

## 연결리스트, 스택, 큐 이론 및 실습

# 연결리스트(Linked List)

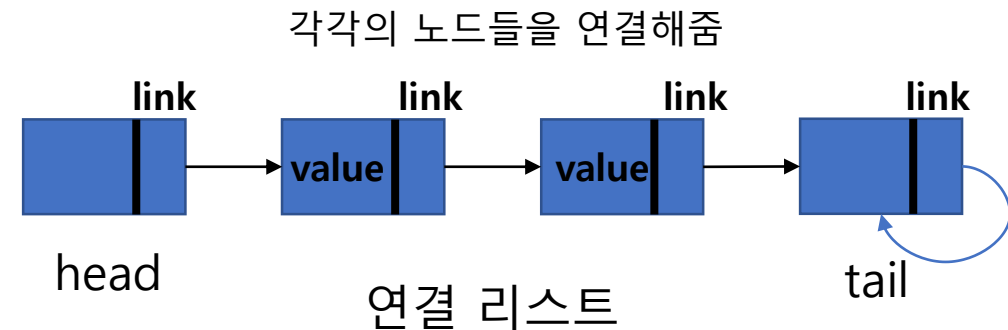
## 배열과 연결리스트의 차이점

- 배열은 정적인 자료 구조이지만, 연결리스트는 동적인 자료 구조이다.
- 배열은 메모리의 연속된 공간을 차지하는데 비해서 연결 리스트는 메모리의 연속된 공간을 차지하지 못한다.
- 자료의 순서를 바꾸어야 할 경우 배열의 경우에는 당기고 미는 등의 메모리 복사가 필요하지만, 연결 리스트의 경우에는 링크가 가리키는 방향만 바꾸어 주면 된다.

```
char a[5] = {'a' , 'b' , 'c' , 'd' , 'e'};
```

e
d
c
b
a

배열



# 연결리스트(Linked List)

## 연결리스트 구현 코드

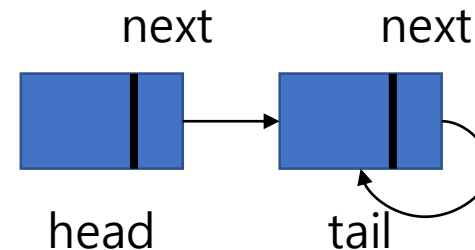
- 선언

```
typedef struct _node {  
    int key;  
    struct _node *next; // 다음 노드를 가리킬 구조체 포인터  
}node;
```

```
node *head, *tail; // 맨 처음 노드와 맨 마지막 노드(head 와 tail 노드에는 key 값을 넣지 않음)
```

- 초기화

```
void init_list(void)  
{  
    head = (node *)malloc(sizeof(node));  
    tail = (node *)malloc(sizeof(node));  
    head->next = tail;  
    tail->next = tail;  
}
```



# 연결리스트(Linked List)

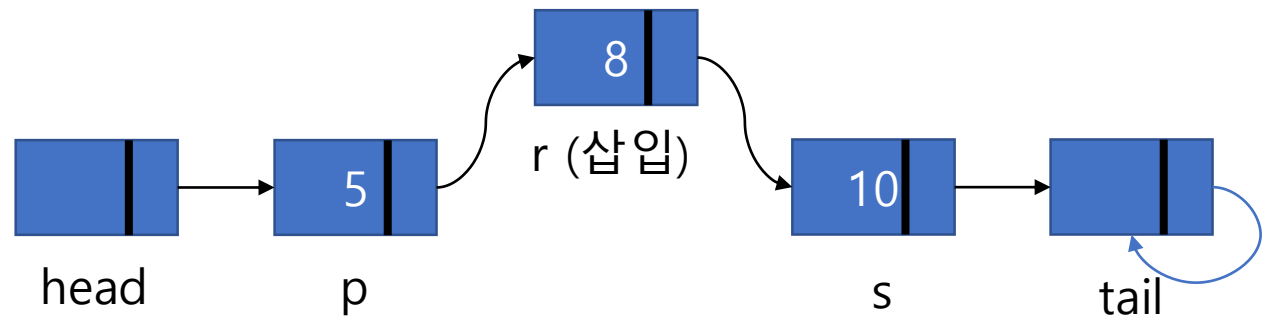
- 크기 순서대로 정렬해서 삽입

```
node *ordered_insert(int k)
{
    node *s, *p, *r;
    p = head;
    s = p->next;

    while (s->key <= k && s != tail)
    {
        p = p->next;
        s = p->next;
    }

    r = (node *)malloc(sizeof(node));
    r->key = k;
    p->next = r;
    r->next = s;

    return r;
}
```

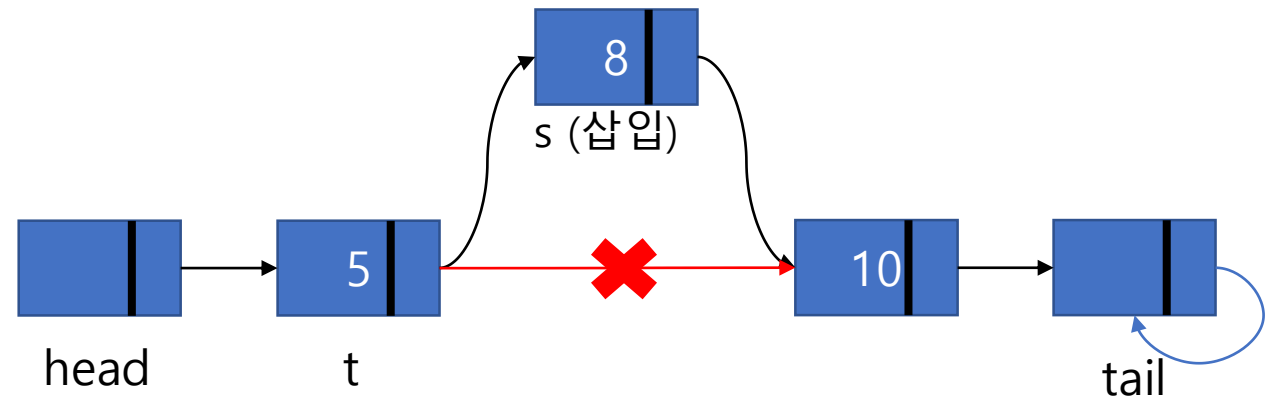


# 연결리스트(Linked List)

- t 노드 다음에 삽입

```
node *insert_after(int k, node *t)
{
    node *s;
    s = (node *)malloc(sizeof(node));
    s->key = k;
    s->next = t->next;
    t->next = s;

    return s;
}
```



