자료구조(스택,큐)

유튜브 주소 : https://youtu.be/kipfKUvd9ss

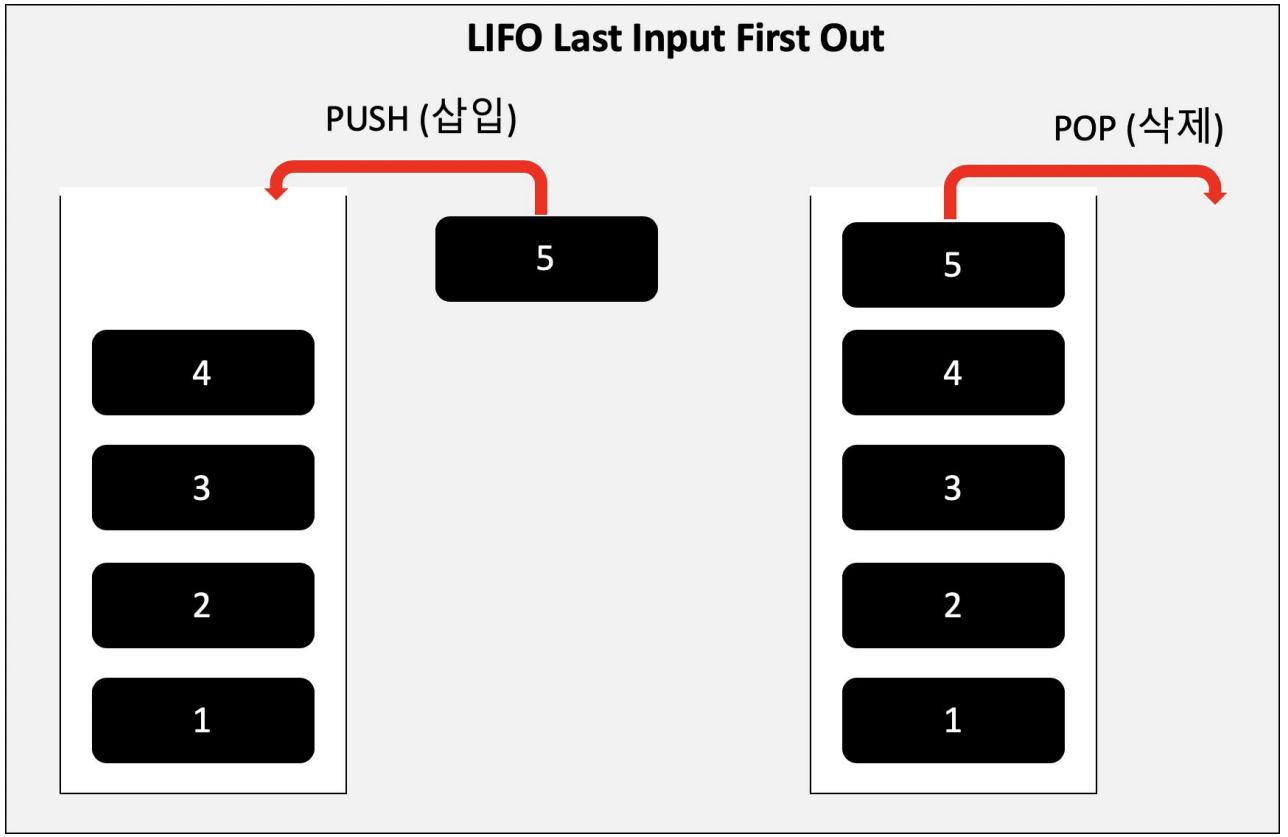
스택

큐

구현 코드 분석

스택

- 데이터를 쌓아 올린 형태의 자료구조
- 데이터를 제한적으로 접근할 수 있음
  - 한쪽 끝에서만 데이터를 넣거나 뺄 수 있는 구조
- LIFO(Last In First Out)



