

기초프로그래밍

제17장 동적 메모리 할당

Sangsoo Lim

CSAI

Dongguk University

동적 메모리 할당

- 프로세스의 메모리 구조

- 코드 영역: 프로그램 실행 코드, 함수들이 저장되는 영역
- 스택 영역: 매개변수, 지역 변수, 중괄호(블록) 내부에 정의된 변수들이 저장되는 영역
- 데이터 영역: 전역 변수, 정적 변수들이 저장되는 영역
- 힙 영역: 프로그램이 실행 되는 동안 동적으로 메모리를 할당할 수 있는 영역



동적 메모리 할당

```
#include <stdio.h>

int a = 10; // 전역 변수

int main(void)
{
    int num1 = 10, num2 = 20; // 지역 변수
    static int s = 20;        // 정적 지역 변수

    // 변수 값 출력
    printf("변수 값 출력: %d %d %d %d\n", a, num1, num2, s);

    // 코드 영역 주소 (함수 주소)
    printf("코드 영역 주소: %p %p\n", (void *)main, (void *)printf);

    // 스택 영역 주소 (지역 변수)
    printf("스택 영역 주소: %p %p\n", (void *)&num1, (void *)&num2);

    // 데이터 영역 주소 (전역 변수, 정적 변수)
    printf("데이터 영역 주소: %p %p\n", (void *)&a, (void *)&s);

    return 0;
}
```

동적 메모리 할당

- 동적 메모리 할당

- 힙영역에 할당 된다.
- 런타임 중(실행 시간)에 이루어 진다.
- 프로그래머가 동적 메모리 할당을 요구한다.

코드 영역
(실행 코드, 함수)

스택 영역
(지역 변수, 매개 변수)

데이터 영역
(전역 변수, 정적 변수)

힙 영역
(동적 메모리 할당)

동적 메모리 할당

- 동적 메모리 할당이 필요한 이유

1. 선언된 배열 요소의 수가 사용된 요소 수 보다 많은 경우 (메모리 낭비)

```
int array[5];           // 선언된 배열 요소 수: 5개(20바이트)  
array[0]=10, array[1]=20, array[2]=30; // 사용된 배열 요소 수: 3개(12바이트)
```

2. 선언된 배열 요소의 수가 사용된 요소의 수보다 적은 경우 (메모리 부족)

```
int array[2];           // 선언된 배열 요소 수: 2개(8바이트)  
array[0]=10, array[1]=20, array[2]=30; // 사용된 배열 요소 수: 3개(12바이트)
```

동적 메모리 할당

- 동적 메모리 할당이 필요한 이유

3. 배열 선언 시 배열 길이에 변수를 설정한 경우 **에러** 발생

```
int a=5;  
int array[a]    // 배열 선언 시 배열 a를 배열 길이로 사용
```

```
void init(int a)  
{  
    int array[a];    // 배열 선언 시 함수의 인자(지역 변수) a를 배열 길이로 사용
```



결론: 프로그래머가 필요한 메모리 크기를 예측할 수 없다.
따라서 동적 메모리 할당의 필요하다.

동적 메모리 할당

- 동적 메모리 **할당 함수**와 **해제 함수**
 - 헤더파일 : **stdlib.h**

종류	함수	성공	실패
메모리 할당 함수	<code>#include <stdlib.h></code> <code>void* malloc(size_t size)</code>	할당된 메모리의 시작 주소 반환	NULL 반환
메모리 할당 함수	<code>#include <stdlib.h></code> <code>void* calloc(size_t num, size_t size)</code>	할당된 메모리의 시작 주소 반환	NULL 반환
메모리 할당 함수	<code>#include <stdlib.h></code> <code>void* realloc(void* p, size_t size)</code>	재할당된 메모리의 시작 주소 반환	NULL 반환
메모리 해제 함수	<code>#include <stdlib.h></code> <code>void free(void* p)</code>	할당된 메모리 해제	-

