

## 6 주차 코벤져스's PICK

### 미션 1

13-6

```
#include <stdio.h>
#include <stdlib.h>

typedef struct stack
{
    int top;
    int capacity;
    int* array;
} Stack;

Stack* createStack(int capacity)
{
    Stack* stack = (Stack*)malloc(sizeof(Stack));
    stack -> capacity = capacity;
    stack -> top = -1;
    stack -> array = (int *)malloc(stack -> capacity*sizeof(int));
    return stack;
}

int isFull(Stack* stack)
{
    return stack -> top == stack -> capacity - 1;
}

int isEmpty(Stack* stack)
{
    return stack -> top == -1;
}

void push(Stack* stack, int item)
{
    if(isFull(stack))
        return;
    stack -> array[++stack -> top] = item;
    printf("%d pushed to stack\n", item);
}
```

```

// 수정한 부분
int pop(Stack* stack)
{
    if (isEmpty(stack))
    {
        printf("Wa"); //비어 있을 때 경고음 내기
        return -9999;
    }
    int temp = stack -> array[stack->top--];
    return temp;
}

// 수정한 부분
int peek(Stack* stack)
{
    if (isEmpty(stack))
        return -9999;
    return stack -> array[stack -> top];
}

int main()
{
    Stack* stack = createStack(100);

    push(stack, 10);
    push(stack, 20);
    push(stack, 30);
    push(stack, 40);

    printf("%d pop from stack\n", pop(stack));
    printf("%d pop from stack\n", pop(stack));

    push(stack, 50);
    printf("%d pop from stack\n", pop(stack));
    printf("%d pop from stack\n", pop(stack));
    printf("%d pop from stack\n", pop(stack));
    printf("%d pop from stack\n", pop(stack));

    //메모리 해제
    free(stack -> array);
    return 0;
}
[출력]

```

```
10 pushed to stack
20 pushed to stack
30 pushed to stack
40 pushed to stack
40 pop from stack
30 pop from stack
50 pushed to stack
50 pop from stack
20 pop from stack
10 pop from stack
-9999 pop from stack
$
```

## 미션 2

6-8

```
#include <stdio.h>
#include <stdlib.h>

typedef struct stackNode
{
    int data;
    struct stackNode *next;
} StackNode;

StackNode *createStackNode(int data)
{
    StackNode *node = (StackNode *)malloc(sizeof(StackNode));
    node->data = data;
    node->next = NULL;
    return node;
}

int isEmpty(StackNode *root)
{
    return !root;
}

void push(StackNode **root, int data)
{
    StackNode *node = createStackNode(data); // 새로운 노드를 만든다.
    node->next = *root;                      // 기존의 root 를 next 에 연결한다.
    *root = node;                           // root 를 새로운 노드로 변경한다.

    printf("%d pushed to stack\n", data);
}

int pop(StackNode **root)
{
    if (isEmpty(*root))
    {
        return -9999;
    }

    int popped;
```

```

StackNode *temp = *root; // root 노드의 주소를 temp 에 저장한다.
popped = temp->data;      // root 노드의 data 를 popped 변수에 저장한다.
*root = temp->next;       // root 노드를 기존 root 의 next 노드로 변경한다.
free(temp);              // 이전 root 노드(temp)를 제거한다.

return popped;
}

int peek(StackNode **root)
{
    if (isEmpty(*root))
        return -9999;
    return (*root)->data;
}

int main()
{
    StackNode *root = NULL;

    push(&root, 10);
    push(&root, 20);
    push(&root, 30);
    push(&root, 40);

    printf("%d pop from stack\n", pop(&root));
    printf("%d pop from stack\n", pop(&root));

    push(&root, 50);
    printf("%d peeked from stack\n", peek(&root));
    printf("%d pop from stack\n", pop(&root));
    printf("%d pop from stack\n", pop(&root));
    printf("%d pop from stack\n", pop(&root));
    printf("%d pop from stack\n", pop(&root));
    return 0;
}

```

우선 push 함수에서는 createStackNode 를 이용해 새로운 노드를 만들고,

새로운 노드의 next 에 root 노드를 연결한 다음 root 를 새로운 노드로 변경해줬습니다.

root 는 stack 의 가장 상단 노드를 의미합니다.

stack에서는 가장 상단의 노드부터 꺼내야 하기 때문에 root를 꺼내야 할 노드인 최상단 노드로 설정했습니다.

pop을 할 때는 root 노드를 꺼내고, root 노드의 next 노드를 root로 설정했습니다.

