스택

Stack

스택 ADT

: 데이터를 바닥에서부터 쌓아올리는 구조

LIFO : Last In First Out 구조

(후입선출) : 마지막에 넣은 것이 가장 먼저 제거됨.

: 요소의 삽입과 삭제가 한쪽 끝에서만 이루어짐.

스택과 리스트 비교

리스트는 삽입, 삭제 등의 연산을 리스트 내의 어디서든 행할 수 있다.

스택의 경우 삽입과 삭제 등의 연산이 한쪽 끝에서만 이루어진다.

스택의 중요성

프로세스 주소공간의 스택영역도 스택ADT를 기반으로함.

문서편집기의 '되돌리기(undo)' 기능도 스택 기반이다.

Ctrl+Z를 생각하면 쉽다.

이처럼 컴퓨터의 많은 영역에서 스택이 사용되고있다.

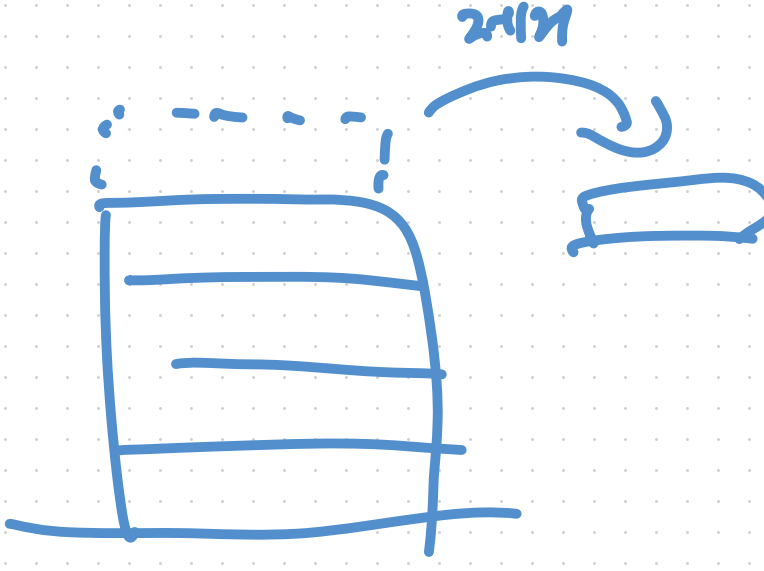
스택의 핵심 기능

- 삽입과 제거 연산 ★

- 그 외의 기능은 보조연산이다.

삽입

제거



스택의 두 가지 구현 방법

1. 배열

장점 : 스택 생성 최초에 용량 만큼의 노드를 한 번에 생성할 수 있다.

단점 : 용량을 동적으로 변경해줘야한다.

2. 링크드 리스트

장점 : 스택 용량에 제한을 두지 않아도 된다.

단점 : 배열과 달리 인덱스를 이용해 노드에 접근할 수 없기 때문에 노드에 대한 포인터를 갖고 있어야 한다.
(자신의 뒤에 존재하는)

배열 기반 스택 : 노드 표현

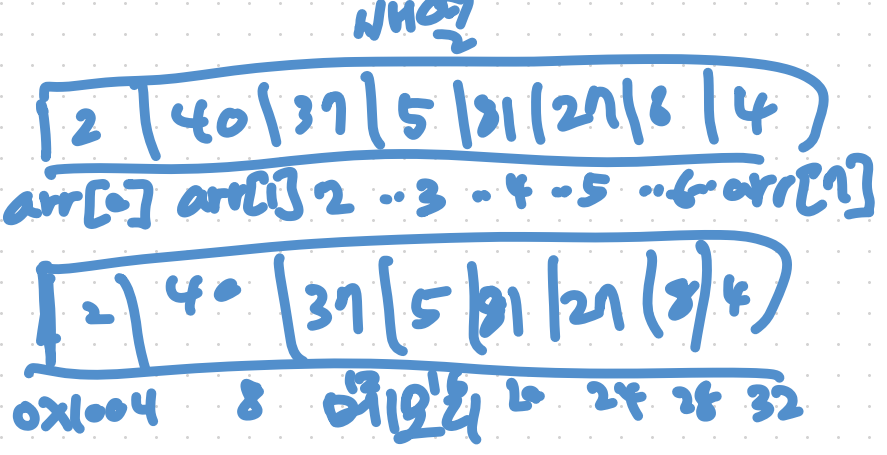
배열을 기반으로 구현하는 스택은 노드의 위치를 배열의 인덱스로 알 수 있기 때문에 포인터가 필요하지 않다.

