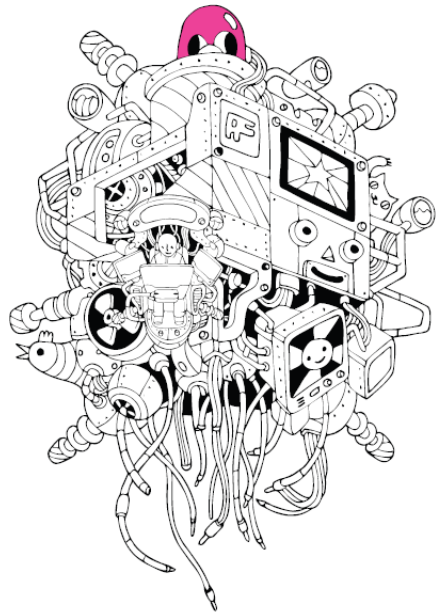
프로그래밍 언어 I



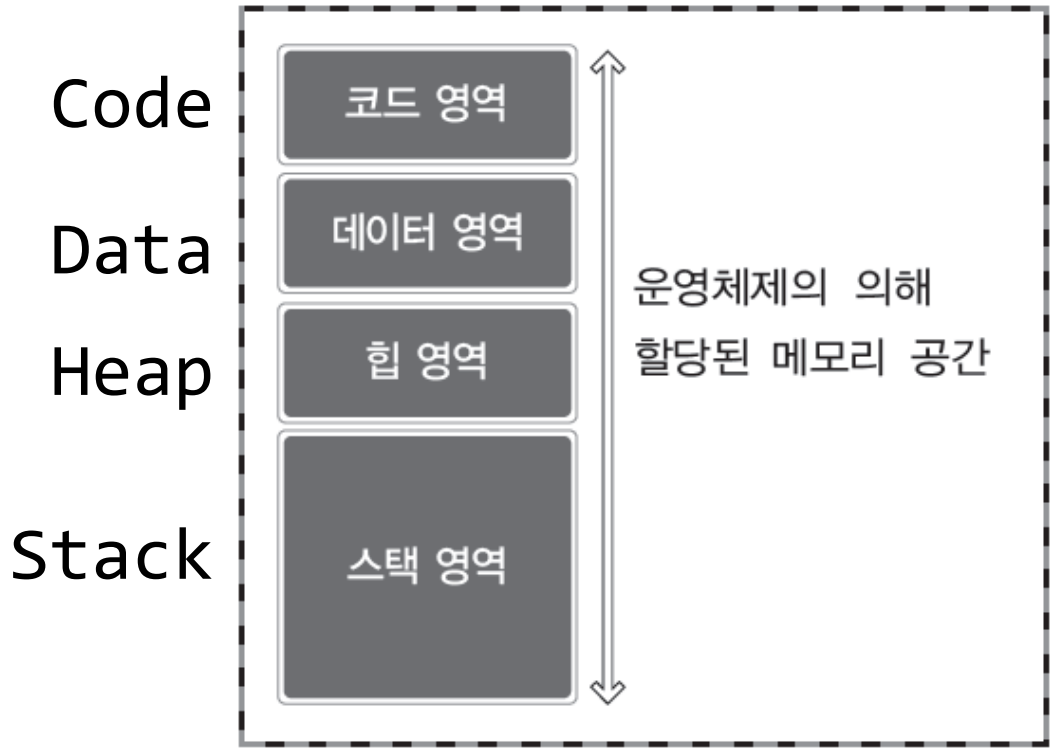
Chapter 25. 메모리 관리와 메모리의 동적 할당

프로그래밍 언어 I

Chapter 25-1. C언어의 메모리 구조

Chapter 25. 메모리 관리와 메모리의 동적 할당

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.

