

개정3판

Visual
Studio
2017

쉽게 풀어쓴

C언어 EXPRESS

천인국 지음

제17장 동적 메모리와 연결 리스트



이번장에서 학습할 내용



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

동적 메모리 할당에
대한 개념을
이해하고 응용으로
연결 리스트를
학습합니다.





동적 할당 메모리의 개념

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

메모리도 필요할 때마다
요청해서 사용하면 좋은데...





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



동적 메모리 할당 절차

동적 메모리
1바이트가 필요합니다.



사용후엔 반드시
반납해주세요.



동적 할당받았던
메모리를 반납합니다.



감사합니다.



동적 메모리 할당

Syntax: 동적메모리할당

예

```
int *p;
```

```
p = (int *)malloc(100*sizeof(int));
```

```
// 100개의 정수 할당
```

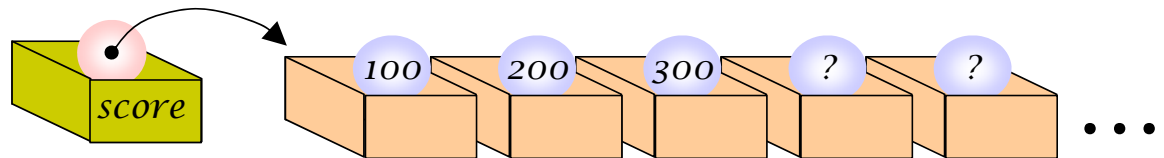
동적 메모리의 주소

필요한 바이트 수



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

Syntax: 동적메모리해제

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

score가 가리키는 동적 메모리를 반납한다.



예제 #1

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int *list;
    list = (int *)malloc(3 * sizeof(int));
    if (list == NULL) { // 반환값이 NULL인지 검사
        printf("동적 메모리 할당 오류\n");
        exit(1);
    }
    list[0] = 10;
    list[1] = 20;
    list[2] = 30;
    free(list);
    return 0;
}
```

동적 메모리 할당

동적 메모리 해제



예제 #2

- 성적 처리 프로그램을 작성한다고 하자. 사용자한테 학생이 몇 명인지 물어보고 적절한 동적 메모리를 할당한다. 사용자로부터 성적을 받아서 저장하였다가 다시 출력한다.

```
학생의 수: 3
학생 #1 성적:100
학생 #2 성적:90
학생 #3 성적:80
=====
학생 #1 성적:100
학생 #2 성적:90
학생 #3 성적:80
=====
```



예제 #2

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int *list;
    int i, students;
    printf("학생의 수: ");
    scanf("%d", &students);
    list = (int *)malloc(students * sizeof(int));
    if (list == NULL) { // 반환값이 NULL인지 검사
        printf("동적 메모리 할당 오류\n");
        exit(1);
    }
}
```



예제 #2

```
for (i = 0; i < students; i++) {  
    printf("학생 #%d 성적: ", i + 1);  
    scanf("%d", &list[i]);  
}  
printf("=====\n");  
for (i = 0; i < students; i++) {  
    printf("학생 #%d 성적: %d \n", i + 1, list[i]);  
}  
printf("=====\n\n");  
free(list);  
return 0;  
}
```



예제 #3

- 구조체를 저장할 수 있는 공간을 할당받아서 사용해본다.



예제

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Book {
    int number;
    char title[50];
};
```

```
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[0].number = 1;
```

```
    strcpy(p[0].title, "C Programming");
```

```
    p[1].number = 2;
```

```
    strcpy(p[1].title, "Data Structure");
```

```
    free(p);
```

```
    return 0;
```

```
}
```

구조체 배열 할당



10개의 문자열을 저장하는 동적 메모리

- 최대 10개의 문자열을 저장할 수 있는 동적 메모리를 생성해보자.

