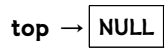
2. 연결 스택

연결 스택(linked stack)은 **연결리스트**를 이용하여 스택을 구현한 것이다.

// 연결스택 동작 및 특징

---

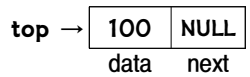
① **초기 상태**



top 포인터는 NULL을 가진다.

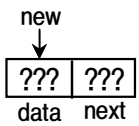
```
linked_stack *top = NULL;
```

② push(100)

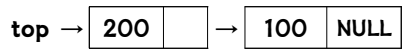


top 포인터가 첫 번째 노드를 가리킨다.

```
linked_stack *new;
new = (linked_stack *)malloc(sizeof(linked_stack)); //메모리 할당
new->data = 100;
new->next = top;
top = new;
```



③ push(200)



top 포인터는 최근에 생성된 노드를 가리킨다.

- 스택을 연결리스트로 구현하면 첫 번째 자료가 저장된 노드의 포인터가 NULL이다.

- 연결스택은 **스택**의 **크기**를 미리 **제한**할 필요가 없다.(순차스택은 미리 결정함)
- 연결스택은 기억공간이 필요할 때 **동적** 할당하는 방식이다.

- 연결스택은 스택이 가득찬 경우(overflow)는 조사하지 않아도 된다.
- 연결스택은 메모리 용량이 충분하다고 가정하므로

◈ 연결리스트를 이용한 스택(연결스택)

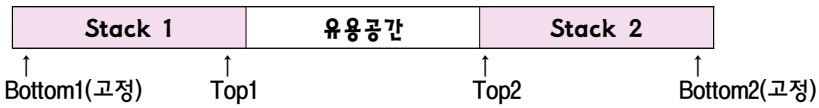
```
#define NULL  0
typedef struct linked_stack linked_stack;
struct linked_stack{
    int data;
    linked_stack *next;
};
linked_stack *top = NULL;                                   //초기상태 : top 포인터를 NULL로 한다
int empty(){ return ( top==NULL ? 1 : 0 ); }
void push(int key){
    linked_stack *new;
    new = (linked_stack *)malloc(sizeof(linked_stack));   //메모리 할당
    new->data = key;
    new->next = top;
    top = new;
}
int pop(){
    int key;
    linked_stack *temp;
    if( empty() ){ printf("Stack Underflow...\n"); exit(1); }
    else{ temp = top;
          key = top->data;
          top = top->next;
          free(temp);                                       //메모리 반납(회수)
          return(key);
        }
}
void scan_linked_stack(linked_stack *scan){
    int i;
    if( empty() ){ printf("Empty...\n"); return; }
    printf("\nStack pointer : %x", scan);
    printf("\nLinked Stack Data : ");
    while(scan){ printf("%d ", scan->data); scan=scan->next; } printf("\n");
}
void main(){
    int sw, data;
    while(1){
      printf("\nPush : 1, Pop : 2, 3 : 종료 => "); scanf("%d", &sw);
      switch(sw){
        case 1 : printf("\nPush data ==> "); scanf("%d", &data);
                 push(data); break;
        case 2 : printf("\nPop Data===>%c", pop()); break;
        default : printf("\n프로그램 종료.....\n"); exit(0);
      }
      scan_linked_stack(top);
    }
}
```

탐구

스택 Overflow 처리

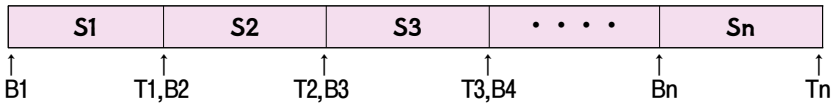
하나의 프로그램이 실행되기 위해서 여러 개의 스택이 동시에 필요할 수 있다.
이때, 스택 Overflow 발생을 최소화하면서 메모리 낭비를 억제하는 방안이다.

