자료구조(스택,큐,연결 리스트)

https://www.youtube.com/watch?v=1aDKJFDW4Mg

스택

큐

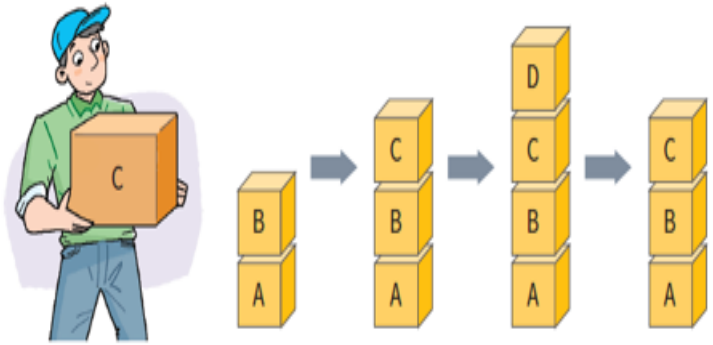
연결 리스트

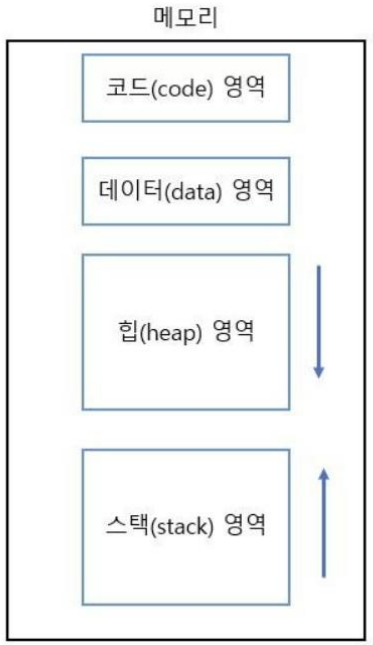
코드

스택

- 스택: 위로 쌓아서 올리는 의미

- LIFO(Last-In First-Out):가장 최근에 들어온 데이터가 가장 먼저 OUT
- 데이터를 한쪽에서만 넣고 뺄 수 있는 구조

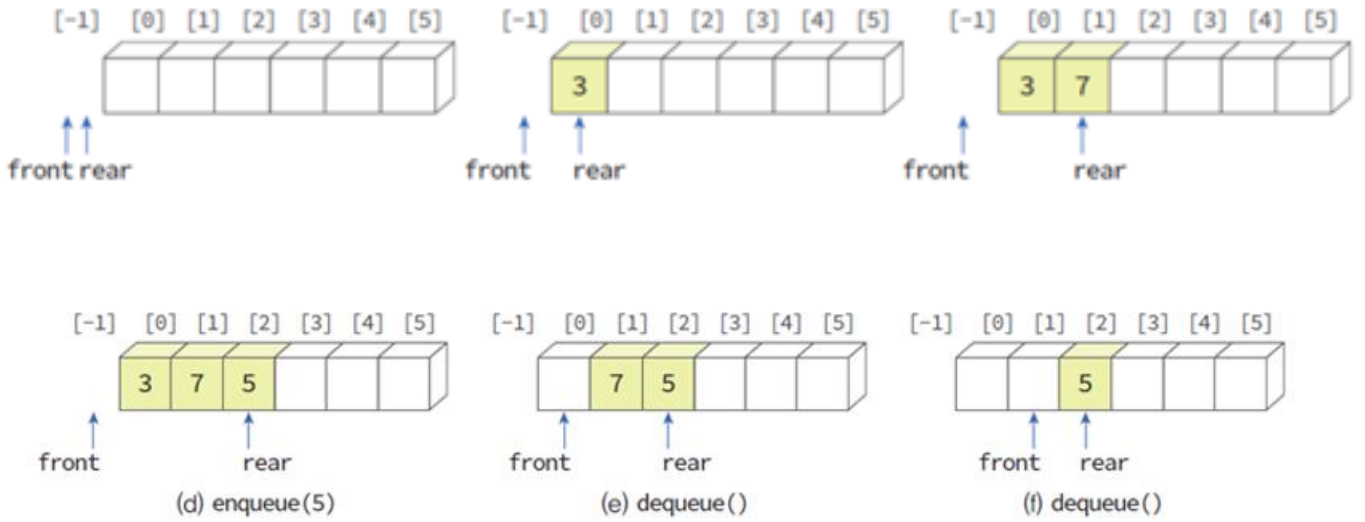




큐

- 큐: 먼저 들어온 데이터가 가장 먼저 나가는 구조

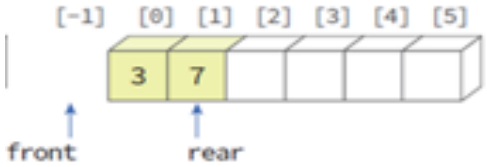
- FIFO(First-In First-Out): 가장 먼저 들어온 데이터가 가장 먼저 OUT



