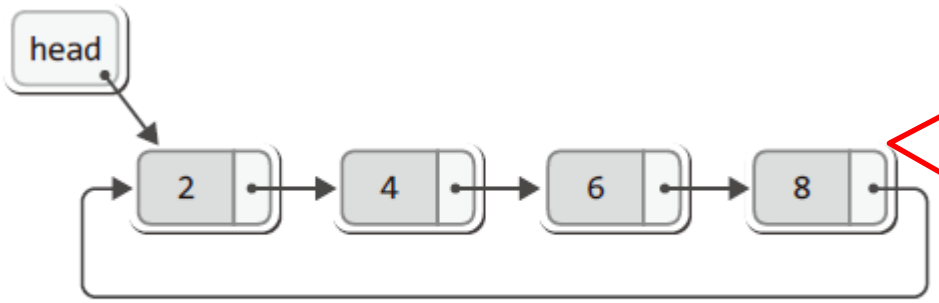
Chapter 05.
연결리스트3

6주차

원형연결리스트

▶ [그림 05-1 : 단순 연결 리스트]

단순 연결 리스트의 마지막 노드의 next멤버는 NULL.



1. 머리에 추가
2. 꼬리에 추가

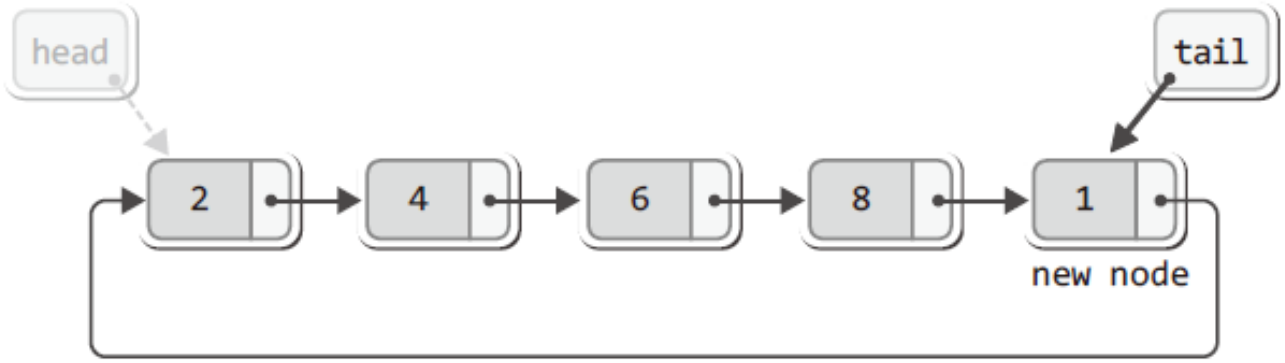
단점 : 위 두 경우 모두 마지막 노드의 next멤버를 수정해야 함

▶ [그림 05-2: 원형 연결 리스트]

원형 연결 리스트의 마지막 노드의 next맴버는 첫 번째 노드 주소.

(장점 : 머리와 꼬리 포인터를 각각 유지하지 않아도 됨.)

변형된 원형연결리스트



▶ [그림 05-6: 변형된 원형 연결 리스트]

꼬리를 가리키는 포인터 변수는? : tail

머리를 가리키는 포인터 변수는? : tail->next

하나의 포인터만 유지하면서 머리와 꼬리에 모두 손쉽게 노드 추가 가능.

변형된 원형연결리스트
(구현 범위)

· 조회관련 LFirst	이전에 구현한 연결 리스트와 기능 동일
· 조회관련 LNext	원형 연결 리스트를 계속해서 순환하는 형태로 변경!
· 삭제관련 LRemove	이전에 구현한 연결 리스트와 기능 동일
· 삽입관련	앞과 뒤에 삽입이 가능하도록 두 개의 함수 정의!
· 정렬관련	정렬과 관련된 부분 전부 제거
· 이외의 부분	이전에 구현한 연결 리스트와 기능 동일

원형연결리스트
(구조체 정의)

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#define TRUE      1
#define FALSE     0

typedef struct _node {
        int data;
        struct _node * next;
} Node;

typedef struct _CLL
{
        Node * tail;
        Node * cur;
        Node * before;
        int count;
} List;
```

노드 구조체
(자기참조구조체)

리스트 구조체

원형연결리스트
(리스트 초기화)