메모리 영역별로 저장되는 데이터의 유형

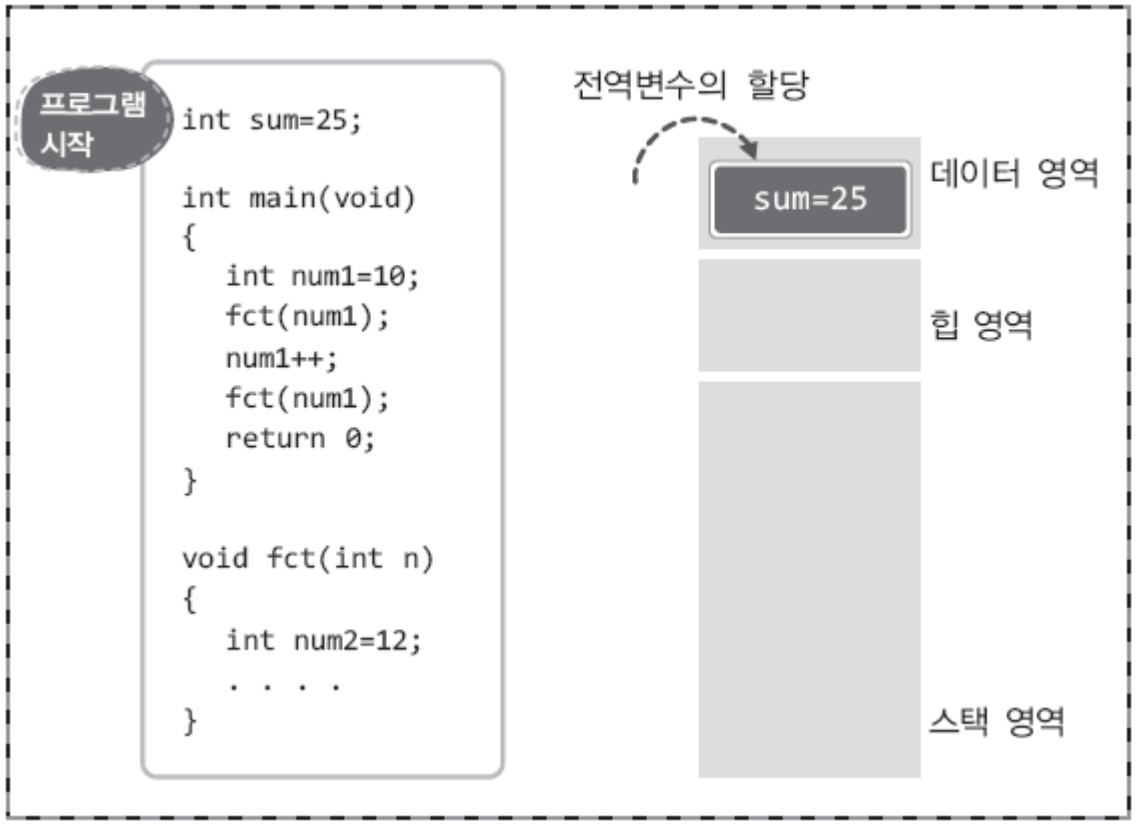
실행할 프로그램의 코드가 저장되는 메모리 공간.
CPU는 코드 영역에 저장된 명령문을 하나씩 가져다가 실행

