연결 리스트 (Linked list)

자기 참조 구조체란?

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

struct EMP
{
    char name[20];
    int age;
    char address[40];
    struct EMP* next;
};
```

- 자기 참조 구조체는 구조체의 멤버로 자신의 구조체 자료형을 가리키는 포인터 멤버를 가짐
- 포인터 멤버 next를 이용해서 자신과 같은 구조체 주소를 참조함
- 마지막 노드의 포인터 필드는 끝을 나타내기 위해 NULL로 설정됨

연결 리스트

- 연결 리스트는 자기 참조 구조체로 생성됨
- 다음에 연결된 노드의 주소를 next에 저장하면서 연결되는 구조
- 연결 리스트를 구성할 때 사용되는 구조체 포인터 변수 head와 tail이 필요함
  - head
    - 첫 번째 노드를 가리킴
  - tail
    - 마지막 노드를 가리킴
    - 노드가 추가될 때마다 새로고침

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    struct EMP
    {
        char name[20];
        int age;
        char address[40];
        struct EMP* next;
    } *cur;

    struct EMP* head, * tail, * del;
    head = tail = del = NULL;

    while (1) {
        cur = (struct EMP*)malloc(sizeof(struct EMP));

        printf("성명 : (입력종료 : end)");
        gets(cur->name);
        if (!strcmp(cur->name, "end")) {
            break;
        }
        printf("나이 : ");
        scanf("%d%*c", &cur->age);
        printf("주소 : ");
        gets(cur->address);
        cur->next = NULL;

        if (head == NULL)
            head = tail = cur;
        else {
            tail->next = cur;
            tail = cur;
        }
    }
```

```
    free(cur);
    printf("\n");

    cur = head;
    while (cur) {
        printf("%s, %d, %s \n", ptr->name, ptr->age, ptr->add
        cur = cur->next;
    }

    cur = head;
    while (cur) {
        del = cur;
        cur = cur->next;
        free(del);
    }

    return 0;
}
```

- 연결 리스트를 생성, 출력, 삭제하는 예시 코드임
- 연결 리스트의 검색, 추가, 삭제의 구현 방법
  - 검색
    - 출력의 방식과 마찬가지로 노드들의 next를 탐색하면서 조건문으로 판별
  - 추가
    - 맨 뒤에 추가
      - 새로운 노드를 현재 tail의 next로 설정
      - 새로운 노드를 새로운 tail로 설정
    - 중간에 추가
      - 추가하고 싶은 자리의 전 노드의 next를 새로운 노드로 변경
      - 새로운 노드의 next를 원래 다음 노드로 설정
    - 맨 앞에 추가
      - 새로운 노드의 next를 head로 설정
      - 새로운 노드를 새로운 head로 설정

    - 삭제
        - 원하는 노드의 이전 노드의 next를 원하는 노드의 다음 노드로 설정
        - 원하는 노드를 삭제
        - 만약 원하는 노드가 head라면
            - 다음 노드를 head로 설정
        - 만약 원하는 노드가 tail이라면
            - 이전 노드를 tail로 설정

연결 리스트의 종류

- 단순 / 단방향 연결 리스트 (Singly Linked List)
    - 기본적인 연결 리스트
    - 구현이 쉬움
    - head를 잃어버리면 메모리 누수 발생

- 이중 / 양방향 연결 리스트 (Doubly Linked List)
    - 이전 노드로의 이동 가능
    - head나 tail 중 하나가 사라져도 모든 노드 탐색 가능
    - Singly보다 구현하기 어려움

- 원형 / 순환 연결 리스트 (Circular linked list)
    - 단순 연결에서 마지막 노드의 next 포인터를 처음 노드를 가리키게 함
    - 이중 순환 연결도 가능

- 다차원 연결 리스트 (Multidimensional Linked List)
    - next 포인터가 여러개로 이루어져 있음
    - 양방향이면 next 포인터 수만큼 prev 포인터가 존재

(ex) number of nextptr == N, total ptr == 2*N)

- 구현이 가장 어려움 (위의 연결보다)
- 순차 탐색을 통해 정보를 찾아야 함. (O(1)에 해결 불가)

연결 리스트의 활용

- 추상적 자료형 (Abstract Data Type)의 대부분을 구현하는 데에 사용

구현 자료형

- 스택, 큐, 덱
- 트리
  - 이진 트리. 이진 탐색 트리
  - B-tree, B+tree
  - AVL tree
  - 이진 힙
  - 트라이
  - 레드-블랙 트리
- 리스트
- 그래프