14주차 2차시 메모리 관리

【학습목표】

1. C언어의 메모리 구조를 설명할 수 있다.
2. 메모리의 동적 할당을 설명할 수 있다.

학습내용1 : C언어의 메모리 구조

1. 서식에 따른 데이터 입출력: fprintf, fscanf

```
char name[10]="홍길동";
char sex='M';
int age=24;
fprintf(fp, "%s %c %d", name, sex, age);
```

fprintf 함수를 이용하면 어떻게 텍스트 & 바이너리 데이터를 동시에 출력할 수 있을까?
fprintf 함수는 printf 함수와 그 사용방법이 매우 유사하다. 다만 fp를 대상으로 조합이 된 문자열이 출력(저장)될 뿐이다.

```
char name[10];
char sex;
int age;
fscanf(fp, "%s %c %d", name, &sex, &age);
```

fscanf 함수를 이용하면 어떻게 텍스트 & 바이너리 데이터를 동시에 입력할 수 있을까?
sprintf 함수는 printf 함수와 그 사용방법이 매우 유사하다. 다만 fp를 대상으로 서식문자의 조합 형태대로 데이터가 입력될 뿐이다.

2. fprintf & fscanf 관련 예제

```
int main(void)
{
    char name[10];
    char sex;
    int age;
    FILE * fp=fopen("friend.txt", "wt");
    int i;
    for(i=0; i<3; i++)
    {
        printf("이름 성별 나이 순 입력: ");
        scanf("%s %c %d", name, &sex, &age);
        getchar();  // 버퍼에 남아있는 \n의 소멸을 위해서
        fprintf(fp, "%s %c %d", name, sex, age);
    }
    fclose(fp);
    return 0;
}
```

저장하는 데이터가 문자열이므로 텍스트 모드로 개방한다!

```
이름 성별 나이 순 입력: 정은영 F 22
이름 성별 나이 순 입력: 한수정 F 26
이름 성별 나이 순 입력: 이영호 M 31
```

```
int main(void)
{
    char name[10];
    char sex;
    int age;
    FILE * fp=fopen("friend.txt", "rt");
    int ret;
    while(1)
    {
        ret=fscanf(fp, "%s %c %d", name, &sex, &age);
        if(ret==EOF)
            break;
        printf("%s %c %d \n", name, sex, age);
    }
    fclose(fp);
    return 0;
}
```

```
정은영 F 22
한수정 F 26
이영호 M 31
```

3. Text/Binary의 집합체인 구조체 변수 입출력

```
typedef struct fren
{
    char name[10];
    char sex;
    int age;
} Friend;
```

```
int main(void)
{
    FILE * fp;
    Friend myfren1;
    Friend myfren2;
    /*** file write ***/
    fp=fopen("friend.bin", "wb");
    printf("이름, 성별, 나이 순 입력: ");
    scanf("%s %c %d", myfren1.name, &(myfren1.sex), &(myfren1.age));
    fwrite((void*)&myfren1, sizeof(myfren1), 1, fp);
    fclose(fp);
    /*** file read ***/
    fp=fopen("friend.bin", "rb");
    fread((void*)&myfren2, sizeof(myfren2), 1, fp);
    printf("%s %c %d \n", myfren2.name, myfren2.sex, myfren2.age);
    fclose(fp);
    return 0;
}
```

바이너리 모드로 통째로 구조체 변수를 저장

바이너리 모드로 통째로 구조체 변수를 복원

```
이름, 성별, 나이 순 입력: Jungs M 27
Jungs M 27
```

구조체 변수의 입출력은 생각보다 어렵지 않다.
fread & fwrite 함수 기반으로 통째로 입출력 하면 된다.

* 파일 위치 지시자란?

- FILE 구조체의 멤버 중 하나.
- read 모드로 오픈 된 파일 위치 지시자: “어디까지 읽었더라?”에 대한 답
- write 모드로 오픈 된 파일 위치 지시자: “어디부터 이어서 쓰더라?”에 대한 답
- 즉, Read/Write에 대한 위치 정보를 갖고 있다.

