
void pushList(StackList* stack, int item);
int popList(StackList* stack);
void deleteList(StackList* stack);
```

Linked Stack을 구현하기 위한 코드는 위와 같다.

struct NODE{…}는 Linked List를 구현하기 위한 데이터가 들어갈 구조체이고, 이 노드의 헤드노드는 **struct StackList{…}**를 통해 생성된다. 즉, StackList 구조체가 헤드노드를 가리키는 포인터의 역할을 한다.

Linked Stack 은 StackList* newStackList()함수를 통해 생성할 수 있다. 이 함수는 헤드노드를 Heap 영역에 생성하여 이 노드를 가리키는 구조체 포인터를 반환한다.

```
StackList* linkedStack = newStackList();
```

따라서 위와 같은 코드로 linked Stack 을 선언할 수 있다.
void initList(StackList* stack)함수는 처음 생성된 StackList 의 top 변수와 count 변수를 초기화 해주는 역할을 하는데, 이는 newStackList();에 포함되어 있어서 stack 이 생성될 때, 실행되게 된다.

void deleteList(StackList* stack);함수는 linkedStack 의 사용으로 생성된 모든 메모리를 해제하는 함수이다.

이외의 함수에 대한 기능은 일반적인 stack 의 기능과 동일하므로 설명을 생략한다.

2. 메모리 사용에 대한 고찰

다음과 같은 코드를 통해서 Array를 이용한 스택의 메모리 사용을 관찰하였다.

```
	//array stack test
	STACK* arrStack = creatStack();
	int i = 0;
	for (i = 0; i < 11; i++) {
		Push(arrStack, i);
	}

	for (i = 0; i < 11; i++) {
		Pop(arrStack);
	}

	deleteStack(arrStack);
```

arrStack이란 포인터변수에 스택을 생성하여 할당한다. 그리고 0부터 11까지의 정수를 반복해서 스택에 push한다. 이때 Push()함수를 통해 스택에 저장된 값들과 메모리 주소를 확인할 수 있다.

그리고 11번 pop을 실행하게 되고, 마지막에는 스택을 삭제한다.

실행결과의 일부는 다음과 같다.

```
stack:
9 0000020ABA19EC94
8 0000020ABA19EC90
7 0000020ABA19EC8C
6 0000020ABA19EC88
5 0000020ABA19EC84
4 0000020ABA19EC80
3 0000020ABA19EC7C
2 0000020ABA19EC78
1 0000020ABA19EC74
0 0000020ABA19EC70
Stack is full, cannot  add element
Stack is empty, cannot delet element
```

이 실행결과를 보면 스택에 저장된 값들과 저장된 위치의 주소 값이 위에서부터 차례대로 출력된 것을 확인할 수 있다. 이때 메모리의 주소 값들이 4씩 차이가 나는 것을 알 수 있는데, 이것은 int형 array를 이용해서 스택을 구현했기 때문이다.

9	8	7	6	5	4	3	2	1	0

위의 그림과 같이 데이터가 메모리에 차례대로 저장 돼있는 것이다.

다음으로 Linked Stack의 메모리 사용을 관찰하여 보았다.

```
        //linked stack test
        StackList* linkedStack = newStackList();
        for (i = 0; i < 11; i++) {
                pushList(linkedStack, i);
        }

        for (i = 0; i < 11; i++) {
                popList(linkedStack);
        }

        deleteList(linkedStack);
```

위와 같은 코드를 이용해 스택의 메모리 사용을 관찰하였다.

linkedStack이라는 변수에 stack의 헤드노드를 가리키는 포인터를 생성하여 할당하고, 0부터 11까지의 정수를 push한다. 이때 pushList()라는 함수를 통해서 stack에 저장된 값들과 메모리의 주소를 확인할 수 있다. 그리고 이 스택에 pop을 11번 실행하고, 마지막에는 이 스택을 해제하게 된다.

실행결과의 일부는 다음과 같다.