① 이중 스택 : 하나의 기억공간에서 2개의 스택을 운영한다.



Top1=Top2이면 overflow가 발생

② 다중 스택 : 하나의 기억공간에서 여러 개의 스택을 운영한다.



- 각 스택에 B와 T포인터를 둔다. 이들 위치는 모두 **유동적**이다.

- 스택이 **비어** 있으면 Ti == Bi
- 스택이 **가득** 차면 Ti == Bi+1이 된다.

- 임의 스택에 Overflow가 발생하면 빈 공간을 찾아 **리패킹(repacking)**을 한다.

- 리패킹 작업은 기억공간을 재분배하고, 스택의 자료를 옮기고, 포인터 값이 재조정된다.
- 리패킹 알고리즘으로 Knuth와 Jan Garwick's Repacking이 있다.

// 시스템 스택

```
void main( ) {
  f1( );
  f2( );
}
```

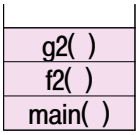
```
f1( ) {
  g1( );
   :
}
```

```
f2( ) {
  g2( );
   :
}
```

```
g1( ) {
   ⋮
   ⋮
}
```

```
g2( ) {
   ⋮
   ⋮
}
```

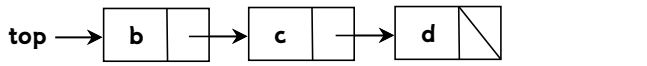
• 프로그램에서 호출하는 함수를 처리하기 위하여 시스템 스택을 사용한다.
• 위의 C 프로그램에서 함수 g2( )가 실행되는 시점의 Stack 상태는 다음과 같다.



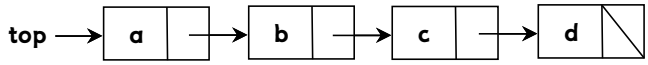
→ stack에는 함수의 현재 상태와 복귀주소 등이 기억된다.

기출문제 분석

1. 다음은 연결리스트를 이용하여 스택을 표현한 것이다. 이에 대한 설명으로 옳지 않은 것은? (단, push는 스택에 자료를 삽입하는 연산이고, pop은 스택에서 자료를 삭제하는 연산이다) [2014년 국가 7급]



(ㄱ) push 연산 수행 전 스택



(ㄴ) push 연산 수행 후 스택

---

① 스택에 가장 최근에 입력된 자료는 top이 지시한다.
② 스택에 입력된 자료 중 d가 가장 오래된 자료이다.
③ (ㄴ)에서 자료 c를 가져오려면 pop 연산이 2회 필요하다.
④ (ㄱ)에서 자료의 입력된 순서는 d, c, b이다.

♤ 연결리스트를 이용한 스택

• (ㄴ)에서 자료 c를 가져오려면 pop 연산이 2회 필요하다.(×)
→ (ㄴ)에서 자료 c를 가져오려면 pop 연산이 3회 필요하다.

정답 : ③

2. 다음 C 코드는 단순 연결리스트를 이용하여 스택의 pop() 함수를 구현한 것이다. (가), (나)에 들어갈 내용을 바르게 연결한 것은? [2022년 국가 7급]

```
struct stack {
  int data;
  struct stack* link;
};
struct stack* top = NULL;
int pop(){
  struct stack* temp = top;
  int item;
  if (!temp){ printf("Stack is empty.\n"); exit(1); }
  item = temp->data;
  [ (가) ] ;
  free(temp);
  [ (나) ] ;
}
```

	(가)	(나)
①	temp = temp->link	return item
②	temp = temp->link	return top->data
③	top = temp->link	return item
④	top = temp->link	return top->data

