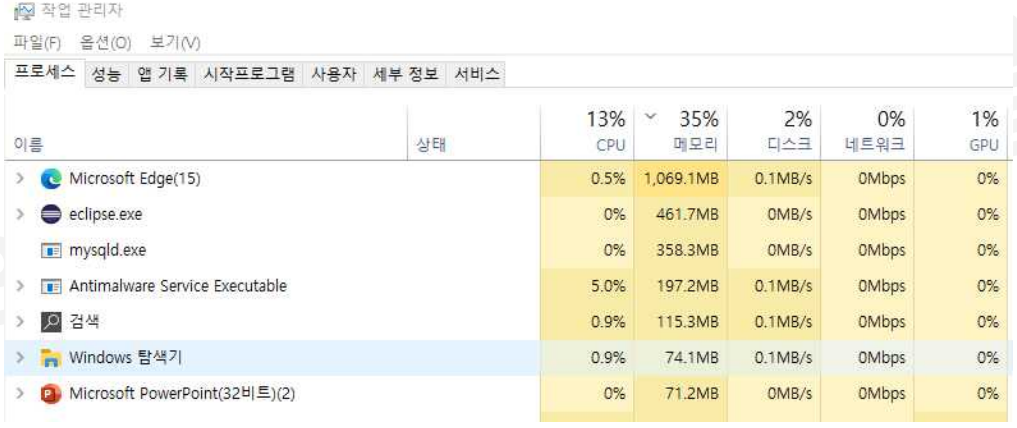
작성자 : 김기희 (cupid072@naver.com)

동적 할당

- 학습목표
  - C 프로그램 실행 메모리의 세그먼트 종류를 기술할 수 있다.
  - 각 세그먼트를 사용하는 요소를 기술할 수 있다.
  - Heap Segment의 특징을 설명할 수 있다.
  - 동적 할당의 필요성을 설명할 수 있다.
  - 동적 할당 함수를 사용하여 메모리를 확보할 수 있다.
  - 동적 할당 받은 메모리의 크기를 조정할 수 있다.
  - 동적 할당 받은 메모리를 해제할 수 있다.
  - 1차원 배열 및 2차원 배열 공간을 동적할당 할 수 있다.
  - 포인터 배열 공간을 동적할당 할 수 있다.

C 프로그램 메모리 구조

- 프로그램은 운영체제로부터 메모리를 할당 받고 할당 받은 메모리를 사용하여 프로그램을 실행한다.
- 프로그램 실행을 위해 운영체제로부터 할당 받은 메모리 공간을 프로세스(process)라고 한다.



C 프로그램 메모리 구조

- C 프로그램은 운영체제로부터 할당 받은 공간을 용도에 따라 여러 구역으로 나누어 사용한다.
- 이때 용도에 따라 나누어진 구역을 세그먼트(segment)라고 한다.

배치		Segment	
MCU	Embeded		
RAM	RAM	BSS	초기화되지 않은 전역 변수
		DATA	초기화된 전역 변수
ROM / FLASH	RAM	CONST	변수 초기화 데이터
		RODATA	상수 값 데이터
		TEXT	기계어 코드
RAM	RAM	HEAP	동적할당
RAM	RAM	STACK	자동변수

BSS & DATA Segment

- BSS와 DATA 세그먼트는 전역 변수와 static 변수에 할당에 사용되는 메모리 영역이다.
- BSS는 비초기화 영역이라고 초기 값을 지정하지 않은 전역 변수의 할당에 사용된다.

```
int a;                  // 초기 값이 지정되지 않은 전역 변수
int ary[10];            // 초기 값이 지정되지 않고 전역으로 선언된 배열
```

- DATA는 초기화 영역이라고 하며 초기 값이 지정된 전역 변수와 static 변수의 할당에 사용된다.

```
int a = 0;              // 초기 값이 지정된 전역 변수
int ary[10] = {0};      // 초기 값이 지정되어 전역으로 선언된 배열
void func() {
  static int a;         // static 변수
}
```

CONST, RODATA, TEXT Segment

- CONST Segment는 변수 초기화 데이터가 저장되는 영역이다.

```
int age = 25;                                   // 변수 초기화 데이터 25
struct person_t person = { "홍길동", 20 };      // 구조체 변수 초기화 데이터 "홍길동", 20
```