# 연결리스트(Linked List)

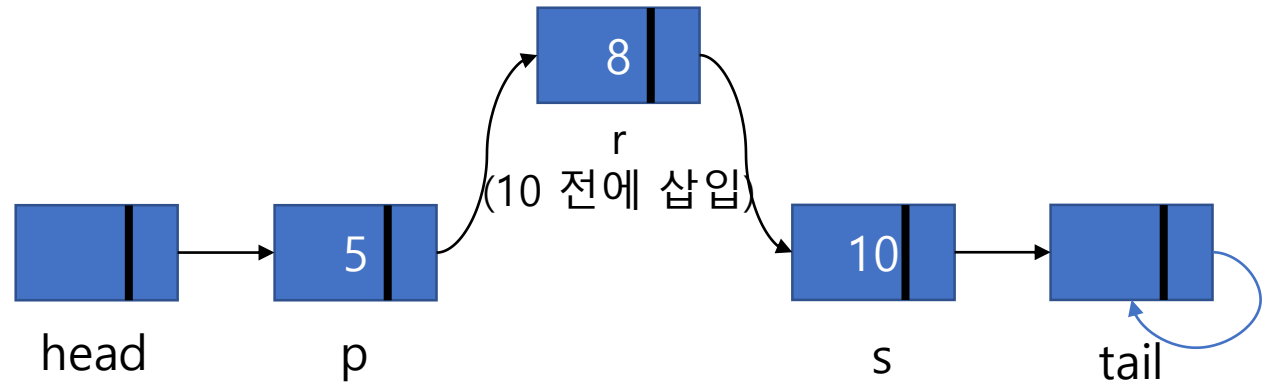
- k 전에 t 삽입

```
node *insert_node(int t, int k) // insert t before k
{
    node *s, *p, *r;
    p = head;
    s = p->next;

    while (s->key != k && s != tail)
    {
        p = p->next;
        s = p->next;
    }

    if (s != tail)
    {
        r = (node *)malloc(sizeof(node));
        r->key = t;
        p->next = r;
        r->next = s;
    }

    return p->next;
}
```



# 연결리스트(Linked List)

- 모든 노드 값 출력

```
void print_list(node *t)
{
    printf("Wn");
    while (t != tail)
    {
        printf("%-8d", t->key);
        t = t->next;
    }
}
```

/\* 호출 할 때는 print\_list(head->next);  
이렇게 호출 하면 head->next 부터 tail 전까지 모든 노드 값들이  
출력됨. \*/

# 연결리스트(Linked List)

- k 값을 가지고 있는 노드 찾기

```
node *find_node(int k)
{
    node *s;
    s = head->next;

    while (s->key != k && s != tail)
        s = s->next;

    return s;
}

/* head->next 노드 부터 시작해서 k값을 찾을 때까지 계속
next로 이동함. k값의 노드 주소를 반환함. */
```



# 연결리스트(Linked List)

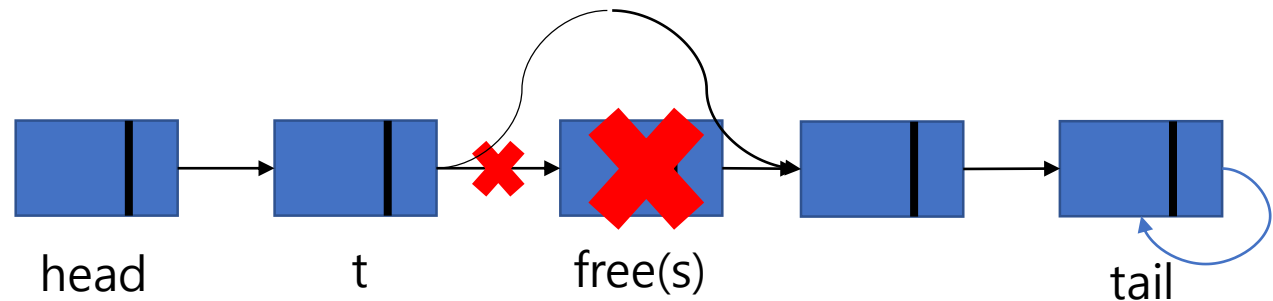
- t 노드의 다음 노드 삭제

```
int delete_next(node *t)
{
    node *s;

    if (t->next == tail) // t 노드의 다음이 tail일 경우 삭제하지 않음
        return 0;

    s = t->next;
    t->next = t->next->next;
    free(s);

    return 1;
}
```



