

## 포인터의 개념[복습]

- 포인터는 메모리 주소를 값으로 가지는 데이터 형(type)이다.
- C 언어에서는 어떤 타입 **T**에 대해서 **T의 포인터 타입(T \*)**이 존재한다.  
**int \*      float \***
- 포인터 타입에는 주소연산자(&)와 역참조(간접 지시) 연산자(\*)가 사용된다.

9주차 과제 2번 : 다음 printf문장에 의하여 출력되는 num1, num2의 값은?

```
main()
{
    int num1 = 100, num2;
    float ratio = 0.5;
    num2 = cal(&num1, ratio);
    printf("%d == %d\n",
           num1, num2);
}
```

```
int cal(int *a, float b)
{
    int c;
    c = (*a) * b;
    *a = (*a) * 2;
    return c;
}
```

# 연결리스트의 개념(복습)

## ◆ 연결 리스트의 구조

- [데이터, 링크]의 형태의 노드를 기본 단위로 연결되어 있음

데이터	링크
-----	----

- 데이터 필드(data field) : 표현하려는 값을 저장
- 링크 필드(link field) : 다음 노드의 주소를 저장

## 단순 연결 리스트의 구현(복습)

### ◆ 연결 리스트를 생성하기 위해 필요한 기능

- (1) 노드의 구조 정의 – 자기 참조구조체
- (2) 노드 생성 : malloc() 함수 사용
- (3) 노드의 데이터 필드와 링크 필드에 값을 할당

### ◆ [자기참조구조체의 예제]

```
typedef struct simple_list *simple_pointer;  
struct simple_list {  
    char stae[3];  
    int count;  
    simple_pointer next;  
};
```

```
typedef struct list_node *list_pointer;  
struct list_node {  
    int data;  
    list_pointer link;  
};
```

## 단순 연결 리스트의 구현

```
void main()
{
    list_pointer ptr;
    ptr = make_node();
    print_list(ptr);
    printf("list안의 data 합 : %d\n", nodesum(ptr));
    printf("list안의 node 수 : %d\n", nodenumber(ptr));
}

list_pointer make_node()
{
    list_pointer first, second, third;
    first = (list_pointer)malloc(sizeof(struct list_node));
    second = (list_pointer)malloc(sizeof(struct list_node));
    third = (list_pointer)malloc(sizeof(struct list_node));
    first->data = 100; first->link = second;
    second->data = 200; second->link = third;
    third->data = 300; third->link = NULL;
    return first;
}
```

## 단순 연결 리스트의 구현

```
void print_list(list_pointer ptr) {  
    printf("The list contains: ");  
    for (; ptr; ptr = ptr -> link)  
        printf("%4d", ptr -> data);  
    printf("\n");  
}
```

```
int nodenumber(list_pointer ptr) {  
    int count = 0;  
    for (; ptr; ptr = ptr -> link)  
        count++;  
    return count;  
}
```

# 원형연결리스트(Circular linked list)

## ◆ 원형 연결 리스트의 개념

마지막 노드의 링크가 첫 번째 노드(head node)를 가리키는 리스트

### 장점

- 한 노드에서 다른 모든 노드로의 접근이 가능

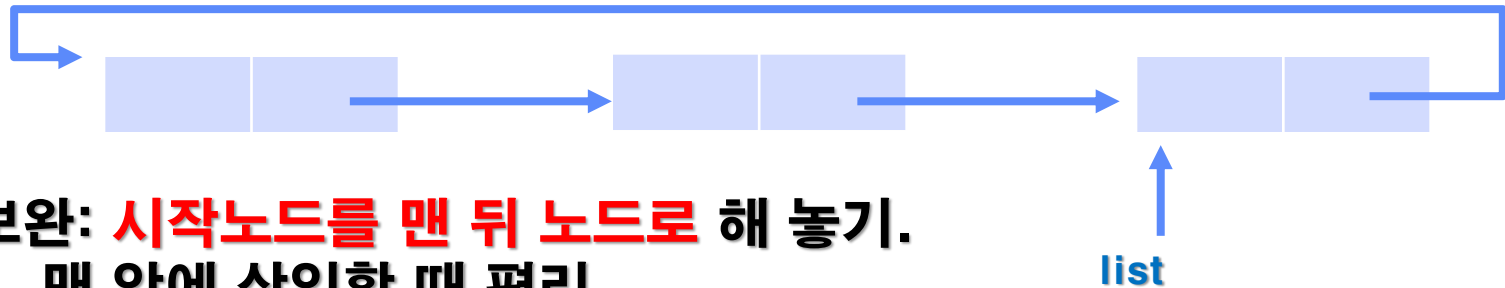
### 주의

- 검색할 때 무한루프에 빠지지 않도록 검색종료 시킬 조건을 정해야 함

## 원형연결리스트의 구조



단점: 앞에 삽입하려면, 맨 뒤 원소까지 찾아가야 함.