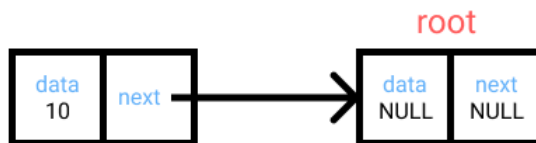
글로 설명하기에는 너무 복잡한 것 같아 그림으로 표현해봤습니다!

## 1. PUSH

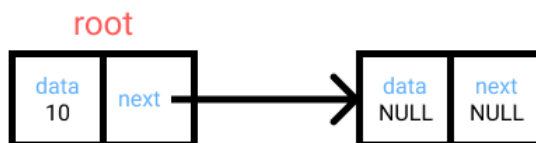
1. 새로운 노드를 만든다.



2. 기존의 root를 next에 연결한다.

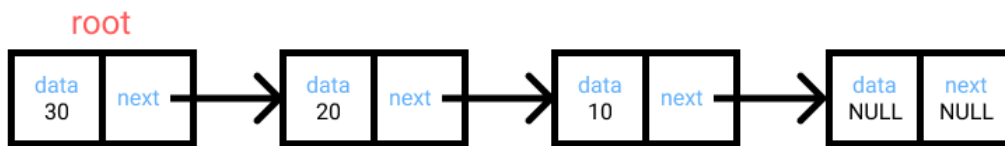


3. root를 새로운 노드로 변경한다.

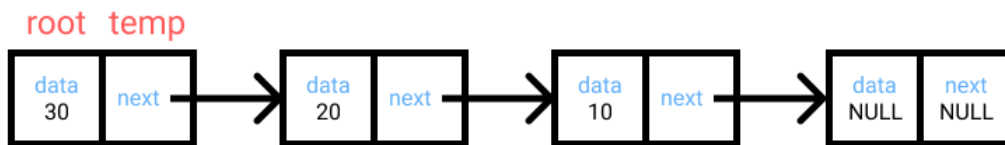


## 2. POP

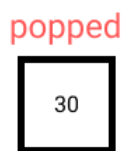
10, 20, 30을 차례대로 저장한 모습



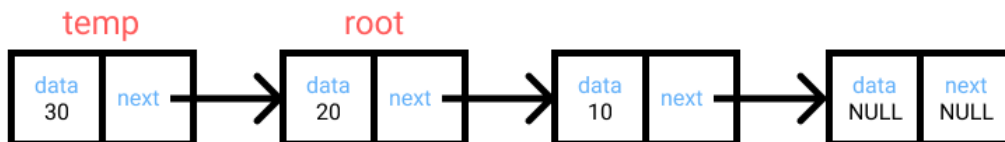
1.root 노드의 주소를 temp에 저장한다.



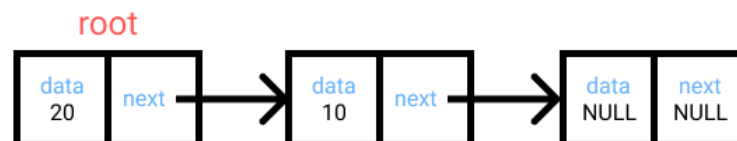
2. root 노드의 data를 popped 변수에 저장한다.



3. root 노드를 기존 root의 next 노드로 변경한다.



4. 이전 root 노드 (temp)를 제거한다.



### 미션 3

2-9

```
void enqueue(Queue *queue, int item)
{
    if (isFull(queue))
    {
        return;
    }
    queue->rear++;
    queue->rear %= (queue->capacity);
    queue->array[queue->rear] = item;
    queue->size++;
    printf("%d enqueued to queue\n", item);
}

int dequeue(Queue *queue)
{
    if (isEmpty(queue))
    {
        return -9999;
    }
    int item = 0;
    item = queue->array[queue->front++];
    queue->front %= (queue->capacity);
    queue->size--;
    return item;
}
```

double-linked list 로 구현한 Queue 입니다.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct node
{
    int val;
    struct node *prev;
    struct node *next;
} Node;

typedef struct queue
```



```

{
    Node *front;
    Node *rear;
    int size;
} Queue;

void queue_init(Queue *queue);
void enqueue(Queue *queue, int val);
int dequeue(Queue *queue);

int main(void)
{
    Queue* queue = (Queue *)malloc(sizeof(Queue));

    queue_init(queue);

