Data Struct - 2

25NOVA Study

송동선

오늘 배울 것

1. 동적할당
2. 연결리스트

1158번 – 요세푸스 문제

요세푸스 문제 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	130035	65554	45916	49.163%

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

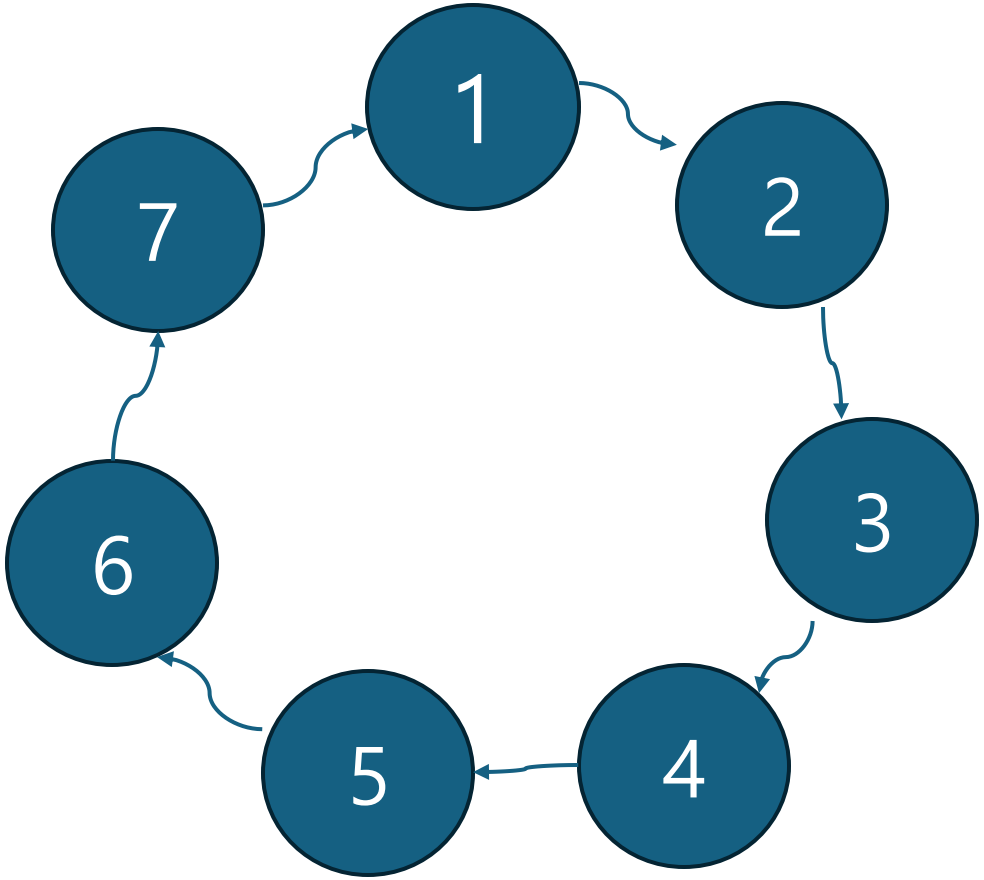
1158번 – 요세푸스 문제

- 시간 제한 : 2초
- 메모리 제한 : 256MB
- 입력값 : $1 \le K \le N \le 5,000$

1158번 – 요세푸스 문제

- 시간 제한 : 2초 약 200,000,000회 연산 가능
- 메모리 제한 : 256MB
- 입력값 : $1 \le K \le N \le 5,000$
  - N^2 = 25,000,000 (가능)
  - N^3 = 125,000,000,000 (**불가능**)
  - NlogN = ?

1158번 – 요세푸스 문제



요세푸스 문제 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	130035	65554	45916	49.163%

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.