♤ 단순 연결리스트 - 연결 스택

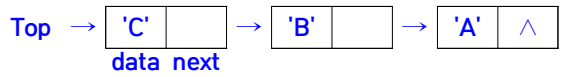
```
int pop(){
  struct stack* temp = top;
  int item;
  if (!temp){ printf("Stack is empty.\n"); exit(1); }
  item = temp->data;              // top이 가리키는 데이터 대입
  (가) top = temp->link ;         // top이 가리키는 노드 삭제
  free(temp);
  (나) return item ;              // top이 가리키는 데이터 반환
}
```

top → [200 |] → [100 | NULL]

top → [100 | NULL]

정답 : ③

3. 다음은 연결 스택을 구현한 것이다. 스택에 하나의 데이터('D')를 삽입(push)하려고 한다. 이를 위한 올바른 작업순서는? [2020년 군무 7급]



ㄱ. Get_node(P) (의미 : P 노드 생성)

ㄴ. Top = P

ㄷ. P->next = Top

ㄹ. P->data = 'D'

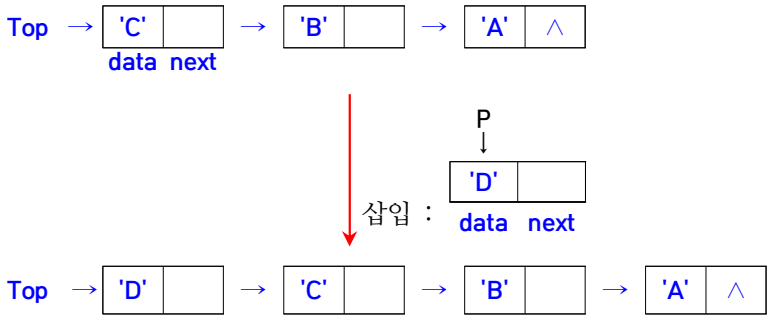
ㅁ. Top = P->next

---

① ㄱ,ㄴ,ㄷ,ㄹ ② ㄱ,ㄹ,ㄷ,ㄴ

③ ㄱ,ㄷ,ㄹ,ㅁ ④ ㄱ,ㄹ,ㄷ,ㅁ

♤ 연결 스택 – 데이터('D')를 삽입



ㄱ. Get_node(P) (의미 : P 노드 생성)

↓

ㄹ. P->data = 'D'

↓

ㄷ. P->next = Top

↓

ㄴ. Top = P

정답 : ②

4. 다음은 연결 스택(linked stack)에서 새로운 노드를 삽입하는 C 언어 프로그램이다. ㉠, ㉡에 들어갈 내용으로 옳게 짝지은 것은? (단, top은 스택의 첫 번째 노드를 가리키는 포인터이며, top이 NULL이면 공백 스택이다) [2017년 국가 7급]

```
typedef struct node {
    int key;
    struct node *link;
} StackNode;
StackNode *top = NULL;
void push(int data) {
  StackNode *tmp = (struct node *)malloc(sizeof(StackNode));
  if (tmp == NULL) {
    printf("Memory allocation is failed.\n");
    exit(1);
  }
  tmp→key = data;
  if (top == NULL) {
    tmp→link = NULL;
    top = tmp;
  }
  else {
    [          ㉠          ];
    [          ㉡          ];
  }
}
```

	㉠	㉡
①	tmp→link = top	top = tmp
②	top→link = tmp	tmp = top→link
③	top→link = tmp	tmp = top
④	tmp→link = top	top = tmp→link

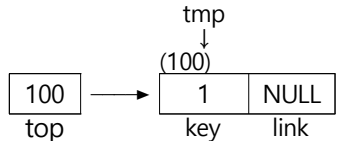
♤ **연결 스택(linked stack)**

- 연결 스택에 새로운 자료를 입력하는 알고리즘이다.
- 이런 유형의 문제는 연결 스택 그림을 그려서 풀면 실수하지 않을 것이다.

◈ 연결 스택 진행 과정 예.