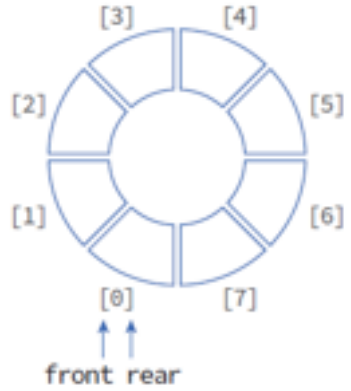
큐

- 큐의 종류

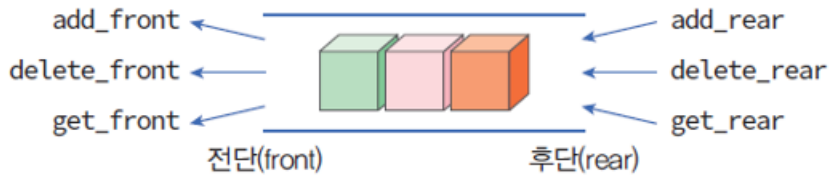
- 선형 큐



- 원형 큐



- 덱(deque: double-ended queue)



연결 리스트

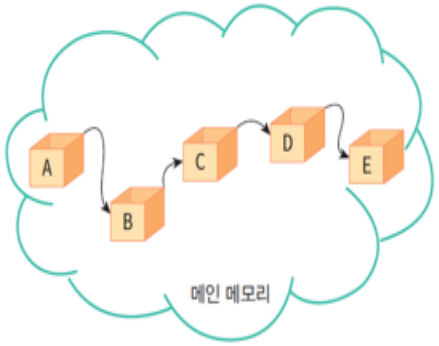
- 연결 리스트

  리스트는 노드(node)들로 구성되어 있음.

  노드는 데이터 필드와 링크 필드로 구성

  데이터 필드: 리스트의 데이터, 즉 데이터 값을 저장하는 필드

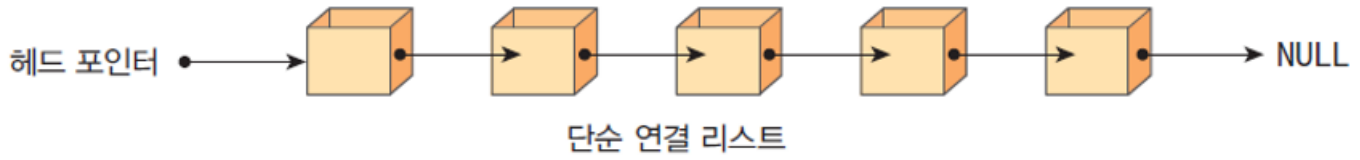
  링크 필드: 다른 노드의 주소 값을 저장하는 필드



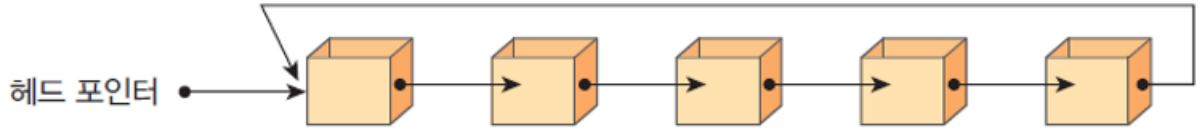

```
struct node {
  int data;
  struct node* link;
}
```