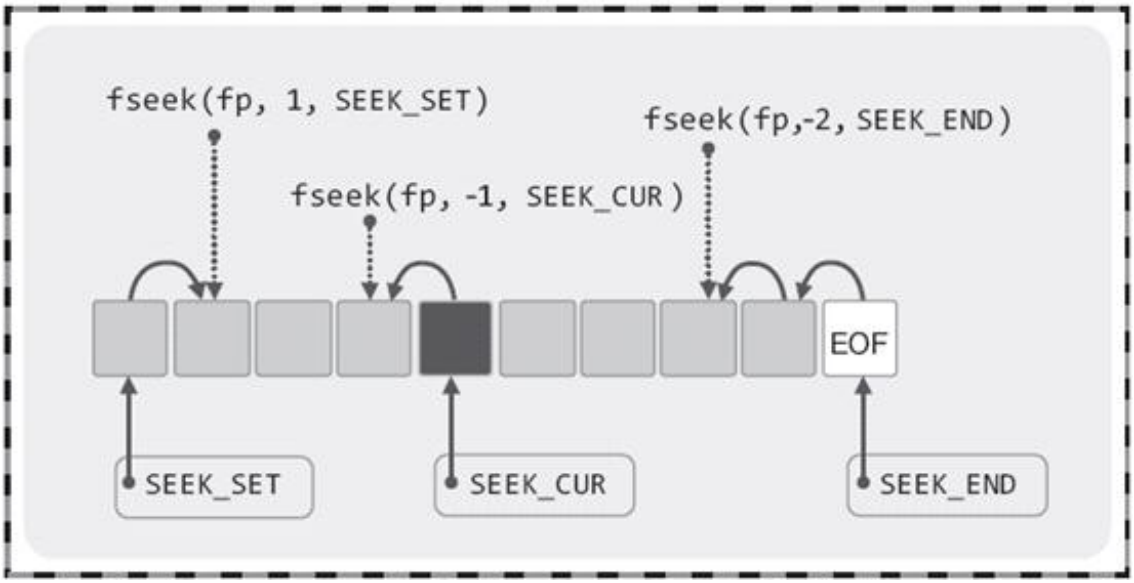
따라서 파일 입출력과 관련이 있는 fputs, fread, fwrite와 같은 함수가 호출될 때마다 파일 위치 지시자의 참조 위치는 변경이 된다.

4. 파일 위치 지시자의 이동: fseek

■ 파일 위치 지시자의 참조 위치를 변경시키는 함수

```
#include <stdio.h>
int fseek(FILE * stream, long offset, int wherefrom);
```

➔ 성공 시 0, 실패 시 0이 아닌 값을 반환



■ fseek 함수의 호출결과로 인한 파일 위치 지시자의 이동 결과

매개변수 wherefrom 이...	파일 위치 지시자는...
SEEK_SET(0) 이라면	파일 맨 앞에서부터 이동을 시작
SEEK_CUR(1) 이라면	현재 위치에서부터 이동을 시작
SEEK_END(2) 이라면	파일 맨 끝에서부터 이동을 시작

5. fseek 함수의 호출의 예

```
int main(void)
{
    /* 파일생성 */
    FILE * fp=fopen("text.txt", "wt");
    fputs("123456789", fp);
    fclose(fp);
    /* 파일개방 */
    fp=fopen("text.txt", "rt");
    /* SEEK_END test */
    fseek(fp, -2, SEEK_END);
    putchar(fgetc(fp));
    /* SEEK_SET test */
    fseek(fp, 2, SEEK_SET);
    putchar(fgetc(fp));
    /* SEEK_CUR test */
    fseek(fp, 2, SEEK_CUR);
    putchar(fgetc(fp));
    fclose(fp);
    return 0;
}
```

123456789 e(eof)
123456789 e(eof)
123456789 e(eof)
123456789 e(eof)
123456789 e(eof)
123456789 e(eof)

836

6. 현재 파일 위치 지시자의 위치는?: ftell

■ 현재 파일 위치자의 위치 정보를 반환하는 함수!

```
#include <stdio.h>
long ftell(FILE * stream);
```

