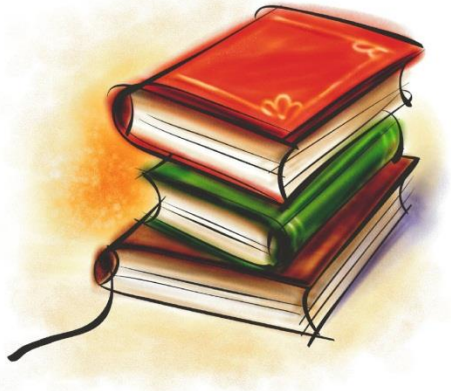
CSE2010 자료구조론

Week 4: Stack 1

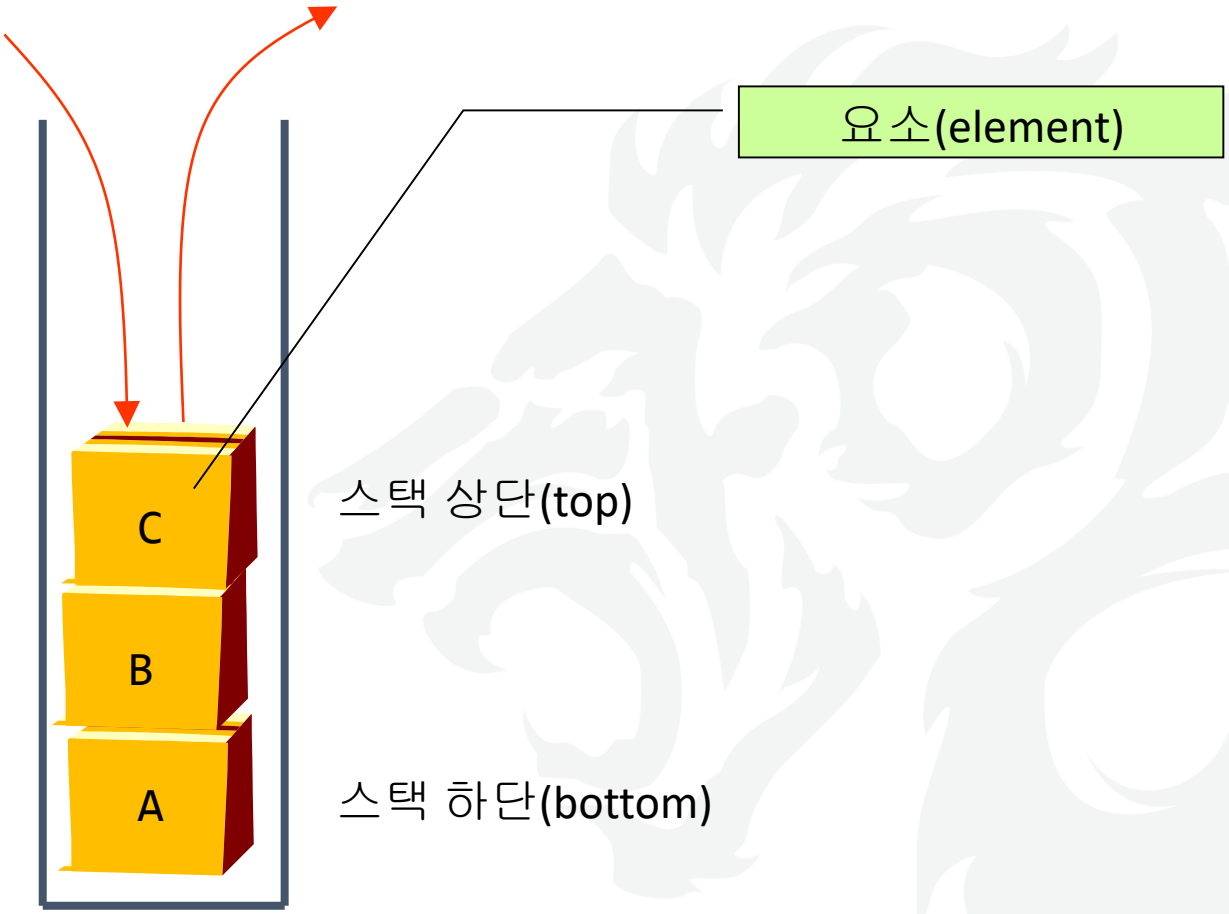
ICT융합학부 한진영

스택(stack)?