동적 메모리 할당 함수, 해제 함수

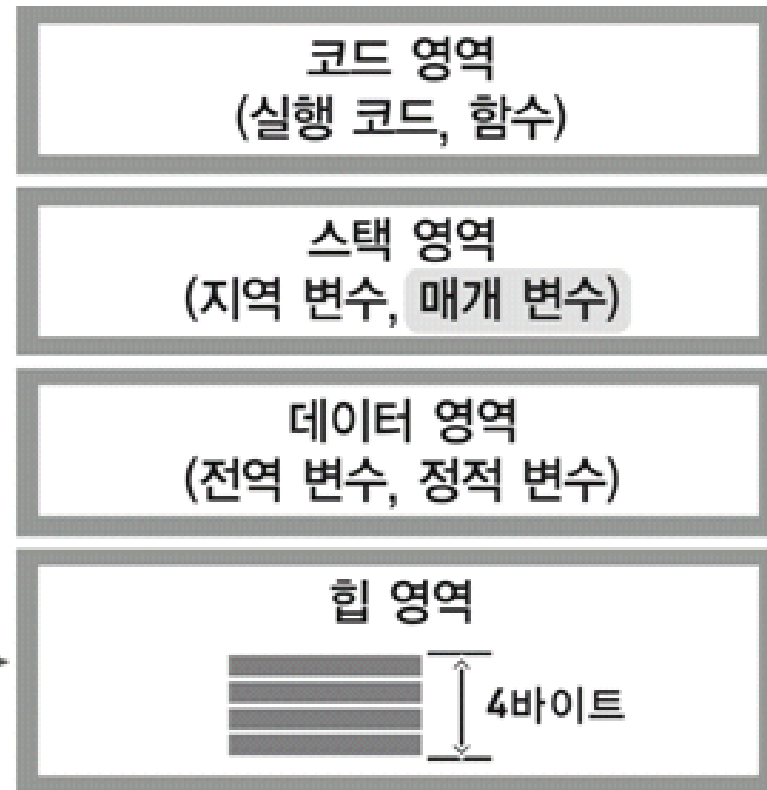
- malloc() 함수와 free() 함수

종류	함수	성공	실패
메모리 할당 함수	void* malloc(size_t size);	할당된 메모리의 시작 주소 반환	NULL 반환
메모리 해제 함수	void free(void* p);	할당된 메모리 해제	

동적 메모리 할당 함수, 해제 함수

- `malloc(4)` 함수를 이용한 동적 메모리 할당

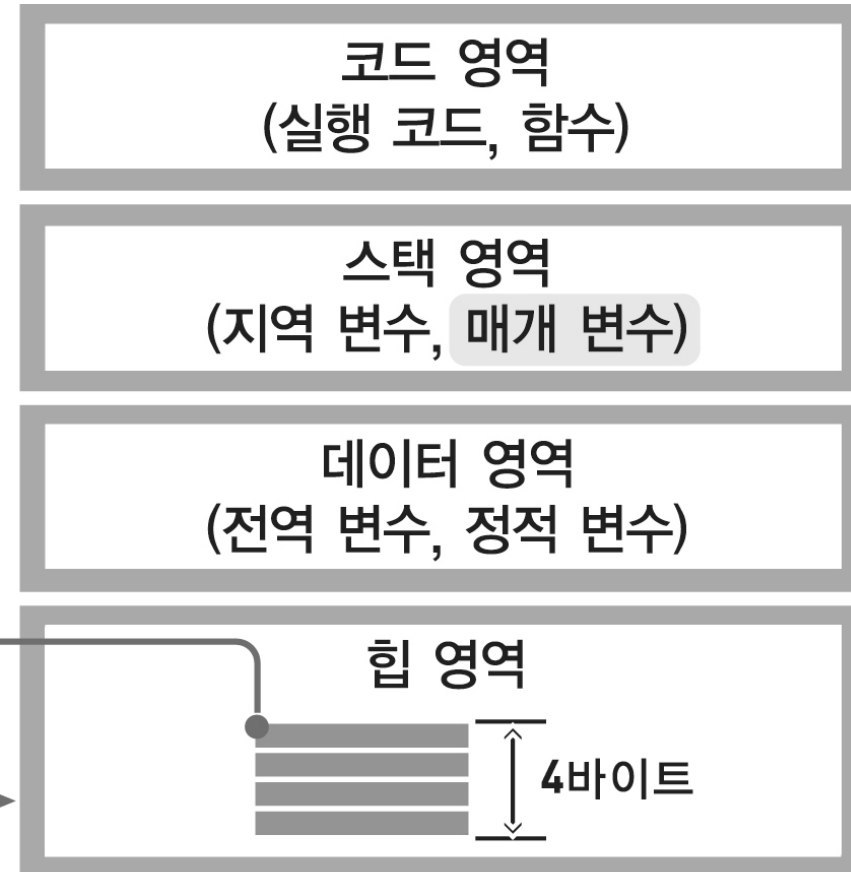
```
int main(void)
{
    int* p=NULL;
    p=(int*) malloc(4);
    return 0;
}
```



