

스택

1. 스택의 이해 : 스택의 개념과 구조

● 스택(stack)

- 스택에 저장된 원소는 top으로 정한 것만 접근 가능
 - top의 위치에서만 원소를 삽입하므로, 먼저 삽입한 원소는 밑에 쌓이고, 나중에 삽입한 원소는 위에 쌓이는 구조
 - 마지막에 삽입(Last-In)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(First-Out)됨 ➡ **후입선출 구조 (LIFO, Last-In-First-Out)**

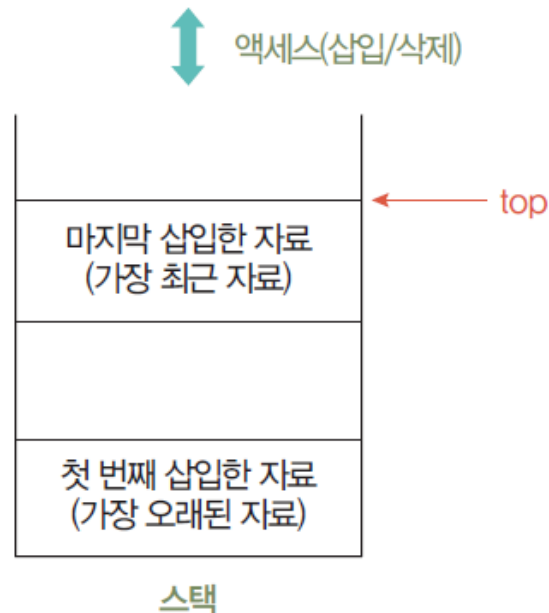


그림 5-2 스택의 구조

1. 스택의 이해 : 스택의 개념과 구조

● 스택의 연산

- 스택에서의 삽입 연산 : **push**
- 스택에서의 삭제 연산 : **pop**
- 스택에서의 원소 삽입/삭제 과정
 - 공백 스택에 원소 A, B, C를 순서대로 삽입하고
한번 삭제하는 연산과정 동안의 스택 변화

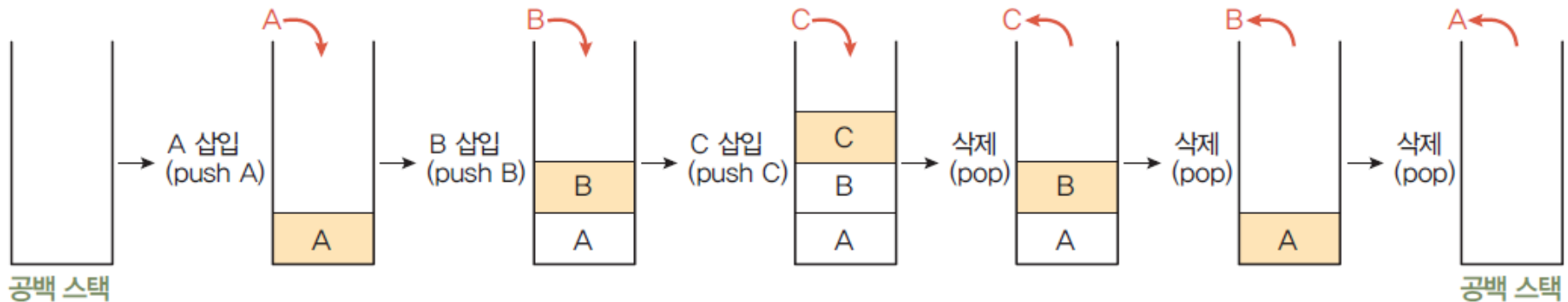


그림 5-6 스택의 데이터 삽입(push)과 삭제(pop) 과정

1. 스택의 이해

● 스택의 push() 알고리즘

① $top \leftarrow top + 1;$

- 스택 S에서 top이 마지막 자료를 가리키고 있으므로 그 위에 자료를 삽입하려면 먼저 top의 위치를 하나 증가
- 만약 이때 top의 위치가 스택의 크기(stack_SIZE)보다 크다면 오버플로우(overflow)상태가 되므로 삽입 연산을 수행하지 못하고 연산 종료

② $S(top) \leftarrow x;$

- 오버플로우 상태가 아니라면 스택의 top이 가리키는 위치에 x 삽입

알고리즘 5-1 스택의 원소 삽입

```
push(S, x)
  ① top ← top + 1;
  if (top > stack_SIZE) then overflow;
  else
    ② S(top) ← x;
end push()
```