- 스택 = 쌓아 놓은 더미

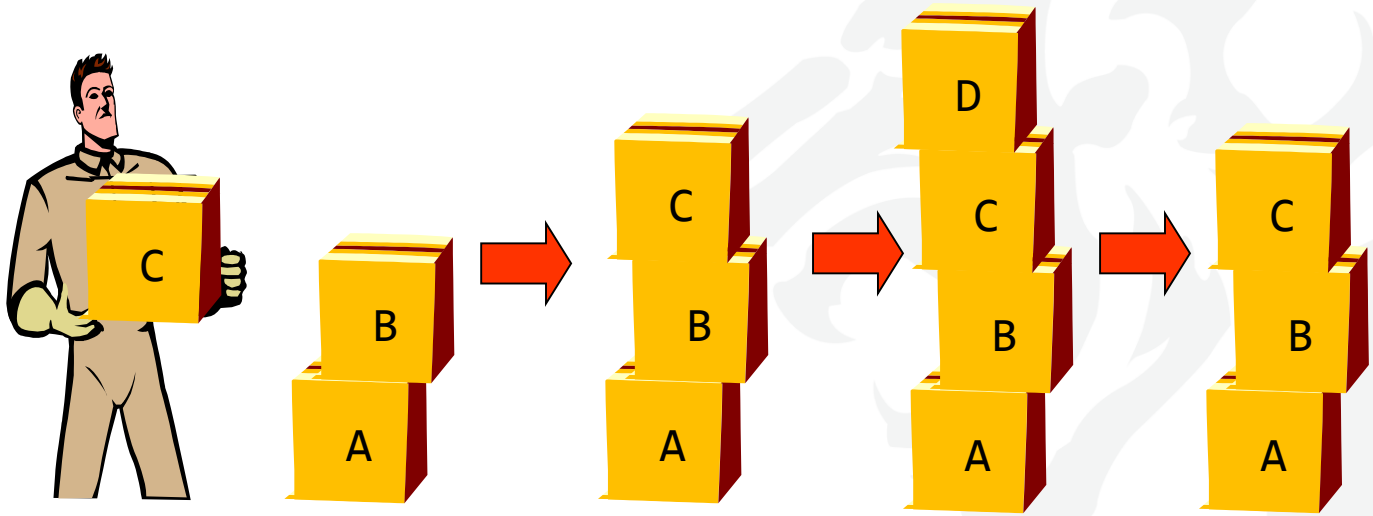


스택의 구조



스택의 특징

- 후입선출(LIFO:Last-In First-Out)
  - 가장 최근에 들어온 데이터가 가장 먼저 나감



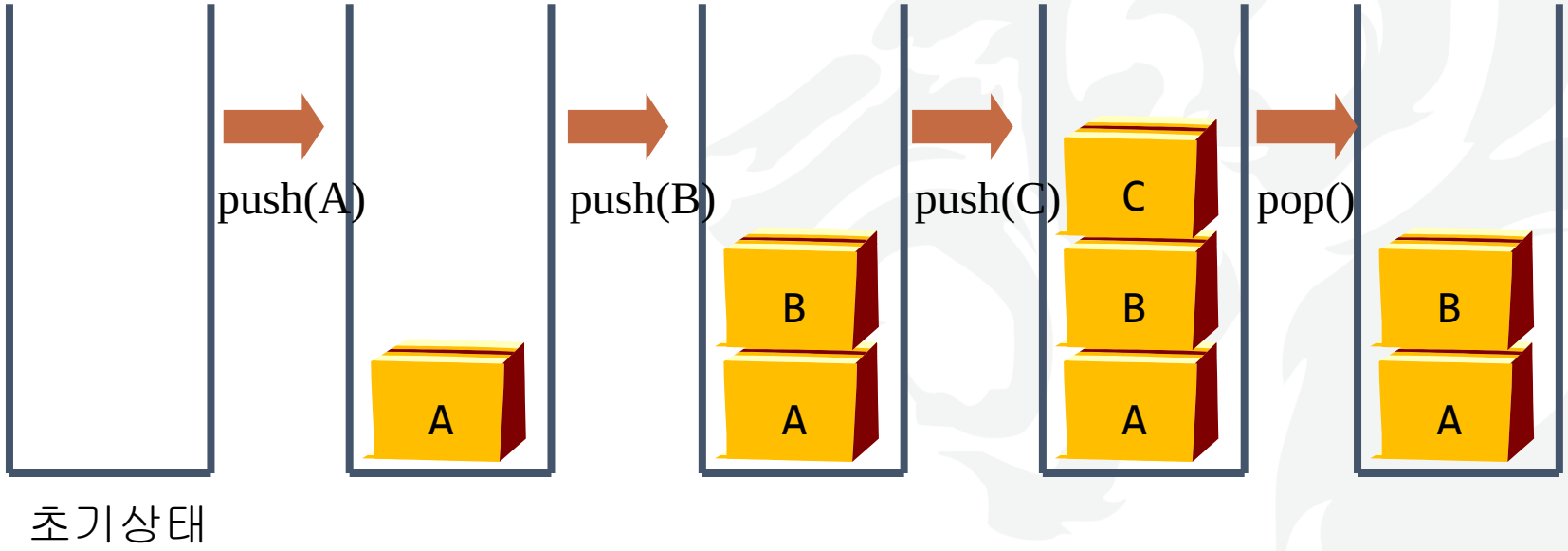
스택 ADT

- 스택 ADT

·객체: n개의 element형의 요소들의 선형 리스트
·연산:
- create() ::= 스택을 생성
- is_empty(s) ::= 스택이 비어있는지를 검사
- is_full(s) ::= 스택이 가득 찼는가를 검사
- push(s, e) ::= 스택의 맨 위에 요소 e를 추가
- pop(s) ::= 스택의 맨 위에 있는 요소를 삭제
- peek(s) ::= 스택의 맨 위에 있는 요소를 삭제하지 않고 반환

스택 연산(1)

- push(): 스택에 데이터를 추가
- pop(): 스택에서 데이터를 삭제