⇒ 따라서 노드는 데이터만 담는 구조체로 표현된다.

```
typedef struct _Node {
    int data;
} Node;
```

+ 추가설명

배열은 논리적인 순서와 물리적인 순서가 같다



배열의 인덱스 덕분에 노드의 위치에 접근하는 것은 매우 쉽다.

스택의 최상위 노드의 위치만
알고 있으면 그 외의

배열 기반 스택 : 스택 구조체에 필요한 세가지 필드

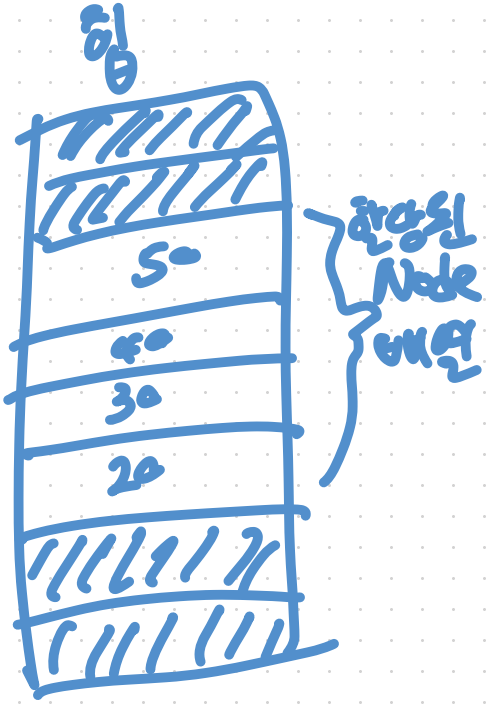
- 용량 : 스택이 얼마만큼의 노드를 가질 수 있는지 알기 위함.
- 최상위 노드의 위치 : 삽입/제거 연산 시 최상위 노드에 접근할 수 있도록 하기 위함.
- 노드 배열 : 스택에 쌓이는 노드를 보관하기 위함.

⇒ 링크드 리스트 때는 노드만 만들었는데, 왜 스택은 노드 구조체에 스택 구조체까지 만드나요?

링크드 리스트는 노드끼리 연결해주기만 하면 될 것이고,
스택의 경우, 노드를 담을 공간이 필요하기 때문에 스택 구조체도 만들어줘야 한다.

배열기반 스택 : 스택 구조체 구현

```
typedef struct _ArrayStack{
    int maxSize;
    int top;
    Node* nodeArray;
} ArrayStack;
```



Node 포인터는
힙 메모리영역에
할당한 배열의
첫번째 요소를 가리킨다.
(포인터로 배열은 사용할 것)

배열기반스택 : 스택 생성

```
Void create_Stack (ArrayStack** stack, int size){
1 (*stack)= (ArrayStack*) malloc (sizeof(ArrayStack));
2 (*Stack)→nodeArray =(Node*)malloc(sizeof(Node)*maxSize);
3 (*Stack)→maxSize = size;
4 (*Stack)→top = -1;
}
```

추가선언

1. ArrayStack을 힙 메모리 영역에 할당하기위해 사용

2. 매개변수 size만큼 노드를 미리 생성하는데 사용

3. 스택의 용량저장

4. C언어에서는 첫번째 인덱스가 0이므로, 현재스택에 아무것도 들어가지 않기 때문에 -1로 초기화함.
   top은 노드가 삽입되면 1씩 증가, 삭제되면 1씩 감소함.

배열기반스택 : 스택소멸

```
Void destroy_Stack(ArrayStack* stack){
    free(stack->nodeArray);
    free(stack);
}
```

[배열기반 스택: 노드 삽입 (push)]

: 최상위 노드의 인덱스(top)에서 1을 더한 위치에 새 노드를 입력한다.

```
void push(ArrayStack* stack, int data)
{
    stack->top++;
    stack->nodeArray[stack->top].data = data;
}
```

배열기반스택 : 노드 제거 (pop)

: 최상위 노드의 인덱스를 1 감소시키고, 최상위 노드의 데이터를 반환하도록 한다.

```
int pop(ArrayStack* stack) {
    int popNode = stack->top--;
    return stack->nodeArray[popNode].data;
}
```