# 연결리스트(Linked List)

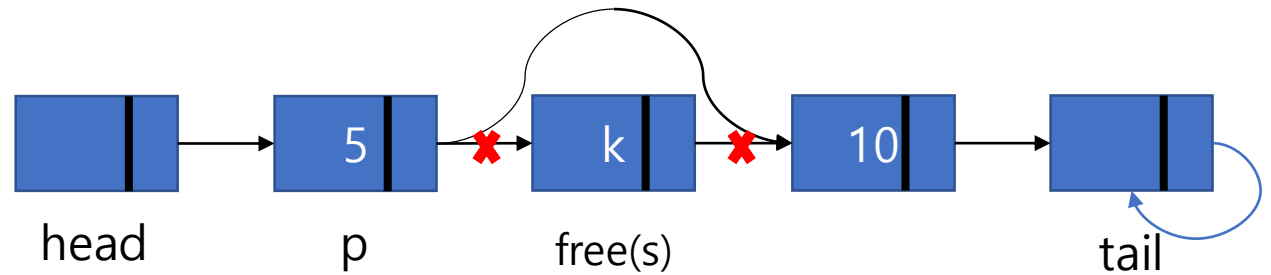
- k 값을 갖는 노드 삭제

```
int delete_node(int k)
{
    node *s, *p;

    p = head;
    s = p->next;

    while (s->key != k && s != tail)
    {
        p = p->next;
        s = p->next;
    }

    if (s != tail)
    {
        p->next = s->next;
        free(s);
        return 1;
    }
    else
        return 0;
}
```

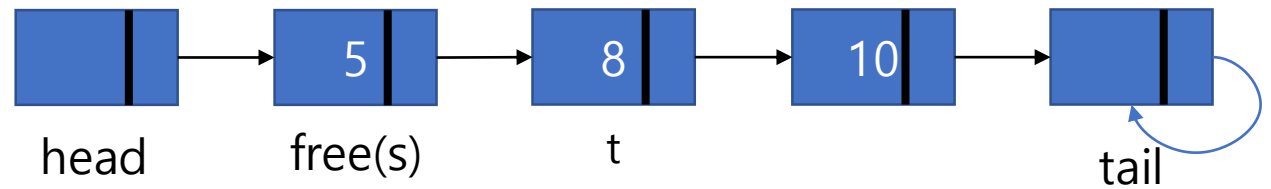


# 연결리스트(Linked List)

## - 모든 노드 삭제

```
node *delete_all(void)
{
    node *s, *t;
    t = head->next;

    while (t != tail)
    {
        s = t;
        t = t->next;
        free(s);
    }
    head->next = tail;
    return head;
}
```



# 연결리스트(Linked List)

```
- main 문 {
    int main()
    node *t;
```


```
    init_list();
    ordered_insert(10);
    ordered_insert(5);
    ordered_insert(8);
    ordered_insert(3);
    ordered_insert(1);
    ordered_insert(7);
    ordered_insert(8);
```

```
    printf("Initial Linked list is ");
    print_list(head->next);
```

```
    printf("Finding 4 is %ssuccessful", find_node(4) == tail ? "un" : "");
```

```
    t = find_node(5);
    printf("Finding 5 is %ssuccessful", t == tail ? "un" : "");
```

```
    printf("Inserting 9 after 5");
    insert_after(9, t);
    print_list(head->next);
```

 C:\Users\Owner\source\repos\data\_str\Debug\data\_str.exe

```
Initial Linked list is
1      3      5      7      8      8      10
Finding 4 is unsuccessful
Finding 5 is successful
Inserting 9 after 5
1      3      5      9      7      8      8      10
```

# 연결리스트(Linked List)

```

t = find_node(10);
printf("WnDeleting next last node");
delete_next(t);
print_list(head->next);

t = find_node(3);
printf("WnDeleting next 3");
delete_next(t);
print_list(head->next);

printf("WnInsert node 2 before 3");
insert_node(2,3);
print_list(head->next);

printf("WnDeleting node 2");

if (!delete_node(2))
printf("Wn deleting 2 is unsuccessful");

print_list(head->next);

```

```

printf("WnDeleting node 1");
delete_node(1);
print_list(head->next);

printf("WnDeleting all node");
delete_all();
print_list(head->next);

system("pause");
return 0;

```

}

```

Deleting next last node
1      3      5      9      7      8      8      10
Deleting next 3
1      3      9      7      8      8      10
Insert node 2 before 3
1      2      3      9      7      8      8      10
Deleting node 2
1      3      9      7      8      8      10
Deleting node 1
3      9      7      8      8      10
Deleting all node

```

# 연결리스트(Linked List)

## - 결과 화면

C:\Users\Owner\source\repos\data\_str\Debug\data\_str.exe

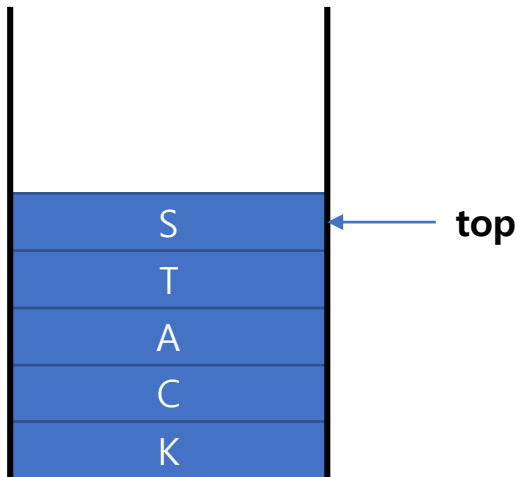
```
Initial Linked list is
1 3 5 7 8 8 10
Finding 4 is unsuccessful
Finding 5 is successful
Inserting 9 after 5
1 3 5 9 7 8 8 10
Deleting next last node
1 3 5 9 7 8 8 10
Deleting next 3
1 3 9 7 8 8 10
Insert node 2 before 3
1 2 3 9 7 8 8 10
Deleting node 2
1 3 9 7 8 8 10
Deleting node 1
3 9 7 8 8 10
Deleting all node
계속하려면 아무 키나 누르십시오 . . .
```

