# 쉽게 풀어쓴 C언어 Express


제17장 동적메모리와 연결리스트

# 이번 장에서 학습할 내용

- 동적 메모리 할당의 이해
- 동적 메모리 할당 관련 함수
- 연결 리스트

# 동적 할당 메모리의 개념

- 프로그램이 메모리를 할당받는 방법
  - 정적(static)
  - 동적(dynamic)

# 정적 메모리 할당

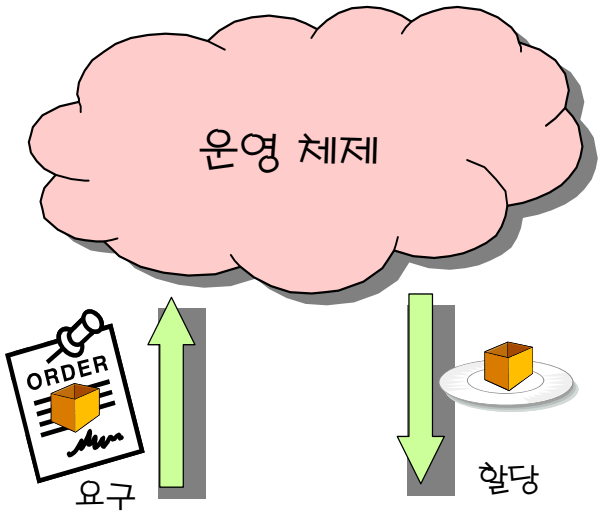
- 정적 메모리 할당
  - 프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당받는 것
  - 메모리의 크기는 프로그램이 시작하기 전에 결정

  - (예) int score_s[100];

  - 처음에 결정된 크기보다 더 큰 입력이 들어온다면 처리하지 못함
  - 더 작은 입력이 들어온다면 남은 메모리 공간은 낭비

# 동적 메모리 할당

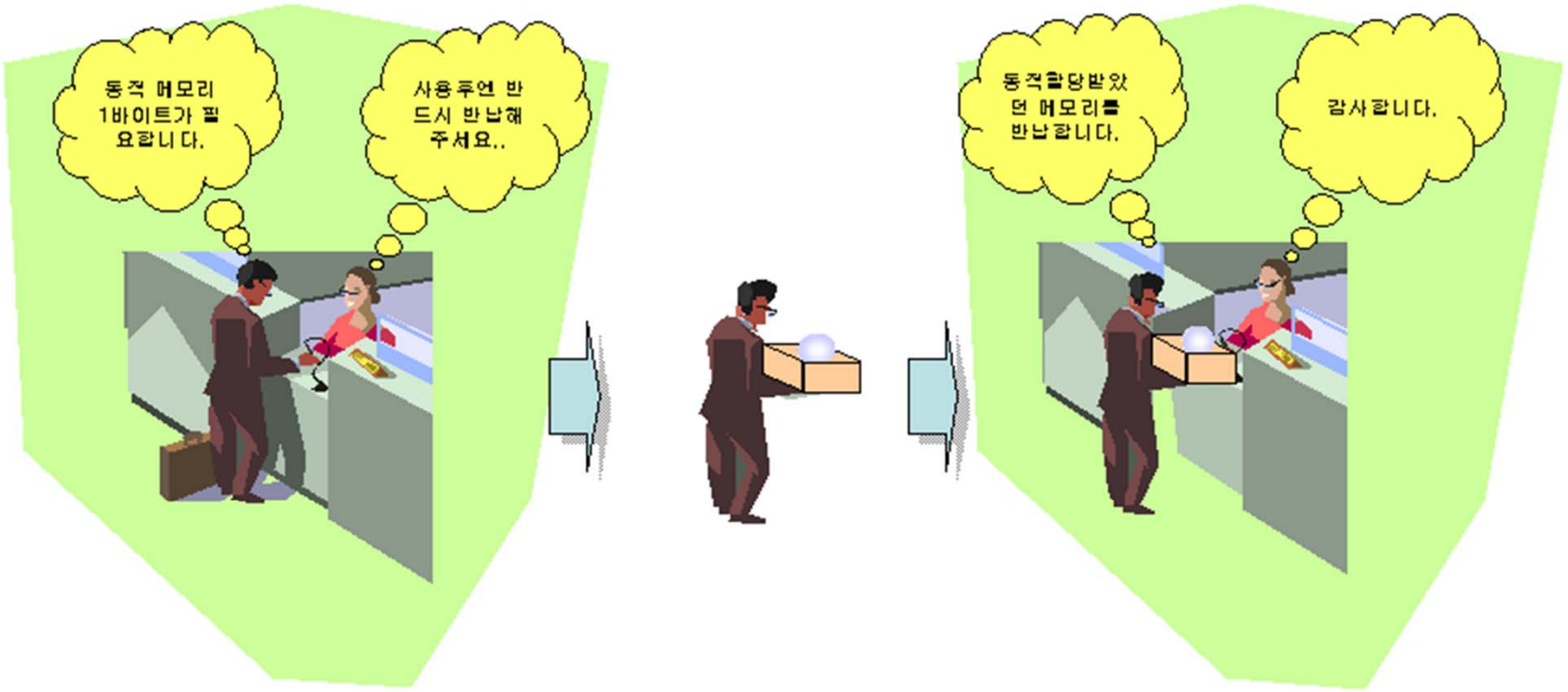
- 동적 메모리 할당
  - 실행 도중에 동적으로 메모리를 할당받는 것
  - 사용이 끝나면 시스템에 메모리를 반납
  - score = (int *) malloc(100*sizeof(int));
  - 필요한 만큼만 할당을 받고 메모리를 매우 효율적으로 사용
  - malloc() 계열의 라이브러리 함수를 사용




```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
 int *p;
 p = (int *)malloc( sizeof(int) );
 ...
}
```