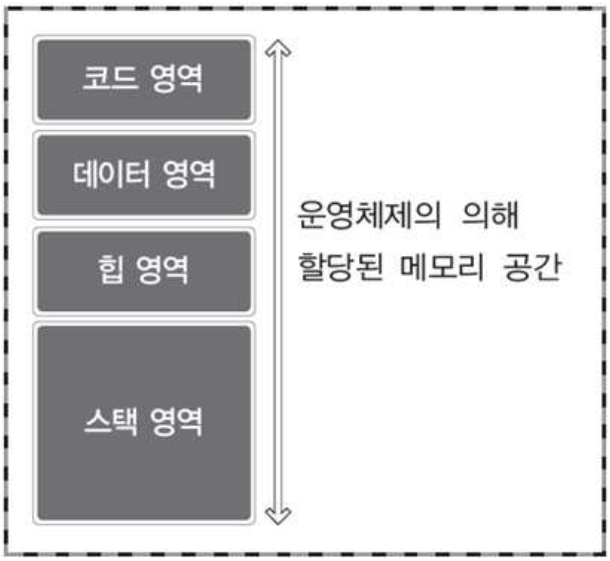
➡ 파일 위치 지시자의 위치 정보 반환

```
int main(void)
{
    long fpos;
    int i;
    /* 파일생성 */
    FILE * fp=fopen("text.txt", "wt");
    fputs("1234-", fp);
    fclose(fp);
    /* 파일개방 */
    fp=fopen("text.txt", "rt");
    for(i=0; i<4; i++)
    {
        putchar(fgetc(fp));
        fpos=ftell(fp);현재 위치 저장
맨 뒤로이동fseek(fp, -1, SEEK_END);
        putchar(fgetc(fp));
        fseek(fp, fpos, SEEK_SET);
    }                    저장해 놓은위치 복원
    fclose(fp);
    return 0;
}
```

```
1-2-3-4-
```

학습내용2 : 메모리의 동적 할당

1. 메모리의 구성

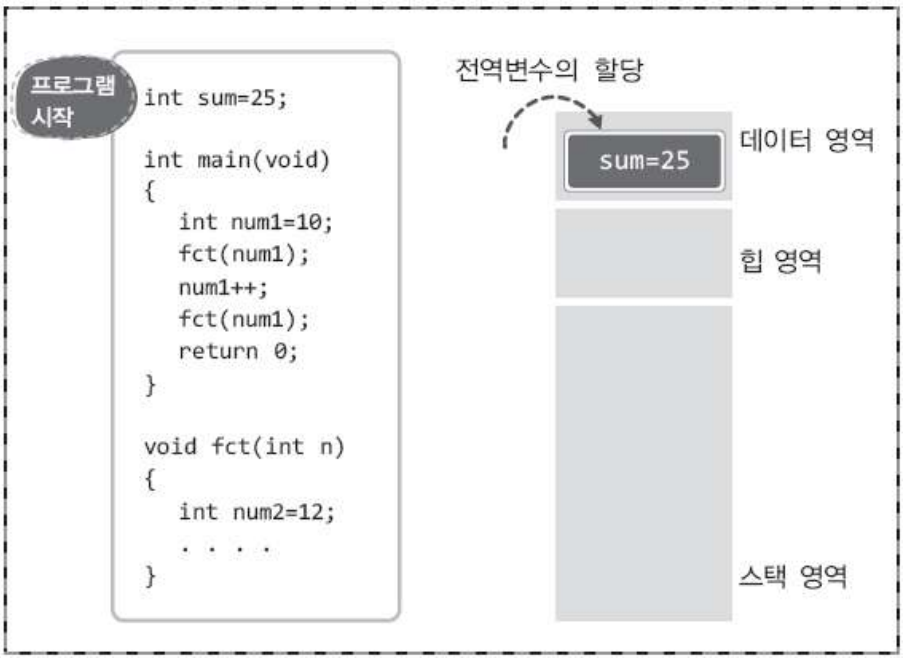


■ 메모리 공간을 나눠놓은 이유는 커다란 서랍장의 수납공간이 나뉘어 있는 이유와 유사하다.
■ 메모리 공간을 나눠서 유사한 성향의 데이터를 묶어서 저장을 하면, 관리가 용이해지고 메모리의 접근 속도가 향상된다.

