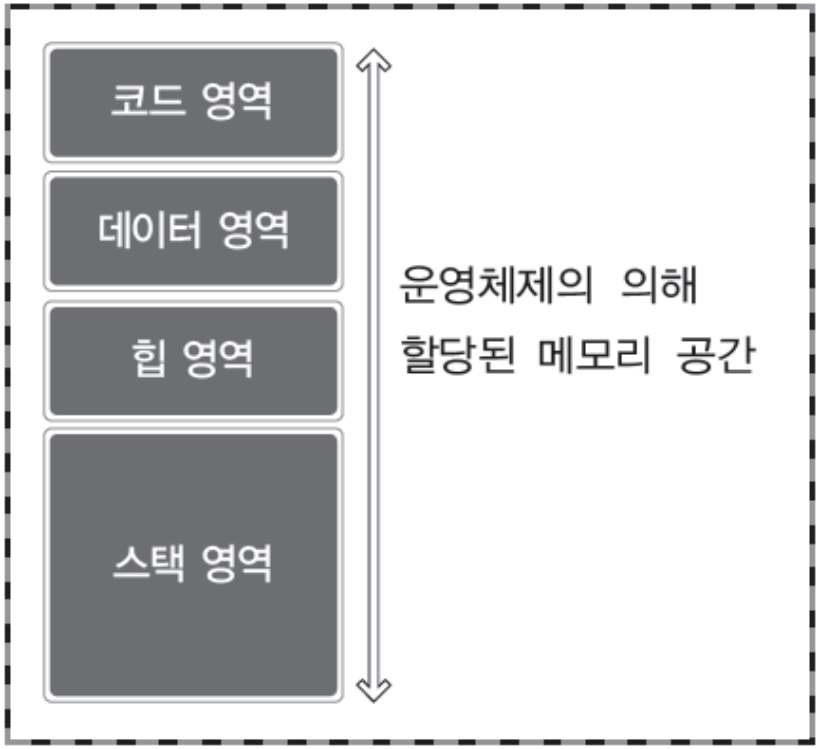
CHAPTER 10

동적 메모리 할당의 이해
리스트 : 순차 선형 리스트와 연결 리스트
비대면

학습 목표

- 동적 메모리 할당의 이해

10.1 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 접근속도가 향상된다.

10.2 메모리 영역별로 저장되는 데이터의 유형

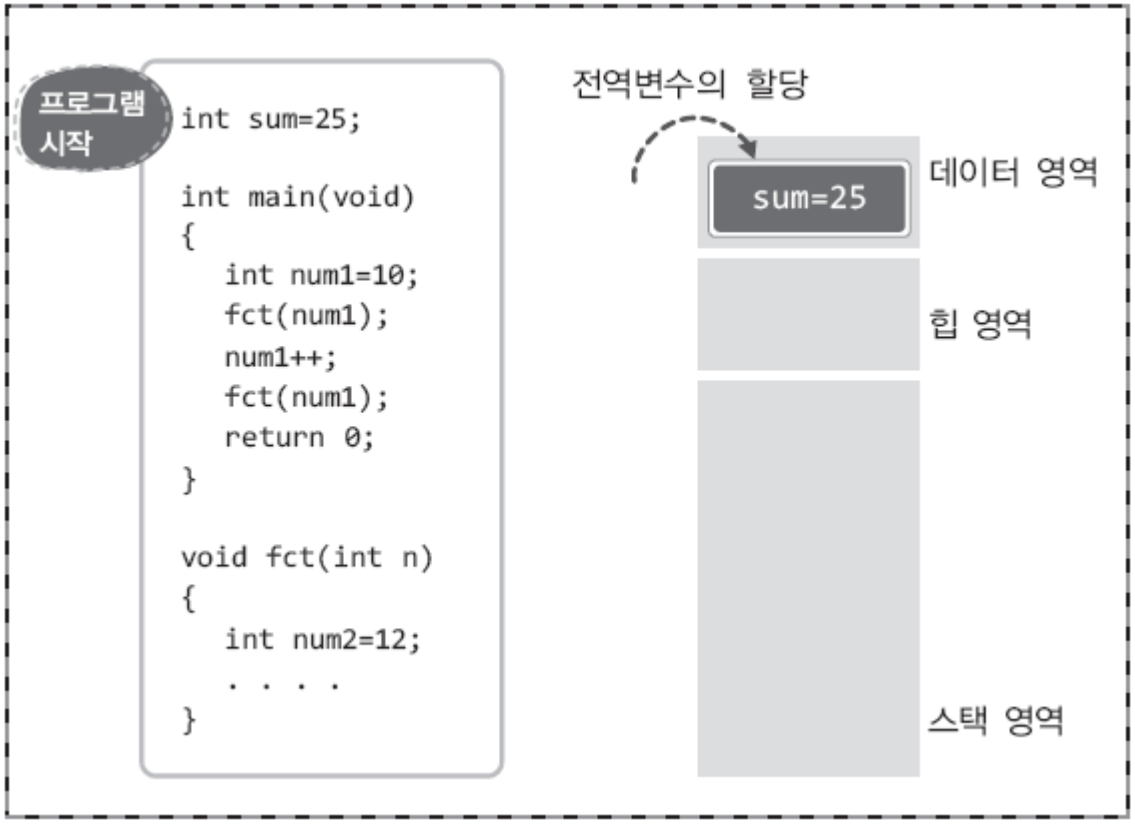
실행할 프로그램의 코드가 저장되는 메모리 공간.
CPU는 코드 영역에 저장된 명령문을 하나씩 가져다가 실행

