해달 자료구조 부트캠프

1. OT&ADT, 스택

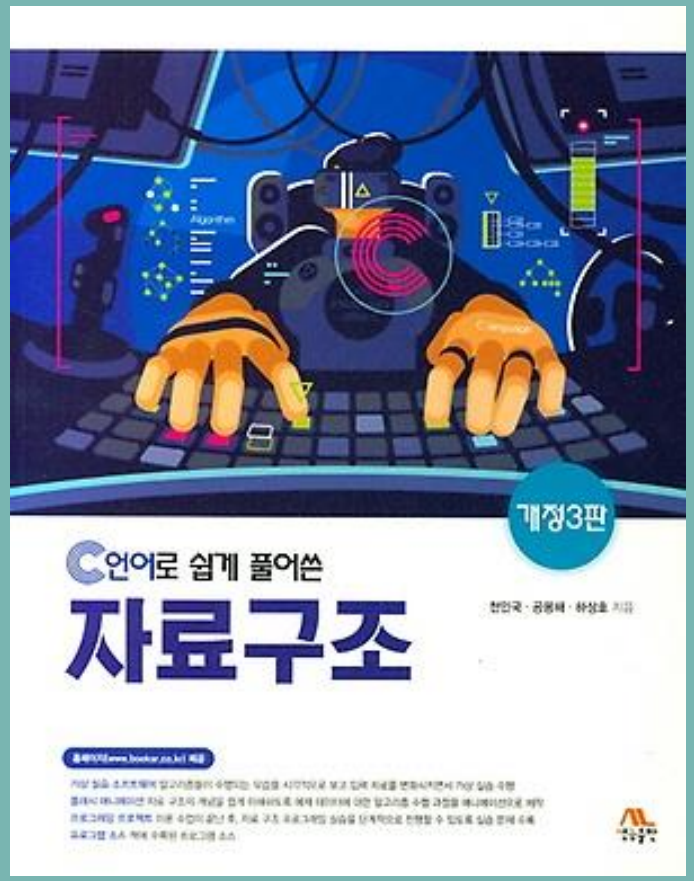
Table of Contents

Part 1

OT

Part 1

자료구조 부트캠프 소개



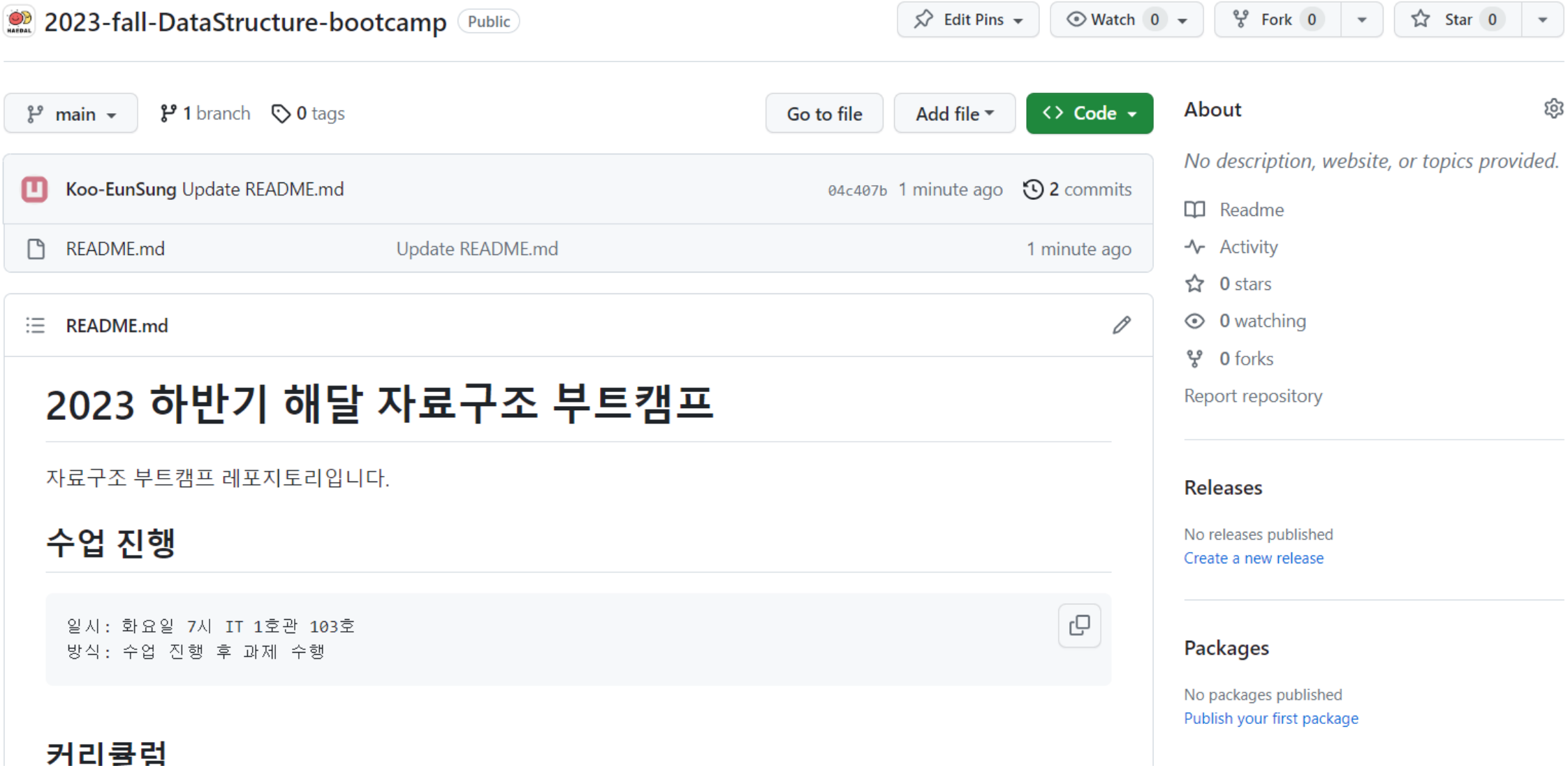
목표	자료구조 개념과 친숙해지기
일시	매주 화요일 오후 7시 IT1호관 103호
대상	자료구조는 처음이다 혹은 자료구조 공부를 더 해보고 싶으신 분들

Part 1

부트캠프 일정 안내

번호	날짜	상세
1주차	9/19	OT, ADT&스택
2주차	9/26	큐
3주차	10/3	연결리스트
4주차	10/10	트리