프로그램

# 동적 메모리 할당 절차




동적 메모리 할당 동적 메모리 사용 동적 메모리 반납

# 동적 메모리 할당 예제

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        int *score;
        int i;
        score = (int *)malloc( 100*sizeof(int) );
        if( p == NULL )                    // 반환값이 NULL인지 검사
        {
                printf("동적 메모리 할당 오류\n");
                exit(1);
        }
        for(i=0 ; i<100 ; i++)
                score[i] = 0;
        free(p);
        return 0;
}
```

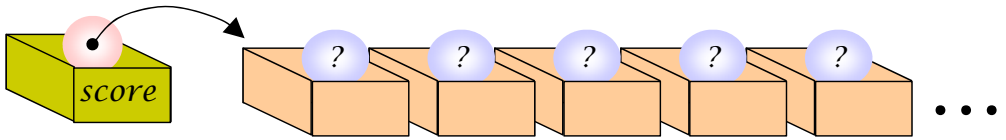
동적 메모리 할당

동적 메모리 해제

# 동적 메모리 할당

- void *malloc(size_t  size)
  - size는 바이트의 수
  - malloc()함수는 메모리 블럭의 첫 번째 바이트에 대한 주소를 반환
  - 만약 요청한 메모리 공간을 할당할 수 없는 경우에는 NULL값을 반환

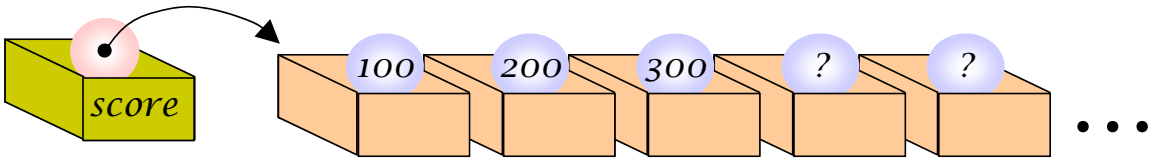
```
int *score;
score = (int *)malloc(100*sizeof(int));
if( score == NULL ){
        … // 오류 처리
}
```



# 동적 메모리 사용

- 할당받은 공간은 어떻게 사용하면 좋을까?

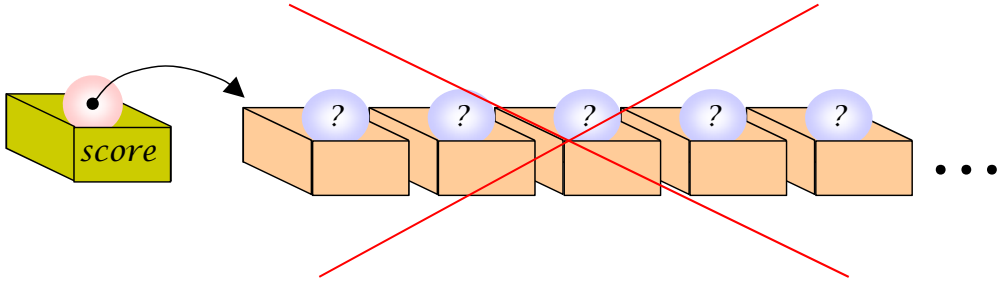
- 첫 번째 방법: 포인터를 통하여 사용
  - *score = 100;
  - *(score+1) = 200;
  - *(score+2) = 300;
  - …
- 두 번째 방법: 동적 메모리를 배열과 같이 취급
  - score[0] = 100;
  - score[1] = 200;
  - score[2] = 300;
  - …



# 동적 메모리 반납

- void free(void *ptr)
  - free()는 동적으로 할당되었던 메모리 블록을 시스템에 반납
  - ptr은 malloc()을 이용하여 동적 할당된 메모리를 가리키는 포인터