스택 연산

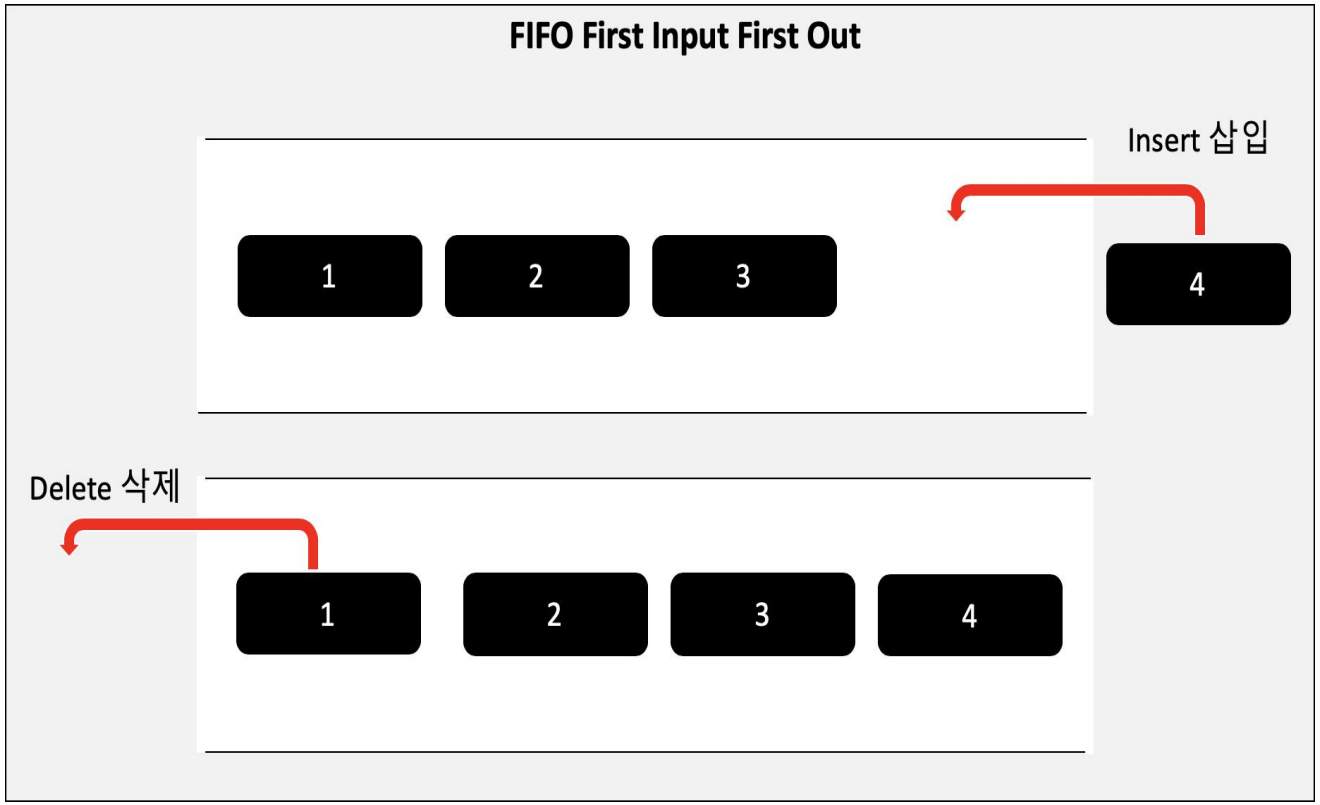
- Peek(top)
  - 스택의 맨 위의 원소를 제거하지 않고 반환
- Push
  - 스택의 맨 위에 데이터를 추가
- Pop
  - 스택의 맨 위에 있는 데이터를 제거한 후 반환
- full
  - 스택이 가득 차있으면 true반환 / 아닐 경우 false 반환
- empty
  - 스택이 비어있을 경우 true반환 / 아닐 경우 false 반환

스택 활용 사례

- 재귀 알고리즘
- 웹 브라우저 방문 기록(뒤로가기)
- 실행 취소(undo)
- 수식의 괄호 검사(연산자 우선 순위 산출을 위함)
- 후위 표기법 계산

큐

- 줄을 서서 기다리는 형태의 자료구조
- 데이터의 입력과 출력이 다름
  - 끝부분으로 데이터 삽입, 앞부분에서 데이터 삭제
- FIFO(First In First Out)
- 인큐 - 큐에 데이터를 삽입
- 디큐 – 큐에서 데이터를 삭제



