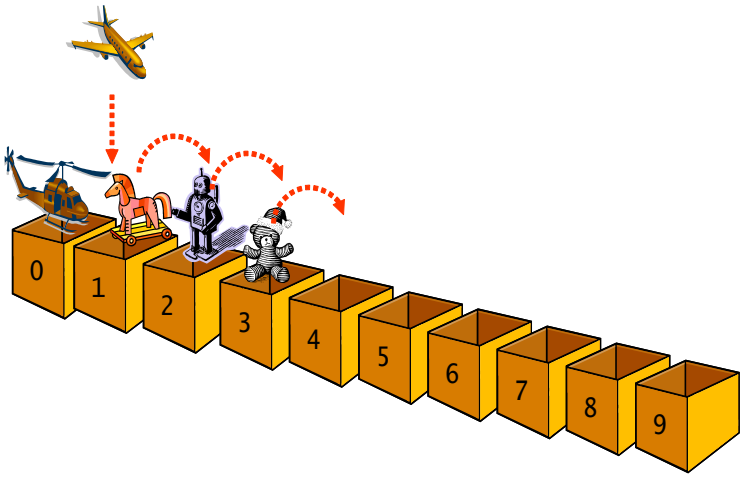
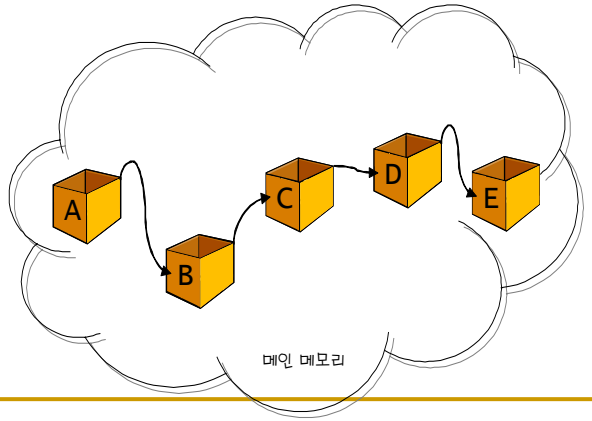
동적 메모리와 연결리스트 Part 2

연결 리스트

- 배열(array)
  - 장점: 구현이 간단하고 빠르다
  - 단점: 크기가 고정된다.
  - 중간에서 삽입, 삭제가 어렵다.

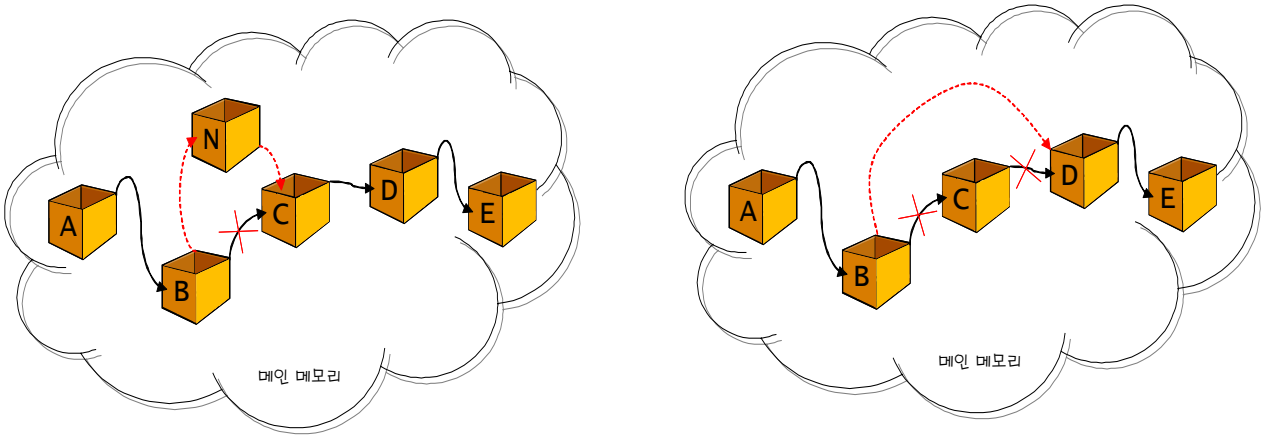


- 연결 리스트(linked list)
  - 각각의 원소가 포인터를 사용하여 다음 원소의 위치를 가리킨다.