2. 메모리 영역별로 저장되는 데이터의 유형

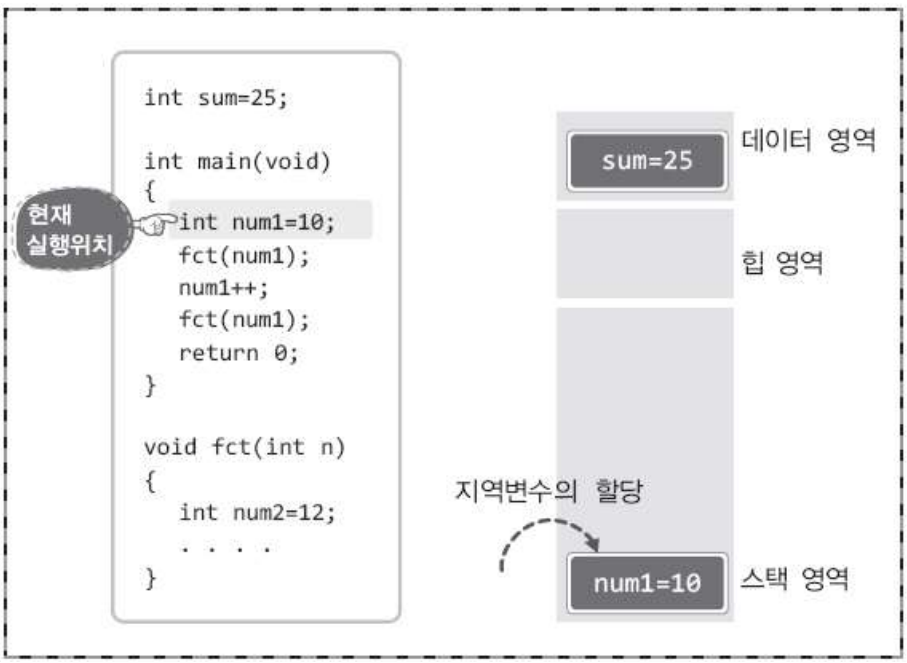
코드 영역	실행할 프로그램의 코드가 저장되는 메모리 공간. CPU는 코드 영역에 저장된 명령문을 하나씩 가져다가 실행
데이터 영역	전역변수와 static 변수가 할당되는 영역. 프로그램 시작과 동시에 할당되어 종료 시까지 남아있는 특징의 변수가 저장되는 영역
힙 영역	프로그래머가 원하는 시점에 메모리 공간에 할당 및 소멸을 하기 위한 영역
스택 영역	지역변수와 매개변수가 할당되는 영역 함수를 빠져나가면 소멸되는 변수를 저장하는 영역