문제 분석, 입력과 출력

- 입력
  - N : 사람 수 (1부터 N까지)
  - K : 제거할 사람의 간격

- 출력
  - 제거된 순서대로 출력

  -> (N, K) – 요세푸스 순열

문제 분석, 핵심 동작

1. 원형 구조
2. 순차 탐색
3. 특정 위치에서 제거
4. 반복적인 순회

배열 활용

회차 0 (초기 상태)

---

arr: [1] [2] [3] [4] [5] [6] [7]

idx: ↑

---

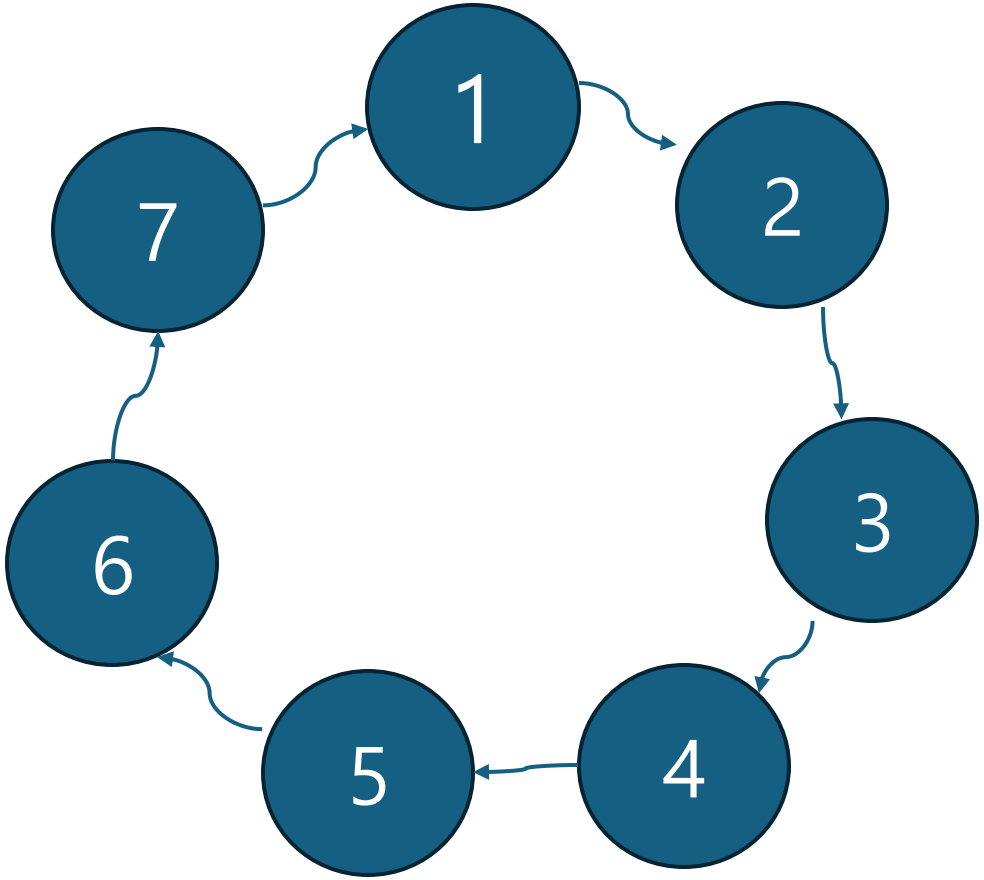
step = 0, 제거 대상 아직 없음

배열 활용

1. 삭제의 비효율성?
    1. 이미 삭제된 값의 인덱스도 메모리를 차지한다.
    2. 매번 들어있는 값이 유효값인지 확인 해야한다.

2. 입력이 커졌을 때 순회가 부담
    1. N=10000일때
    2. 9999번째 수를 제거한 뒤 하나의 유효값과 9999개의 -1가 저장된 메모리를 최악의 경우 9999번 순회해야 할 가능성