```
int *score;
score = (int *)malloc(100*sizeof(int));
...
free(score);
```



# 예제

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        char *pc = NULL;
        int i = 0;
        pc = (char *)malloc(100*sizeof(char));
        if( pc == NULL )        {
                printf("메모리 할당 오류\n");
                exit(1);
        }
        for(i=0;i<26;i++){
                pc[i] = 'a'+i;          // 알파벳 소문자를 순서대로 대입
        }
        pc[i] = 0;              // NULL 문자 추가
        printf("%s\n", pc);
        free(pc);
        return 0;
}
```

```
abcdefghijklmnopqrstuvwxyz
```

# 예제

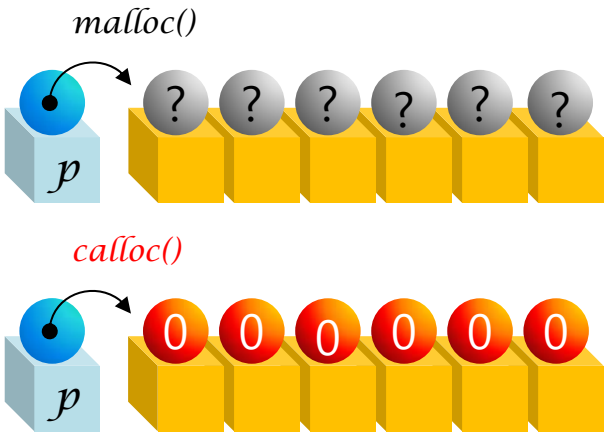
```
struct Book {
      int number;
      char title[10];
};

int main(void)
{
      struct Book *p;

      p = (struct Book *)malloc(2 * sizeof(struct Book));

      if(p == NULL){
        printf("메모리 할당 오류\n") ;
        exit(1);
      }

      p->number = 1;
      strcpy(p->title,"C Programming");

      (p+1)->number = 2;
      strcpy((p+1)->title,"Data Structure");

      free(p);
      return 0;
}
```

구조체 배열 할당

# calloc()

```
void *calloc(size_t    n,   size_t    size);
```

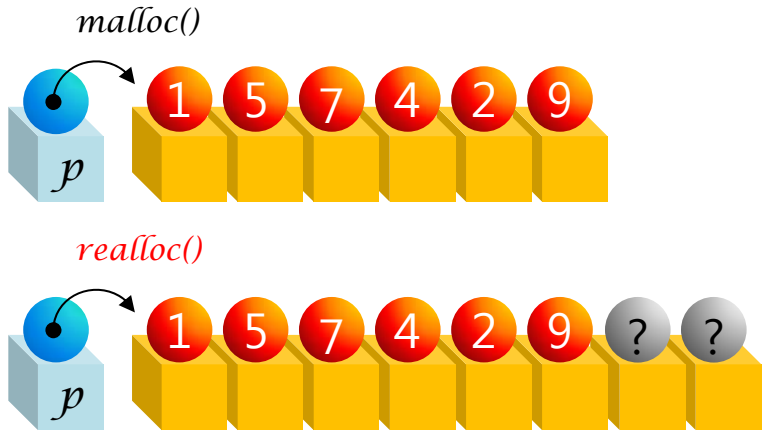
- calloc()은  0으로 초기화된 메모리 할당
- 항목 단위로 메모리를 할당
- (예)
- int *p;
- p = (int *)calloc(5, sizeof(int));



# realloc()

```
void *realloc(void *memblock,     size_t   size);
```

- realloc() 함수는 할당하였던 메모리 블록의 크기를 변경
- (예)
- int *p;
- p = (int *)malloc(5 * sizeof(int)));
- p = realloc(p,   7 * sizeof(int)));



# 예제

```
#include <stdio.h>
#include <stdlib.h>
#define INCREMENT 100                    // 한 번에 증가시키는 크기
double *scores = NULL;
int add_score(double new_score)
{
        static int limit = 0;            // 현재 동적 배열의 최대 크기
        static int count = 0;            // 현재 동적 배열의 크기
        if( count < limit ){
                scores[count++]= new_score;
        } else {
                int new_limit = limit + INCREMENT;
                double *new_array = realloc(scores, new_limit*sizeof(double));
                if( new_array == NULL ){
                        fprintf(stderr, "메모리 할당 오류\n");
                }
                else {                          scores = new_array;
                        limit = new_limit;
                        scores[count++] = new_score;
                }
                return count;
        }
}
```