## - 소스 코드

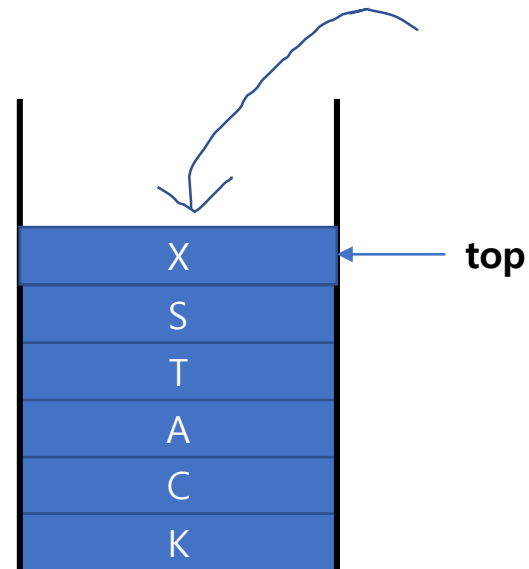
[https://github.com/ahnsangjae/MY\\_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Linkedlist](https://github.com/ahnsangjae/MY_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Linkedlist)

## 스택의 개념

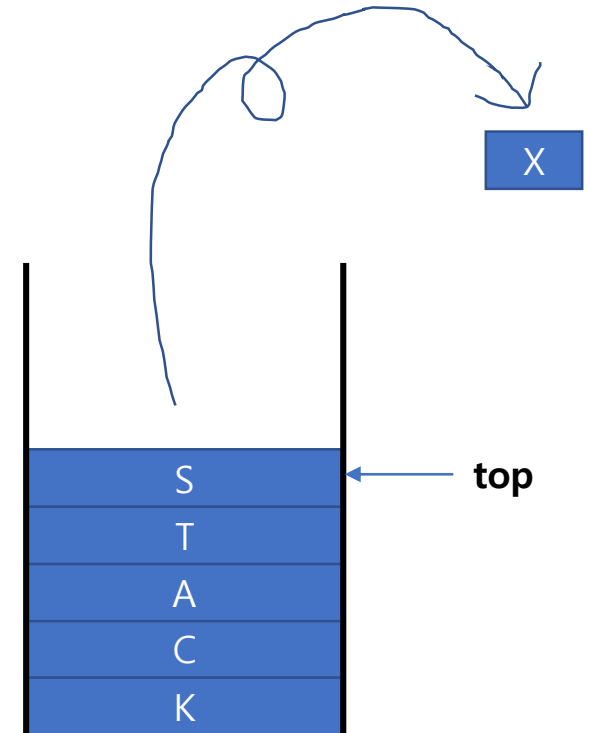
- 밑이 막힌 긴 통(무언가를 넣는 곳과 빼내는 곳이 같음)
- 입, 출구가 같기 때문에 먼저 들어간 것은 밑에 있게 되고 나중에 들어간 것이 위에 있다.
- 결국 제일 나중에 들어간 것이 제일 먼저 나오게 된다. LIFO(Last In First Out) 구조.
- 스택의 조작 방법 (push, pop 동작)



스택의 모양



X를 push



X를 pop

## 배열로 스택 구현하기!

- 스택 배열, top 변수 및 초기화

```
#define MAX 10
```

```
int stack[MAX];
```

```
int top;
```

```
void init_stack(void)
```

```
{
```

```
    top = -1; // 스택이 비어 있을 때는 top = -1
```

```
}
```



# 스택(Stack)

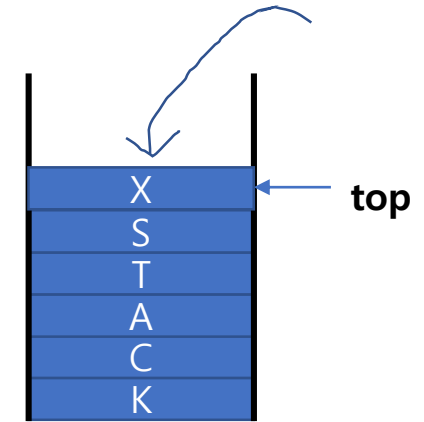
## - push, pop 함수 구현

```
int push(int t)
{
    if (top >= MAX - 1)
    {
        printf("Stack overflow.");
        return -1;
    }

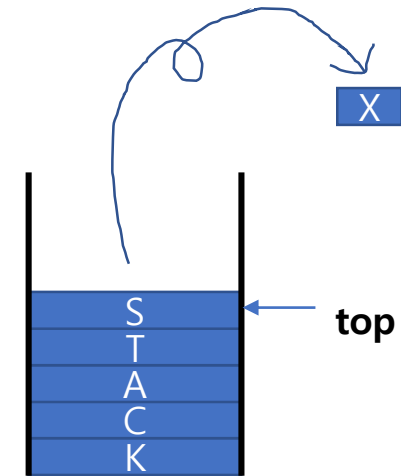
    stack[++top] = t;    // 전위 연산자
    return t;
}

int pop(void)
{
    if (top < 0)
    {
        printf("Stack underflow.");
        return -1;
    }

    return stack[top--];    // 후위 연산자
}
```



X를 push



X를 pop

```
void print_stack(void)
{
    int i;
    printf("Wn stack contents : Top ----> BottomWn");

    for (i = top; i >= 0; i--)    // top 부터 0번째(Bottom) 까지 순서대로 출력
    {
        printf("%-6d", stack[i]);
    }
}
```

## - main 문 소스코드

```
int main()
{
    int i;
    init_stack();

    printf("Push 5, 4, 7, 8, 2, 1");
    push(5);
    push(4);
    push(7);
    push(8);
    push(2);
    push(1);
    print_stack();

    printf("WnWnPop");
    i = pop();
    print_stack();
    printf("Wn popping value is %dWn", i);
```