연결 리스트

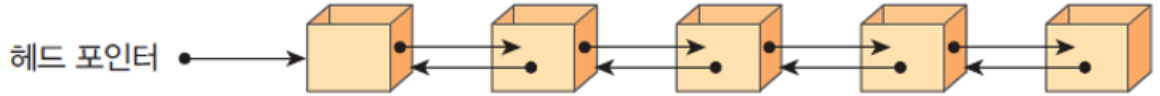
연결 리스트의 종류



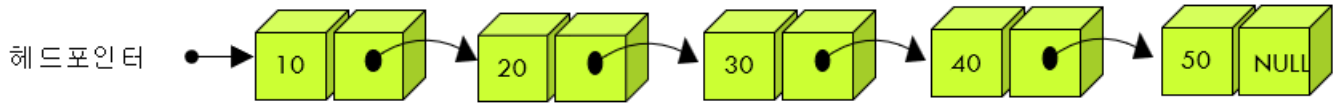
단순 연결 리스트



원형 연결 리스트



이중 연결 리스트

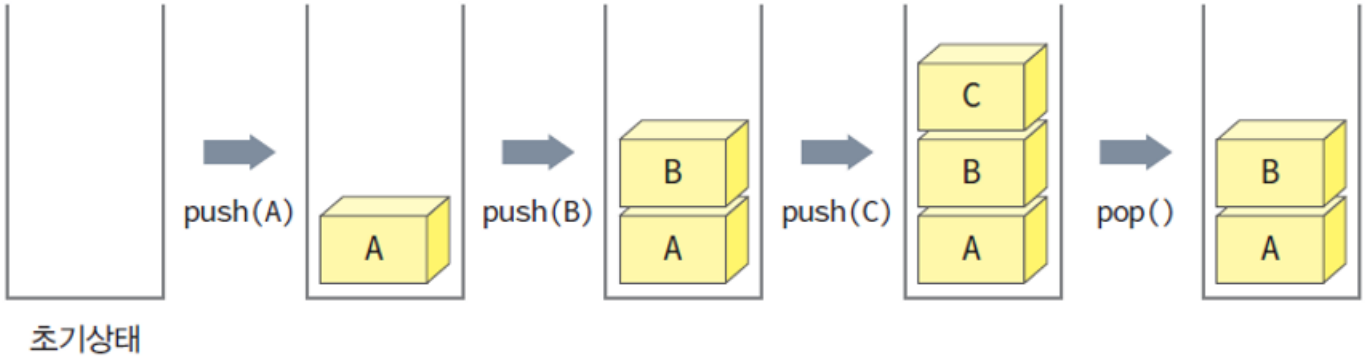


스택

- 스택의 연산

  Push(): 스택에 데이터를 추가

  Pop(): 스택에서 데이터를 삭제



스택_코드

ked_list_stack

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

typedef struct _Data
{
    int value;
    struct _Data* next;
}Data;


void init_linkedlist(void);
void push(int);
void pop(void);
void print_list(void);


Data* head;
Data* tail;
```

```
void init_linkedlist(void)
{
    head = (Data*)malloc(sizeof(Data));
    tail = (Data*)malloc(sizeof(Data));

    head->next = tail;
    tail->next = tail;
}

void push(int inputValue)
{
    Data* newData;

    newData = (Data*)malloc(sizeof(Data));

    newData->value = inputValue;
    newData->next = head->next;
    head->next = newData;
}
```

```
void pop(void)
{
    Data* nextData;
    int returnVal;

    if (head->next == tail)
    {
        printf("₩n This Linked List is empty₩n");
        exit(0);
    }

    nextData = head->next;
    returnVal = nextData->value;
    printf("returnVal is %d₩n", returnVal);

    head->next = nextData ->next;
    free(nextData);

    printf("%d is pop₩n", returnVal);
}
```

```
int main()
{


    init_linkedlist();

    push(3);
    push(5);
    push(7);
    push(10);

    print_list();

    pop();
    pop();
    pop();
    pop();
    pop();

    return 0;
```

d_list_stack (전역 범위)