```
문자열 0: test string  
문자열 1: test string  
문자열 2: test string  
문자열 3: test string  
문자열 4: test string  
문자열 5: test string  
문자열 6: test string  
문자열 7: test string  
문자열 8: test string  
문자열 9: test string
```



예제

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(void)
{
    char* list[10];
    for (int i = 0; i < 10; i++) {
        list[i] = (char*)malloc(100 * sizeof(char));
        if (list[i] == NULL) {
            printf("malloc() 실패!\n\n");
            exit(1);
        }
        strcpy(list[i], "test string");
    }
    for (int i = 0; i < 10; i++) {
        printf("문자열 %d: %s\n", i, list[i]);
    }
    return 0;
}
```

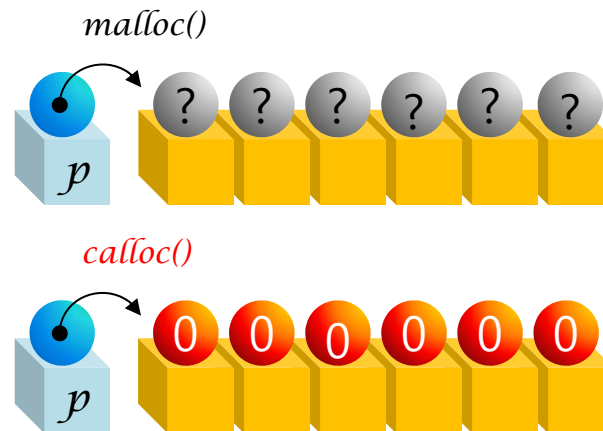


calloc()

- calloc()은 0으로 초기화된 메모리 할당
- calloc()은 항목 단위로 메모리를 할당
- (예)

```
int *p;
```

```
p = (int *)calloc(5, sizeof(int));
```





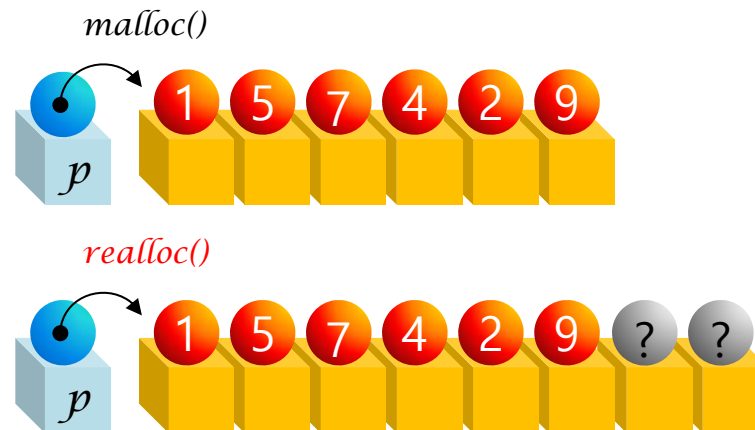
realloc()

- realloc() 함수는 할당하였던 메모리 블록의 크기를 변경
- (예)

```
int *p;
```

```
p = (int *)malloc(5 * sizeof(int));
```

```
p = realloc(p, 7 * sizeof(int));
```





예제

- 아래의 코드는 2개의 정수를 저장하는 동적 메모리를 할당을 받는다.
이 공간을 정수 3개를 저장할 수 있는 공간으로 확장한다

```
정수 2개를 저장할 공간이 필요  
정수 3개를 저장할 공간으로 확장  
10 20 30
```



예제

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

int main()
{
    printf("정수 2개를 저장할 공간이 필요 \n");
    int *list = (int *)malloc(sizeof(int) * 2);
    int i;
    int *list_new;

    list[0] = 10;
    list[1] = 20;
    printf("정수 3개를 저장할 공간으로 확장 \n");

    list_new = (int *)realloc(list, sizeof(int) * 3);
    list_new[2] = 30;