스택 연산(2)

- create(): 스택을 생성
- is_empty(s): 스택이 공백상태인지 검사
- is_full(s): 스택이 포화상태인지 검사
- pop(s): 요소를 스택에서 완전히 삭제하면서 가져옴
- peek(s): 요소를 스택에서 삭제하지 않고 보기만 하는 연산

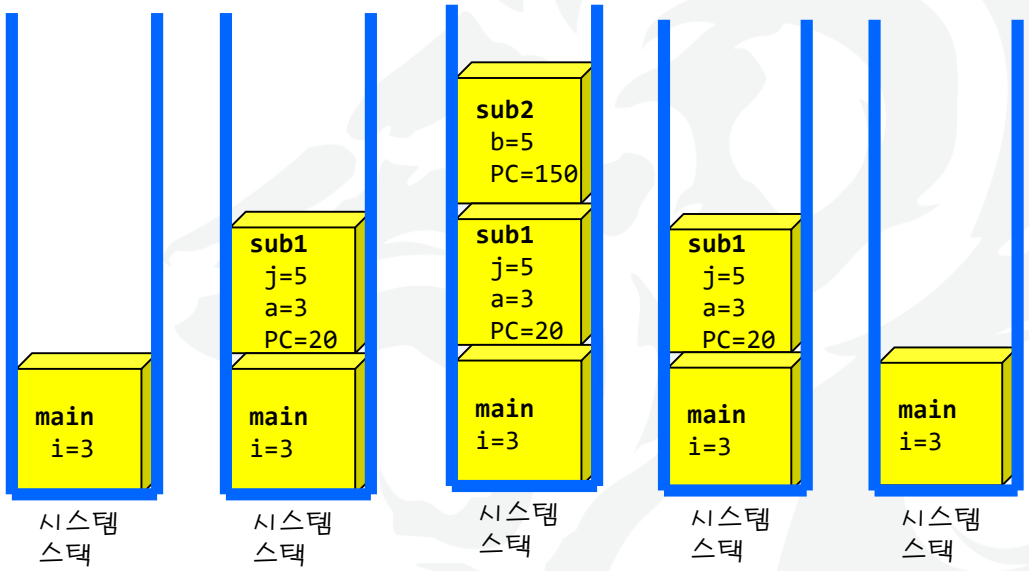
스택의 용도 예(1)

- 자료의 출력 순서가 입력 순서의 역순으로 이뤄져야 할 때
  - 입력: (A,B,C,D,E) -> 출력: (E,D,C,B,A)
- 에디터에서 되돌리기(undo) 기능
  - 최근 수행한 명령어들 중에서 가장 최근에 수행한 것부터 되돌리기
- 함수 호출에서 복귀주소(PC: Program Counter) 기억
  - 시스템 스택: 컴퓨터 OS만 사용함, 사용자는 접근 안됨
  - 시스템 스택에는 함수가 호출될 때마다 활성화 레코드(activation record)가 만들어지고, 여기에 복귀주소가 기록됨