Linked list 활용



동적 할당

- 프로그램 실행 중에 메모리를 할당한다.
- Heap 영역 사용

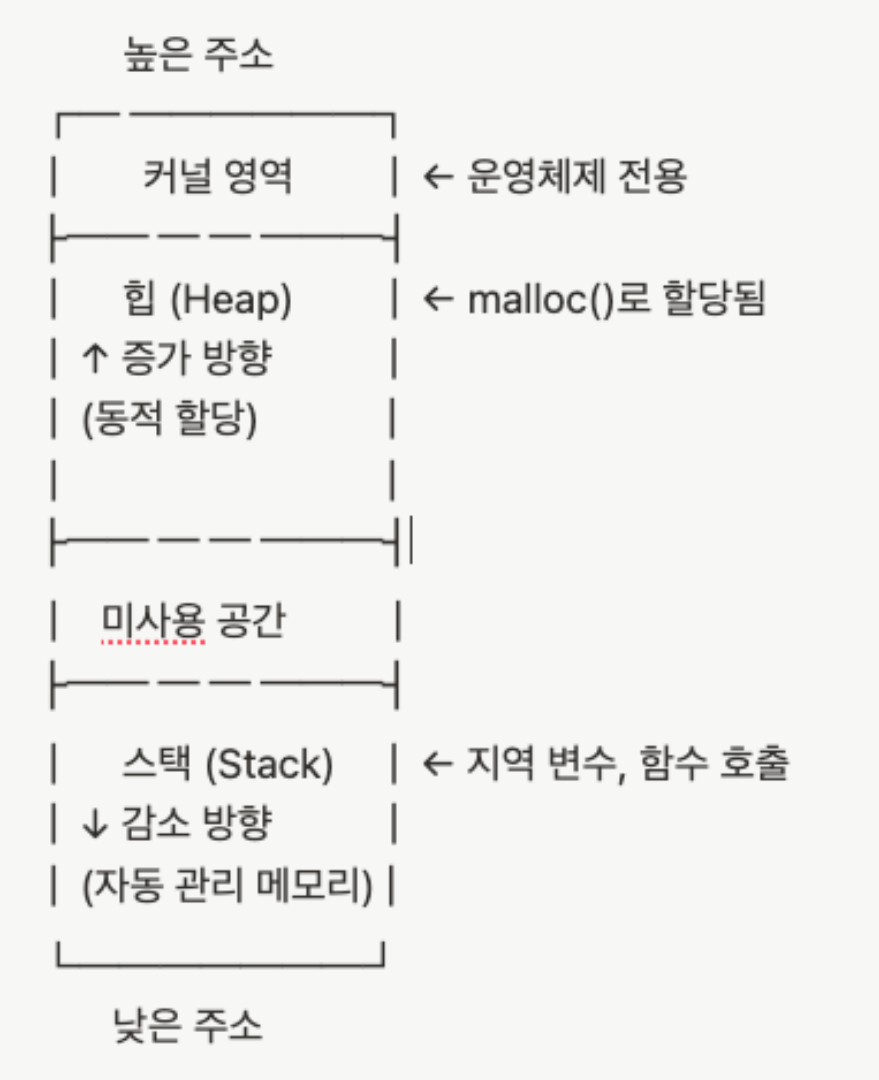
	설명	장점	단점
정적 할당	int arr[100]; 같이 **컴파일 타임**에 크기가 정해지는 방식	빠른 접근 속도	크기 변경 불가, 메모리 낭비
동적 할당	malloc(), calloc() 등을 사용하여 **실행시간**에 크기를 결정하는 방식	유연한 크기 조절	속도 저하, 메모리 누수 위험

동적 할당

- 컴파일 타임 (compile time)
  - 코드가 번역되어 기계어로 번역될 때

- 런타임 (run time)
  - 실제 실행되는 시간
  - main 함수가 시작된 이후

동적 할당



Heap : 실행 중 메모리 할당 시 사용하는 공간
Stack : 함수 호출 시 자동으로 사용되는 공간

동적할당

```c
int *p = (int*)malloc(sizeof(int));
*p = 42;
```

```
Heap:
0x5000 ——— [ 42 ]          ← *p 가 가리키는 값 (int 형 공간)

Stack:
p = 0x5000                 ← p 자체는 스택에 저장된 포인터 변수
```