    for (i = 0; i < 3; i++)
        printf("%d ", list_new[i]);
    printf("\n");
    return 0;
}
```



정가 점검

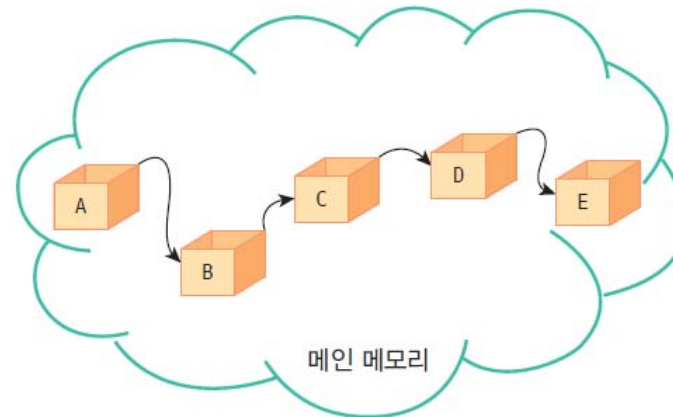
1. 동적 할당 후에 메모리 블록을 초기화하여 넘겨주는 함수는 _____이다.
2. 할당되었던 동적 메모리의 크기를 변경하는 함수는 _____이다.
3. 동적 메모리 할당에서의 단위는 _____이다.
4. `malloc()`이 반환하는 자료형은 _____이다.





연결 리스트

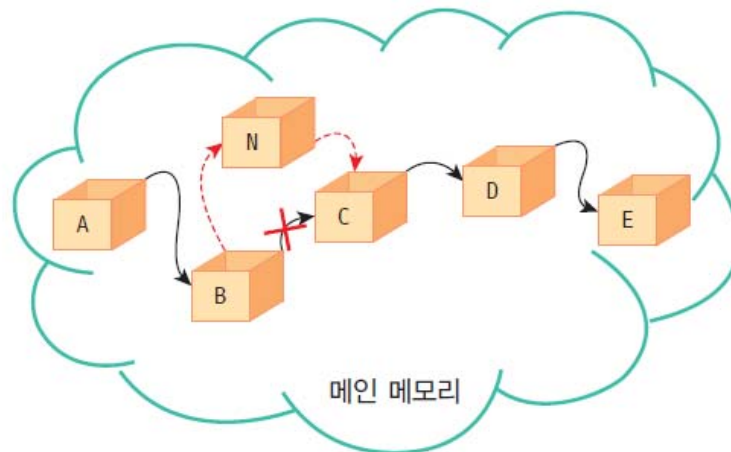
- 배열(array)
 - ▣ 장점: 구현이 간단하고 빠르다
 - ▣ 단점: 크기가 고정된다. 중간에서 삽입, 삭제가 어렵다.
- 연결 리스트(linked list)
 - ▣ 각 항목이 포인터를 사용하여 다음 항목을 가리킨다.
 - ▣ 랜덤 접근이 어렵다.





연결 리스트의 장단점

- 중간에 데이터를 삽입, 삭제하기 쉽다.

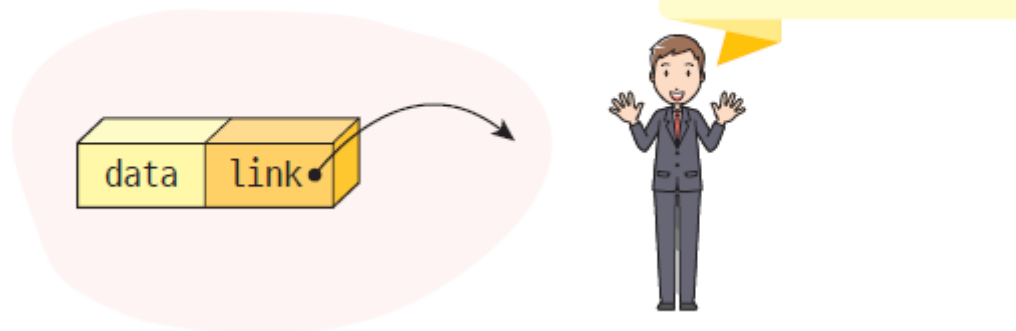


- 데이터를 저장할 공간이 필요할 때마다 동적으로 공간을 만들어서 쉽게 추가