전역변수와 static 변수가 할당되는 영역.
프로그램 시작과 동시에 할당되어 종료 시까지 남아있는 특징의 변수가 저장되는 영역

프로그래머가 원하는 시점에 메모리 공간에 할당 및 소멸을 하기 위한 영역

지역변수와 매개변수가 할당되는 영역
함수를 빠져나가면 소멸되는 변수를 저장하는 영역

10.3 프로그램의 실행에 따른 메모리의 상태 변화1

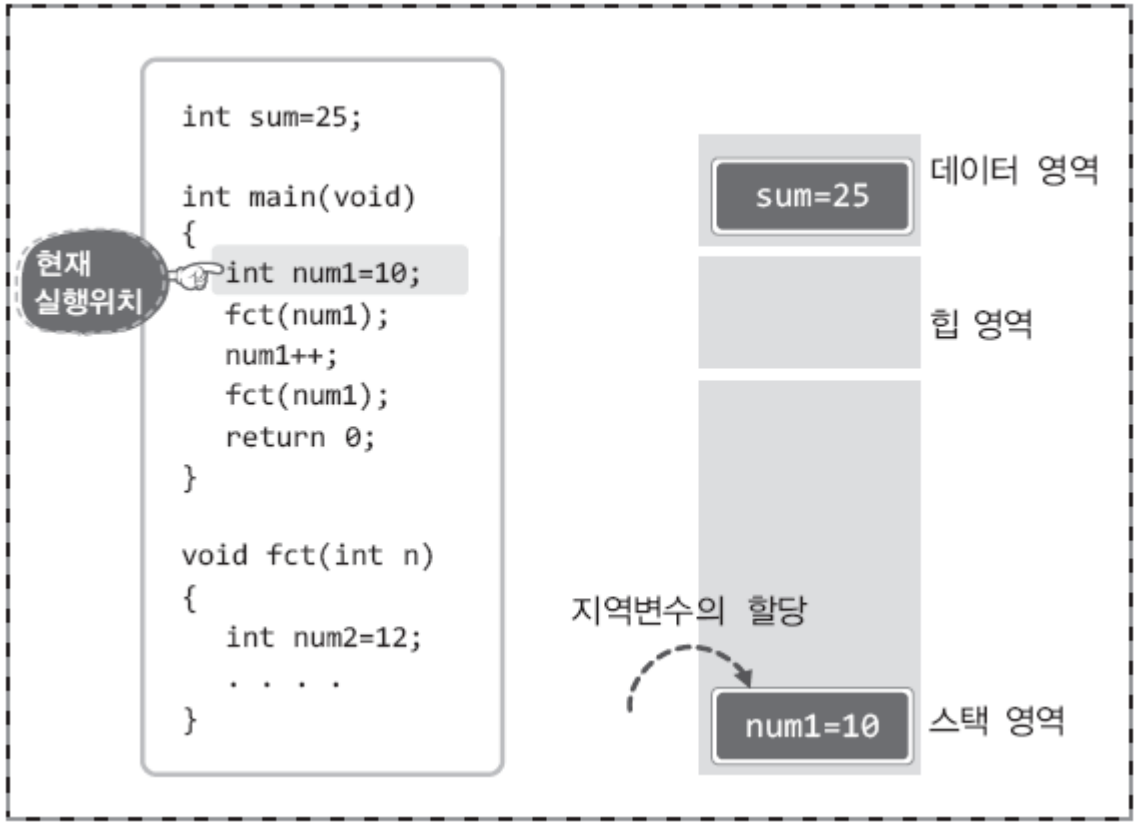


프로그램의 시작:

전역변수의 할당 및 초기화

실행의 흐름/

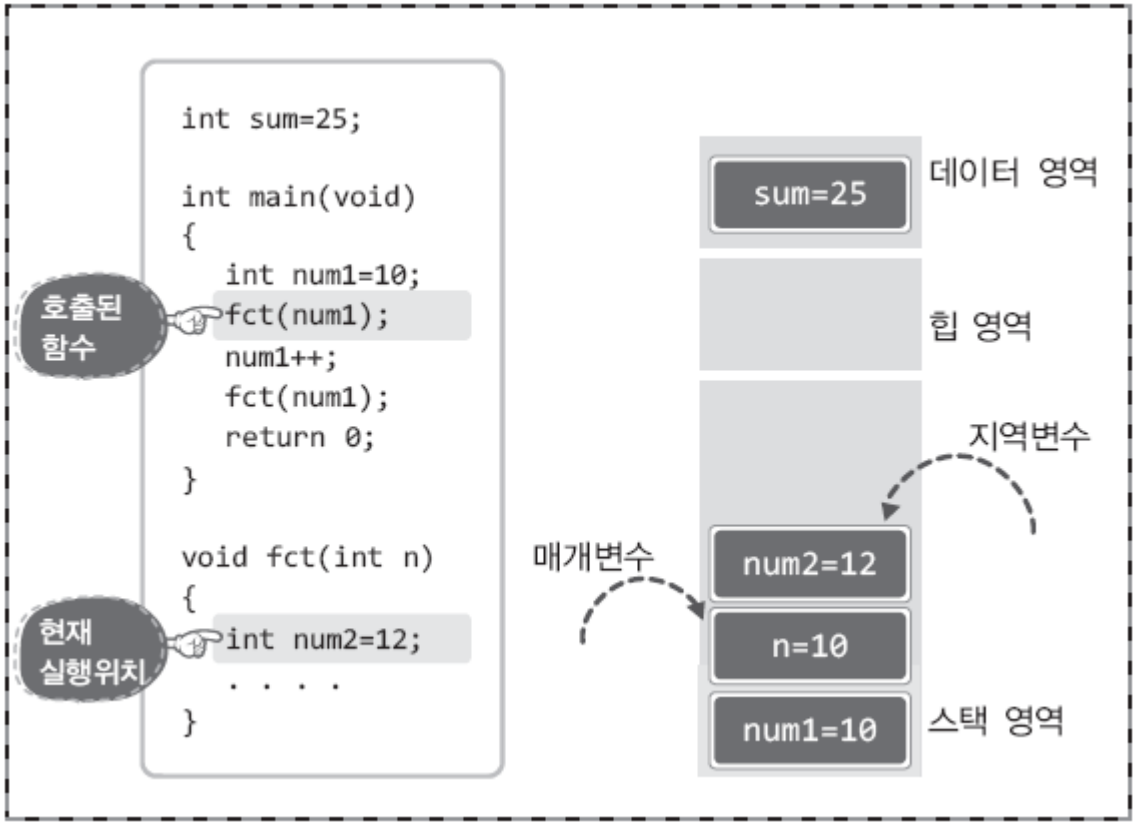
10.4 프로그램의 실행에 따른 메모리의 상태 변화2



main 함수의 호출 및 실행

실행의 흐름2

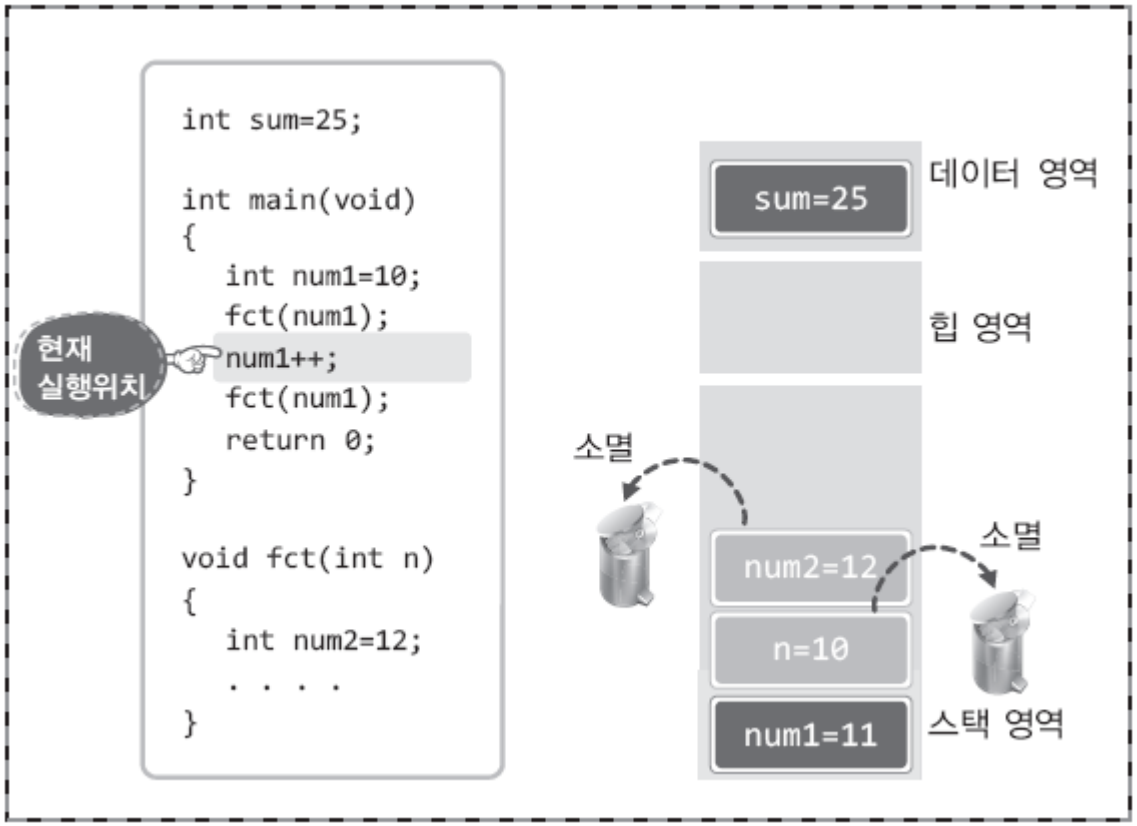
10.5 프로그램의 실행에 따른 메모리의 상태 변화3



fct 함수의 호출

실행의 흐름3

10.6 프로그램의 실행에 따른 메모리의 상태 변화4

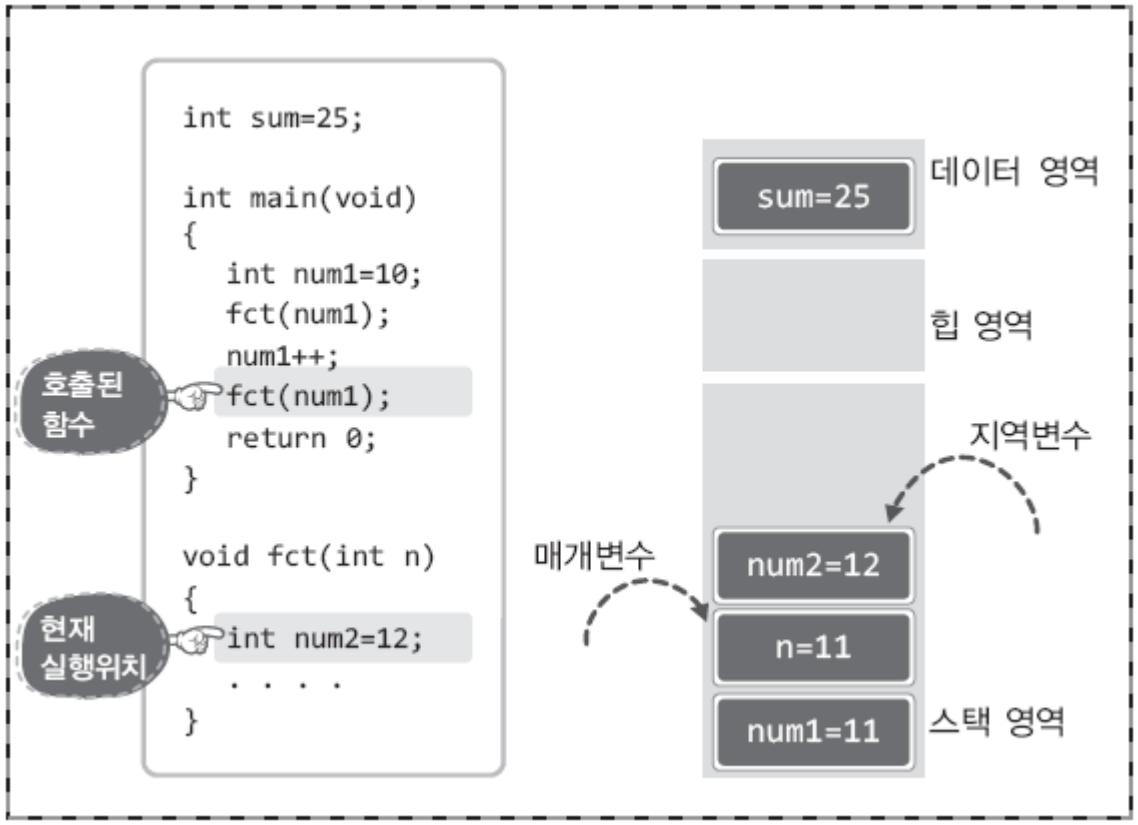


fct 함수의 반환

그리고 main 함수 이어서 실행

실행의 흐름4

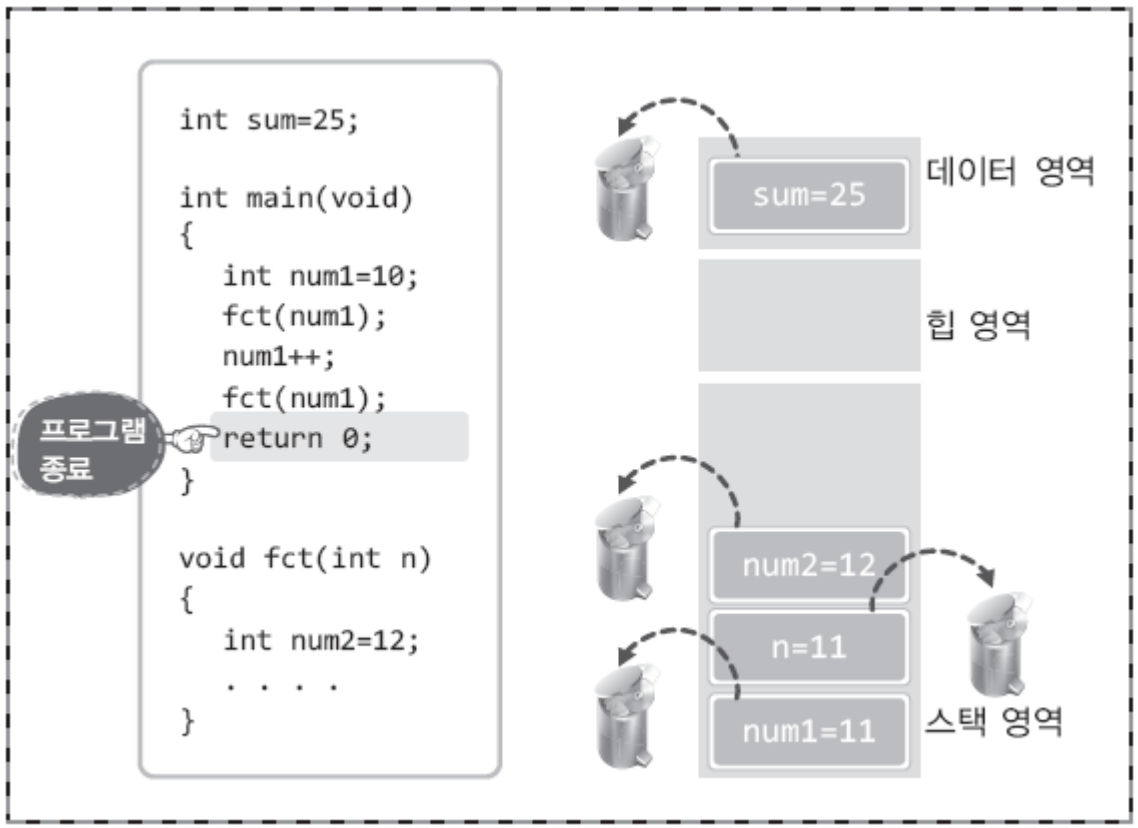
10.7 프로그램의 실행에 따른 메모리의 상태 변화5



fct 함수의 재호출 및 실행

실행의 흐름5

10.8 프로그램의 실행에 따른 메모리의 상태 변화6



fct 함수의 반환

및 main 함수의 반환

실행의 흐름6 (프로그램 종료)

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

메모리의 동적 할당

10.9 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;