연결 리스트의 장단점

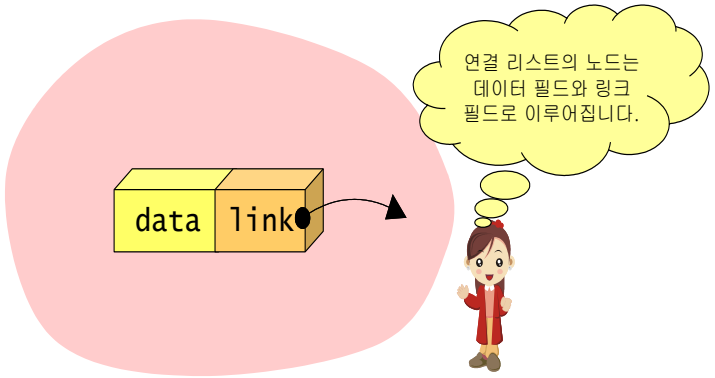
- 중간에 데이터를 삽입, 삭제하는 경우



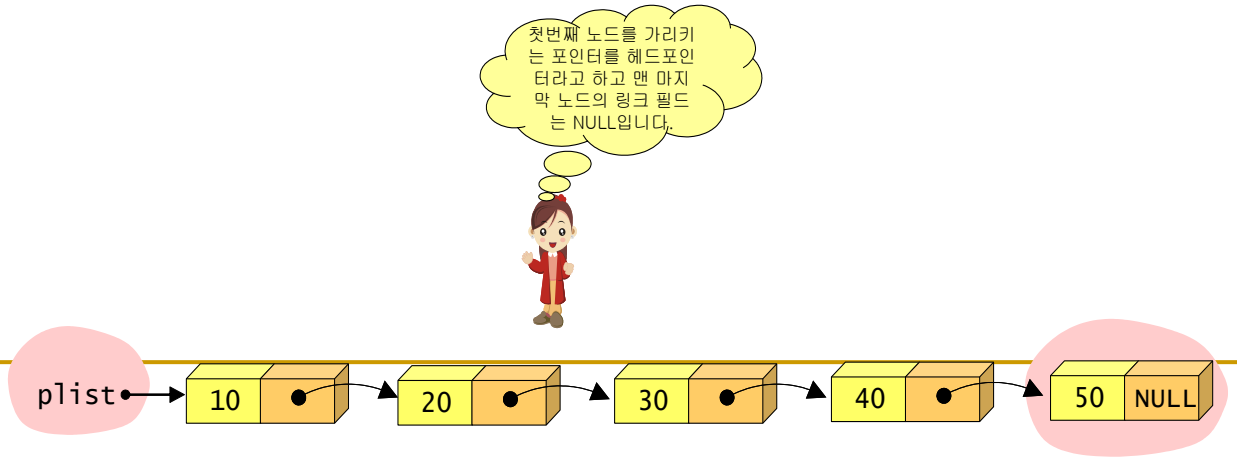
- 데이터를 저장할 공간이 필요할 때마다 동적으로 공간을 만들어서 쉽게 추가
- 구현이 어렵고 오류가 나기 쉽다.

연결 리스트의 구조

- 노드(node) = **데이터 필드**(data field)+ **링크 필드**(link field)



- **헤드 포인터**(head pointer)**: 첫번째 노드를 가리키는 포인터**



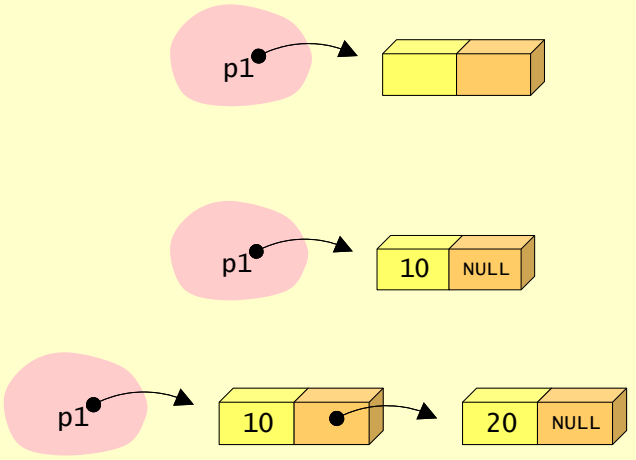
자기 참조 구조체

- **자기 참조 구조체(self-referential structure)**는 특별한 구조체로서 구성 멤버 중에 같은 타입의 구조체를 가리키는 포인터가 존재하는 구조체

```
// 노드의 정의
typedef struct NODE {
        int data;
        struct NODE *next;
} NODE;
```

간단한 연결 리스트 생성

```
NODE *p1;
p1 = (NODE *)malloc(sizeof(NODE));

p1-> data = 10;
p1-> next = NULL;

NODE *p2;
p2 = (NODE *)malloc(sizeof(NODE));
p2-> data = 20;

p2-> next = NULL;
p1-> next = p2;

free(p1);
free(p2);
```