- 구현이 어렵고 오류가 나기 쉽다.
- 중간에 있는 데이터를 빠르게 가져올 수 없다.



연결 리스트의 구조

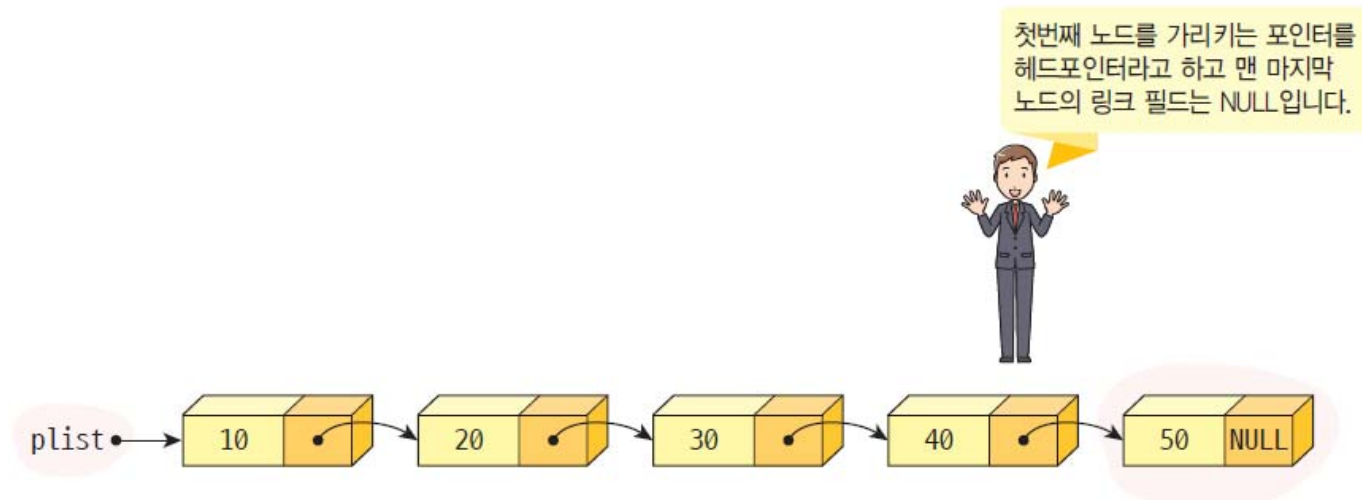
- 노드(node) = 데이터 필드(data field)+ 링크 필드(link field)





연결 리스트의 구조

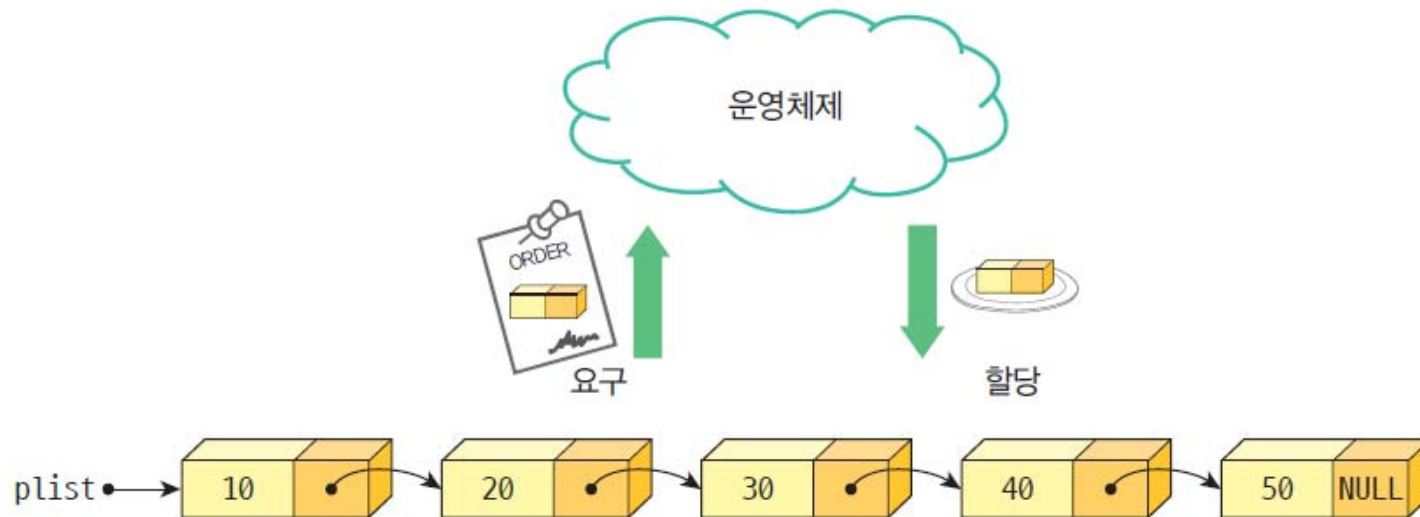
- 헤드 포인터(head pointer): 첫번째 노드를 가리키는 포인터





노드 생성

- 노드들은 동적으로 생성된다.

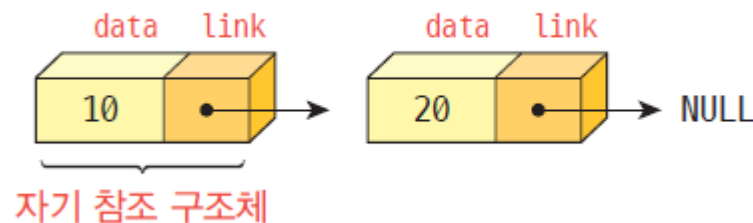




자기 참조 구조체

- 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

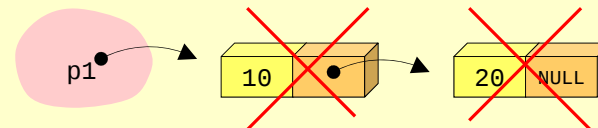
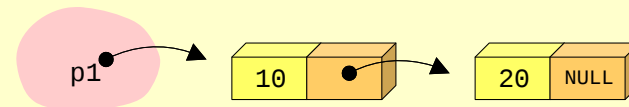
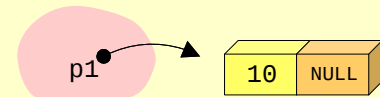
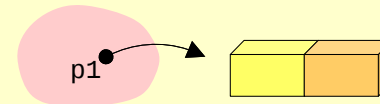
```
NODE *p1;  
p1 = (NODE *)malloc(sizeof(NODE));
```