스택의 용도 예(2)

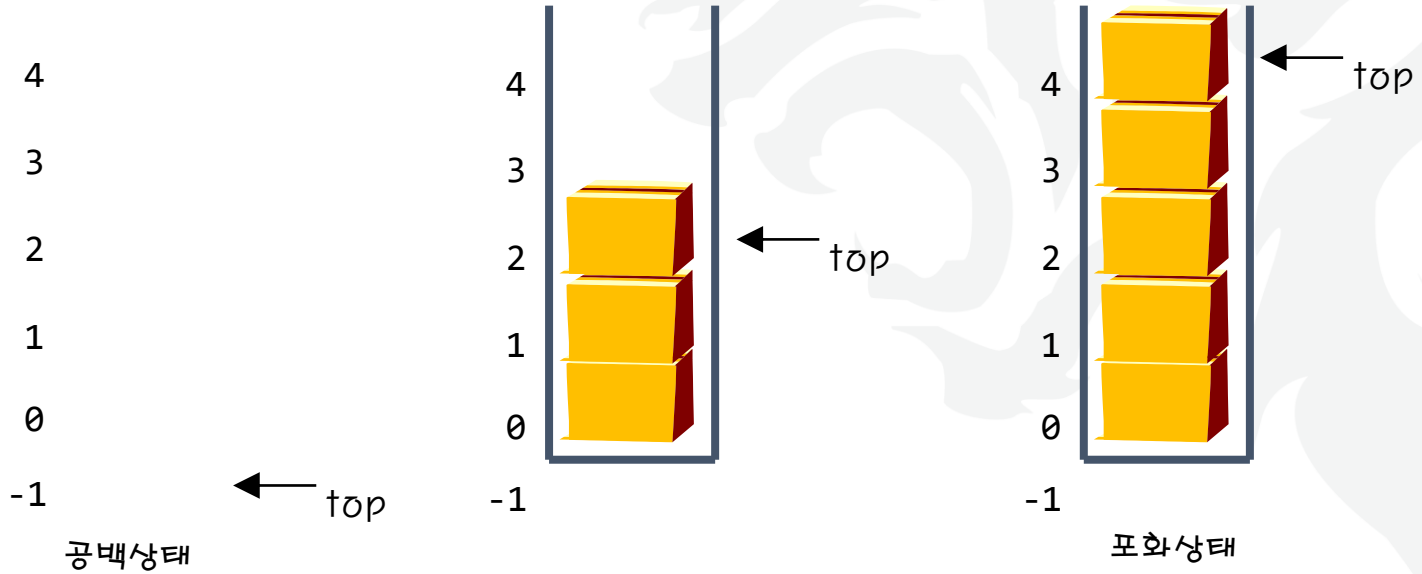
- 함수 호출시 복귀주소(PC: Program Counter)를 시스템 스택에 저장함

```
1       int main()
        {
          int i=3;
20         sub1(i);
           ...
        }

100     int sub1(int a)
        {
          int j=5;
150        sub2(j);
           ...
        }

200     void sub2(int b)
        {
           ...
        }
```



배열을 이용한 스택의 구현

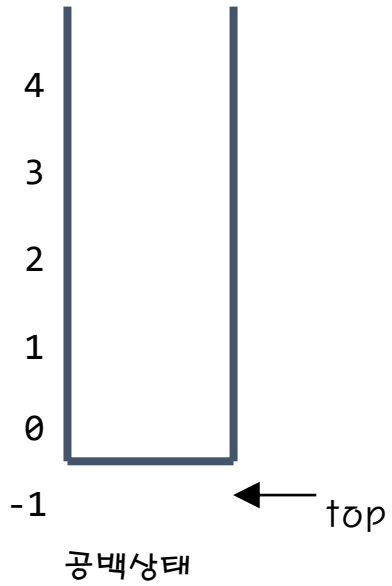
- 1차원 배열 stack[MAX_STACK_SIZE]
- top 변수: 스택에서 가장 최근에 입력되었던 자료를 가리킴
  - 가장 먼저 들어온 요소는 stack[0]에, 가장 최근에 들어온 요소는 stack[top]에 저장
  - 스택이 공백상태이면 top은 -1