    enqueue(queue, 10);
    enqueue(queue, 20);
    enqueue(queue, 30);
    enqueue(queue, 40);

    printf("%d dequeued from queue\n\n", dequeue(queue));
    printf("%d dequeued from queue\n\n", dequeue(queue));
    printf("%d dequeued from queue\n\n", dequeue(queue));
    printf("%d dequeued from queue\n\n", dequeue(queue));
    printf("%d dequeued from queue\n\n", dequeue(queue));
}

void queue_init(Queue *queue)
{
    queue->front = (Node *)malloc(sizeof(Node));
    queue->rear = (Node *)malloc(sizeof(Node));

    queue->front->val = -1;
    queue->front->prev = NULL;
    queue->front->next = queue->rear;

    queue->rear->val = -1;
    queue->rear->prev = queue->front;
    queue->rear->next = NULL;

    queue->size = 0;
}

```

```

void enqueue(Queue *queue, int val)
{
    Node *new_node = (Node *)malloc(sizeof(Node));
    Node *last_node = queue->rear->prev;

    new_node->val = val;
    new_node->prev = last_node;
    new_node->next = queue->rear;

    last_node->next = new_node;
    queue->rear->prev = new_node;

    queue->size++;
    printf("%d enqueued to queue\n", val);
}

```

```

int dequeue(Queue *queue)
{
    if (queue->size == 0) return -9999;

    Node *target_node = queue->front->next;
    Node *next_front_node = target_node->next;
    int val = target_node->val;

    queue->front->next = next_front_node;
    next_front_node->prev = queue->front;

    free(target_node);
    queue->size--;
    return val;
}

```

## 미션 4

8-4

```
typedef struct node{
    int data;
    struct node* next;
} Node;

void append(Node* head, int data) {
    // 이 부분을 작성해 주세요!
}

int getLastNode (Node* head, int k) {
    // 이 부분을 작성해 주세요!
}

void printList(Node* head) {
    // 이 부분을 작성해 주세요!
}

int main() {
    Node* head = (Node*)malloc(sizeof(Node));
    append(head, 9);
    append(head, 8);
    append(head, 4);
    append(head, 14);
    append(head, 5);

    printList(head);

    printf("Wn%dth last node is %dWn", 2, getLastNode(head, 2));
}
```

- 최종 코드 및 실행 결과

< 최종 코드 >

```
#include <stdio.h>
#include <stdlib.h>

// 노드 구조체
typedef struct node {
    int data;
    struct node* next;
```

```
} Node;
```

```
// 노드 삽입 함수
```

```
void append(Node* head, int data) {
```

```
    // 데이터를 넣을 노드를 생성
```

```
    Node *new_node = (Node*)malloc(sizeof(Node));
```

```
    new_node->data = data;
```

```
    new_node->next = NULL;
```

```
    // NULL 값을 가리키는 노드를 찾기 위한 curr 포인터
```

```
    Node *curr = head->next;
```

```
    // curr 이 가리키는 노드의 이전 노드를 가리키는 포인터
```

```
    Node *pre = head;
```

```
    // curr 포인터가 NULL 이 아닐 때까지 반복
```

```
    while (curr != NULL)
```

```
    {
```

```
        pre = curr;
```

```
        curr = curr->next;
```

```
    }
```

```
    // 해당 위치에 노드 삽입
```

```
    pre->next = new_node;
```

```
}
```

```
// 뒤에서 k 번째 노드의 값을 구하는 함수
```

```
int getLastNode(Node* head, int k) {
```

```
    Node *curr = head->next;
```

```
    // curr 포인터가 가리키는 노드보다 k 전의 노드를 가리킴
```

```
    Node *pre = head;
```

```
    while (curr != NULL)
```

```
    {
```

```
        curr = curr->next;
```

```
        // pre 포인터가 curr 포인터와 k 노드 만큼 떨어진 위치 유지
```

```
        if (--k <= 0)
```

```
            pre = pre->next;
```

```

    }

    return pre->data;
}

// 모든 노드를 출력하는 함수
void printList(Node* head) {

    Node *curr = head->next;

    while (curr != NULL)
    {
        printf("%d ", curr->data);
        curr = curr->next;
    }

    free(curr);
}

int main() {
    Node* head = (Node*)malloc(sizeof(Node));
    head->next = NULL;

    append(head, 9);
    append(head, 8);
    append(head, 4);
    append(head, 14);
    append(head, 5);