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

프로그램의 실행에 따른 메모리의 상태 변화1

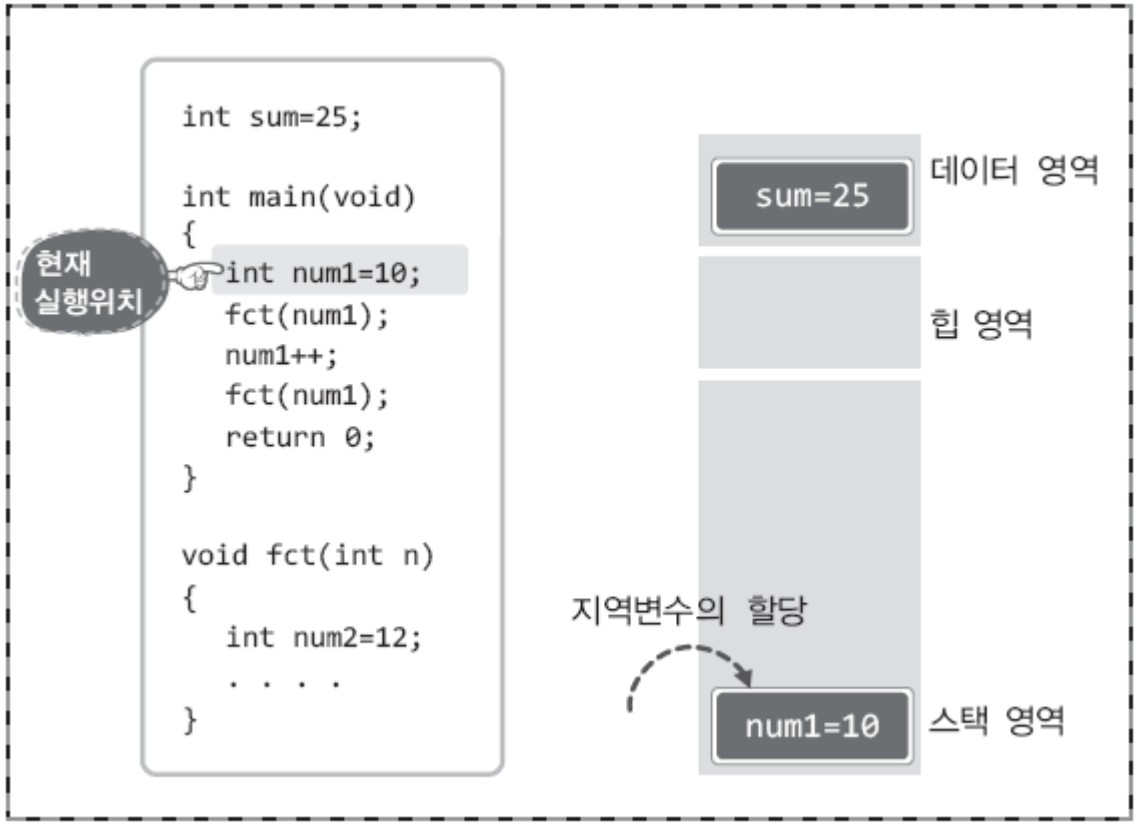


프로그램의 시작:

전역변수의 할당 및 초기화

실행의 흐름!

