자료구조(F012) - 4장 실습 과제

G5_쓴맛불좀도둑들(송유민, 김성준, 김현민, 김진형, 김다원)

01번 - 정답 : 3번, **02번** - 정답 : 2번, **03번** - 정답 : 1번, **04번** - 정답 : 1번,

05번 - 정답 : 4번, **06번** - 정답 : 2번, **07번** - 정답 : 3번, **08번** - 정답 : 2번,

09번 - 정답 : 3번, **10번** - 정답 : 2번, **11번** - 정답 : 1번, **12번** - 정답 : 2번,

13번 - 정답 : 2번, **14번** - 정답 : 4번, **17번** - 정답 : 4번, **18번** - 정답 : 4번,

19번 - 정답 : 2번,

15번

<정답>

순차 자료구조는 물리적 순서와 논리적 순서가 같은 일반적인 배열이다.

원소의 위치를 찾아 접근하기 쉽다는 장점이 있다.

그러나 배열이 갖는 메모리 사용의 비효율성 문제가 있다.

그에 반해 연결 자료주고는 주소를 이용해 접근하기 때문에

물리적 구조와 논리적 구조가 다를 수 있다.

순차 자료구조보다 좀 더 효율적으로 메모리를 사용할 수 있다.

연결 자료구조에서는 주소를 저장하기 위한 단위로 노드를 사용한다.

16번

<정답>

단순 연결 리스트 : 노드가 하나의 링크 필드에 의해서

다음 노드와 연결되는 구조를 가지고 있다.

다음 노드에 대한 참조만을 가진 가장 단순한 형태의 연결 리스트이다.

예를 들어 a 다음이 b인 것은 참조를 통해 알 수 있지만

c 전에 b가 나오는건 알 수 없다.

원형 연결 리스트 : 마지막 노드가 NULL 대신 처음 노드를 가리키는 구조이다.

링크를 따라 계속 순회하면 이전 노드에 접근할 수 있다.

이중 연결 리스트 : 단순 연결 리스트의 구종ㅐ서 이전 노드의 참조도 가능하도록

양쪽 방향으로 순회 할 수 있게 노드를 연결한 리스트이다.

관리해야할 참조가 두 개가 있기 때문에 삽입 및 정렬의 경우

단순 연결 리스트에 비해 작업량이 더 많고 자료구조의 크기가 약간 커진다.

20번

<소스코드>

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct _node {
    int data;
    struct _node* next;
} node;

node* head;
node* tail;

void init_list(void) {
    head = (node*)malloc(sizeof(node));
    tail = (node*)malloc(sizeof(node));
    head->next = tail; //처음엔 둘 밖에 없음
    tail->next = tail; //끝은 NULL이 아니라 자기 자신
}

node* ordered_insert(int k) {
    node* p = head;
    node* i = (node*)malloc(sizeof(node));
    i->data = k;
    if (p->next == tail) //암것도 없음
    {
        head->next = i;
        i->next = tail;
        return i;
    }
```

```c
    while (p != tail)
    {
        if (p->next->data >= k) //이러면 k는 그 전에 들가야됨
        {
            i->next = p->next;
            p->next = i;
            return i;
        }
        if (p->next == tail) //위에서 안끝나고 여기까지 왔으면 k가 젤 크단 뜻
        {
            i->next = tail;
            p->next = i;
            return i;
        }
        p = p->next;
    }
}
node* print_list(node* t) {
    node* p = t; //t값을 오염시키면 안 됨.
    while (p != tail)
    {
        printf("%d ", p->data);
        p = p->next;
    }
    printf("₩n₩n");
    return p;
}
```

```
int delete_node(int k) {
    node* p = head;
    while (p != tail)
    {
        if (p->next->data == k)
        {
            p->next = p->next->next;
            printf("지웠음!!₩n₩n");
            return k;
        }
        p = p->next;
    }
    printf("그런거 없다₩n₩n");
    return -1;
}

//int _tmain(int argc, _TCHAR* argv[]) {
int main() {
    node* t;

    init_list();
    ordered_insert(10);
    ordered_insert(5);
    ordered_insert(8);
    ordered_insert(3);
    ordered_insert(1);
    ordered_insert(7);

    printf("₩nInitial Linked list is ");
    print_list(head->next);

    delete_node(8);
    print_list(head->next);

    return 0;
}
```

<실행 결과>

```
Initial Linked list is List: 1 3 5 7 8 10
List: 1 3 5 7 10
```