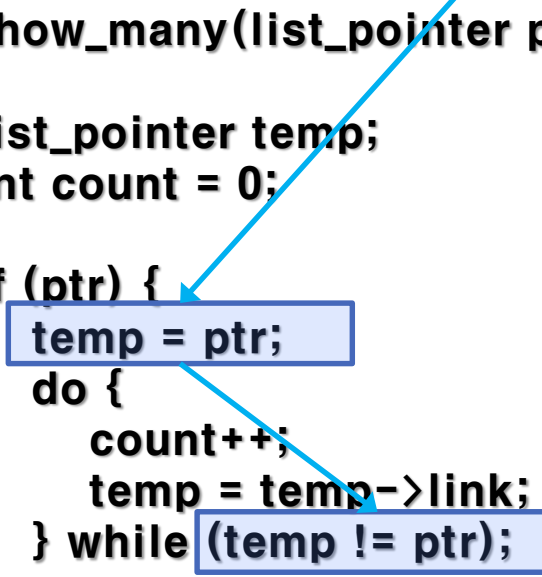
보완: 시작노드를 맨 뒤 노드로 해 놓기.  
맨 앞에 삽입할 때 편리.

# 원형연결리스트의 길이 계산함수

주의 : 검색할 때 무한루프에 빠지지 않도록  
검색종료 시킬 노드를 정해야 함.

```
int how_many(list_pointer ptr)
{
    list_pointer temp;
    int count = 0;

    if (ptr) {
        temp = ptr;
        do {
            count++;
            temp = temp->link;
        } while (temp != ptr);
    }
    return count;
}
```





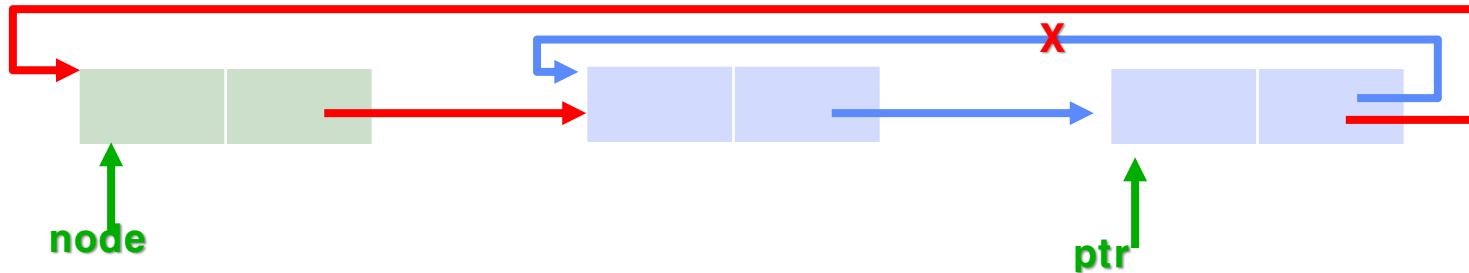
# 원형연결리스트 활용 프로그래밍연습(1)

- 원형연결리스트 구현을 위한 자기참조구조체를 정의해 보자.
  - 구조체 이름 : `struct cnode`
  - 구조체 포인터 형 선언(`typedef` 사용)할 때의 이름 : `npointer`
  - 구조체 내용 : 번호(`num, int`), 등급(`grade, char`), 연결(`link`)
  - 시작노드주소를 위한 “`ptr`” 변수를 만들자.

```
typedef struct cnode *npointer;
struct cnode {
    int num;
    char grade;
    npointer link;
};
npointer ptr = NULL;
```

## 원형연결리스트 활용 프로그래밍연습(2)

[insert\_front() 함수]



```
void insert_front(npointer node)
{
    if (!ptr) {
        node -> link = node;
        ptr = node;
    }
    else {
        node -> link = ptr -> link;
        ptr -> link = node;
    }
}
```

## 원형연결리스트 활용 프로그래밍연습(3)

[c\_print() 함수]

```
void c_print()
{
    npointer temp = ptr;
    if (ptr) {
        do {
            temp = temp -> link;
            printf("%d : %c\n", temp->num, temp->grade);
        } while ( temp != ptr );
    }
}
```

# 원형연결리스트 활용 프로그래밍연습(4)

[main() 함수]

```
void main()
{
    int cond=1, i;
    npointer temp;

    while (cond) {
        temp = (npointer) malloc(sizeof(struct cnode));
        printf("Enter id and grade : ");
        scanf("%d %c", &(temp->num), &(temp->grade));
        insert_front(temp);
        printf("Continue?(0/1) : ");
        scanf("%d", &cond);
    }
    c_print();
    printf( "The number of nodes in this list = %d\n" , how_many());
}
```

## 원형연결리스트 활용 프로그래밍연습(5)

[how\_many() 함수]

```
int how_many()
{
    npointer temp;
    int count = 0;
    if (ptr) {
        temp = ptr;
        do {
            count++;
            temp = temp->link;
        } while (temp != ptr);
    }
    return count;
}
```