동적 메모리 할당 함수, 해제 함수

- 할당된 메모리의 시작 주소 형변환 (int*)

```
int main(void)
{
    int* p=NULL;
    p=(int*) malloc(4);
    return 0;
}
```



동적 메모리 할당 함수, 해제 함수

- `free()` 함수를 이용한 동적 메모리 해제

```
int main(void)
{
    int* p=NULL;
    p=(int*) malloc(4);
    ...
    free(p);
    return 0;
}
```



동적 메모리 할당 함수, 해제 함수

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

int main(void)
{
    int* p = NULL; // 포인터 초기화

    // 정수형 크기만큼 동적 메모리 할당
    p = (int*)malloc(sizeof(int));

    if (p == NULL) {
        printf("힙 영역에 동적 메모리 할당 실패\n");
        return 1; // 메모리 할당 실패 시 종료
    }

    // 할당된 메모리에 값 저장
    *p = 10;

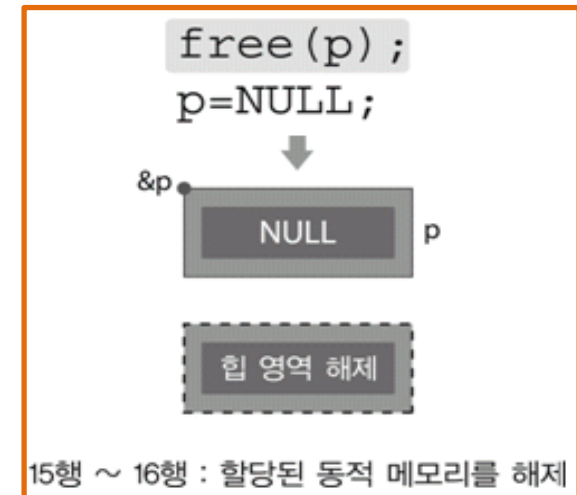
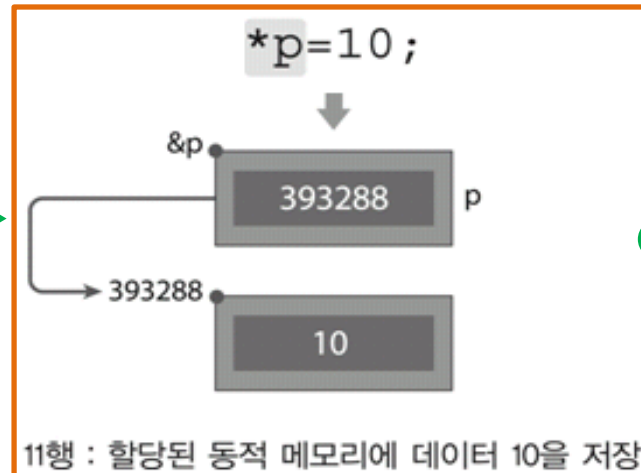
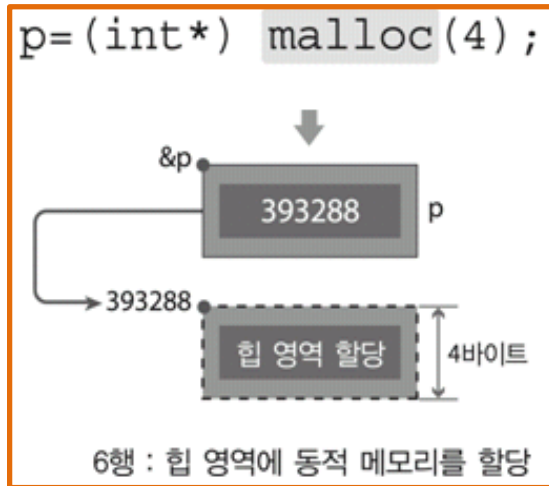
    // 주소와 저장된 값 출력
    printf("할당된 주소: %p\n", (void*)p);
    printf("저장된 값 : %d\n", *p);