[https://github.com/ahnsangjae/MY\\_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Stack\\_array](https://github.com/ahnsangjae/MY_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Stack_array)

 C:\Users\Owner\source\repos\data\_str\Debug\data\_st

```
Push 5, 4, 7, 8, 2, 1
stack contents : Top ----> Bottom
1      2      8      7      4      5

Pop
stack contents : Top ----> Bottom
2      8      7      4      5
popping value is 1
```

# 스택(Stack)

```

printf("\nPush 3, 2, 5, 7, 2");
push(3);
push(2);
push(5);
push(7);
push(2);
print_stack();

printf("\n\nNow stack is full, push 3");
push(3);
print_stack();

printf("\n\nInitialize stack");
init_stack();
print_stack();

printf("\n\nNow stack is empty, pop");
i = pop();
print_stack();
printf("\n popping value is %d\n", i);

system("pause");
return 0;
}

```

```

Push 3, 2, 5, 7, 2
stack contents : Top ----> Bottom
2      7      5      2      3      2      8      7      4      5

Now stack is full, push 3
stack overflow.
stack contents : Top ----> Bottom
2      7      5      2      3      2      8      7      4      5

Initialize stack
stack contents : Top ----> Bottom

Now stack is empty, pop
stack underflow.
stack contents : Top ----> Bottom

popping value is -1
계속하려면 아무 키나 누르십시오 . . .

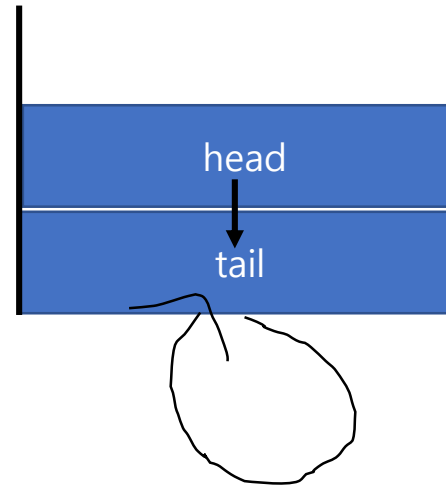
```

## 연결리스트로 스택 구현하기!

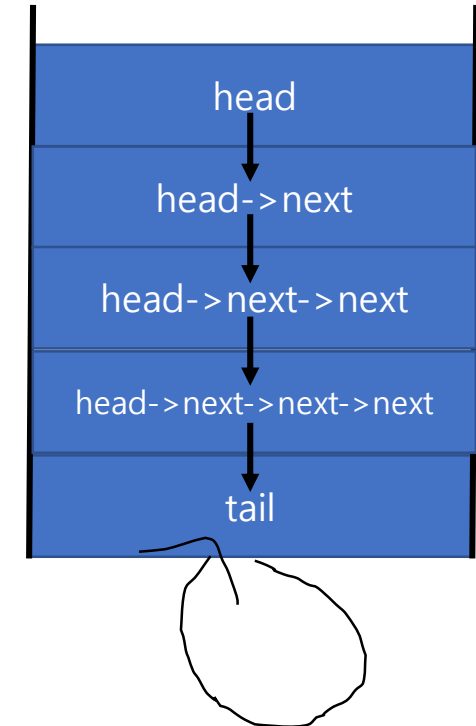
```
typedef struct _node{
    int key;
    struct _node *next;
}node;

node *head, *tail;

void init_stack(void)
{
    head = (node *)malloc(sizeof(node));
    tail = (node *)malloc(sizeof(node));
    head->next = tail;
    tail->next = tail;
}
```