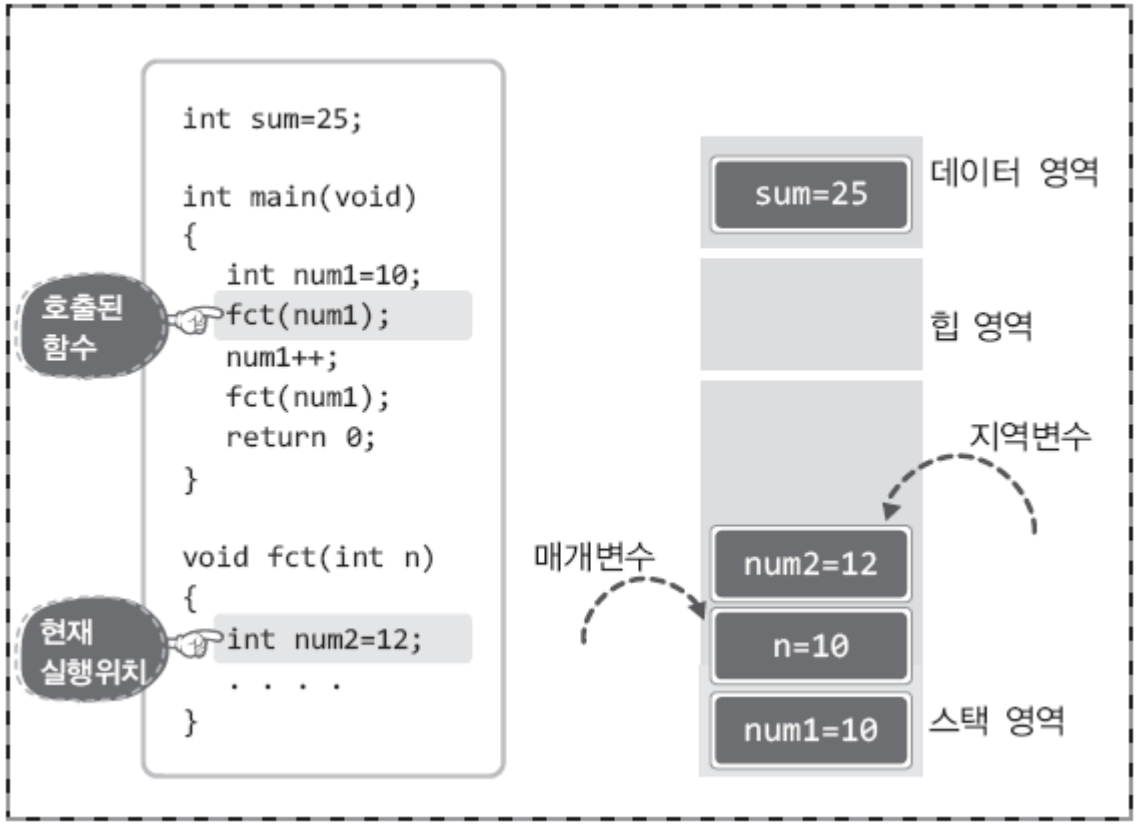
프로그램의 실행에 따른 메모리의 상태 변화2



main 함수의 호출 및 실행

실행의 흐름2

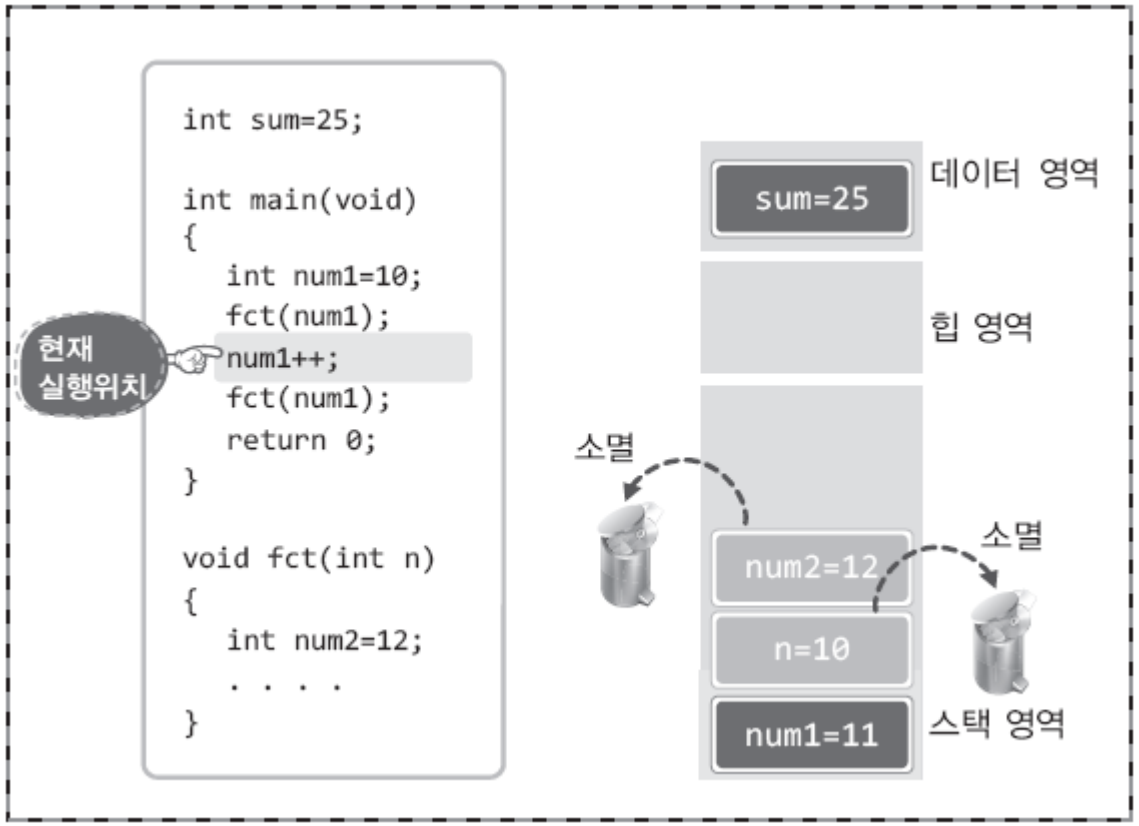
프로그램의 실행에 따른 메모리의 상태 변화3



fct 함수의 호출

실행의 흐름3

프로그램의 실행에 따른 메모리의 상태 변화4

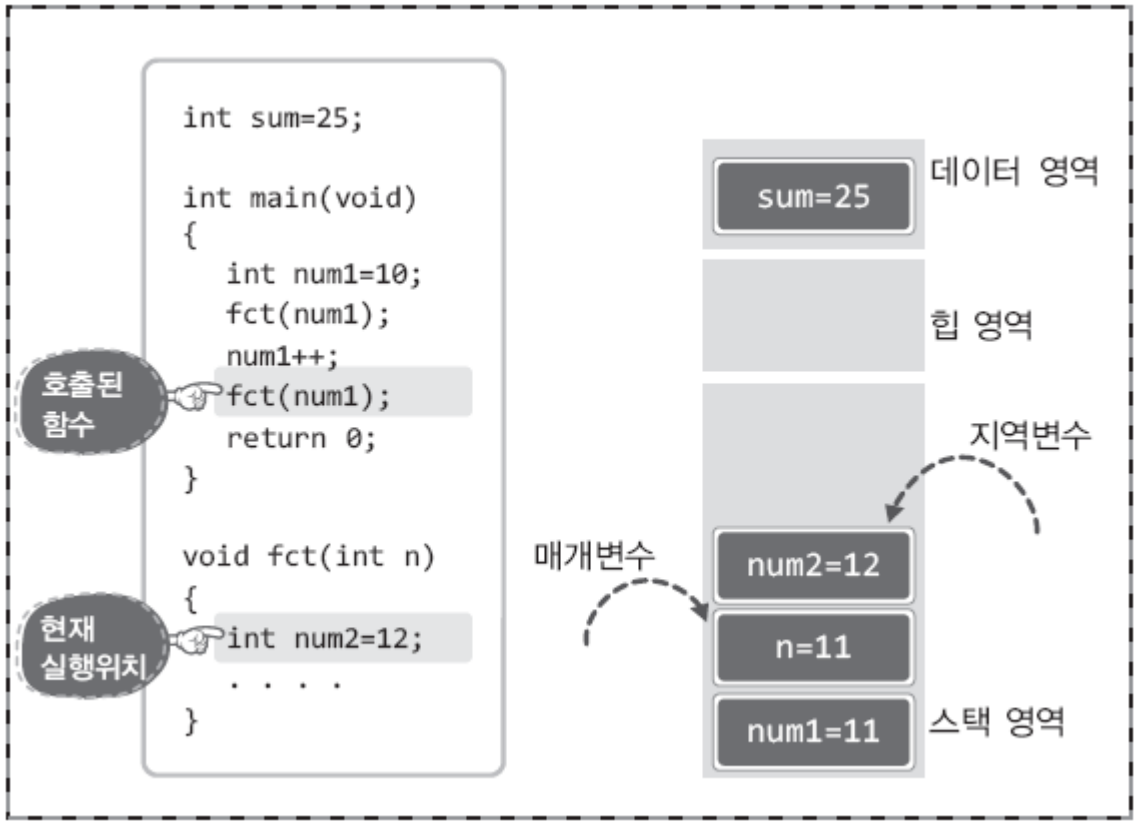


fct 함수의 반환

그리고 main 함수 이어서 실행

실행의 흐름4

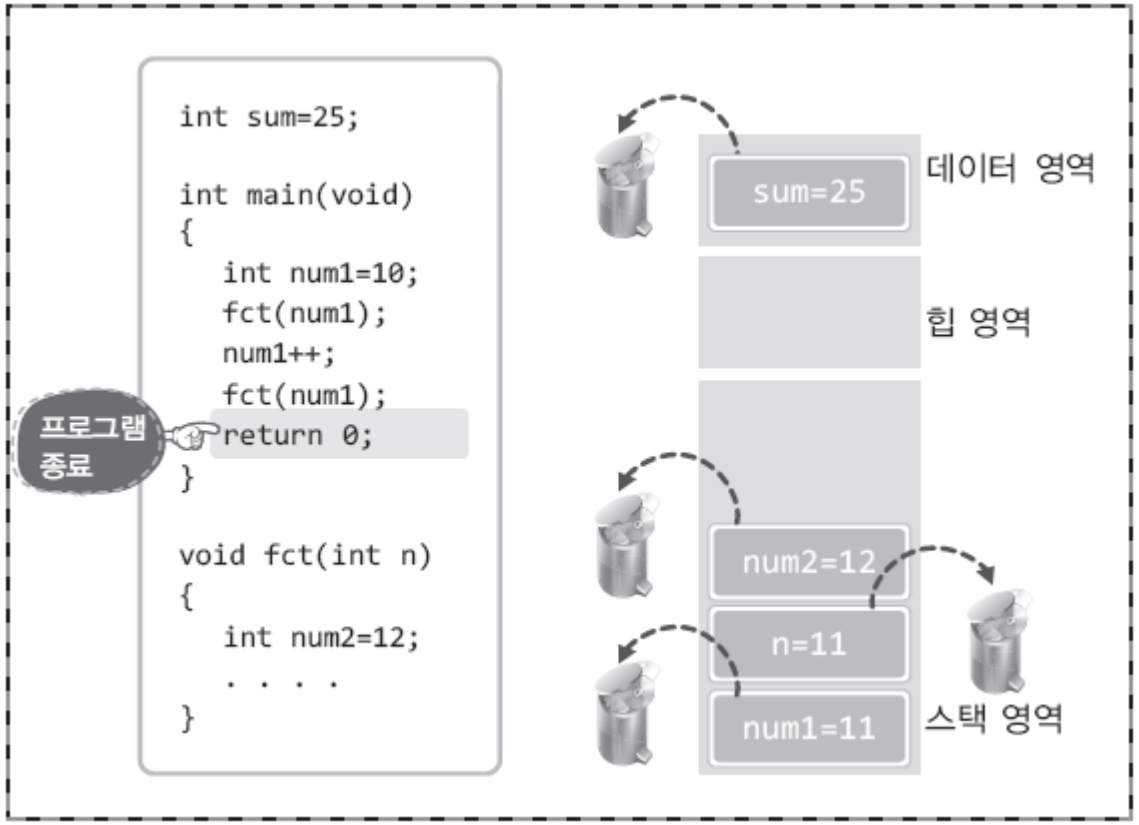
프로그램의 실행에 따른 메모리의 상태 변화5



fct 함수의 재호출 및 실행

실행의 흐름5

프로그램의 실행에 따른 메모리의 상태 변화6



fct 함수의 반환

및 main 함수의 반환

실행의 흐름6 (프로그램 종료)

프로그래밍 언어 I

Chapter 25-2. 메모리의 동적 할당

Chapter 25. 메모리 관리와 메모리의 동적 할당

메모리 할당

프로그램이 메모리를 할당 받는 방법

- 정적 메모리 할당 (static memory allocation)
- 동적 메모리 할당 (dynamic memory allocation)