    // 메모리 해제 및 포인터 초기화
    free(p);
    p = NULL;

    return 0;
}
```

동적 메모리 할당 함수, 해제 함수

- 동적 메모리 할당



동적 메모리 할당 함수, 해제 함수

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

int main(void)
{
    // 문자형과 정수형 배열 메모리 동적 할당
    char* p1 = (char*)malloc(2 * sizeof(char)); // 문자 2개 저장 공간
    int* p2 = (int*)malloc(2 * sizeof(int)); // 정수 2개 저장 공간

    // 메모리 할당 실패 확인
    if (p1 == NULL || p2 == NULL) {
        printf("메모리 할당 실패\n");
        return 1;
    }

    // 값 저장
    p1[0] = 'A';
    p1[1] = 'B';
    p2[0] = 10;
    p2[1] = 20;

    // 주소 및 값 출력
    printf("주소: %p %p %p %p\n",
           (void*)&p1[0], (void*)&p1[1],
           (void*)&p2[0], (void*)&p2[1]);

    printf("값 : %c %c %d %d\n",
           p1[0], p1[1],
           p2[0], p2[1]);

    // 메모리 해제 및 포인터 초기화
    free(p1);
    p1 = NULL;

    free(p2);
    p2 = NULL;

    return 0;
}
```

동적 메모리 할당 함수, 해제 함수

- **calloc()** 함수를 이용한 동적 메모리 할당

종류	함수	반환 값
메모리 할당 함수	<code>void* calloc(size_t num, size_t size);</code>	성공 : 할당된 메모리의 시작 주소 반환 실패 : NULL 반환

- **calloc()** 함수와 **malloc()** 함수와의 차이

`int* p1=calloc(4, 4);` → 함수의 입력 인자 2개

==

`int* p2=malloc(16);` → 함수의 입력 인자 1개

동적 메모리 할당 함수, 해제 함수

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

int main(void)
{
    int i = 0;
    int* p = (int*)calloc(4, sizeof(int)); // 정수 4개 크기만큼 메모리 할당 및 0으로 초기화

    if (p == NULL) {
        printf("힙 영역에 동적 메모리 할당 실패\n");
        return 1; // 할당 실패 시 비정상 종료
    }

    // 값 할당 및 출력
    for (i = 0; i < 4; i++) {
        p[i] = i; // 값 할당
        printf("주소: %p\n", (void*)&p[i]); // 요소 주소 출력
        printf("값 : %d\n", p[i]); // 요소 값 출력
    }

    // 메모리 해제 및 포인터 초기화
    free(p);
    p = NULL;

    return 0;
}
```

동적 메모리 할당 함수, 해제 함수

```
/* 17-5.c */
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    // 정수 하나를 위한 동적 메모리 할당
    int* p1 = (int*)calloc(1, sizeof(int)); // 0으로 초기화됨
    int* p2 = (int*)malloc(sizeof(int));    // 초기화되지 않음 (쓰레기 값)

    // 할당된 값 출력
    printf("p1 값 (calloc으로 할당): %d\n", *p1);
    printf("p2 값 (malloc으로 할당): %d\n", *p2); // 경고: 초기화되지 않은 값을 읽음

    // 메모리 해제
    free(p1);
    p1 = NULL;

    free(p2);
    p2 = NULL;

    return 0;
}
```

동적 메모리 할당 함수, 해제 함수

- **realloc()** 함수를 이용한 동적 메모리 **재할당**
 - malloc(), calloc() 함수는 동적 메모리를 할당 후 **메모리 변경 불가**
 - **realloc()** 함수로 해결

종류	함수	반환 값
메모리 할당 함수	<code>void* realloc(void* p, size_t size);</code>	성공 : 재할당된 메모리의 시작 주소 반환 실패 : NULL 반환

동적 메모리 할당 함수, 해제 함수

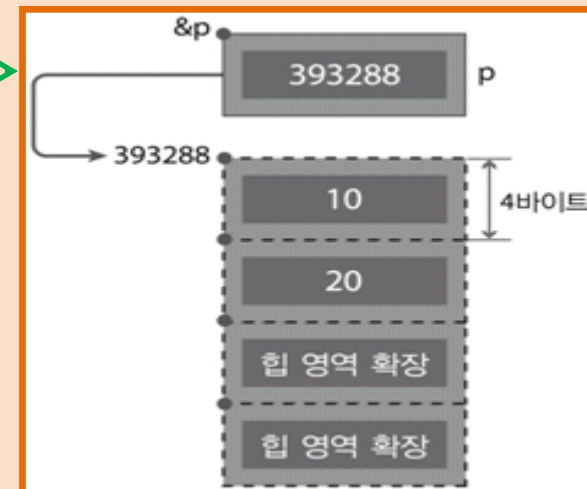
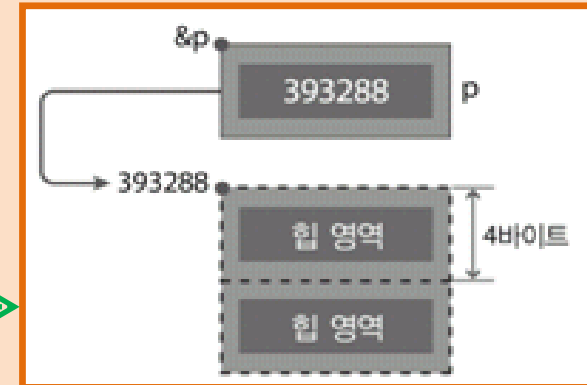
```
/* 17-6.c */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i=0;