```
linked stack:
9 0000020ABA19D830
8 0000020ABA19D9C0
7 0000020ABA19D470
6 0000020ABA19D420
5 0000020ABA19D600
4 0000020ABA19D560
3 0000020ABA19D8D0
2 0000020ABA19D5B0
1 0000020ABA19D6A0
0 0000020ABA19DB50
스택이 최대크기입니다. (최대 스택 사이즈:10)
스택이 비었습니다.
```

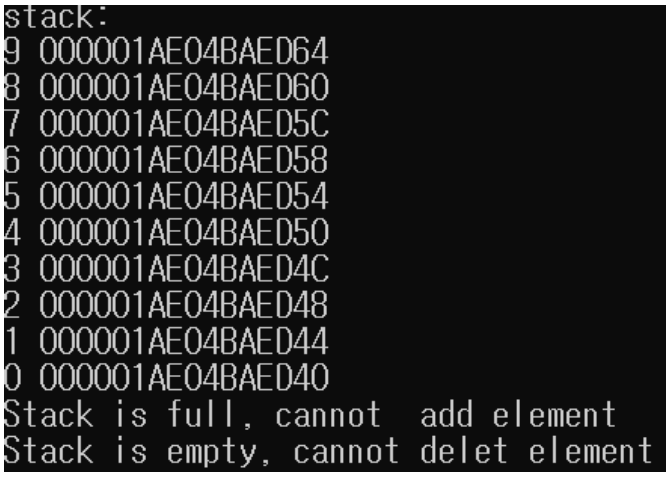
이때 확인할 수 있는 것은 메모리의 주소 값들이 일정한 규칙을 가지지 않고 있다는 것이다. 이는 이 stack이 Push를 할 때마다 새로운 노드를 동적할당해서 직전의 노드에 연결시키는 linked list의 형태이기 때문이다.

이 stack의 메모리형태의 일부를 추상적으로 그려본다면 다음과 같다.

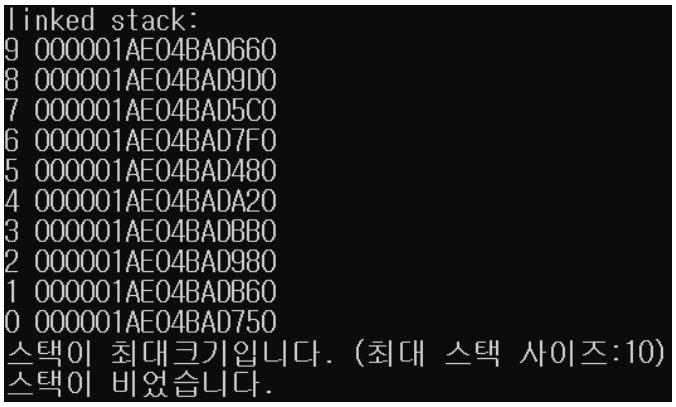
...	Adress: #2 Data: 0 Link: null		Adress: #4 Data: 1 Link: #2			Adress: #45 Data: 2 Link: #4		Adress: #50 Data: 3 Link: #45				Adress: #58 Data: 4 Link: 50	...

Push를 실행할 때마다 새로운 노드를 동적할당하여 데이터를 저장한 후 직전의 노드의 주소값을 함께 저장하여 직전 노드와 연결시키는 방식인 것이다. 따라서 한꺼번에 여러 개의 공간을 할당하는 array와 달리 linked list를 이용한 stack은 메모리 주소가 불규칙 적이다.

위의 코드들을 다시한번 실행시켜 보았다.



<- Array Stack



<- Linked Stack

위와 같은 결과를 볼 수가 있었는데, 두개의 스택 모두 다 메모리의 주소가 달라졌지만 array는 계속해서 메모리 주소가 4씩 차이가 난다는 규칙을 가지고 있는 것을 다시 확인할 수 있었다. 그리고 linked stack은 메모리가 불규칙 적으로 할당된다는 것을 다시 확인할 수 있었다.

본 과제를 해결하면서 알게 된 것은 linked list로 구현한 stack은 메모리 할당 위치가 불규칙 적이라는 것이다. 또한, push를 할 때마다 새로운 노드를 생성하는 연산을 하기 때문에 array에 비해서 시간이 더 소모된다는 것을 알 수 있다.