3. 프로그램의 실행에 따른 메모리의 상태 변화 1



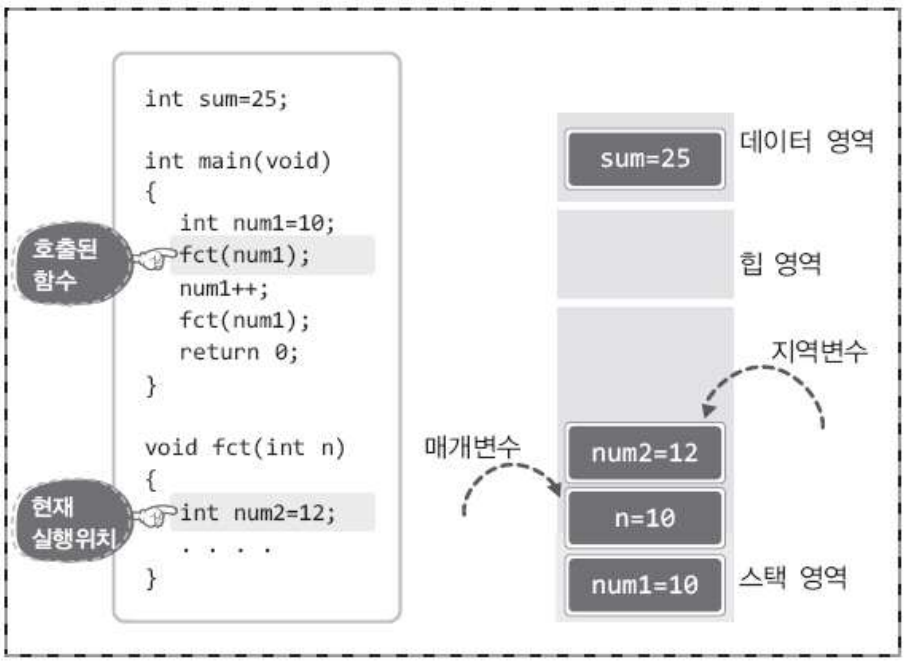
프로그램의 시작: 전역변수의 할당 및 초기화

4. 프로그램의 실행에 따른 메모리의 상태 변화 2



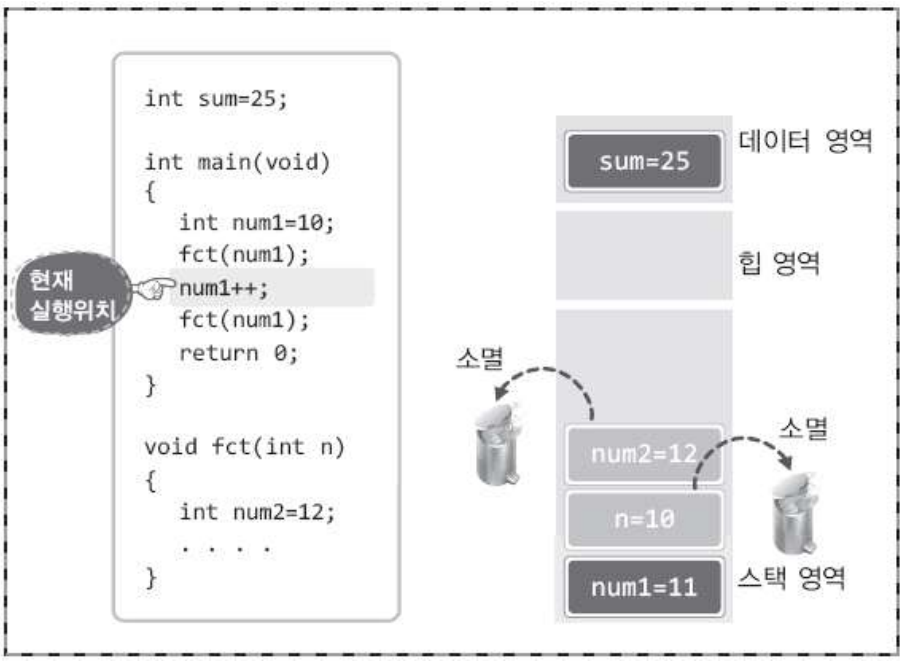
main 함수의 호출 및 실행

5. 프로그램의 실행에 따른 메모리의 상태 변화 3



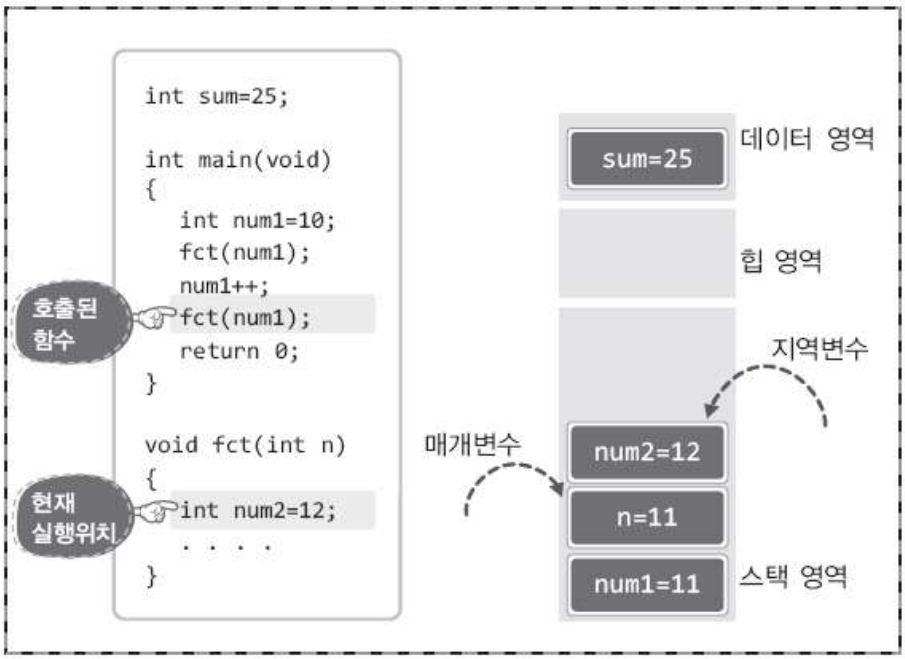
fct 함수의 호출

6. 프로그램의 실행에 따른 메모리의 상태 변화 4



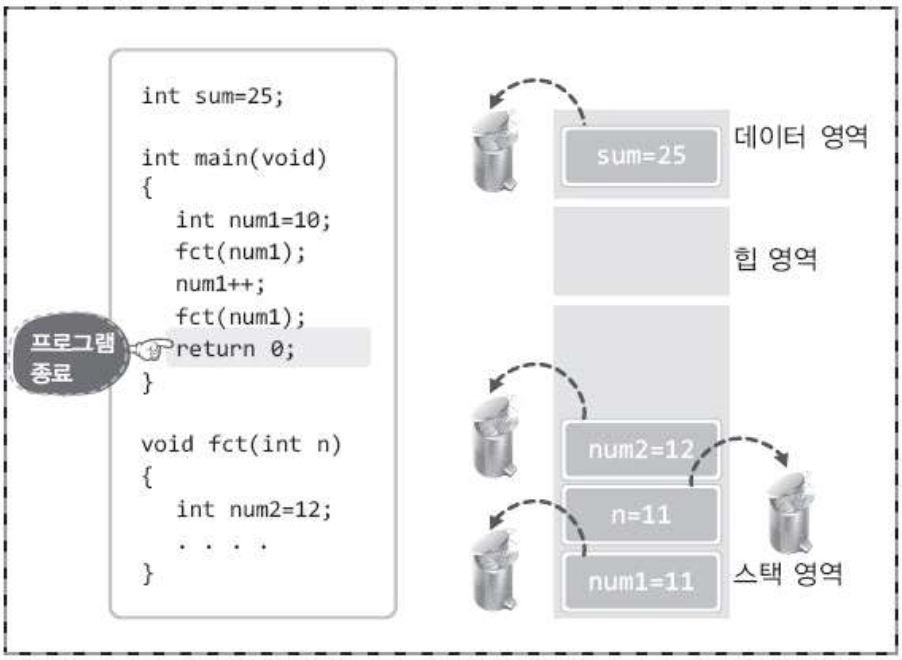
fct 함수의 반환 그리고 main 함수 이어서 실행

7. 프로그램의 실행에 따른 메모리의 상태 변화 5



fct 함수의 재호출 및 실행

8. 프로그램의 실행에 따른 메모리의 상태 변화 6



fct 함수의 반환 및 main 함수의 반환, 프로그램 종료

■ 함수의 호출순서가 main → fct1 → fct2이라면 스택의 반환은(지역변수의 소멸은) 그의 역순인 fct2 → fct1 → main으로 이루어진다는 특징을 기억하자!

9. 전역변수와 지역변수로 해결이 되지 않는 상황

```
char * ReadUserName(void)
{
    char name[30];
    printf("What's your name? ");
    gets(name);
    return name;
}

int main(void)
{
    char * name1;
    char * name2;
    name1=ReadUserName();
    printf("name1: %s \n", name1);
    name2=ReadUserName();
    printf("name2: %s \n", name2);
    return 0;
}
```

변수 name은 ReadUserName 함수호출 시 할당이 되어야 하고, ReadUserName 함수가 반환을 하더라도 계속해서 존재해야 한다. 그런데 전역변수도 지역변수도 이러한 유형에는 부합하지 않는다!