# 예제

```
int main(void)
{
        int i, size;
        double value, total=0.0;
        size = 3;
        for(i=0;i<size;i++) {
                printf("성적: ");
                scanf("%lf", &value);
                add_score(value);
        }
        for(i=0;i<size;i++){
                total += scores[i];
        }
        printf("평균: %f\n", total/size);     free(scores);
        return 0;
}
```

```
성적: 10
성적: 20
성적: 30
평균: 20.000000
```

# 중간 점검

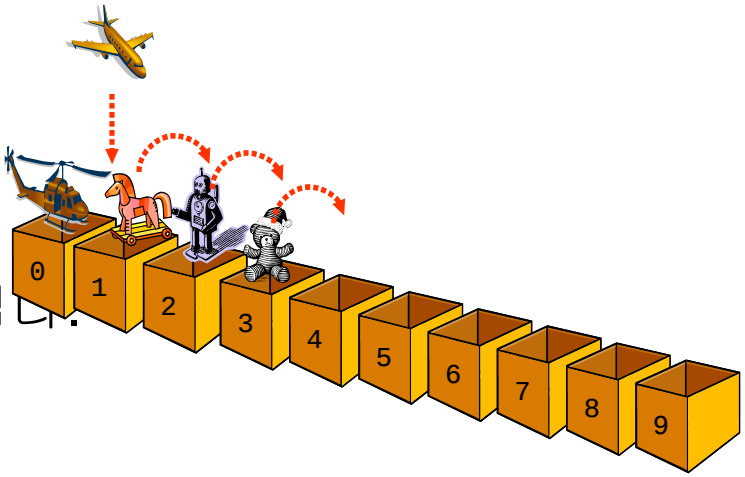
1. 프로그램의 실행 도중에 메모리를 할당받아서 사용하는 것을 _____이라고 한다.
2. 동적으로 메모리를 할당받을 때 사용하는 대표적인 함수는 _____이다.
3. 동적으로 할당된 메모리를 해제하는 함수는 _____이다.
4. 동적 메모리 함수의 원형은 헤더파일 _____에 정의되어 있다.

# 중간 점검

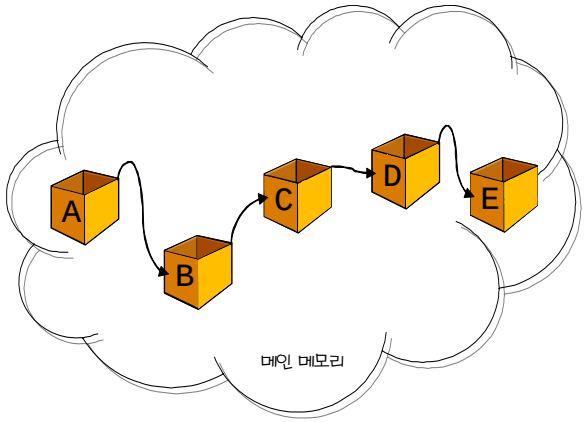
1. 동적 할당 후에 메모리 블록을 초기화하여 넘겨주는 함수는 _____이다.
2. 할당되었던 동적 메모리의 크기를 변경하는 함수는 _____이다.
3. 동적 메모리 할당에서의 단위는 _____이다.
4. malloc()이 반환하는 자료형은 _____이다.

# 연결 리스트

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

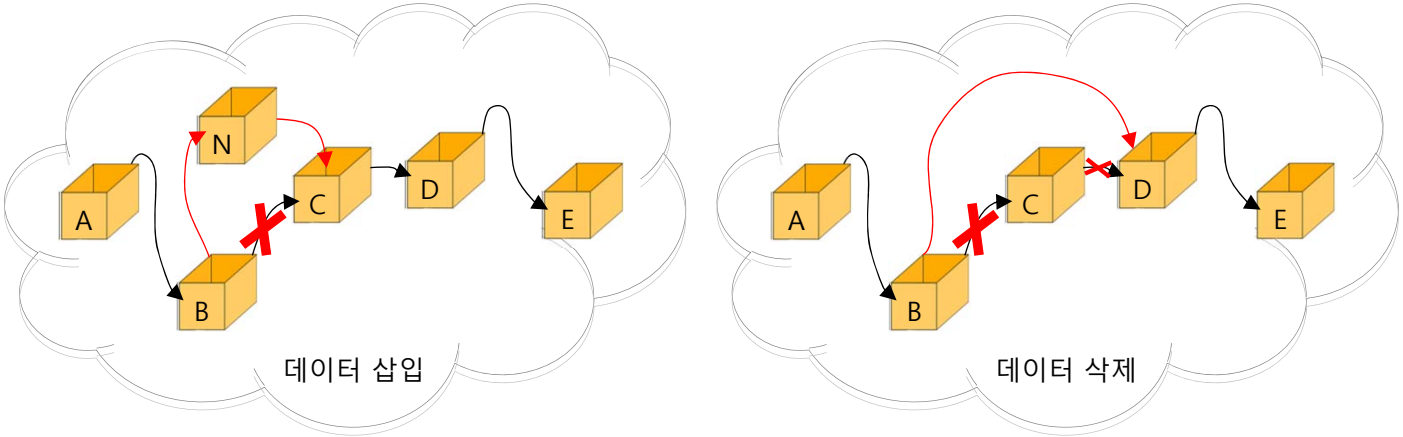



- 연결 리스트(linked list)
  - 각각의 원소가 포인터를 사용하여 다음 원소의 위치를 가리킨다.