정적 메모리 할당 (Static Memory Allocation)

- 프로그램이 시작되기 전에 미리 정해진 크기의 메모리를 할당 받는 것
- 메모리의 크기는 프로그램이 시작하기 전에 결정

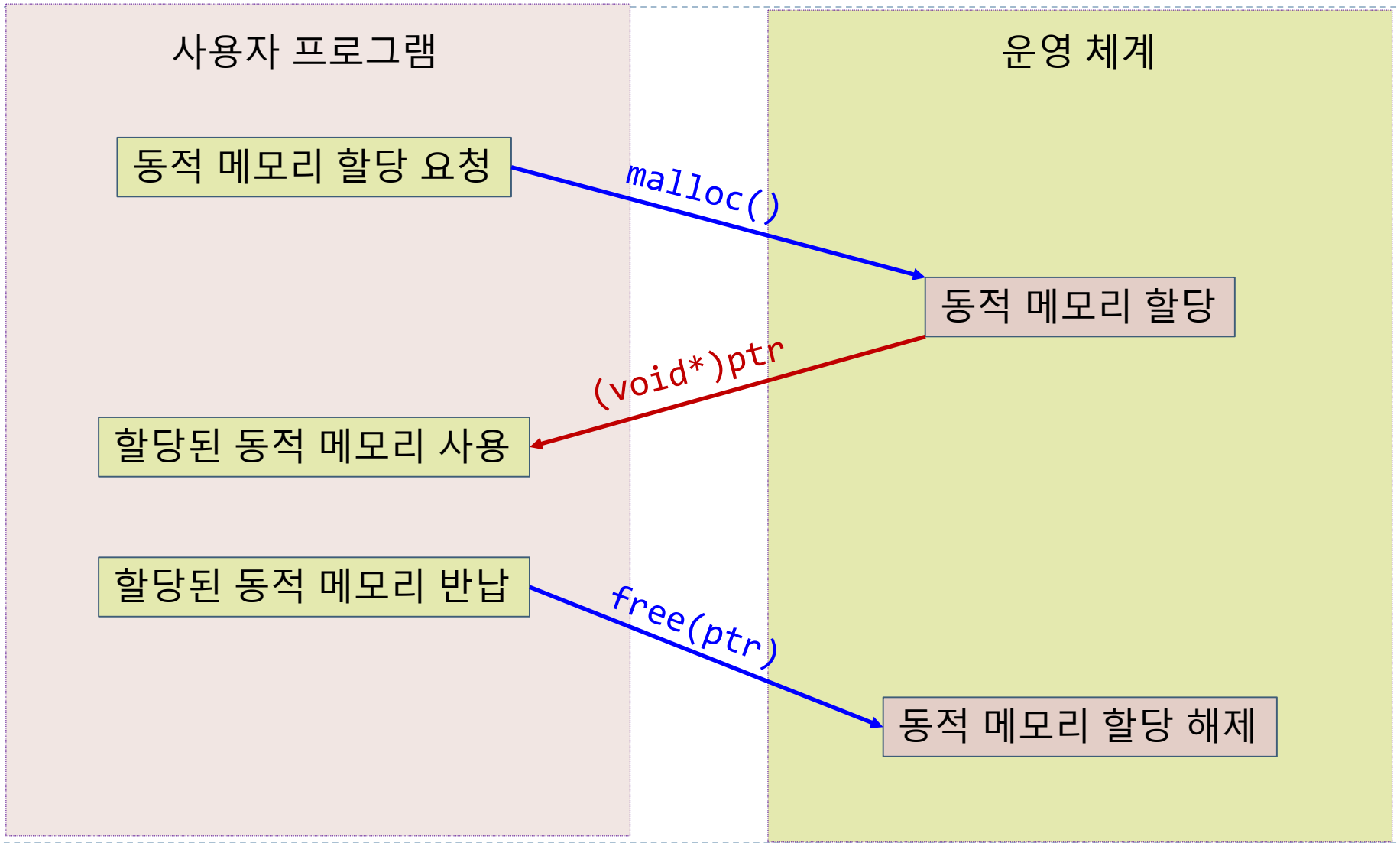
```
int sarray[10];
```

- 처음에 결정된 개수보다 더 많은 개수의 입력이 들어온다면 처리하지 못함
- 처음에 결정된 개수보다 더 적은 개수 입력이 들어온다면 남은 메모리 공간은 낭비

동적 메모리 할당 (Dynamic Memory Allocation)