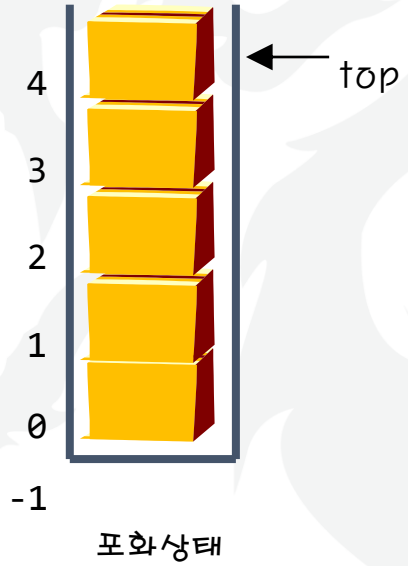


배열을 이용한 스택의 구현: is_empty, is_full

```
is_empty(S)

if top = -1
     then return TRUE
     else return FALSE
```

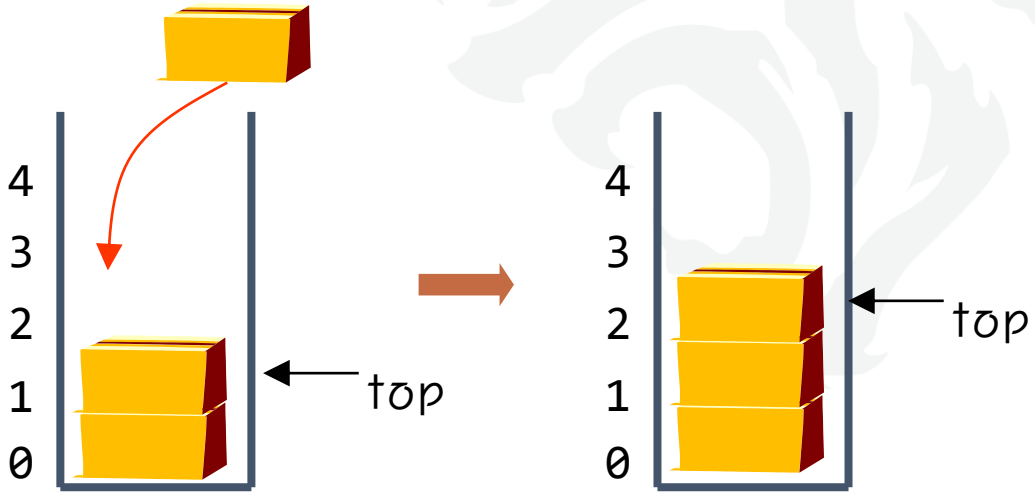
```
is_full(S)

if top = (MAX_STACK_SIZE-1)
     then return TRUE
     else return FALSE
```





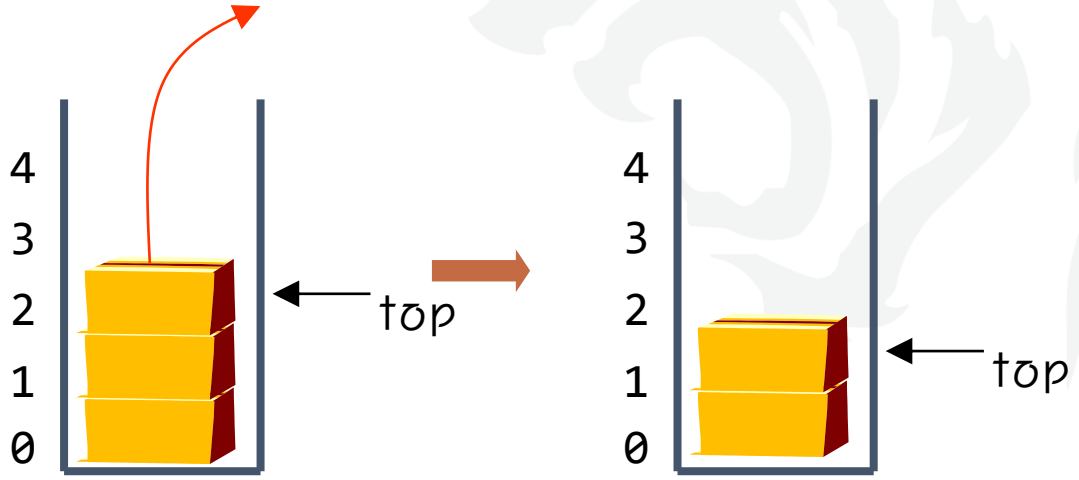
배열을 이용한 스택의 구현: push

```
push(S, x)


if is_full(S)
      then error "overflow"
      else top ← top+1
           stack[top] ← x
```