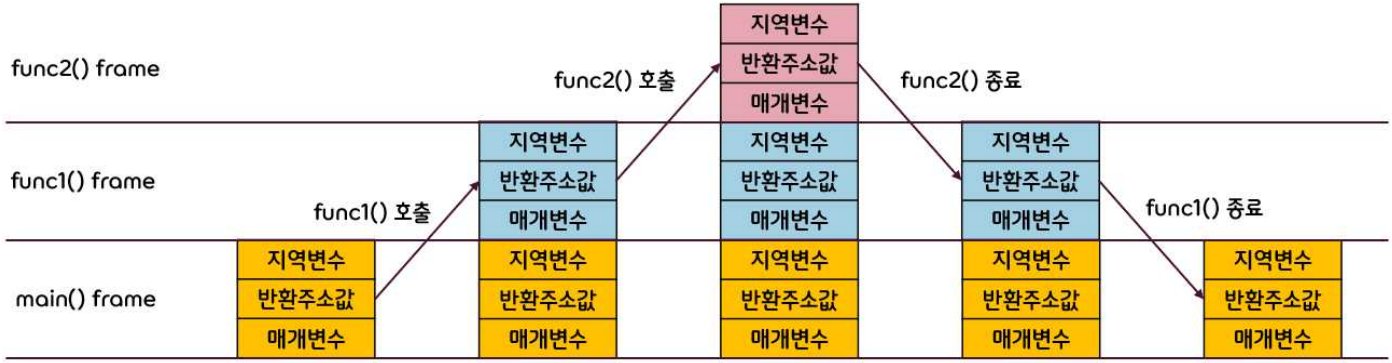
- RODATA Segment는 ReadOnly DATA라고 하여 상수 값 데이터가 저장된다.

```
printf("이름 : %s\n", name);                    // 출력 포멧 문자열 "이름 : %\n"
```

- TEXT Segment는 함수 실행 코드가 저장된다.

STACK Segment

- 함수가 호출되면 Stack(Function Call Stack) 세그먼트에 함수 실행을 위한 공간이 할당되고 할당된 공간을 사용하여 함수가 실행된다.
- 함수 실행을 위해 Stack 상에 할당된 공간을 stack frame이라고 한다.
- Stack 세그먼트는 함수의 호출과 관계되는 지역변수(매개변수 포함)가 저장되는 영역이다.
- 더 이상 Stack Frame을 생성할 수 없을 경우 Stack Overflow가 발생한다.



HEAP Segment

- HEAP Segment는 동적 할당에 사용되는 영역이다.
- 프로그램에 의해 자동으로 생성되고 관리되는 STACK Segment와는 달리 HEAP Segment는 프로그래머에 의해 생성되고 소멸된다.
- 프로그래머에 의해 생성되고 소멸된다는 것은 동적할당 함수 호출에 의해 HEAP Segment 상의 영역을 할당 받고 할당 해제 함수 호출에 의해 할당 영역을 반환한다는 것을 의미한다.
- HEAP Segment는 동적 할당 함수 호출에 의해 할당 되며 동적 할당 해제 함수 호출에 의해 반환된다.
- 그러므로 사용이 완료된 동적 메모리는 반드시 동적 할당 해제 함수 호출을 통해 반환해야 한다.
- 동적 할당 메모리를 반환하지 않으면 메모리릭(메모리 누수) 현상이 발생되어 더 이상 프로그램을 실행 불가능한 상태에 빠질 수 있다.

동적 할당이 필요한 이유

- 효율적인 메모리 사용

  학생 성적 처리 시 학생의 수를 알 수 없거나 유동적인 경우
  학생 수를 입력 받아 입력 받은 수만큼의 배열 공간을 생성하여 사용

- 큰 용량의 메모리를 필요로 하는 경우

  함수 내 지역 변수로 할당 가능한 메모리의 용량 한계는 1MB (커널 설정에 따라 다름) 이며 용량 한계를 초과하는 경우 Stack Overflow 에러가 발생한다.

- 함수 종료 이후에도 데이터를 유지하고자 하는 경우

  함수 내 지역 변수는 함수가 종료되면 함수 Stack frame 소멸로 인해 함께 소멸된다.