    printList(head);

    printf("\n%dth last node is %d\n", 2, getLastNode(head, 2));
}

```

실행결과

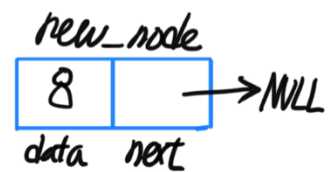
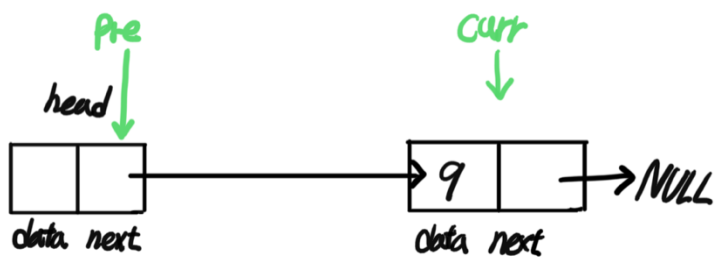
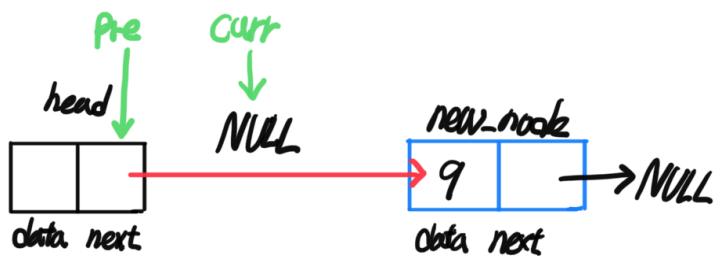
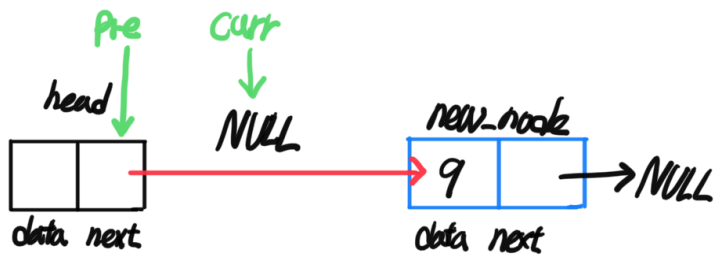
```

9 8 4 14 5
2th last node is 14
계속하려면 아무 키나 누르십시오 . . .