배열을 이용한 스택의 구현: pop

```
pop(S)

if is_empty(S)
      then error "underflow"
      else e ← stack[top]
           top←top-1
           return e
```



배열을 이용한 스택 구현#1: 전역변수(1)

```
#define MAX_STACK_SIZE 100
typedef int element;
element stack[MAX_STACK_SIZE];
int top = -1;

// 공백 상태 검출 함수
int is_empty( )
{
    return (top == -1);
}
// 포화 상태 검출 함수
int is_full( )
{
    return (top == (MAX_STACK_SIZE-1));
}
```

배열을 이용한 스택 구현#1: 전역변수(2)

```
// 삽입함수
void push(element item)
{
       if( is_full() ) {
              fprintf(stderr,"스택 포화 에러₩n");
              return;
        }
        else stack[++top] = item;
}
// 삭제함수
element pop( )
{
    if( is_empty( ) ) {
              fprintf(stderr, "스택 공백 에러₩n");
              exit(1);
     }
     else return stack[top--];
}
```

배열을 이용한 스택 구현#1: 전역변수(3)

```
// 피크함수
element peek( )
{
    if( is_empty( ) ) {
            fprintf(stderr, "스택 공백 에러₩n");
            exit(1);
     }
     else return stack[top];
}
```

배열을 이용한 스택 구현#1: 매개변수(1)

```
typedef int element;
typedef struct {
    element stack[MAX_STACK_SIZE];
    int top;
} StackType;


// 스택 초기화 함수
void init(StackType *s)
{
    s->top = -1;
}
// 공백 상태 검출 함수
int is_empty(StackType *s)
{
    return (s->top == -1);
}
```

배열을 이용한 스택 구현#1: 매개변수(2)

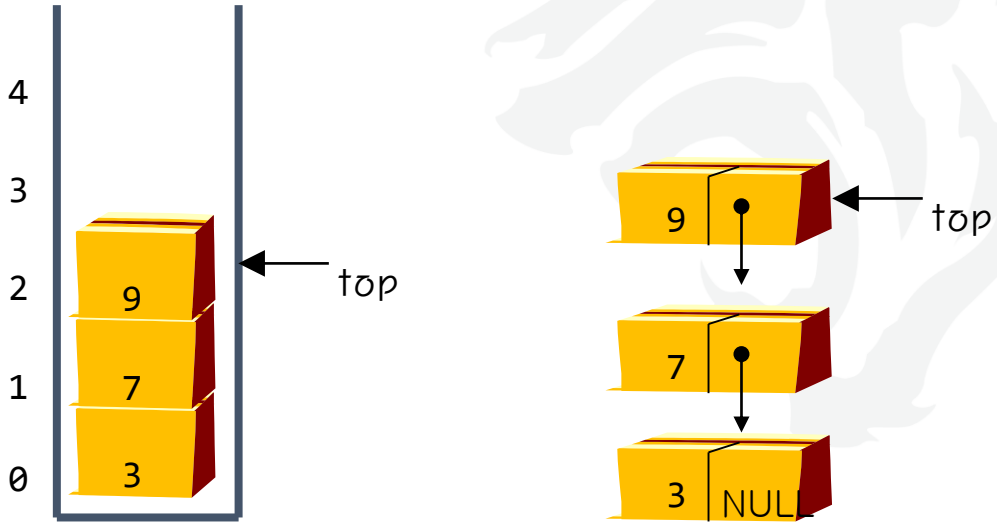
```
// 포화 상태 검출 함수
int is_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE-1));
}

// 삽입함수
void push(StackType *s, element item)
{
        if( is_full(s) ) {
                fprintf(stderr,"스택 포화 에러₩n");
                return;
         }
         else s->stack[++(s->top)] = item;
}
```

배열을 이용한 스택 구현#1: 매개변수(3)

```
// 삭제함수
element pop(StackType *s)
{
    if( is_empty(s) ) {
            fprintf(stderr, "스택 공백 에러₩n");
            exit(1);
     }
     else return s->stack[(s->top)--];
}
// 피크함수
element peek(StackType *s)
{
    if( is_empty(s) ) {
            fprintf(stderr, "스택 공백 에러₩n");
            exit(1);
     }
     else return s->stack[s->top];
}
```