- 실행 도중에 동적으로 메모리를 할당 받는 것
- 사용이 끝나면 시스템에 메모리를 반납
- 필요한 만큼만 메모리를 할당을 받아 메모리를 매우 효율적으로 사용
- `malloc()` 계열의 라이브러리 함수를 사용

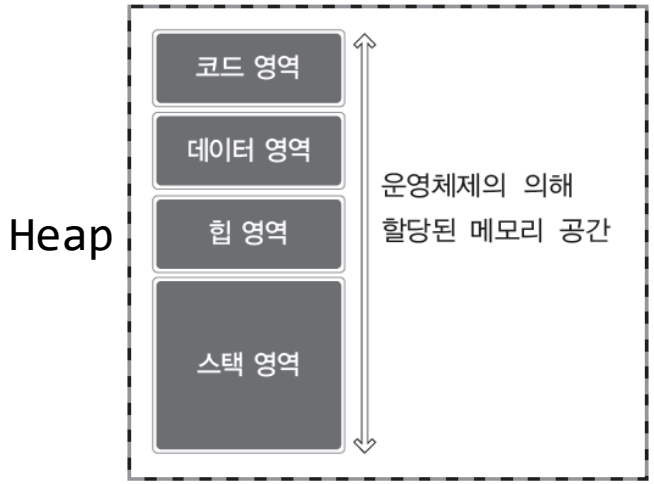
동적 메모리 할당 절차


사용자 프로그램
운영 체계
동적 메모리 할당 요청
malloc()
동적 메모리 할당
(void*)ptr
할당된 동적 메모리 사용
할당된 동적 메모리 반납
free(ptr)
동적 메모리 할당 해제

힙(heap) 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() 함수와 free() 함수 호출의 기본 모델

malloc() 함수의 반환형이 void형 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 malloc()의 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 일반적인 호출형태

힙(heap) 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!

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!

프로그램 실행 시점에 정해지는 데이터의 개수 (1-1)

예제: `malloc1`

1. 프로그램이 실행되면 사용자로부터 데이터의 개수 *N*을 입력 받는다.
2. *N*개의 정수를 저장하는데 필요한 크기의 메모리를 할당 받는다.
3. *N*개의 난수를 발생시켜 할당 받은 메모리에 저장한다.
4. *N*개의 데이터에 대해 평균값, 최소값, 최대값을 구한다.

프로그램 실행 시점에 정해지는 데이터의 개수 (1-2)

예제: malloc1

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int num_of_data;
   int *buffer, i;

   printf("원하는 데이터 개수는?: ");
   scanf("%d", &num_of_data);

   // 메모리 할당
   buffer = (int *)malloc(sizeof(int) * num_of_data);
   if (buffer == NULL)
   {
      printf("메모리 할당 실패.\n");
      return -1;
   }
```

```c
   // 할당받은 메모리에 난수 써 넣기
   for (i = 0; i < num_of_data; i++)
      buffer[i] = rand();

   // 최대, 최소, 평균 구하기
   int min = buffer[0];
   int max = buffer[0];
   long sum = buffer[0];
   for (i = 1; i < num_of_data; i++)
   {
      sum += buffer[i];
      if (buffer[i] < min)
         min = buffer[i];
      if (buffer[i] > max)
         max = buffer[i];
   }
   double avg = (double)sum / num_of_data;
   printf("min=%d, max=%d, avg=%.f\n",
          min, max, avg);

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

실행결과

```
원하는 데이터 개수는?: 1000
min=28, max=32757, avg=16397
```

프로그램 실행 시점에 정해지는 데이터의 개수 (2-1)

예제: `malloc2`

`"MIT_BIH_100_LEAD2.bin"`

1. 지정된 바이너리 파일의 크기 *N*을 알아낸다.
2. *N* 바이트 크기의 메모리를 할당 받는다.
3. 지정된 바이너리 파일에 저장된 내용을 읽어 할당된 메모리에 저장한다.
4. 메모리에 저장된 데이터에 대해 평균값, 최소값, 최대값을 구한다.

프로그램 실행 시점에 정해지는 데이터의 개수 (2-2)

예제: malloc2

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    char* filename = "MIT_BIH_100_LEAD2.bin";
    long file_length;
    short *buffer;
    int num_of_data;

    // 파일 크기 알아내기
    FILE* fp = fopen(filename, "rb");
    if (fp == NULL)
    {
        printf("%s 파일을 열수 없음\n", filename);
        return -1;
    }
    fseek(fp, 0, SEEK_END);
    file_length = ftell(fp);
    num_of_data = file_length / sizeof(short);
    rewind(fp);

    // 메모리 할당
    buffer = (short*)malloc(file_length);
    if (buffer == NULL)
    {
        printf("메모리 할당 실패\n");
        return -2;
    }