1. 스택의 이해

● 스택의 pop() 알고리즘

① return S(top);

- 공백 스택이 아니면 top에 있는 원소를 삭제하고 반환

② top \leftarrow top-1;

- 스택의 마지막 원소가 삭제되면 그 아래 원소, 즉 스택에 남아 있는 원소 중에서 가장 위에 있는 원소가 top이 되어야 하므로 top 위치를 하나 감소

알고리즘 5-2 스택의 원소 삭제

```
pop(S)
  if (top = 0) then underflow;
  else {
    ① return S(top);
    ② top  $\leftarrow$  top - 1;
  }
end pop()
```

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

● 연결 자료구조를 이용한 스택의 구현

■ 단순 연결 리스트를 이용하여 구현

- 스택의 원소 : 단순 연결 리스트의 노드
 - 스택 원소의 순서 : 노드의 링크 포인터로 연결
 - push : 리스트의 마지막에 노드 삽입
 - pop : 리스트의 마지막 노드 삭제
- 변수 top : 단순 연결 리스트의 마지막 노드를 가리키는 포인터 변수
 - 초기 상태 : top = null

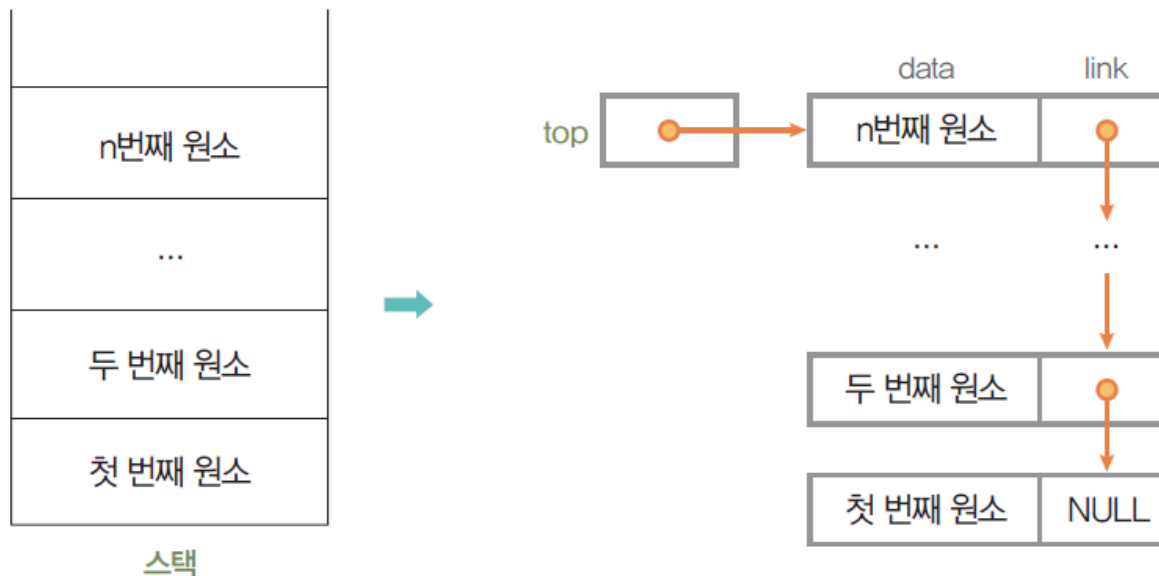
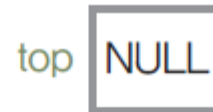