    int* p=(int*) malloc(sizeof(int)*2);
    p[0]=10;
    p[1]=20;

    p=(int*) realloc(p, sizeof(int)*4);
    p[2]=30;
    p[3]=40;

    for(i=0; i<4; i++)
        printf("p[%d] : %d \n", i, p[i]);

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



동적 메모리 할당 함수, 해제 함수

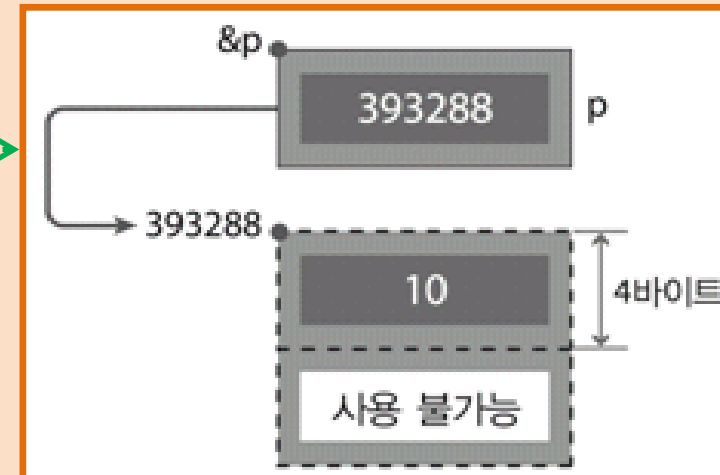
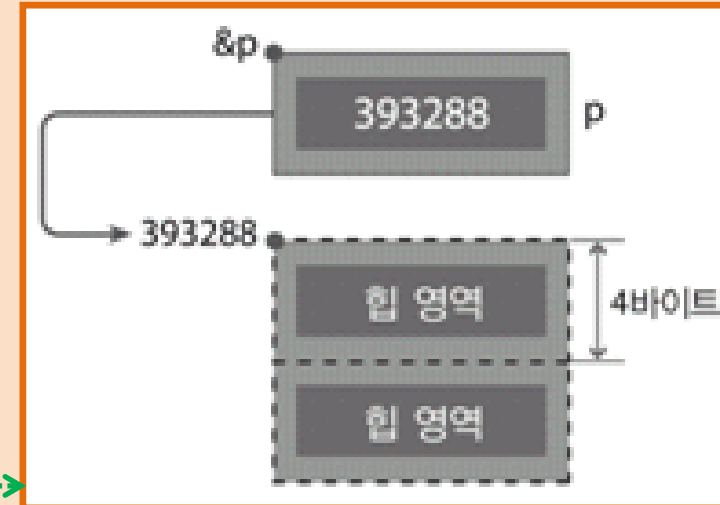
```
/* 17-7.c */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int i=0;

    int* p=(int*) malloc(sizeof(int)*2);
    p[0]=10;
    p[1]=20;

    p=(int*) realloc(p, sizeof(int)*1);
    p[0]=30;

    for(i=0; i<2; i++)
        printf("p[%d] : %d \n", i, p[i]);

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



동적 메모리 할당 함수, 해제 함수

- 메모리 영역의 특징

특징	코드, 스택, 데이터 영역	힙영역
메모리 할당	컴파일 시간에 할당	런타임 시간(실행 시간)에 할당
메모리 해제	자동 해제	free() 함수로 해제
메모리 관리	컴파일러	프로그래머

Summary

- 프로세스의 메모리 구조
 - 코드 영역, 스택 영역, 데이터 영역, 힙 영역
- 동적 메모리 할당의 필요성
- 동적 메모리 할당 함수들 `malloc()`, `calloc()`, `realloc()`
- 동적 메모리 해제 함수 `free()`