# 연결 리스트의 장단점

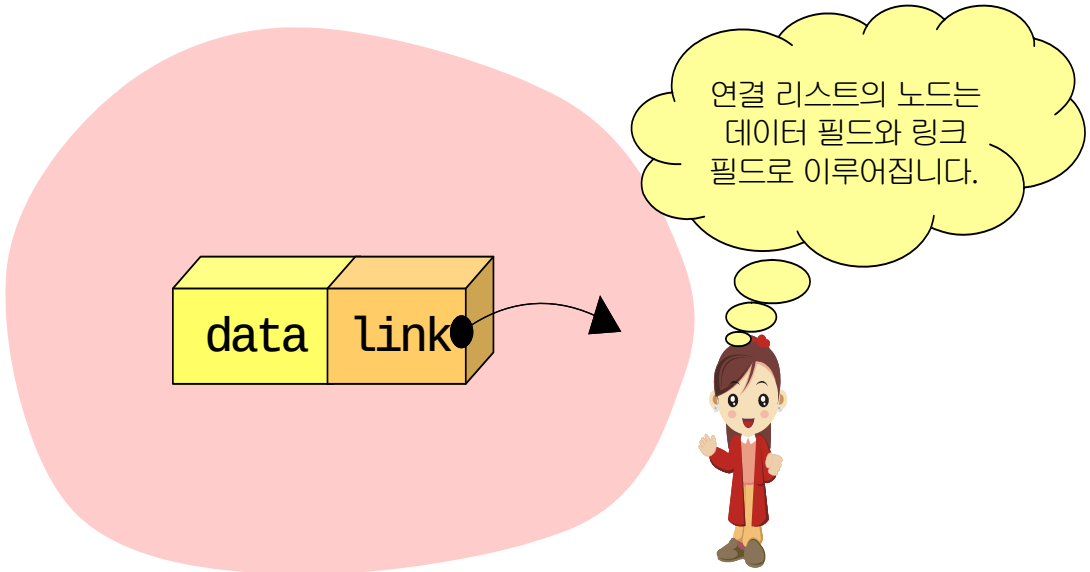
- 중간에 데이터를 삽입, 삭제하는 경우




- 데이터를 저장할 공간이 필요할 때마다 동적으로 공간을 만들어서 쉽게 추가
- 구현이 어렵고 오류가 나기 쉽다.
- 중간에 있는 데이터를 빠르게 가져올 수 없다.