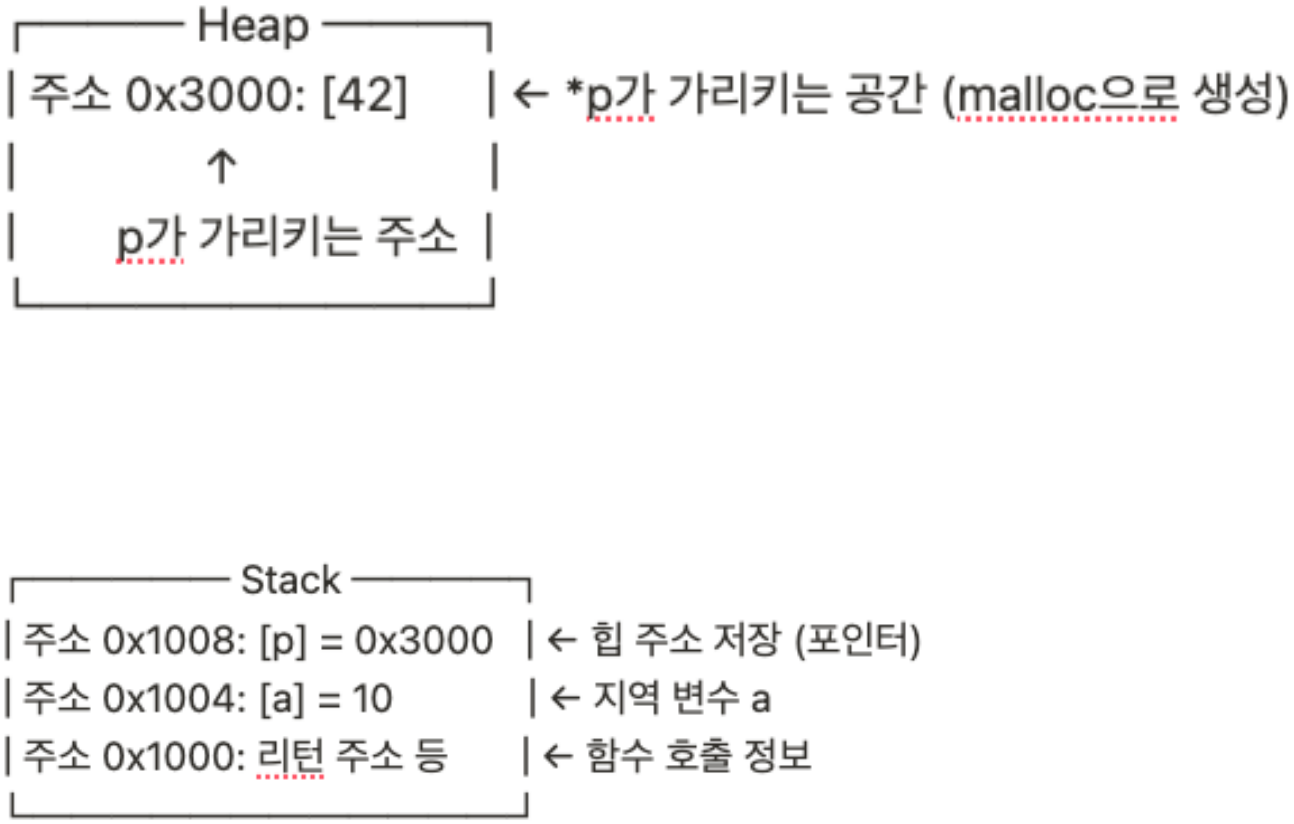
구성 요소	설명
int *p	p는 int형 데이터를 가리키는 **포인터 변수**
malloc(sizeof(int))	int 크기만큼의 메모리를 **Heap 영역에 동적 할당**
sizeof(int)	int 타입의 크기(보통 4바이트)를 계산함
(int*)	malloc()이 반환하는 void*를 int* 타입으로 **형변환 (type casting)**
=	할당된 메모리의 시작 주소를 포인터 p에 저장