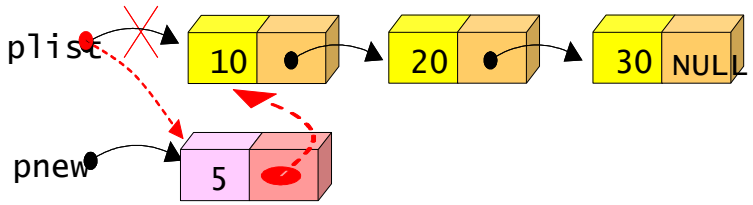


연결 리스트의 삽입 연산

```
NODE *insert_NODE(NODE *plist, NODE *pprev, int item);
```

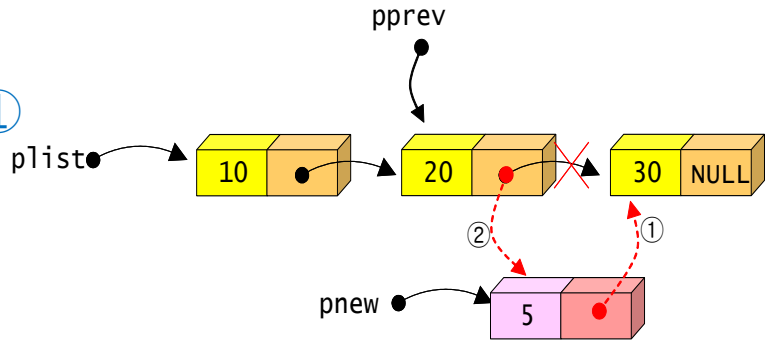
1. 리스트의 처음에 삽입하는 경우

```
pnew -> next = plist;
plist = pnew;
```



2. 리스트의 중간에 삽입하는 경우 (순서가 중요)

```
pnew -> next = pprev -> next;// ①
pprev -> next = pnew;// ②
```



연결 리스트의 삽입 연산

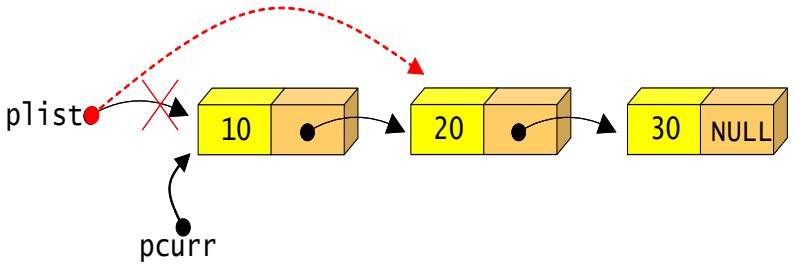
```
NODE *insert_node(NODE *plist, NODE *pprev, int item)
{
        NODE *pnew = NULL;

        if( !(pnew = (NODE *)malloc(sizeof(NODE))) )
        {
                printf("메모리 동적 할당 오류\n");
                exit(1);
        }

        pnew->data = item;
        if( pprev == NULL )      // 연결 리스트의 처음에 삽입
        {
                pnew->next = plist;
                plist = pnew;
        }
        else                     // 연결 리스트의 중간에 삽입
        {
                pnew->next = pprev->link;
                pprev->next = pnew;
        }
        return plist;
}
```

연결 리스트의 삭제 연산

```
NODE *delete_node(NODE *plist, NODE *pprev, NODE *pcurr);
```

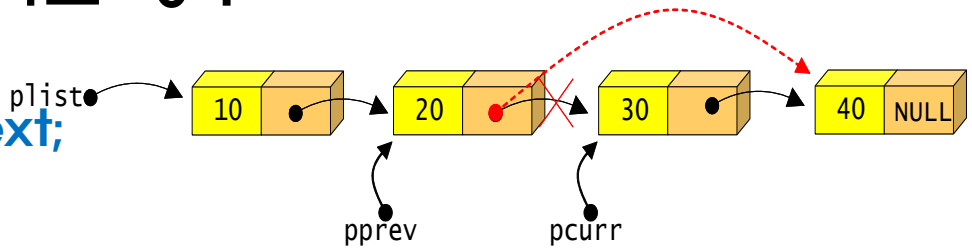
1. 리스트의 처음을 삭제하는 경우

```
plist = pcurr->next;
free(pcurr);
```