그림 5-8 단순 연결 리스트를 이용한 연결 스택

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

- 원소 A, B, C를 순서대로 삽입하면서 스택을 생성한 후에 원소 하나를 삭제하는 과정

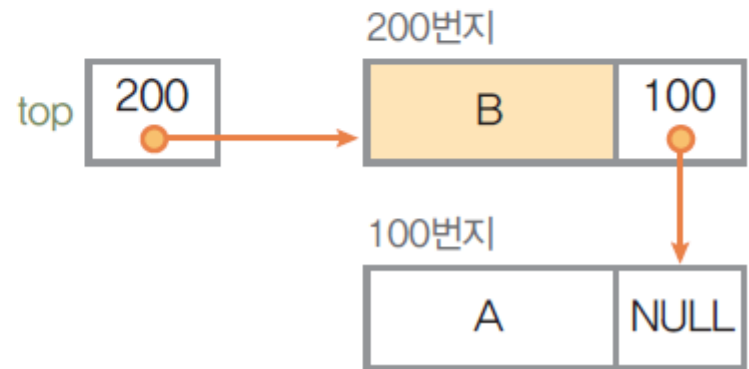
1 공백 스택 생성 : `createStack(S);`



2 원소 A 삽입 : `push(S, A);`

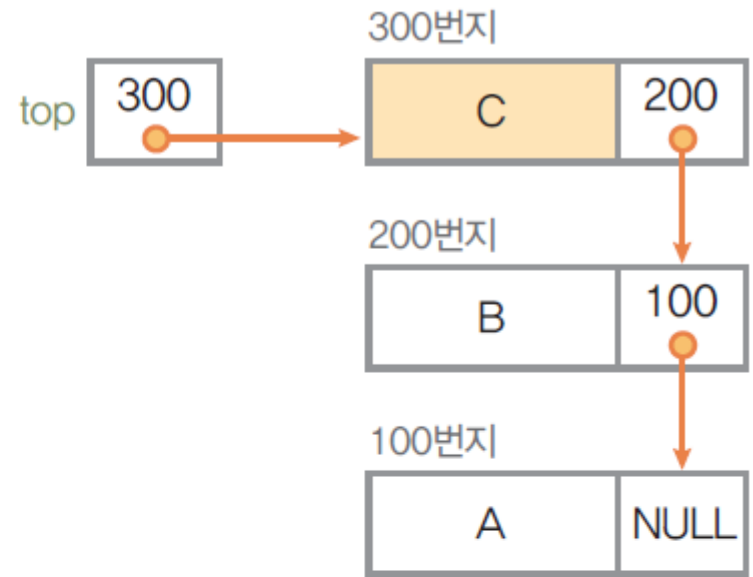


3 원소 B 삽입 : `push(S, B);`

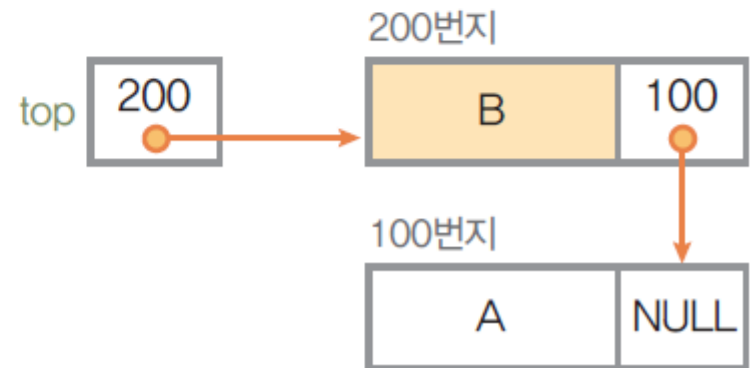


2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

4 원소 C 삽입 : `push(S, C);`



5 원소 삭제 : `pop(S);`



2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

예제 5-2

연결 자료구조를 이용해 연결 스택 구현하기

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04
05  typedef int element;          // 스택 원소(element)의 자료형을 int로 정의
06
07  typedef struct stackNode {    // 스택의 노드를 구조체로 정의
08      element data;
09      struct stackNode *link;
10  } stackNode;
11
12  stackNode* top;               // 스택의 top 노드를 지정하기 위해 포인터 top 선언
13
14  // 스택이 공백 상태인지 확인하는 연산
15  int isEmpty() {
16      if (top == NULL) return 1;
17      else return 0;
18  }
```

2. 스택의 구현: 연결 자료구조를 이용한 스택의 구현