동적 할당

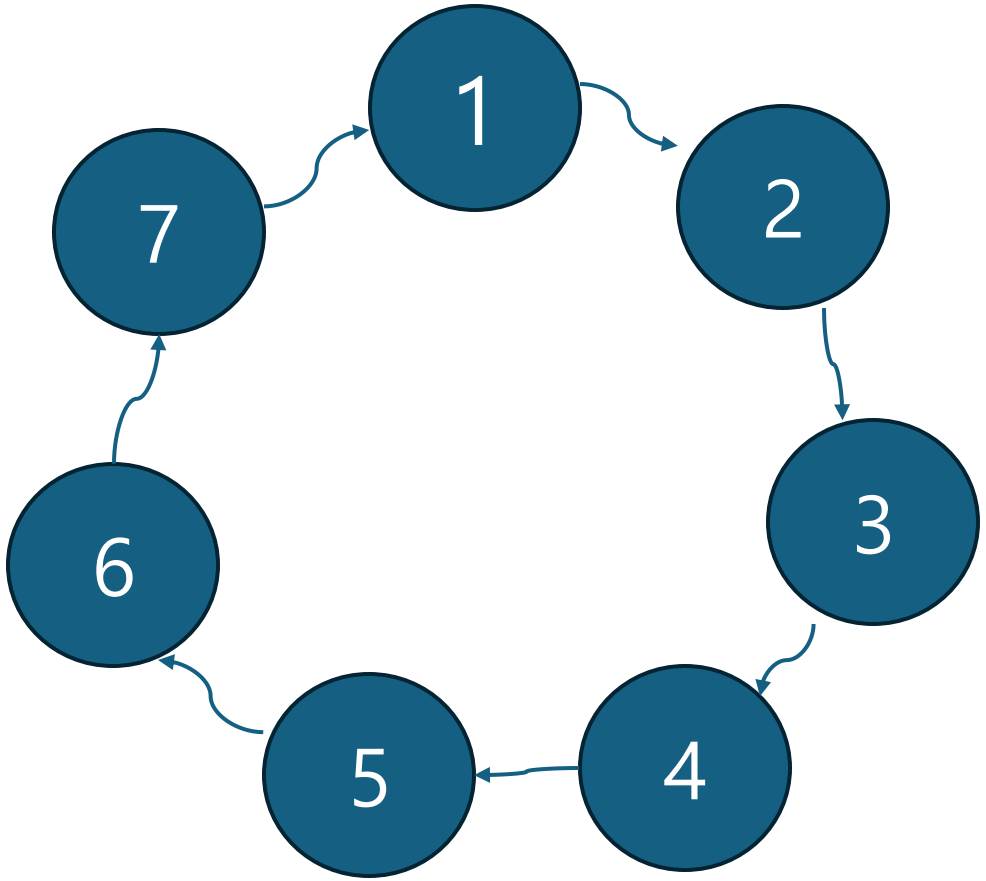
```
void func() {
    int a = 10;
    int *p = (int*)malloc( size: sizeof(int));
    *p = 42;
    free(p);
}
```

동적 할당

```c
void func() {
    int a = 10;
    int *p = (int*)malloc( size: sizeof(int));
    *p = 42;
    free(p);
}
```