스택의 초기 상태

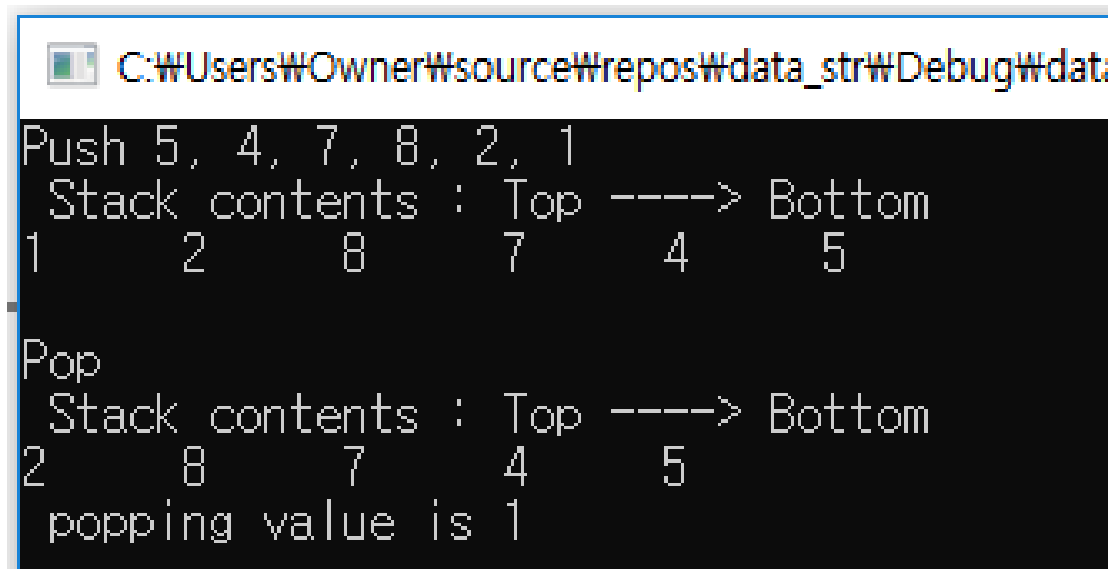


연결리스트로 구현한  
일반적인 스택의 모습

```
int main()
{
    int i;
    init_stack();

    printf("Push 5, 4, 7, 8, 2, 1");
    push(5);
    push(4);
    push(7);
    push(8);
    push(2);
    push(1);
    print_stack();

    printf("\n\nPop");
    i = pop();
    print_stack();
    printf("\n popping value is %d", i);
}
```



```
C:\Users\Owner\source\repos\data_str\Debug\data_str.exe
Push 5, 4, 7, 8, 2, 1
Stack contents : Top ----> Bottom
1      2      8      7      4      5

Pop
Stack contents : Top ----> Bottom
2      8      7      4      5
popping value is 1
```

```

printf("WnWnPush 3, 2, 5, 7, 2");
push(3);
push(2);
push(5);
push(7);
push(2);
print_stack();

printf("WnWnPush 3");
push(3);
print_stack();

printf("WnWnInitialize stack");
clear_stack();
print_stack();

printf("WnWnNow stack is empty, pop");
i = pop();
print_stack();
printf("Wn popping value is %dWn", i);

system("pause");
return 0;
}

```

```

Push 3, 2, 5, 7, 2
Stack contents : Top ----> Bottom
2      7      5      2      3      2      8      7      4      5

Push 3
Stack contents : Top ----> Bottom
3      2      7      5      2      3      2      8      7      4      5

Initialize stack
Stack contents : Top ----> Bottom

Now stack is empty, pop
Stack underflow.
Stack contents : Top ----> Bottom

popping value is -1
계속하려면 아무 키나 누르십시오 . . .

```

## \* 코딩 실습

### - 연결리스트로 스택 구현하기에서

**main 문에서 호출한 push(), pop(), print\_stack(), clear\_stack() 함수 정의부 직접 작성하기!**

clear\_stack() : 스택 안의 모든 노드를 삭제하고 스택을 초기화(head, tail만 남은 상태)하는 함수



## ※ 힌트!!!

```

void push(int k)
{
    node *t;

    if((t = (node *)malloc(sizeof(node))) == NULL)
    {
        printf("Wn out of memory...");
    }
}

```

push할 값  
push할 node 변수

채우기!

// push할 노드의 위치는 head-&gt;next 임!

```

int pop(void)
{
    node *t;
    int i;

    if(head->next == tail)
    {
        printf("Wn Stack underflow");
        return -1;
    }
}

```

pop할 노드를 해제하기 위한 변수  
pop할 값을 받기 위한 변수

채우기!

```

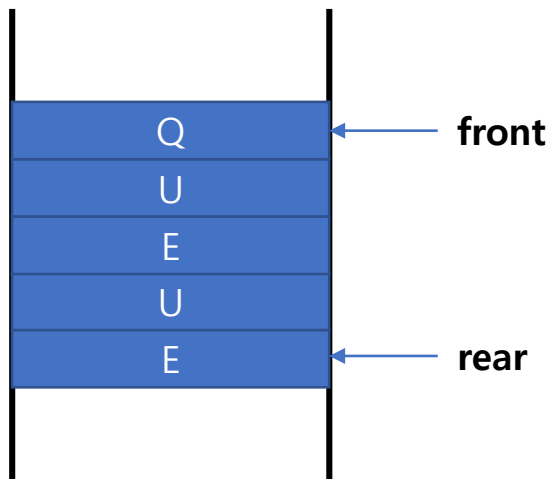
return i;
}

```

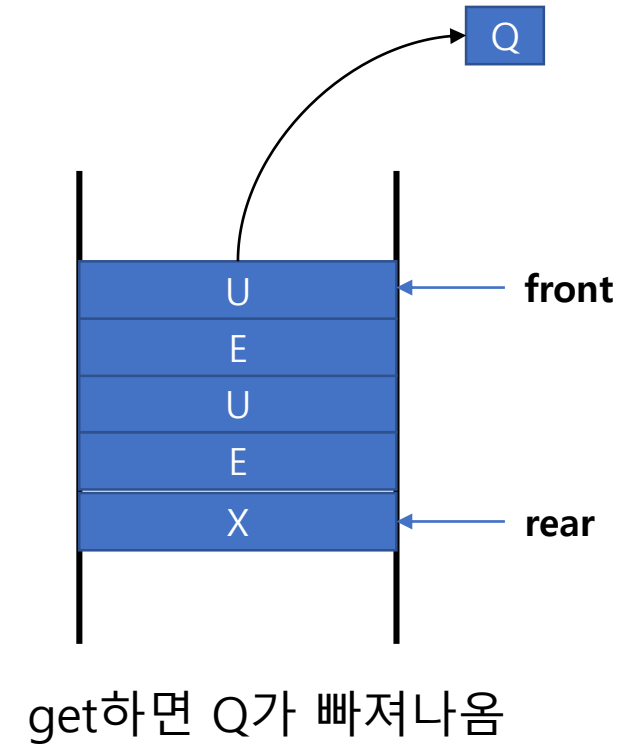
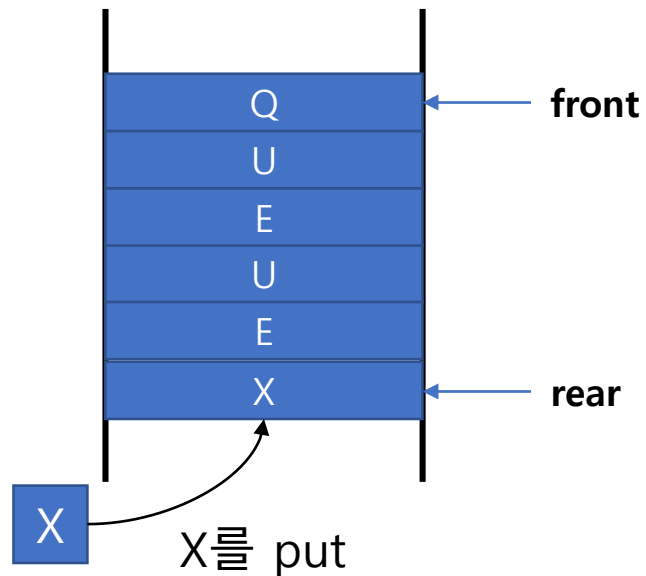
// pop할 노드는 head-&gt;next임!