10. 혹시 전역변수가 답이 된다고 생각하는가?

```
char name[30];
char * ReadUserName(void)
{
    printf("What's your name? ");
    gets(name);
    return name;
}
int main(void)
{
    char * name1;
    char * name2;
    name1=ReadUserName();
    printf("name1: %s \n", name1);
    name2=ReadUserName();
    printf("name2: %s \n", name2);
    printf("name1: %s \n", name1);
    printf("name2: %s \n", name2);
    return 0;
}
```

전역변수는 답이 될 수 없음을 보이는 예제 및 실행결과

```
What's your name? Yoon sung woo
name1: Yoon sung woo
What's your name? Choi jun kyung
name2: Choi jun kyung
name1: Choi jun kyung
name2: Choi jun kyung
```

11. 힙 영역의 메모리 공간 할당과 해제

```
#include <stdlib.h>
void * malloc(size_t size);      // 힙 영역으로의 메모리 공간 할당
void free(void * ptr);    // 힙 영역에 할당된 메모리 공간 해제
```

➜ malloc 함수는 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환

```
int main(void)
{
    void * ptr1 = malloc(4);     // 4바이트가 힙 영역에 할당
    void * ptr2 = malloc(12);    // 12바이트가 힙 영역에 할당
    . . . .
    free(ptr1);    // ptr1이 가리키는 4바이트 메모리 공간 해제
    free(ptr2);    // ptr2가 가리키는 12바이트 메모리 공간 해제
    . . . .
}
```

반환형이 void형 포인터임에 주목!

malloc & free 함수 호출의 기본 모델

12. malloc 함수의 반환형이 void형 포인터인 이유

```
void * ptr1 = malloc(sizeof(int));  voi
d * ptr2 = malloc(sizeof(double));  v
oid * ptr3 = malloc(sizeof(int)*7);
void * ptr4 = malloc(sizeof(double)*9);
```

```
void * ptr1 = malloc(4);
void * ptr2 = malloc(8);
void * ptr3 = malloc(28);
void * ptr4 = malloc(72);
```

malloc 함수의 일반적인 호출형태 　　sizeof 연산 이후 실질적인 malloc의 호출

malloc 함수는 인자로 숫자만 하나 전달받을 뿐이니 할당하는 메모리의 용도를 알지 못한다. 따라서 메모리의 포인터 형을 결정짓지 못한다. 따라서 다음과 곁이 형 변환의 과정을 거쳐서 할당된 메모리의 주소 값을 저장해야 한다.

```
int * ptr1 = (int *)malloc(sizeof(int));
double * ptr2 = (double *)malloc(sizeof(double));
int * ptr3 = (int *)malloc(sizeof(int)*7);
double * ptr4 = (double *)malloc(sizeof(double)*9);
```

malloc 함수의 가장 모범적인 호출형태

13. 힙 영역으로의 접근

```
int main(void)
{
    int * ptr1 = (int *)malloc(sizeof(int));
    int * ptr2 = (int *)malloc(sizeof(int)*7);
    int i;
    *ptr1 = 20;
    for(i=0; i<7; i++)
        ptr2[i]=i+1;
    printf("%d \n", *ptr1);
    for(i=0; i<7; i++)
        printf("%d ", ptr2[i]);
    free(ptr1);
    free(ptr2);
    return 0;
}
```

```
20
1 2 3 4 5 6 7
```

힙 영역으로의 접근은 포인터를 통해서만 이뤄진다.

```
int * ptr = (int *)malloc(sizeof(int));
if(ptr==NULL)
{
    // 메모리 할당 실패에 따른 오류의 처리
}
```

메모리 할당 실패 시 malloc 함수는 NULL을 반환

■ 동적 할당'이라 하는 이유!
컴파일 시 할당에 필요한 메모리 공간이 계산되지 않고, 실행 시 할당에 필요한 메모리 공간이 계산되므로!