연결된 스택

- 연결된 스택(linked stack): 연결리스트를 이용하여 구현한 스택
- 장점: 크기가 제한되지 않음
- 단점: 구현이 복잡하고 삽입이나 삭제 시간이 오래 걸림
  - 동적 메모리 할당 및 해제 때문



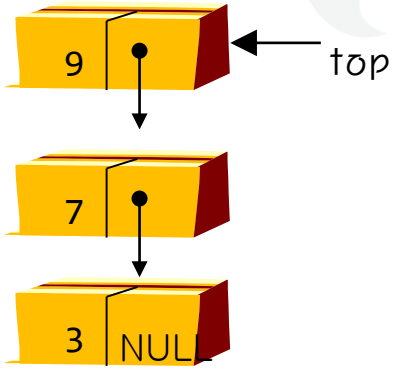
연결된 스택 구조

```
typedef int element;

typedef struct StackNode {
     element item;
     struct StackNode *link;
} StackeNode;

typedef struct {
     StackNode *top;
} LinkedStackType;
```

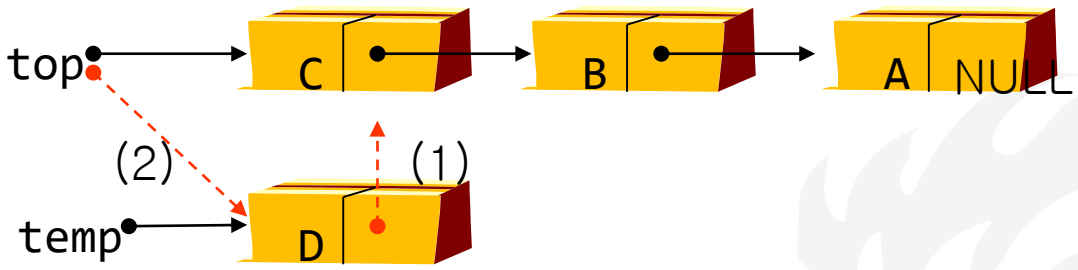
요소의 타입

노드의 타입

연결된 스택의 관련 데이터

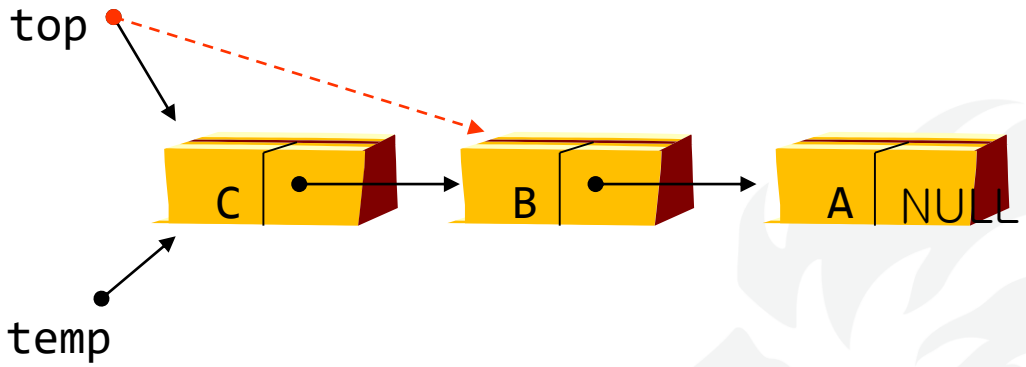


연결된 스택: push




```
// 삽입 함수
void push(LinkedStackType *s, element item)
{
   StackNode *temp=(StackNode *)malloc(sizeof(StackNode));
   if( temp == NULL ){
      fprintf(stderr, "메모리 할당에러\n");
      return;
   }
   else{
      temp->item = item;
      temp->link = s->top;
      s->top = temp;
   }
}
```

연결된 스택: pop




```
// 삭제 함수
element pop(LinkedStackType *s)
{
    if( is_empty(s) ) {
            fprintf(stderr, "스택이 비어있음\n");
            exit(1);
     }
     else{
            StackNode *temp=s->top;
            element item = temp->item;
            s->top = s->top->link;
            free(temp);
            return item;
     }
}
```

Week 4: Stack 1