## 큐의 개념

- 앞 뒤가 뚫린 긴 통
- 뒤에서 뭔가를 집어넣고 앞에서 그것을 빼냄 (입구와 출구가 다름)
- 먼저 들어온 것이 먼저 나오는 FIFO(First In First Out) 구조.
- 큐의 조작 방법 (put, get 동작)



큐의 모양



## 배열로 큐 구현하기!

### - 큐 배열 초기화

```
#define MAX 10

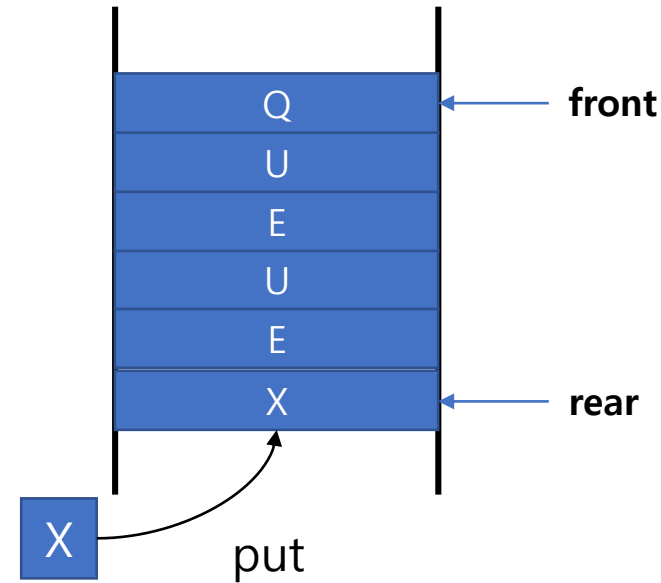
int queue[MAX];
int front, rear;

void init_queue(void)
{
    front = rear = 0;
}

/* front 와 rear가 0에서 출발함.
put 되면 rear가 늘어나고, get 되면 front가 늘어남 */
```

```
int put(int k)
{
    if ((rear + 1) % MAX == front)
    {
        printf("Wn Queue overflow.");
        return -1;
    }

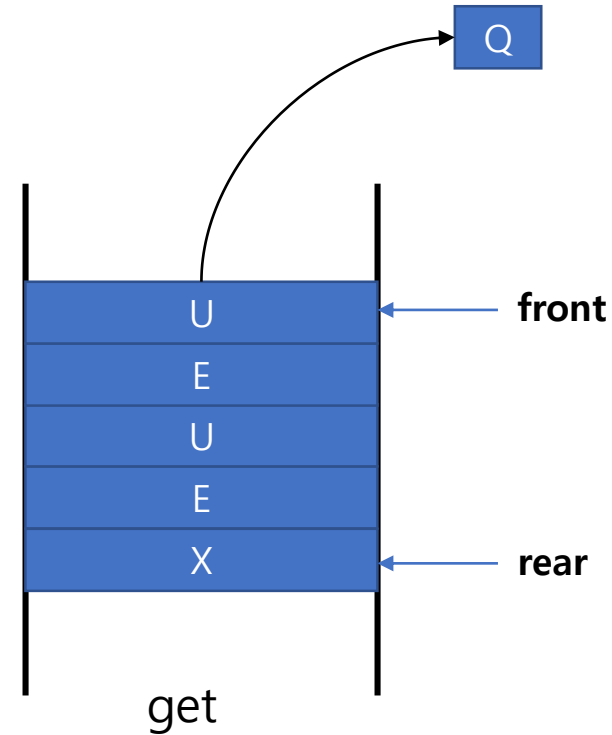
    queue[rear] = k;
    rear = ++rear % MAX;
    return k;
}
```



```
int get(void)
{
    int i;

    if (front == rear)
    {
        printf("Queue underflow.");
        return -1;
    }

    i = queue[front];
    front = ++front % MAX;
    return i;
}
```



```
void clear_queue(void)
{
    front = rear;
}
```

/\* front=rear가 되면 front와 rear 사이에 아무 것도 없어지기 때문에  
큐 안의 모든 값을 제거함 \*/

```
void print_queue(void)
{
    int i;
    printf("\n Queue contentx : Front ----> Rear\n");

    for (i=front; i != rear; i = ++i % MAX)
    {
        printf("%-6d", queue[i]);
    }
}
```

// front 에서 rear 까지 순서대로 출력

```
void main()
{
    int i;
    init_queue();

    printf("Put 5, 4, 7, 8, 2, 1");
    put(5);
    put(4);
    put(7);
    put(8);
    put(2);
    put(1);
    print_queue();

    printf("\n\nGet");
    i = get();
    print_queue();
    printf("\n getting value is %d", i);
}
```

[https://github.com/ahnsangjae/MY\\_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Queue\\_array](https://github.com/ahnsangjae/MY_Workspace/tree/master/%EC%9E%90%EB%A3%8C%EA%B5%AC%EC%A1%B0/Queue_array)