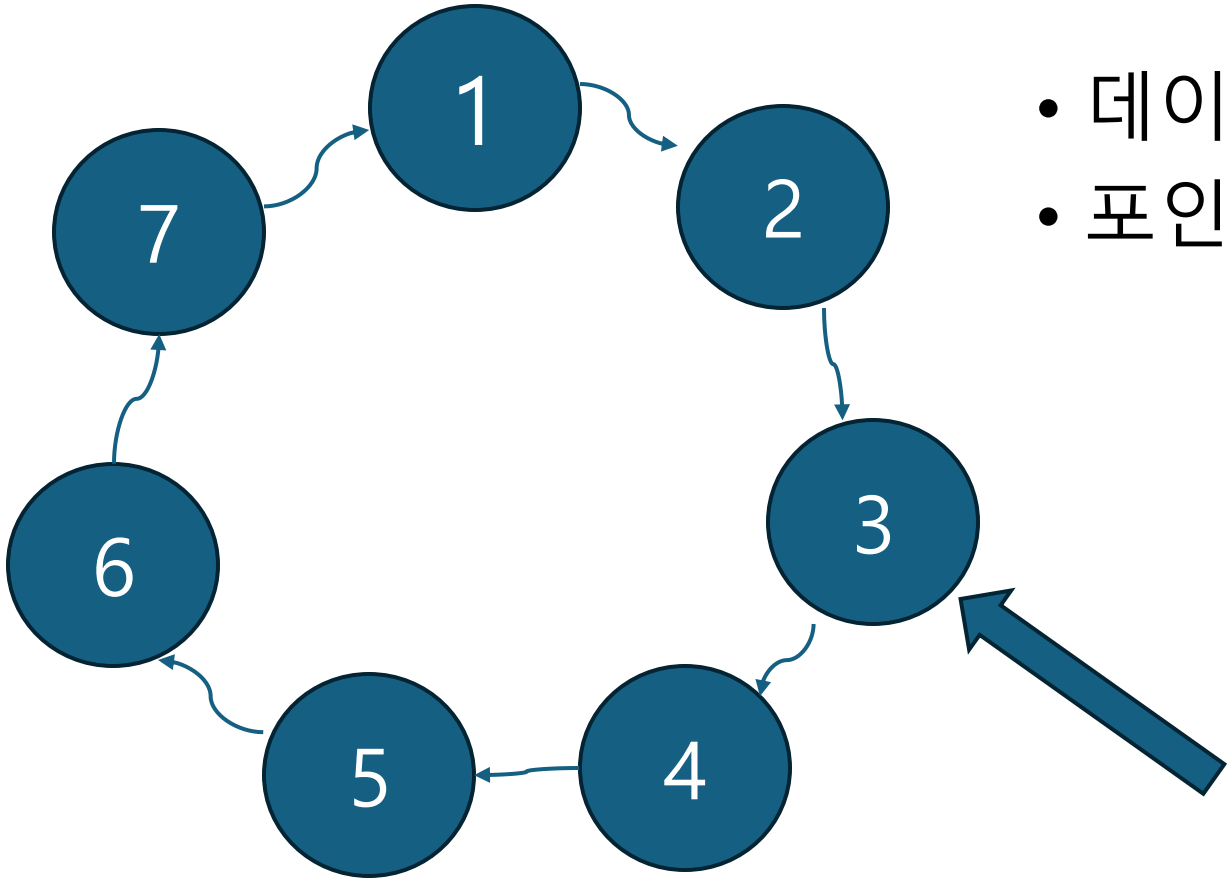


연결리스트



- 데이터를 Node 단위로 저장
- 포인터로 다음 node를 가리키며 연결

연결리스트



- 데이터를 Node 단위로 저장
- 포인터로 다음 node를 가리키며 연결

얘가 담고 있어야 할 정보는?

연결리스트

연결리스트

- 종류

1. 단일 연결 리스트
2. 이중 연결 리스트
3. 원형 연결 리스트

연결리스트

장점

- 필요한 만큼 실시간으로 할당받아 생성 가능
  - 미리 큰 공간을 할당 받을 필요가 없다.
- 삽입 삭제가 빠름

단점

- 접근 속도가 느리다
- 메모리 사용량이 많다
- 구현이 복잡
- 디버깅이 어렵다...

연결리스트

장점

- 필요한 만큼 실시간으로 할당받아 생성 가능
  - 미리 큰 공간을 할당 받을 필요가 없다.
- 삽입 삭제가 빠름

단점

- 접근 속도가 느리다
- 메모리 사용량이 많다
- 구현이 복잡
- 디버깅이 어렵다...

연결리스트

- 스택
- 큐
- 트리
- 그래프

- 다 이걸로 만들겁니다.

연결리스트

```c
typedef struct node {
    int key;
    struct node *left;
    struct node *right;
} Node;

int initializeBST(Node** h);

/* functions that you have to implement */
void inorderTraversal(Node* ptr);     /* recursive inorder traversal */
void preorderTraversal(Node* ptr);    /* recursive preorder traversal */
void postorderTraversal(Node* ptr);   /* recursive postorder traversal */
int insert(Node* head, int key);  /* insert a node to the tree */
int deleteLeafNode(Node* head, int key);  /* delete the leaf node for the key */
Node* searchRecursive(Node* ptr, int key);  /* search the node for the key */
Node* searchIterative(Node* head, int key);  /* search the node for the key */
int freeBST(Node* head); /* free all memories allocated to the tree */

/* you may add your own defined functions if necessary */


int main()
{
    char command;
```

연결리스트

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

// 노드 정의
typedef struct Node {
    int data;           // 데이터를 담을 공간
    struct Node* next;  // 다음 노드를 가리키는 포인터
} Node;
```

연결리스트