```
void print_list(void)
{
    Data* printData;
    int printValue;

    printData = head->next;

    while (1)
    {
        printValue = printData->value;

        if (printData == tail)
        {
            printf("₩n The print is over ₩n₩n");
            break;
        }
        else
        {
            printData = printData->next;
            printf("%d is print value₩n", printValue);
        }
    }
}
```

큐

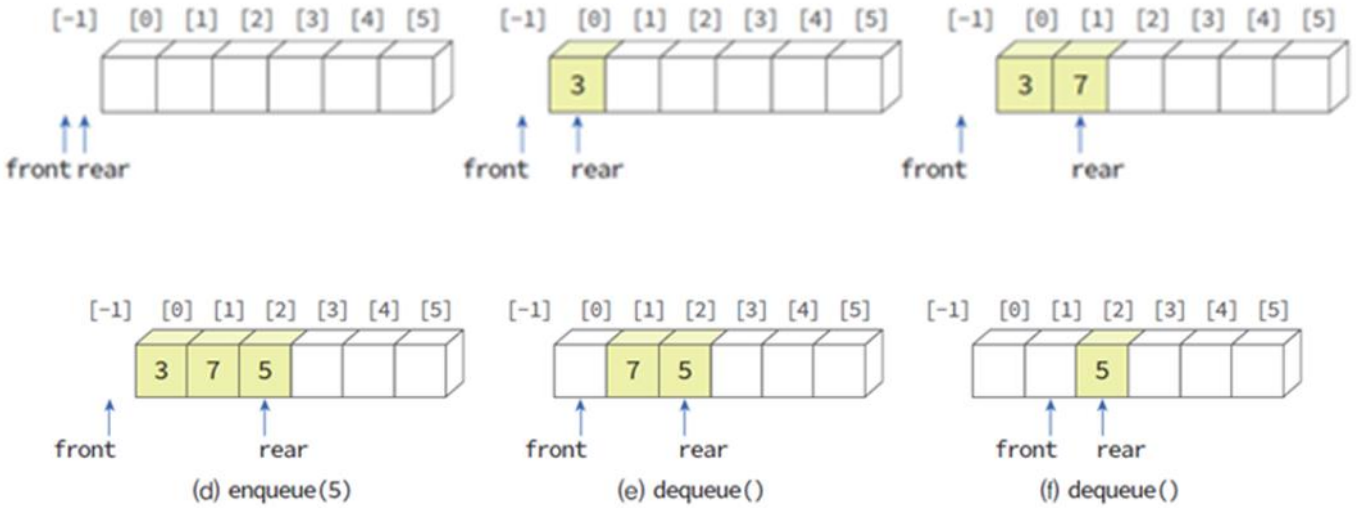
- 큐의 연산

  insert(int)

  : 큐에 데이터를 넣는다.

  remove()

  : 큐에서 데이터를 삭제한다.



큐_코드

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

struct node
{
    int data;
    struct node* next;
};

typedef struct node node;
node* head = NULL, * tail = NULL, * temp = NULL;
void insert(int);
void remove();
void print_queue();
int main()
{
    int x, d;

    do {
        printf("1. 삽입, 2. 삭제, 3. 출력    input: ");
        scanf_s("%d", &x);
        switch (x) {
        case 1:
            printf("₩n데이터 입력 : ");
            scanf_s("%d", &d);
            insert(d);
            break;
        case 2:
            remove();
            break;
        case 3:
            print_queue();
            break;
        }
        printf("₩n₩n");
    } while (1);
```

```c
void insert(int x) {
    temp = (node*)malloc(sizeof(struct node));
    temp->data = x;
    temp->next = NULL;
    if (tail != NULL)
        tail->next = temp;
    tail = temp;
    if (head == NULL)
        head = temp;

    printf("₩n%d가 삽입되었습니다. ₩n", x);
    print_queue();
    return;
}
void remove() {
    int x;
    if (head = NULL) {
        printf("₩nQueue is empty₩n");
        return;
    }

    temp = head;
    head = head->next;
    if (!head)
        tail = NULL;
    x = temp->data;
    free(temp);
    printf("%d가 삭제되었습니다.₩n", x);
    print_queue();
    return;
}
```

```c
void print_queue() {
    if (head == NULL) {
        printf("queue is empty₩n");
        return;
    }
    temp = head;
    printf("현재 큐의 내용 : ");
    while (temp) {
        printf(" <- %d ", temp->data);
        temp = temp->next;
    }
    return;
}
```

Q & A