2. 리스트의 중간을 삭제하는 경우

```
pprev->next = pcurr->next;
free(pcurr);
```

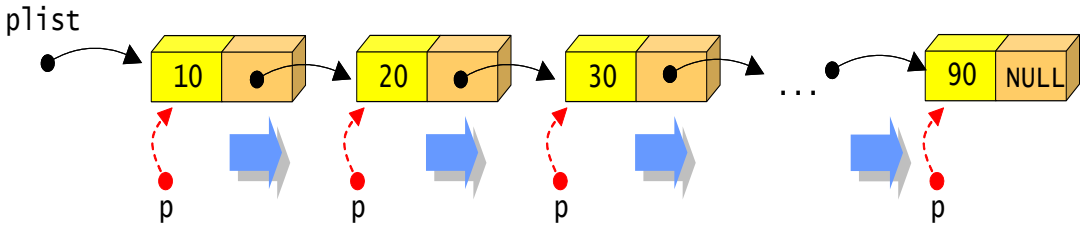


연결 리스트의 삭제 연산

```
NODE *delete_node(NODE *plist, NODE *pprev, NODE *pcurr)
{
        if( pprev == NULL )
                plist = pcurr->next;
        else
                pprev->next = pcurr->next;

        free(pcurr);
        return plist;
}
```

연결 리스트의 순회 연산

```
void print_list(NODE *plist)
{
        NODE *p;

        p = plist;
        printf("( ");

        while( p )
        {
                printf("%d ", p->data);
                p = p->next;
        }
        printf(")\n");
}
```



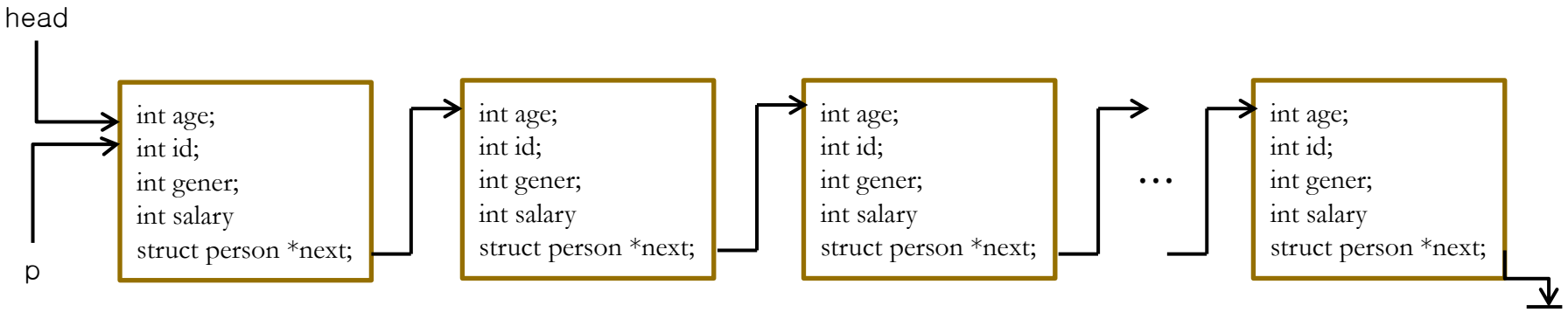
노드의 개수 세기

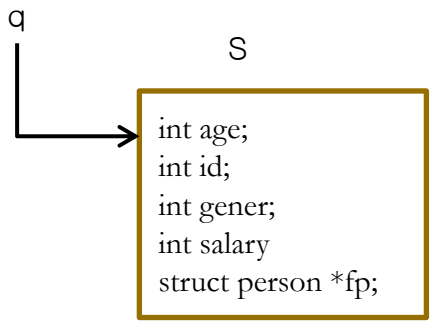
```
1.  int get_length(NODE *plist)
2.  {
3.          NODE *p;
4.          int length = 0;

5.          p = plist;

6.          while( p )
7.          {
8.                  length++;
9.                  p = p->next;
10.         }
11.         printf("리스트의 길이는 %d\n", length);
12.         return length;
13. }
```

합계 구하기

```
1.  int get_sum(NODE *plist)
2.  {
3.          NODE *p;
4.          int sum = 0;

5.          p = plist;

6.          while( p )
7.          {
8.                  sum += p->data;
9.                  p = p->next;
10.         }
11.         printf("리스트의 합계는 %d\n", sum);
12.         return sum;
13. }
```

연결리스트 생성






```
struct person {
        int     age;
        int     id;
        int     gender;
        int     salary;
struct  person  *next;
};
```

과제 7

(1) 10개의 노드를 가지는 연결리스트를 만드는 다음 프로그램을 malloc() 혹은 calloc() 함수를 사용하여 완성하라

```
int main()
{
        struct person {
                int             age;
                int             id;
                int             gender;
                int             salary;
        struct person     *next;
        };

        typedef struct person NODE;

        NODE *head;
        NODE *p;

        여기를 완성하시오
}
```

(2) 10개 노드를 가지는 연결리스트에서 6번 노드와 7번 노드 사이에 새로운 노드를 하나 생성하여 삽입하는 프로그램을 작성하라

(3) 10개 노드를 가지는 연결리스트에서 5번 노드를 삭제하는 프로그램을 작성하라. 단 맨처음 노드는 1번 노드이다