```
19
20 // 스택의 top에 원소를 삽입하는 연산
21 void push(element item) {
22     stackNode* temp = (stackNode *)malloc(sizeof(stackNode));
23     temp -> data = item;
24     temp -> link = top;      // 삽입 노드를 top의 위에 연결
25     top = temp;             // top 위치를 삽입 노드로 이동
26 }
27
28 // 스택의 top에서 원소를 삭제하는 연산
29 element pop() {
30     element item;
31     stackNode* temp = top;
32
33     if (top == NULL) {      // 스택이 공백 리스트인 경우
34         printf("\n\n Stack is empty !\n");
35         return 0;
36     }
```

포인터를 이용한
[알고리즘 5-1] 구현

포인터를 이용한
[알고리즘 5-2] 구현

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

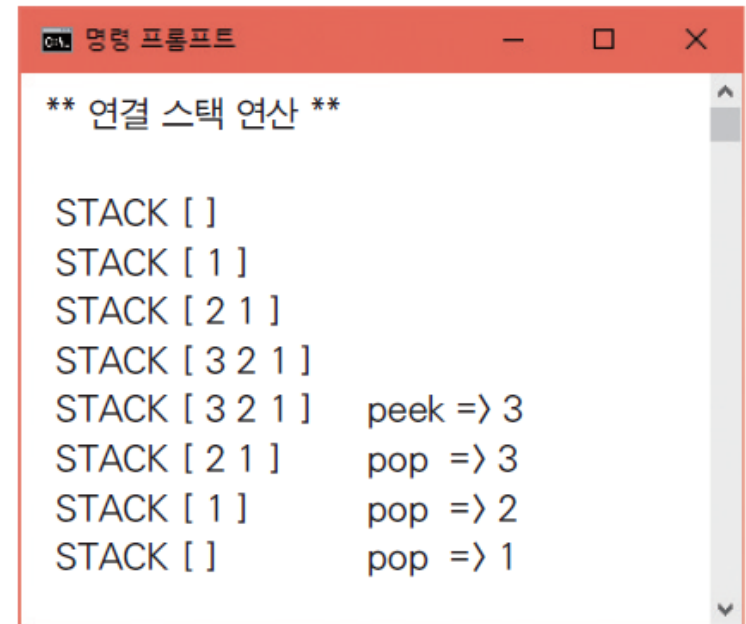
```
37     else {                                // 스택이 공백 리스트가 아닌 경우
38         item = temp -> data;
39         top = temp -> link;                // top 위치를 삭제 노드 아래로 이동
40         free(temp);                       // 삭제된 노드의 메모리 반환
41         return item;                      // 삭제된 원소 반환
42     }
43 }
44
45 // 스택의 top 원소를 검색하는 연산
46 element peek() {
47     if (top == NULL) {                    // 스택이 공백 리스트인 경우
48         printf("\n\n Stack is empty !\n");
49         return 0;
50     }
51     else {                                // 스택이 공백 리스트가 아닌 경우
52         return(top -> data);              // 현재 top의 원소 반환
53     }
54 }
55
```

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

```
56 // 스택의 원소를 top에서 bottom 순서로 출력하는 연산
57 void printStack() {
58     stackNode* p = top;
59     printf("\n STACK [ ");
60     while (p) {
61         printf("%d ", p -> data);
62         p = p -> link;
63     }
64     printf("] ");
65 }
66
67 void main(void) {
68     element item;
69     top = NULL;
70     printf("\n** 연결 스택 연산 **\n");
71     printStack();
72     push(1);    printStack();    // 1 삽입
73     push(2);    printStack();    // 2 삽입
74     push(3);    printStack();    // 3 삽입
```

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

```
75
76     item = peek(); printStack(); // 현재 top의 원소 출력
77     printf("peek => %d", item);
78
79     item = pop(); printStack(); // 삭제
80     printf("\t pop => %d", item);
81
82     item = pop(); printStack(); // 삭제
83     printf("\t pop => %d", item);
84
85     item = pop(); printStack(); // 삭제
86     printf("\t pop => %d", item);
87
88     getchar();
89 }
```