```
void ListInit(List * plist)
{
        plist->tail = NULL;
        plist->cur = NULL;
        plist->before = NULL;
        plist->count = 0;
}
```

원형연결리스트
(머리에 추가)

```
void LInsertFront(List * plist, int data)
{
        Node * newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;

        if (plist->tail == NULL)
        {
                plist->tail = newNode;
                newNode->next = newNode;
        }
        else
        {
                newNode->next = plist->tail->next;
                plist->tail->next = newNode;
        }

        plist->count++;
}
```

첫 노드일 때의 링크부 연결

두 번째 이후 노드일 때의 링크부 연결

원형연결리스트
(꼬리에 추가)

```
void LInsert(List * plist, int data)
{
        Node * newNode = (Node*)malloc(sizeof(Node));
        newNode->data = data;

        if (plist->tail == NULL)
        {
                plist->tail = newNode;
                newNode->next = newNode;
        }
        else
        {
                newNode->next = plist->tail->next;
                plist->tail->next = newNode;
                plist->tail = newNode;
        }

        plist->count++;
}
```

첫 노드일 때의 링크부 연결

두 번째 이후 노드일 때의 링크부 연결

원형연결리스트
(데이터 조회)

```
int LFirst(List * plist, int * pdata) {
        if (plist->tail == NULL)
                return FALSE;

        plist->before = plist->tail;
        plist->cur = plist->tail->next;

        *pdata = plist->cur->data;
        return TRUE;
}
int LNext(List * plist, int *pdata) {
        if (plist->tail == NULL)
                return FALSE;

        plist->before = plist->cur;
        plist->cur = plist->cur->next;

        *pdata = plist->cur->data;
        return TRUE;
}
```

저장된 노드가 없다면

before는 꼬리를
cur는 머리를 가리키게 함.

cur가 가리키는 노드의
데이터부를 반환

저장된 노드가 없다면

before와 cur를 한 칸 씩
전진시킴.

cur가 가리키는 노드의
데이터부를 반환

원형연결리스트
(노드의 삭제)

```
int LRemove(List * plist) {
        Node * rpos = plist->cur;
        int rdata = rpos->data;

        if (rpos == plist->tail) {
            if (plist->tail == plist->tail->next)
                plist->tail = NULL;
            else
                plist->tail = plist->before;
        }

        plist->before->next = plist->cur->next;
        plist->cur = plist->before;

        free(rpos);
        plist->count--;
        return rdata;
}
```

Tail이 가리키는 노드가 삭제대상일 때

Tail이 가리키는 노드가 유일한 노드일 때

삭제될 노드를 보관했다가 링크부 정리 후에 Free시킴.

링크부를 조정하여 cur가 가리키는 노드를 분리시킴. LRemove 후에는 항상 LNext하므로 before는 그대로 둔다. 같은 이유로 cur는 before쪽으로 옮겨줘야 건너뛰는 노드가 없게 된다.

원형연결리스트
(노드 갯수)

```
int LCount(List * plist)
{
        return plist->count;
}

void LPrint(List *plist) {
        int data, count = LCount(plist), i;
        printf("현재 데이터의 수: %d ₩n", count);
        if (LFirst(plist, &data)) {
                count--;
                printf("%d ", data);
                for (i = 0; i < count; i++) {
                        LNext(plist, &data);
                        printf("%d ", data);
                }
        }
        printf("₩n₩n");
}
```

원형연결리스트
(main함수 1/3)

```
int main(void) {
        List list; // 원형 연결 리스트의 생성 및 초기화
        int data, i, nodeNum;
        ListInit(&list);

        LInsert(&list, 3);  LInsert(&list, 4);  LInsert(&list, 5);
        LInsertFront(&list, 2);
        LInsertFront(&list, 1);

        LPrint(&list);
```

원형연결리스트
(main함수 2/3)

```
        nodeNum = LCount(&list);

        if (nodeNum != 0)
        {
                LFirst(&list, &data);
                if (data % 2 == 0)
                        LRemove(&list);

                for (i = 0; i < nodeNum - 1; i++)
                {
                        LNext(&list, &data);
                        if (data % 2 == 0)
                                LRemove(&list);
                }
        }

        LPrint(&list);
        return 0;
}
```

2의 배수를 찾아서 모두 삭제