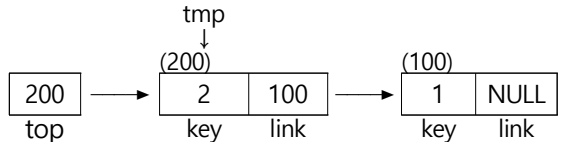
• 초기 상태 : top = NULL

• 키 1 입력 :



← top은 최근에 생성된 노드를 가리킨다.

• 키 2 입력 :



◈ 주어진 프로그램 분석

```
typedef struct node {
   int key;
   struct node *link;
} StackNode;
StackNode *top = NULL;
void push(int data) {          //스택에 자료 입력
   StackNode *tmp = (struct node*)malloc(sizeof(StackNode)); //힙할당
   if (tmp == NULL) {          //힙할당 실패
      printf("Memory allocation is failed.\n");
      exit(1);                 //프로그램 종료
   }
   tmp->key = data;            //새로 할당받은 노드의 key 필드에 키 저장
   if (top == NULL) {          //스택이 비어 있는 경우
      tmp->link = NULL;        //새로 할당받은 노드의 link 필드에 NULL 저장
      top = tmp;               //top이 새로 할당받은 노드를 가리킴
   }
   else {                      //공백 스택이 아닌 경우
      [㉠ tmp->link = top];    //새로 할당받은 노드의 link 필드에 top 값 대입
      [㉡ top = tmp];          //top이 새로 할당받은 노드를 가리킴
   }
}
```

정답 : ①

5. 다음은 연결리스트 스택의 삭제 알고리즘이다. ㉠~㉢에 들어갈 문장으로 옳게 짝지어진 것은? (단, 스택이 비었을 때 top은 NULL이고, top은 스택의 최상위 노드를 가리키는 포인터이다. 또한 스택이 빈 상태에서 맨 처음 삽입되는 노드의 next는 NULL이다) [2015년 국가 7급]

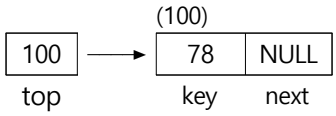
```
typedef struct node{
    int key;
    struct node *next;
} NODE;                 //스택의 노드구조
NODE *top = NULL;   //스택의 최상위 노드를 가리키는 포인터
int pop(void) {
    NODE *temp;
    int ret;
    if (    ㉠    ) {
        printf("\n Stack underflow.");
        return -1;
    }
    ret = top->key;
    (     ㉡     );
    (     ㉢     );
    free(temp);
    return ret;
}
```

	㉠	㉡	㉢
①	top->next == NULL	temp = top->next	top = top->next
②	top->next == NULL	temp = top	top->next = temp->next
③	top == NULL	temp = top	top = top->next
④	top == NULL	temp = top->next	top->next = temp->next

♤ **연결리스트 스택**

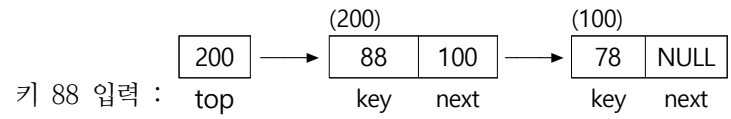
• 연결 스택구조는 다음과 같이 진행된다.

초기 상태 : top = NULL



키 78 입력 : ← top은 최근에 생성된 노드를 가리킨다.

10 http://cafe.daum.net/pass365(홍재연)



// 주어진 프로그램 분석

```
int pop(void)
{
    NODE *temp;             //NODE형 임시 포인터 temp
    int ret;                //반환값
    if(㉠ top == NULL){     //스택이 비어 있는 경우
       printf("\n Stack underflow.");
       return -1;
    }
    ret = top->key;         //ret에 top이 가리키는 노드의 key를 대입
    (㉡ temp = top);        //temp에 top을 대입
    (㉢ top = top->next);   //top에 top이 가리키는 노드의 next를 대입(노드 삭제)
    free(temp);             //삭제한 노드를 메모리 힙에서 제거
    return ret;             //삭제한 노드의 key를 반환
}
```

정답 : ③