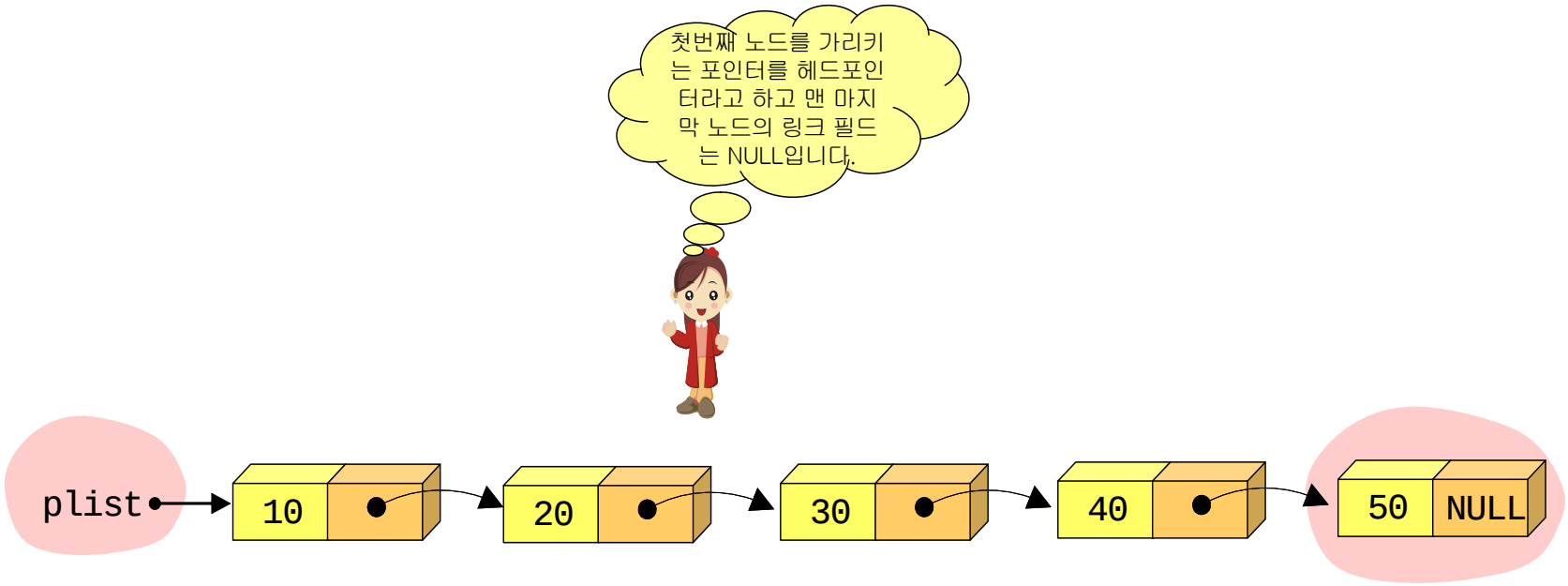
# 연결 리스트의 구조

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



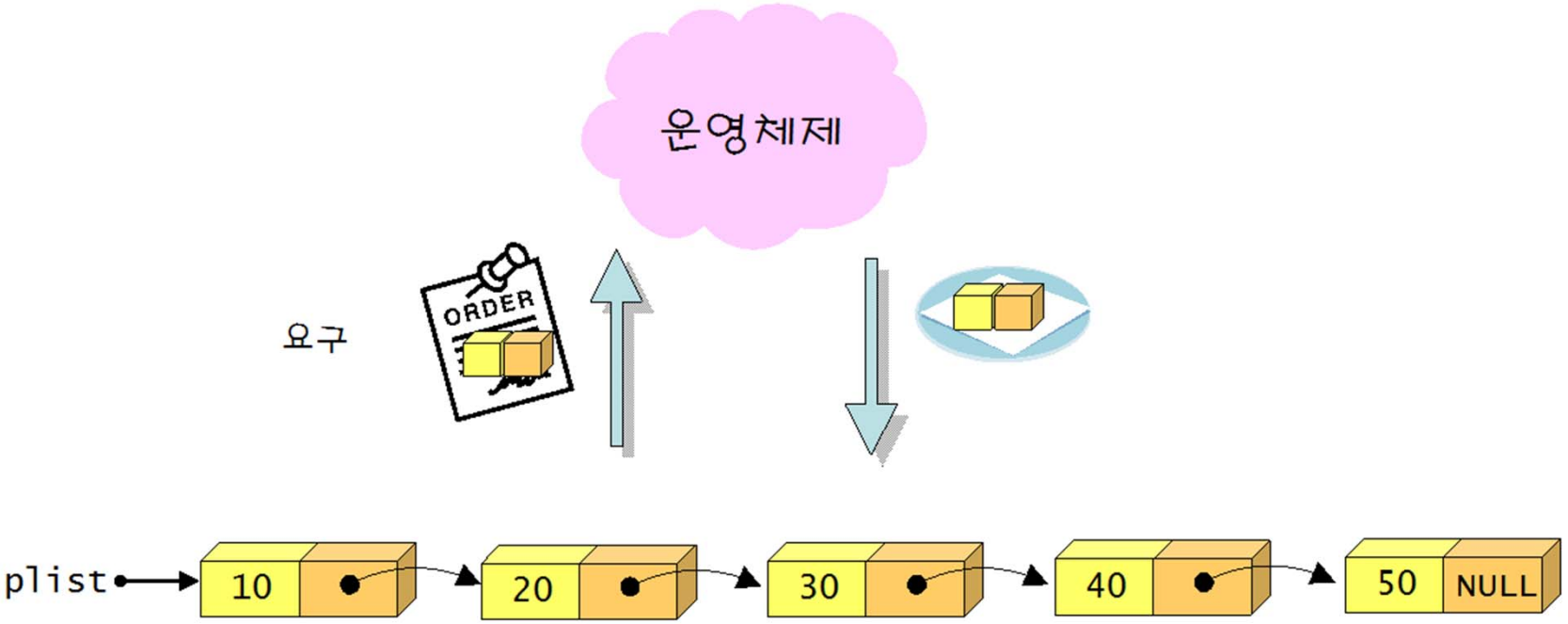
# 연결 리스트의 구조

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



# 노드 생성

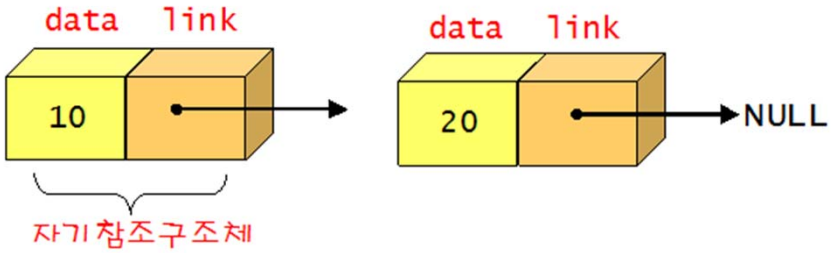
- 노드들은 동적으로 생성된다.



# 자기 참조 구조체

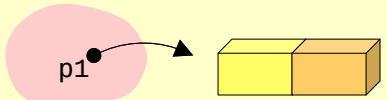
- 자기 참조 구조체**(self-referential structure)**는 특별한 구조체로서 구성 멤버 중에 같은 타입의 구조체를 가리키는 포인터가 존재하는 구조체

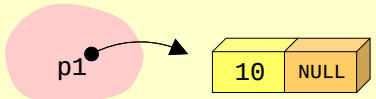
```
typedef  struct NODE {
        int data;
        struct NODE *link;
} NODE;
```

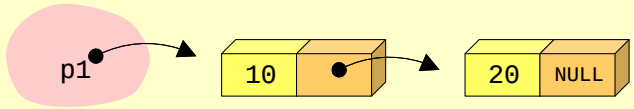


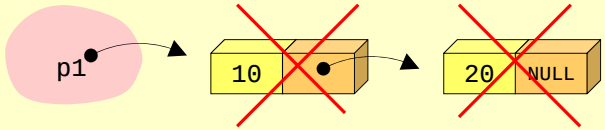
# 간단한 연결 리스트 생성

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

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

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

p2->link = NULL;
p1->link = p2;

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



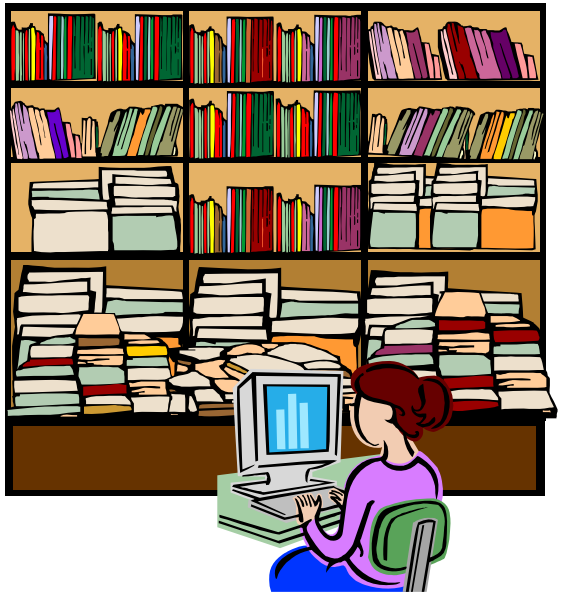



# 연결 리스트의 응용

- 소장하고 있는 책의 목록을 관리하는 프로그램을 작성
  - 연결 리스트를 사용하여서 작성



# 연결 리스트를 이용한 프로그램

```
1    #include <stdio.h>
2    #include <stdlib.h>
3    #include <string.h>
4
5    #define S_SIZE 50
6
7    typedef struct NODE {
8        char title[S_SIZE];
9        int year;