큐 종류

- 선형 큐
  - 막대 모양으로 된 기본적인 큐
  - 크기 제한, 빈 공간 사용이 어려움
- 원형 큐
  - 원형 모양의 큐
  - 선형 큐의 문제점을 보완하고자 나옴
  - 모듈러 연산 사용 - (index + 1) % 배열의 크기
- 우선순위 큐
  - 각 데이터가 우선순위를 지님
  - 완전 이진 트리 구조 이용
  - 들어가는 순서에 관계 없이 우선순위에 맞게 디큐됨

큐 연산

- Add
  - 데이터를 리스트의 끝 부분에 추가
- Delete
  - 리스트의 첫 번째 항목을 제거 후 반환
- Front
  - 큐의 맨 앞을 가리킴(데이터를 꺼낼 수 있는 위치)
- Rear
  - 큐의 맨 뒤를 가리킴(데이터를 넣을 수 있는 위치)

큐 활용 사례

- 그래프 너비 우선 탐색(BFS) 구현
- 캐시 구현
- 프로세스 관리
- 프린터 출력 처리

스택 구현 코드

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

#define MAX_STACK_SIZE 100

typedef int element;
typedef struct {
    element data[MAX_STACK_SIZE];
    int top;
} StackType;

void init_stack(StackType *s)
{
    s->top = -1;
}

int is_empty(StackType *s)
{
    return (s->top == -1);
}

int is_full(StackType *s)
{
    return (s->top == (MAX_STACK_SIZE - 1));
}
```

```c
void push(StackType *s, element item)
{
    if (is_full(s)) {
        fprintf(stderr, _Format: "스택 포화 에러\n");
        return;
    }
    else s->data[++(s->top)] = item;
}

element pop(StackType *s)
{
    if (is_empty(s)) {
        fprintf(stderr, _Format: "스택 공백 에러\n");
        exit( _Code: 1);
    }
    else return s->data[(s->top)--];
}

element peek(StackType *s)
{
    if (is_empty(s)) {
        fprintf(stderr, _Format: "스택 공백 에러\n");
        exit( _Code: 1);
    }
    else return s->data[s->top];
}
```

선형 큐 구현 코드

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_QUEUE_SIZE 5

typedef int element;
typedef struct {
    int  front;
    int rear;
    element  data[MAX_QUEUE_SIZE];
} QueueType;

void error(char *message)
{
    fprintf(stderr, _Format: "%s\n", message);
    exit( _Code: 1);
}

void init_queue(QueueType *q)
{
    q->rear = -1;
    q->front = -1;
}
```

```c
int is_full(QueueType *q)
{
    if (q->rear == MAX_QUEUE_SIZE - 1)
        return 1;
    else
        return 0;
}

int is_empty(QueueType *q)
{
    if (q->front == q->rear)
        return 1;
    else
        return 0;
}
```

```c
void enqueue(QueueType *q, int item)
{
    if (is_full(q)) {
        error( message: "큐가 포화상태입니다.");
        return;
    }
    q->data[++(q->rear)] = item;
}

int dequeue(QueueType *q)
{
    if (is_empty(q)) {
        error( message: "큐가 공백상태입니다.");
        return -1;
    }
    int item = q->data[++(q->front)];
    return item;
}
```

원형 큐 구현 코드

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

#define MAX_QUEUE_SIZE 5
typedef int element;
typedef struct {
    element  data[MAX_QUEUE_SIZE];
    int  front, rear;
} QueueType;

void error(char *message)
{
    fprintf(stderr, _Format: "%s\n", message);
    exit( _Code: 1);
}

void init_queue(QueueType *q)
{
    q->front = q->rear = 0;
}
```

```c
int is_empty(QueueType *q)
{
    return (q->front == q->rear);
}

int is_full(QueueType *q)
{
    return ((q->rear + 1) % MAX_QUEUE_SIZE == q->front);
}

void enqueue(QueueType *q, element item)
{
    if (is_full(q))
        error( message: "큐가 포화상태입니다");
    q->rear = (q->rear + 1) % MAX_QUEUE_SIZE;
    q->data[q->rear] = item;
}
```

```c
element dequeue(QueueType *q)
{
    if (is_empty(q))
        error( message: "큐가 공백상태입니다");
    q->front = (q->front + 1) % MAX_QUEUE_SIZE;
    return q->data[q->front];
}

element peek(QueueType *q)
{
    if (is_empty(q))
        error( message: "큐가 공백상태입니다");
    return q->data[(q->front + 1) % MAX_QUEUE_SIZE];
}
```

Q & A