```
** 연결 스택 연산 **

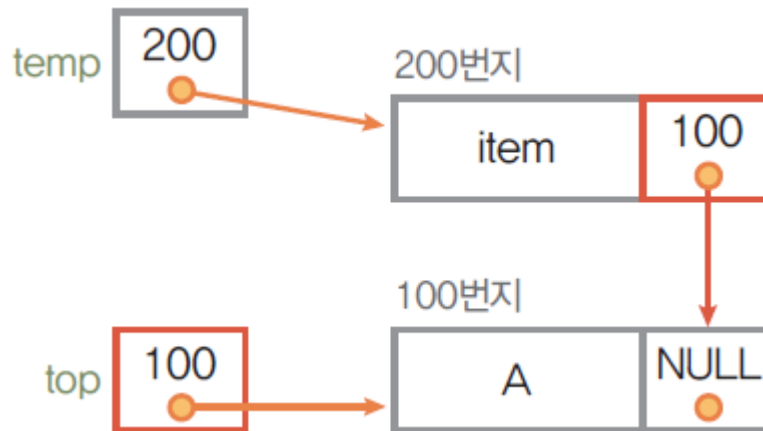
STACK [ ]
STACK [ 1 ]
STACK [ 2 1 ]
STACK [ 3 2 1 ]
STACK [ 3 2 1 ]    peek => 3
STACK [ 2 1 ]      pop  => 3
STACK [ 1 ]        pop  => 2
STACK [ ]          pop  => 1
```

2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

- 20~26행 : 연결 스택에서 삽입 연산을 수행하는 과정
 - 22행 : 원소 item을 저장할 노드에 대한 메모리 할당, 포인터 temp 설정
 - 23행 : 삽입할 노드의 데이터 필드에 원소 item을 저장

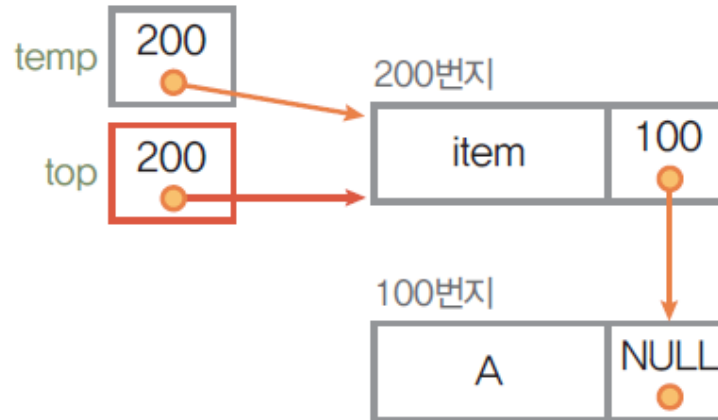


- 24행 : 삽입할 노드의 링크 필드에 포인터 top의 값 저장하면, 새로 삽입한 노드가 현재 스택의 마지막 노드(현재 top이 가리키는 노드)로 연결

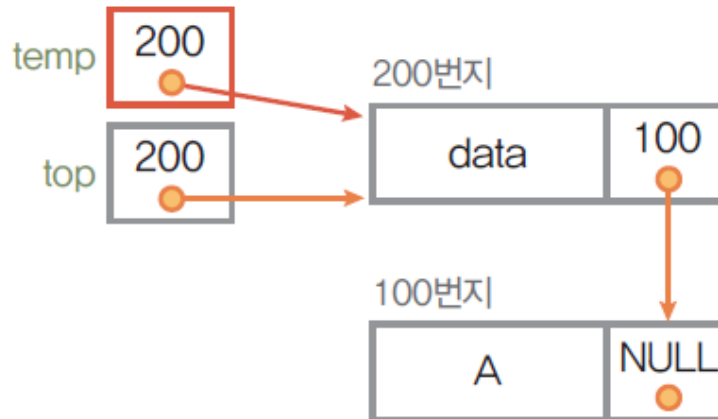


2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

- 25행 : 포인터 temp의 값(삽입 노드의 주소)을 포인터 top에 설정, 새로 삽입한 노드가 스택의 top 노드가 되도록 조정

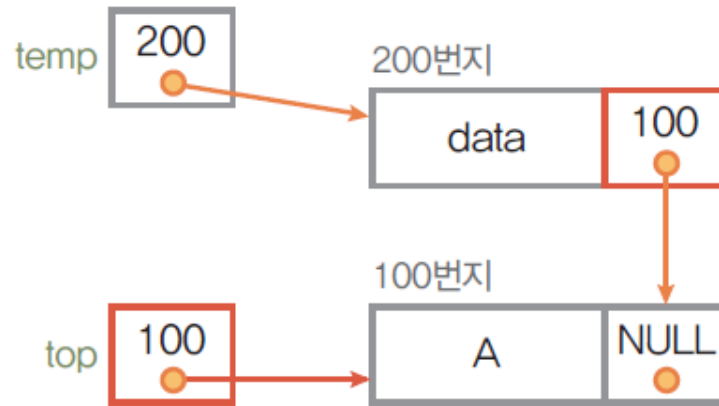


- 28~43행 : 연결 스택에서 삭제하는 연산을 수행하는 과정
 - 31행 : 포인터 temp를 top 노드에 설정하여 삭제할 노드를 가리킴



2. 스택의 구현:연결 자료구조를 이용한 스택의 구현

- 38행 : 스택의 마지막 노드의 데이터 필드값을 변수 item에 저장
- 39행 : 포인터 top의 위치를 현재 마지막 노드의 아래 노드로 이동



- 40행 : 포인터 temp가 가리키는 노드를 메모리 해제
- 41행 : 변수 item의 값, 즉 스택의 top이었던 노드의 데이터를 반환
- 60~63행 : 연결 스택에 노드가 있는 동안, top 노드부터 링크 필드를 따라 아래 노드로 이동하면서 데이터 필드값을 출력. top 노드부터 출력하므로 출력 화면에서 왼쪽이 스택의 마지막 원소인 top이 됨