```
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 사용하여 작성해보자.





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



연결 리스트를 이용한 프로그램

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

#define S_SIZE 50

typedef struct NODE {
    char title[S_SIZE];
    int year;
    struct NODE *link;
} NODE;
```



```
int main(void)
{
    NODE *list = NULL;
    NODE *prev, *p, *next;
    char buffer[S_SIZE];
    int year;
```



```
while(1) {
    printf("책의 제목을 입력하시오: (종료하려면 엔터)");
    gets(buffer);
    if( buffer[0] == '\0' )
        break;
    p = (NODE *)malloc(sizeof(NODE));
    strcpy(p->title, buffer);
    printf("책의 출판 연도를 입력하시오: ");
    gets(buffer);
    year = atoi(buffer);
    p->year = year;
    if( list == NULL )        // 리스트가 비어 있으면
        list = p;           // 새로운 노드를 첫번째 노드로 만든다.
    else                      // 리스트가 비어 있지 않으면
        prev->link = p;      // 새로운 노드를 이전 노드의 끝에
    p->link = NULL;          // 새로운 노드의 링크 필드를 NULL로 설정
    prev = p;
}
```



```
printf("\n");  
// 연결 리스트에 들어 있는 정보를 모두 출력한다.  
p = list;  
while( p != NULL )  
{  
    printf("책의 제목:%s 출판 연도:%d \n", p->title, p->year);  
    p = p->link;  
}  
// 동적 할당을 반납한다.  
p = list;  
while( p != NULL )  
{  
    next = p->link;  
    free(p);  
    p = next;  
}  
return 0;  
}
```



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



장간 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?





mini project: 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? 2
영화 제목:트랜스포머
영화 평점:8.3
영화 제목: 저스티스 리그
영화 평점: 9.0

```
=====
제목 평점
=====
트랜스포머 8.300000
저스티스 리그 9.000000
=====
```



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 **size**에 입력받은 후에, 동적으로 할당

```
movies = (MOVIE *)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 \t 평점 \n");
printf("=====\n");
for(i=0; i<size; i++)
    printf("%s \t %f \n", movies[i].title, movies[i].rating);
printf("\n=====\n");
free(movies);                     // 동적 메모리 공간 해제
return 0;
}
```



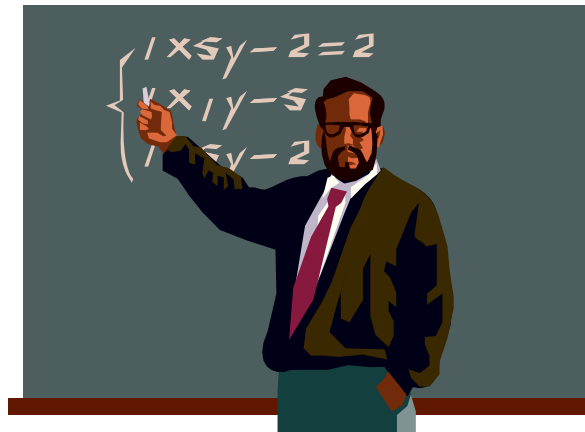
도전문제

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





Q & A





THANK
YOU