 C:\Users\Owner\source\repos\data\_str\Debug\data\_str.e

```
Put 5, 4, 7, 8, 2, 1
Queue contentx : Front ----> Rear
5      4      7      8      2      1

Get
Queue contentx : Front ----> Rear
4      7      8      2      1
getting value is 5
```

```
printf("WnWnPut 3, 2, 5, 7");
put(3);
put(2);
put(5);
put(7);
print_queue();

printf("WnWnNow queue is full, put 3");
put(3);
print_queue();

printf("WnWnInitialize queue");
clear_queue();
print_queue();

printf("WnWnNow queue is empty, get");
i = get();
print_queue();
printf("Wn getting value is %d", i);

system("pause");
}
```

```
Put 3, 2, 5, 7
Queue contentx : Front ----> Rear
4      7      8      2      1      3      2      5      7

Now queue is full, put 3
Queue overflow.
Queue contentx : Front ----> Rear
4      7      8      2      1      3      2      5      7

Initialize queue
Queue contentx : Front ----> Rear

Now queue is empty, get
Queue underflow.
Queue contentx : Front ----> Rear

getting value is -1계속하려면 아무 키나 누르십시오 . . .
```



## 연결리스트로 큐 구현하기!

## - 큐 초기화

```

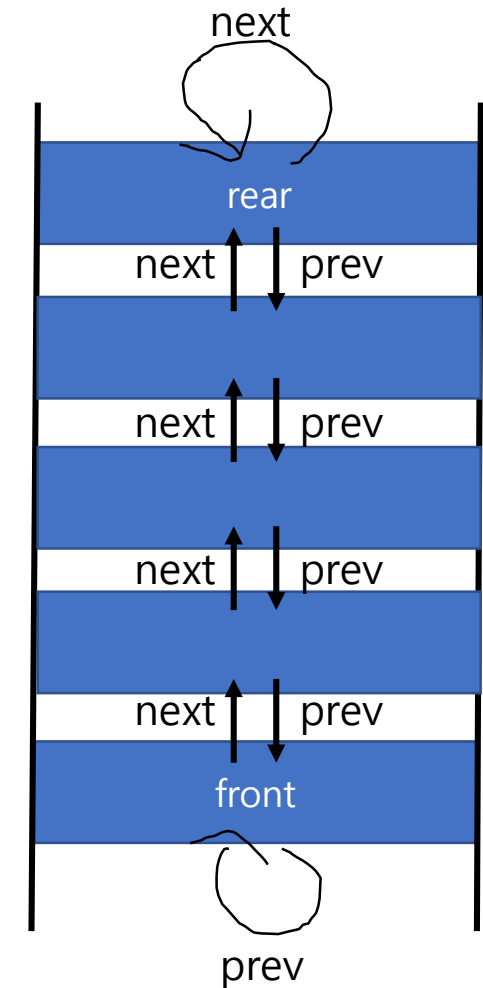
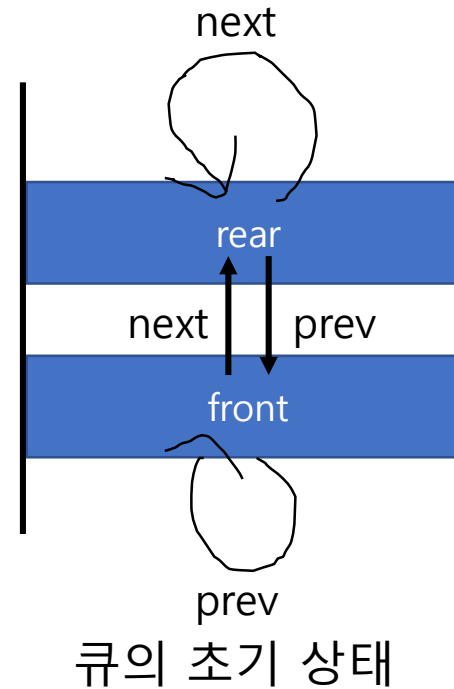
typedef struct _dnode{
    int key;
    struct _dnode *prev;
    struct _dnode *next;
}dnode;

dnode *front, *rear;

void init_queue(void)
{
    front = (dnode *)malloc(sizeof(dnode));
    rear = (dnode *)malloc(sizeof(dnode));
    front->prev = front;
    front->next = rear;
    rear->prev = front;
    rear->next = rear;
}

```

// put할 때 tail 바로 뒤를 쉽게 찾기 위해 prev 사용



연결리스트로 구현한 일반적인 큐의 모습

```
void main()
{
    int i;
    init_queue();

    printf("Put 5, 4, 7, 8, 2, 1");
    put(5);
    put(4);
    put(7);
    put(8);
    put(2);
    put(1);
    print_queue();

    printf("\n\nGet");
    i = get();
    print_queue();
    printf("\ngetting value is %d", i);
}
```

C:\Users\Owner\source\repos\data\_str\Debug

```
Put 5, 4, 7, 8, 2, 1
Queue contents : Front ----> Rear
5      4      7      8      2      1

Get
Queue contents : Front ----> Rear
4      7      8      2      1
getting value is 5
```

```

printf("WnWnPut 3, 2, 5, 7");
put(3);
put(2);
put(5);
put(7);
print_queue();

printf("WnWnPut 3");
put(3);
print_queue();

printf("WnWnInitialize queue");
clear_queue();
print_queue();

printf("WnWnNow queue is empty, get");
i = get();
print_queue();
printf("WnWngetting value is %dWn", i);

system("pause");
}

```

```

Put 3, 2, 5, 7
Queue contents : Front ----> Rear
4      7      8      2      1      3      2      5      7

Put 3
Queue contents : Front ----> Rear
4      7      8      2      1      3      2      5      7      3

Initialize queue
Queue contents : Front ----> Rear

Now queue is empty, get
Queue underflow.
Queue contents : Front ----> Rear

getting value is -1
계속하려면 아무 키나 누르십시오 . . .

```

## \* 코딩 실습

- 연결리스트로 큐 구현하기에서

main 문에서 호출한 `put()`, `get()`, `print_queue()`, `clear_queue()` 함수 정의부 직접 작성하기!