10       struct NODE *link;
11   } NODE;
12
```

노드의 타입을 정의

```
13    int main(void)
14    {
15       NODE *list = NULL;
16       NODE *prev, *p, *next;
17       char buffer[S_SIZE];
18       int year;
19
```

노드를 가리키는 포인터 정의

```
21      while(1)
22      {
23              printf("책의 제목을 입력하시오: (종료하려면 엔터)");
24              gets(buffer);
25              if( buffer[0] == '\0' )
26                      break
27
28              p = (NODE *)malloc(sizeof(NODE));
29              strcpy(p->title, buffer);
30              printf("책의 출판 연도를 입력하시오: ");
31              gets(buffer);
32              year = atoi(buffer);
33              p->year = year;
34
35              if( list == NULL )
36                      list = p;
37              else
38                      prev->link = p;
39              p->link = NULL;
40              prev = p;
41      }
```

동적 메모리 할당

```
44      // 연결 리스트에 들어 있는 정보를 모두 출력한다.
45      p = list;
46      while( p != NULL )
47      {
48              printf("책의 제목:%s 출판 연도:%d \n", p->title, p->year);
49              p = p->link;
50      }
51
52      // 동적 할당을 반납한다.
53      p = list;
54      while( p != NULL )
55      {
56              next = p->link;
57              free(p);
58              p = next;
59      }
```