    // 파일 읽어서 할당받은 메모리에 저장하기
    fread(buffer, sizeof(short), num_of_data, fp);
    fclose(fp);

    // 최대, 최소, 평균값 구하기
    short min = buffer[0], max = buffer[0];
    long sum = buffer[0];
    for (int i = 1; i < num_of_data; i++)
    {
        sum += buffer[i];
        if (buffer[i] < min)
            min = buffer[i];
        if (buffer[i] > max)
            max = buffer[i];
    }
    double avg = (double)sum / num_of_data;
    printf("min=%hd, max=%hd, avg=%.8f\n", min, max, avg);
    free(buffer);
    return 0;
}
```

실행결과

```
min=895, max=1216, avg=961.08148148
```

calloc() 함수와 realloc() 함수

```
#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 size 바이트의 의 크기로 늘리거나 줄인다!

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

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!

프로그램 실행 중에 할당 받은 메모리의 크기 변경 (3-1)

예제: realloc

"MIT_BIH_100_LEAD2.txt"

1. 초기에 적당한 수(예를 들면 10)의 데이터를 저장할 수 있는 메모리를 할당 받는다.
2. 지정된 텍스트 파일에 저장된 데이터를 읽어 할당된 메모리에 저장한다.
3. 파일에 저장된 데이터가 더 존재하면 할당 받은 메모리의 크기를 다시 일정 개수(예를 들면 10) 늘린다.
4. 파일을 모두 읽을 때까지 위 3의 과정을 반복한다.
5. 메모리에 저장된 데이터에 대해 평균값, 최소값, 최대값을 구한다.

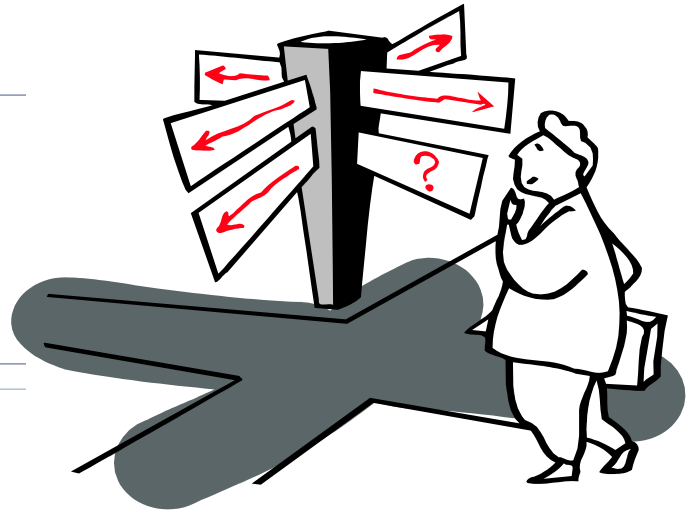
프로그램 실행 중에 할당 받은 메모리의 크기 변경 (3-2)

예제: realloc

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>

#define INCREMENT10

int main(void)
{
     char* filename = "MIT_BIH_100_LEAD2.txt";
     short* buffer = NULL;
     short data;
     unsigned num_of_alloc_data, num_of_used_data;
     int i = 0;

     FILE* fp = fopen(filename, "rt");
     if (fp == NULL)
     {
          printf("%s 파일을 열 수 없습니다.", filename);
          return -1;
     }
     // 초기 메모리 할당
     buffer = (short*)malloc(INCREMENT * sizeof(short));
     if (buffer == NULL)
     {
          printf("초기 메모리 할당 실패\n");
          return -2;
     }
     num_of_alloc_data = INCREMENT;
     num_of_used_data = 0;
   while (fscanf(fp, "%hd", &data) != EOF)
   {
      if (num_of_alloc_data == num_of_used_data)
      {
         unsigned new_buf_size = (num_of_alloc_data + INCREMENT) * sizeof(short);
         short *temp = (short*)realloc(buffer, new_buf_size); // 메모리 할당량 증가
         if (temp == NULL)
         {
            printf("메모리 할당량 증가 실패\n");  return -2;
         }
         buffer = temp;
         num_of_alloc_data += INCREMENT;
      }
      buffer[num_of_used_data++] = data;
   }
   fclose(fp);

   short min = buffer[0], max = buffer[0];
   int sum = buffer[0];
   for (i = 1; i < (int)num_of_used_data; i++)
   {
      sum += (int)buffer[i];
      if (buffer[i] < min)  min = buffer[i];
      if (buffer[i] > max)  max = buffer[i];
   }
   double avg = (double)sum / num_of_used_data;
   printf("min=%hd, max=%hd, avg=%.8f\n", min, max, avg);

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

실행결과

```
min=895, max=1216, avg=961.08148148
```



Chapter 25가 끝났습니다. 질문 있으신지요?