5주차	10/31	우선순위 큐
6주차	11/7	그래프(1)
7주차	11/14	그래프(2)

깃허브 안내



Part 2

ADT

ADT(Abstract Data Type)의 정의

- 자료들과 그 자료들에 대한 연산들을 명기한 것. 구현 방법을 명시하고 있지 않음. (위키백과)

- 실제적인 구현으로부터 분리되어 정의된 자료형.
즉, 자료형을 추상적(수학적)으로 정의함을 의미. (C언어로 쉽게 풀어쓴 자료구조)

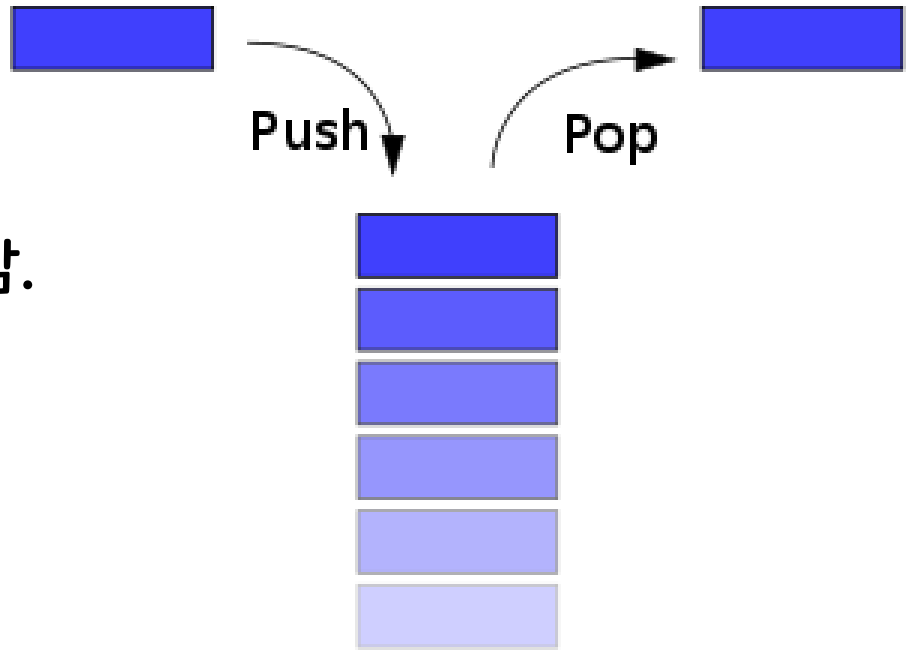
=>ADT는 구체적인 구현 방법은 명시되어 있지 않고, 핵심 개념이나 기능을 간추려낸 자료형