동적 메모리 반납

# 실행 결과

*실행 결과*

```
책의 제목을 입력하시오: (종료하려면 엔터)컴퓨터개론
책의 출판연도를 입력하시오: 2006
책의 제목을 입력하시오: (종료하려면 엔터)C언어
책의 출판연도를 입력하시오: 2007
책의 제목을 입력하시오: (종료하려면 엔터)

책의 제목:컴퓨터개론 출판연도:2006
책의 제목:C언어 출판연도:2007
```

# 중간 점검

1. 연결 리스트에서 다음 노드는 _____로 가리킨다.
2. 연결 리스트의 일반적인 노드는 _____필드와 _____ 필드로 구성되어 있다.
3. 구조체의 멤버 중에 자기 자신을 가리키는 포인터가 존재하는 구조체를 _____라고 한다.
4. 배열과 연결 리스트의 가장 큰 차이점은 무엇인가?

# 실습: 영화 관리 프로그램

- 구조체 배열을 동적 메모리를 이용하여서 생성하고 여기에 영화 정보를 저장했다가 다시 화면에 예쁘게 출력하는 프로그램을 작성하여 보자.
- 영화 정보를 사용자로부터 받는다.

# 실행 결과

```
몇 편이나 저장하시겠습니까? 1
영화 제목:트랜스포머
영화 평점:8.3
========================
제목 평점
========================
트랜스포머 8.300000
========================
```

# 힌트

- 이 문제는 물론 정적 배열을 사용하면 아주 쉬운 문제이지만 여기서 동적 메모리 할당을 이용해보자. 동적 메모리를 사용하면 사용자가 원하는 만큼의 공간을 실행 시간에 할당받을 수 있다. 먼저 영화 정보를 다음과 같이 구조체로 표현한다.

```
typedef struct movie {        // 구조체 타입 정의
  char title[100];    // 영화 제목
  double rating;      // 영화 평점
} MOVIE;
```

- 사용자가 입력하고자 하는 영화의 수를 size에 입력받은 후에, 동적으로 할당

```
movies = (MOVE *)malloc(sizeof(MOVIE)* size);          // 동적 메모리 할당
```

# 예제

```
#include <stdio.h>
typedef struct movie { // 구조체 타입 정의
        char title[100];        // 영화 제목
        double rating;          // 영화 평점
} MOVIE;
int main(void)
{
        MOVIE *movies;          // 동적 메모리 공간을 가리키는 포인터
        int size, i;
        printf("몇 편이나 저장하시겠습니까? ");
        scanf("%d", &size);
        movies = (MOVIE *)malloc(sizeof(MOVIE)* size);          // 동적 메모리 할당
        if( movies == NULL ){
                printf("동적 메모리 할당 오류);
                exit(1);
        }
```

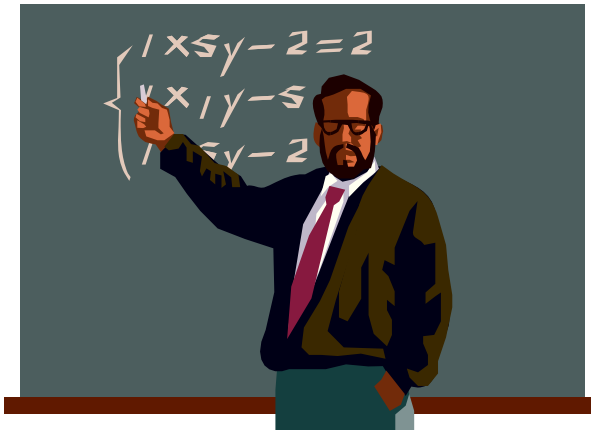
# 예제

```
        for(i=0; i<size ;i++) {                    // size편의 영화 정보 입력
                printf("영화 제목);
                fflush(stdin);                              // 입력 버퍼를 비운다.
                gets(movies[i].title);    // 영화 제목에는 빈칸이 있을 수 있다.
                printf("영화 평점);
                scanf("%lf", &(movies[i].rating));
        }
        printf("======================\n");
        printf("제목                평점);
        printf("======================\n");
        for(i=0;i<size;i++)
                printf("%s \t %f", movies[i].title, movies[i].rating);
        printf("\n======================\n");
        free(movies);                                       // 동적 메모리 공간 해제
        return 0;
}
```

# 도전문제

- 사용자가 입력한 데이터를 파일에 기록하는 코드를 추가해보자.
- 프로그램이 시작할 때 파일에서 데이터를 읽어오는 코드도 추가하여 보자.

# Q & A



# 감사합니다.