```c
void append(Node** head, int value) {
    Node* newNode = (Node*)malloc( size: sizeof(Node));  // 새 노드 동적 할당
    newNode->data = value;          // 값 저장
    newNode->next = NULL;           // 기본적으로 마지막 노드니까 next는 NULL

    if (*head == NULL) {
        // 리스트가 비어 있으면 새 노드가 첫 노드가 됨
        *head = newNode;
    }
    else {
        // 리스트의 끝까지 가서 새 노드를 연결
        Node* temp = *head;
        while (temp->next != NULL)
            temp = temp->next;

        temp->next = newNode;
    }
}
```

연결리스트

```
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d → ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

연결리스트

```
void printList(Node* head) {
    Node* temp = head;
    while (temp != NULL) {
        printf("%d → ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

연결리스트

```
void deleteNode(Node** head, int key) {
    Node* temp = *head;
    Node* prev = NULL;

    // 첫 번째 노드가 삭제 대상일 경우
    if (temp != NULL && temp->data == key) {
        *head = temp->next;
        free(temp);
        return;
    }
```

```
    // 중간 또는 끝 노드 탐색
    while (temp != NULL && temp->data != key) {
        prev = temp;
        temp = temp->next;
    }

    // 값을 찾지 못한 경우
    if (temp == NULL) return;

    // 삭제
    prev->next = temp->next;
    free(temp);
}
```

연결리스트

```c
void insertAt(Node** head, int index, int value) {
    if (index < 0) {
        printf("인덱스는 0 이상이어야 합니다.\n");
        return;
    }

    Node* newNode = (Node*)malloc( size: sizeof(Node));
    newNode->data = value;
    newNode->next = NULL;

    if (index == 0) {
        newNode->next = *head;
        *head = newNode;
        return;
    }

    Node* temp = *head;
    for (int i = 0; temp != NULL && i < index - 1; i++) {
        temp = temp->next;
    }

    if (temp == NULL) {
        printf("인덱스가 리스트 길이보다 큽니다.\n");
        free(newNode);
        return;
    }
```

연결리스트

```
void insertAfter(Node* prev, int value) {
    if (prev == NULL) {
        printf("이전 노드가 NULL입니다.\n");
        return;
    }

    Node* newNode = (Node*)malloc( size: sizeof(Node));
    newNode->data = value;
    newNode->next = prev->next;
    prev->next = newNode;
}
```

연결리스트

```c
int main() {
    Node* head = NULL;

    append(&head, value: 10);
    append(&head, value: 30);
    printf("append 후 리스트: ");
    printList(head);

    insertAfter( prev: head, value: 20);  // 10 다음에 20 삽입
    printf("insertAfter 후 리스트: ");
    printList(head);

    insertAt(&head, index: 2, value: 25); // 인덱스 2 위치에 25 삽입
    printf("insertAt 후 리스트: ");
    printList(head);

    deleteNode(&head, key: 30);
    printf("30 삭제 후 리스트: ");
    printList(head);

    return 0;
}
```

연결 리스트

append 후 리스트: 10 → 30 → NULL
insertAfter 후 리스트: 10 → 20 → 30 → NULL
insertAt 후 리스트: 10 → 20 → 25 → 30 → NULL
30 삭제 후 리스트: 10 → 20 → 25 → NULL

1158번 – 요세푸스 문제

요세푸스 문제 성공

☆

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	130035	65554	45916	49.163%

문제

요세푸스 문제는 다음과 같다.

1번부터 N번까지 N명의 사람이 원을 이루면서 앉아있고, 양의 정수 K(≤ N)가 주어진다. 이제 순서대로 K번째 사람을 제거한다. 한 사람이 제거되면 남은 사람들로 이루어진 원을 따라 이 과정을 계속해 나간다. 이 과정은 N명의 사람이 모두 제거될 때까지 계속된다. 원에서 사람들이 제거되는 순서를 (N, K)-요세푸스 순열이라고 한다. 예를 들어 (7, 3)-요세푸스 순열은 <3, 6, 2, 7, 5, 1, 4>이다.

N과 K가 주어지면 (N, K)-요세푸스 순열을 구하는 프로그램을 작성하시오.