Part 3

스택

스택(Stack)의 개념

스택(stack)은 '쌓다, 더미'라는 뜻.
이처럼, 말 그대로 **데이터들을 쌓아놓는 것**을 말함.



스택의 입출력은 스택의 상단(stack top)에서만 일어남.

나중에 들어온 데이터가 먼저 나가는 방식을
LIFO(Last-In First-Out), 혹은 후입선출이라 부름.

ADT

추상 자료형으로서의 스택은 '0개 이상의 원소를 가지는 선형 리스트의 일종'으로 정의됨.
다음과 같은 연산들로 구성되어 있음.

- is_full: 포화상태 검사
- is_empty: 공백상태 검사
- push: 삽입. 단, 스택이 포화상태라면 정의 불가
- pop: 삭제. 단, 스택이 공백상태라면 정의 불가
- peek: 스택 최상단의 데이터 반환. pop과 마찬가지로 스택이 공백상태라면 정의 불가

구현

스택의 구현에는 2가지 방법이 있음.
1. 배열을 이용하는 방법.
2. 연결 리스트를 이용하는 방법.

배열을 이용할 경우의 장단점
- 장점: 구현 방법이 간단함
- 단점: 스택의 크기가 고정됨

연결리스트를 이용할 경우의 장단점
- 장점: 스택의 크기를 가변적으로 이용할 수 있음
- 단점: 구현이 복잡함

코드 – 배열을 이용한 방법

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

스택을 구조체로 정의하면 여러 개의 스택 이용 가능

```
typedef  struct {
   int student_no;
   char name[20];
   char address[50];
} element;
```

또한, element를 정의하기에 따라 여러 데이터를 넣는 것도 가능해짐.

코드 – 배열을 이용한 방법

```c
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));
}
```

코드 – 배열을 이용한 방법

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

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

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

코드 – 동적 배열을 이용한 방법

```c
typedef int element;
typedef struct {
  element *data;
  int capacity;
  int top;
} StackType;

void init_stack(StackType *s) {
  s->top = -1;
  s->capacity = 1;
  s->data = (element *)malloc(s->capacity * sizeof(element));
}
```

코드 – 동적 배열을 이용한 방법

```c
typedef int element;
typedef struct {
  element *data;
  int capacity;
  int top;
} StackType;

void init_stack(StackType *s) {
  s->top = -1;
  s->capacity = 1;
  s->data = (element *)malloc(s->capacity * sizeof(element));
}
```

코드 – 동적 배열을 이용한 방법

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

void push(StackType *s, element item) {
  if (is_full(s)) {
    s->capacity *= 2;
    s->data=(element *)realloc(s->data, s->capacity * sizeof(element));
    return;
  }
  else s->data[++(s->top)] = item;
}
```

동적 배열을 이용한 스택 구현의 장단점

장점
- 필요할 때마다 스택의 크기를 늘릴 수 있음

단점
- 메모리가 낭비될 수 있음
- 메모리 할당 및 해제에 시간이 추가적으로 소모될 수 있음
- 메모리 누수