}
```

무엇을 반환하는가?

변수 name은 ReadUserName 함수호출 시 할당이 되어야 하고,

ReadUserName 함수가 반환을 하더라도 계속해서 존재해야 한다.

그런데 전역변수도 지역변수도 이러한 유형에는 부합하지 않는다!

10.10 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

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
```

10.11 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

10.12 malloc 함수의 반환형이 void 형 포인터인 이유

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

malloc 함수의 일반적인 호출형태

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

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 같이 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

10.13 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;
}
```

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

메모리 할당 실패 시 malloc 함수는 NULL을 반환

이렇듯 힙 영역으로의 접근은 포인터를 통해서만 이뤄진다.

실행결과

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

'동적 할당'이라 하는 이유!

컴파일 시 할당에 필요한 메모리 공간이 계산되지 않고, 실행 시 할당에 필요한 메모리 공간이 계산되므로!

10.14 free 함수를 호출하지 않으면?

- **free 함수를 호출하지 않으면?**

  할당된 메모리 공간은 메모리라는 중요한 리소스를 계속 차지하게 된다.

- **free 함수를 호출하지 않으면 프로그램 종료 후에도 메모리를 차지하는가?**

  프로그램이 종료되면 프로그램 실행 시 할당된 모든 자원이 반환된다.

- **꼭 free 함수를 호출해야 하는 이유는 무엇인가?**

  fopen 함수와 쌍을 이루어 fclose 함수를 호출하는 것과 유사하다.

- **예제에서 조차 늘 free 함수를 호출하는 이유는 습관을 들이기 위해서인가?**

  맞다! fopen, fclose가 늘 쌍을 이루듯 malloc, free도 쌍을 이루게 하자!

10.15 문자열 반환하는 함수를 정의하는 문제 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
```

실행결과

10.16 calloc & realloc

```
#include <stdlib.h>
void * calloc(size_t elt_count, size_t elt_size);
```

➜ 성공 시 할당된 메모리의 주소 값, 실패 시 NULL 반환

malloc 함수와의 가장 큰 차이점은 메모리 할당을 위한 인자의 전달방식

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!

```
#include <stdlib.h>
void * realloc(void * ptr, size_t size);
```

➜ 성공 시 새로 할당된 메모리의 주소 값, 실패 시 NULL 반환

ptr이 가리키는 힙의 메모리 공간을 size의 크기로 늘리거나 줄인다!

10.17 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!

질문 및 정리