* free 함수를 호출하지 않으면?
① free 함수를 호출하지 않으면?
할당된 메모리 공간은 메모리라는 중요한 리소스를 계속 차지하게 된다.

② free 함수를 호출하지 않으면 프로그램 종료 후에도 메모리를 차지하는가?
프로그램이 종료되면 프로그램 실행 시 할당된 모든 자원이 반환된다.

③ 꼭 free 함수를 호출해야 하는 이유는 무엇인가?
fopen 함수와 쌍을 이루어 fclose 함수를 호출하는 것과 유사하다.

④ 예제에서 조차 늘 free 함수를 호출하는 이유는 습관을 들이기 위해서인가?
맞다! fopen, fclose가 늘 쌍을 이루듯 malloc, free도 쌍을 이루게 하자!

14. 문자열 반환하는 함수를 정의하는 문제의 해결

```
char * ReadUserName(void)
{
    char * name = (char *)malloc(sizeof(char)*30);
    printf("What's your name? ");
    gets(name);
    return name;
}
```

할당!

ReadUserName 함수가 호출될 때마다 새로운 메모리 공간이 할당이 되고 이 메모리 공간은 함수를 빠져나간 후에도 소멸되지 않는다!

```
int main(void)
{
    char * name1;
    char * name2;
    name1=ReadUserName();
    printf("name1: %s \n", name1);
    name2=ReadUserName();
    printf("name2: %s \n", name2);

    printf("again name1: %s \n", name1);
    printf("again name2: %s \n", name2);
    free(name1);
    free(name2);
    return 0;
}
```

소멸!

```
What's your name? Yoon Sung Woo
name1: Yoon Sung Woo
What's your name? Hong Sook Jin
name2: Hong Sook Jin
again name1: Yoon Sung Woo
again name2: Hong Sook Jin
```

15. calloc & realloc

■ malloc 함수와의 가장 큰 차이점은 메모리 할당을 위한 인자의 전달방식

```
#include <stdlib.h>
void * calloc(size_t elt_count, size_t elt_size);
    ➜ 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환
```

elt_count × elt_size 크기의 바이트를 동적 할당한다.
즉, elt_size 크기의 블록을 elt_count의 수만큼 동적할당!
그리고 malloc 함수와 달리 모든 비트를 0으로 초기화!

■ ptr이 가리키는 힙의 메모리 공간을 size의 크기로 늘리거나 줄인다!

```
#include <stdlib.h>
void * realloc(void * ptr, size_t size);
    ➜ 성공 시 새로 할당된 메모리의 주소 값, 실패 시 NULL 반환
```

■ malloc, calloc, realloc 함수호출을 통해서 할당된 메모리 공간은 모두 free 함수호출을 통해서 해제한다.

16. realloc 함수의 보충설명

```
int main(void)
{
    int * arr = (int *)malloc(sizeof(int)*3);   // 길이가 3인 int형 배열 할당
    . . . .
    arr = (int *)realloc(arr, sizeof(int)*5);   // 길이가 5인 int형 배열로 확장
    . . . .
}
```

• malloc 함수! 그리고 realloc 함수가 반환한 주소 값이 같은 경우
✈ 기존에 할당된 메모리 공간을 이어서 확장할 여력이 되는 경우

• malloc 함수! 그리고 realloc 함수가 반환한 주소 값이 다른 경우
✈ 기존에 할당된 메모리 공간을 이을 여력이 없어서 새로운 공간을 마련하는 경우

■ 새로운 공간을 마련해야 하는 경우에는 메모리의 복사과정이 추가됨에 주목!

【학습정리】

1. 화일 위치 지시자의 위치를 알려주는 함수는 ftell이다.

2. C언어의 메모리는 4영역으로 나누어 지며 코드영역, 데이터영역, 스텍영역, 힙영역으로 나뉜다.

3. 힙영역의 메모리 공간 할당은 malloc 함수를 이용하고 할당된 메모리 공간은 free 함수를 통하여 해제한다.

4. 동적 할당'이라 하는 이유는 컴파일 시 할당에 필요한 메모리 공간이 계산되지 않고, 실행 시 할당에 필요한 메모리 공간이 계산되기 때문이다.