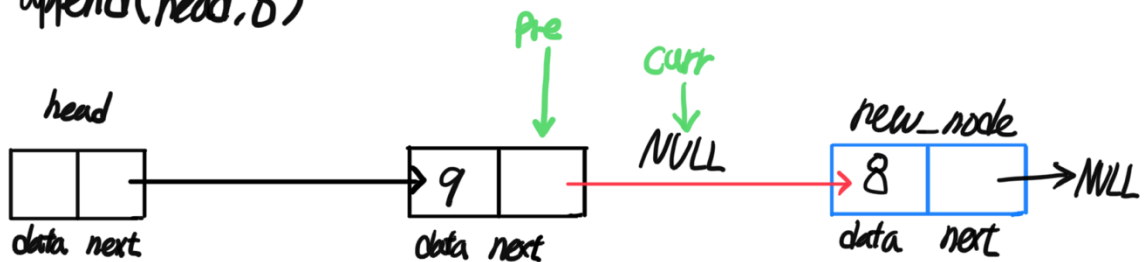
```

## 코드 함수 설명 (그림)

1) void append(Node\* head, int data)



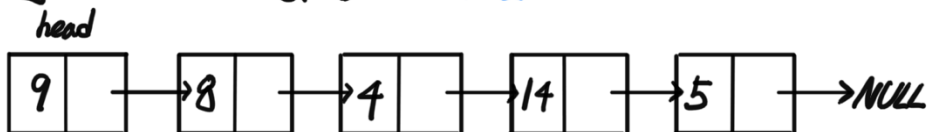
append(head, 8)

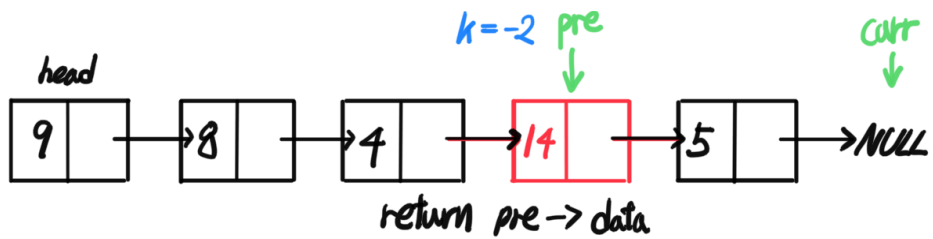
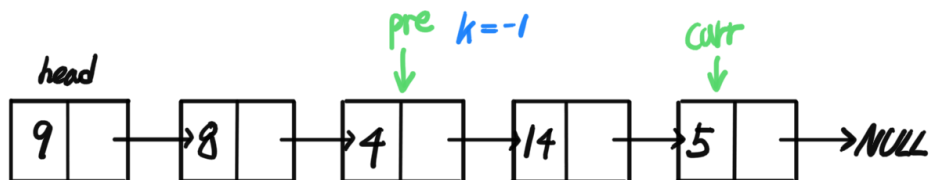
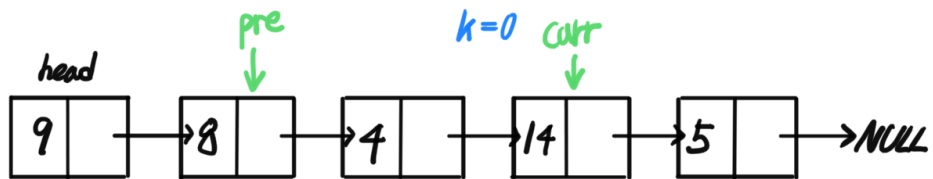
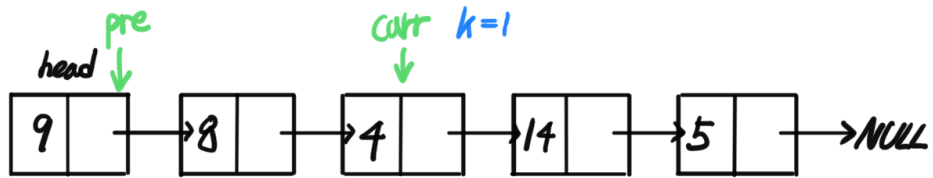
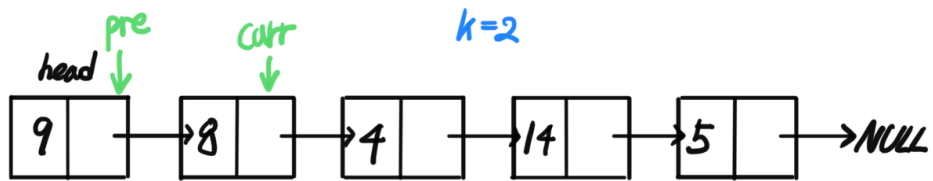


2) int getLastNode(Node\* head, int k)

getLastNode(head, 2)

k=2





15-5

코드 전체적으로 깔끔

```
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct node {
    int data;
    struct node *next;
} Node;
```

```
void append(Node *head, int data) {    // 이 부분을 작성해 주세요!
    if (head->data == 0)
        head->data = data;
    else if (head->next == NULL) {
        head->next = (Node *) malloc(sizeof(Node));
        append(head->next, data);
    }
    else append(head->next, data);
}
```

```
int getLastNode (Node *head, int k) {    // 이 부분을 작성해 주세요!
    int list_length = 1;
    Node *current_head = head;

    while (current_head->next != NULL) {
        current_head = current_head->next;
        list_length++;
    }

    current_head = head;

    for (int i = 0; i < list_length - k; i++) {
        current_head = current_head->next;
    }

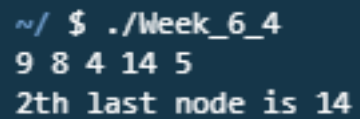
    return current_head->data;
}
```

```
void printList(Node *head) {            // 이 부분을 작성해 주세요!
    if (head == NULL)
        return;
    printf("%d ", head->data);
    printList(head->next);
}
```



```
int main() {  
    Node *head = (Node *) malloc(sizeof(Node));  
    append(head, 9);  
    append(head, 8);  
    append(head, 4);  
    append(head, 14);  
    append(head, 5);  
  
    printList(head);  
    printf("\n%dth last node is %d\n", 2, getLastNode(head, 2));  
}
```

[출력 결과]



```
~/ $ ./Week_6_4  
9 8 4 14 5  
2th last node is 14
```