동적 할당 및 해제 함수

- void* malloc(size_t size);
  - 인수로 주어진 수 만큼의 바이트 크기의 메모리 공간을 할당한 후 할당된 메모리 공간의 선두 번지수를 반환한다.
  - malloc() 함수에 의해 할당된 메모리 공간은 초기화 되지 않은 상태이다.
  - 동적 할당에 실패하는 경우 null 을 반환한다.
  - 동적 할당 가능한 최대 크기는 프로그램이 구동되는 환경(플랫폼)에 따라 다르다. (IBM 기준 16,711,568 Byte)

- void* calloc(size_t count, size_t eltsize);
  - 첫 번째 인수로 주어진 개수 만큼의 메모리 공간을 할당한 후 메모리 공간의 선두 번지수를 반환한다.
  - 두 번째 인수는 첫 번째 인수로 전달된 공간 하나의 크기이다.
  - calloc() 함수에 의해 할당된 메모리 공간은 모든 바이트가 0으로 초기화 된다.
  - 요소 하나 당 크기의 값이 0이거나 용량이 부족하여 동적 할당에 실패하는 경우 null 을 반환한다.

동적 할당 및 해제 함수

- void* realloc(void* ptr, size_t newsize);
  - 첫 번째 인수로 전달된 포인터에 할당된 동적 할당 크기를 축소 또는 확장한다.
  - 연속된 메모리 공간 할당이 불가능 한 경우 새로운 영역을 할당하여 기존 데이터를 복사한 후 새로 할당된 메모리의 선두 번지수를 반환한다.
  - 확장 및 축소 대상이 NULL 인 경우 두 번째 인수의 수 만큼 동적 할당을 한 후 할당 된 메모리의 선두 번지수를 반환한다.
  - 두 번째 인수가 0인 경우 relloc() 함수는 첫 번째 인수로 전달된 포인터에 할당된 메모리를 해제한 후 NULL을 반환한다.

- void free(void *ptr);
  - 인수로 전달된 포인터에 할당된 메모리를 해제한다.
  - malloc(), calloc(), realloc() 함수 호출에 의해 동적 할당된 메모리를 해제한다.
  - 인수로 전달된 포인터가 NULL 인 경우 아무런 작동도 수행되지 않는다.
  - 이미 해제된 메모리 공간에 대한 free() 함수 호출은 에러를 유발한다.
  - 동적 할당 메모리를 해제한다는 것은 자원을 운영체제에 반환한다는 것을 의미한다.

동적 할당 사용시 주의 점

- 동적 메모리 할당 함수 호출 후에는 반드시 동적 할당 성공 여부를 확인하도록 한다.

```
int* jumsu = (int*) malloc(10 * sizeof(int));
또는
int* jumsu = (int *) calloc(10, sizeof(int));
if (jumsu == null) {
    동적 할당 실패에 대한 처리
}
.....
free(jumsu);
jumsu = NULL;
```

- 동적 할당 메모리 사용이 완료된 후에는 반드시 할당 받은 메모리를 해제한다.
- 동적 할당이 해제된 메모리 공간을 가리키는 포인터는 반드시 NULL 값을 대입한다.

1차원 배열의 동적 할당

- 1차원 배열은 연속된 저장공간 이므로 배열 원소의 값을 저장하기 위한 크기 만큼 연속된 공간을 할당 받아야 한다.

- malloc() 함수 사용

```
int *ary = (int*) malloc(10 * sizeof(int));          // int 형 크기 10개를 할당
```

- calloc() 함수 사용

```
int* ary = (int*) calloc(10, sizeof(int));          // int형 크기 10개의 블록을 할당
```

2차원 배열의 동적 할당

- 2차원 배열의 동적할당은 행 * 열 * 자료형의 크기 만큼을 할당하여야 한다.

- malloc() 함수 사용

  int (*ary)[10] = (int(*)[10]) malloc(5 * 10 * sizeof(int));

- calloc() 함수 사용

  int (*ary)[10] = (int(*)[10]) calloc(5, 10 * sizeof(int)); // 또는

  int (*ary)[10] = (int(*)[10]) calloc(5 * 10, sizeof(int));

포인터 배열의 동적 할당

- 포인터 배열 원소 하나의 크기는 주소 상수를 저장하는 포인터 하나의 크기이다.
- 그러므로 포인터 배열의 전체 할당 크기는 포인터 변수의 크기 * 원소의 수가 된다.

- malloc() 함수 사용

```
int** ary = (int**) malloc(ROW * sizeof(int *))
```

- calloc() 함수 사용

```
int** ary = (int**) calloc(ROW, sizeof(int *));
```

- 포인터 변수 ary는 포인터 배열이므로 ary를 구성하는 원소를 참조하기 위해서는 각 원소에 대하여 동적할당을 수행하여야 한다.

```
ary[0] = (int *) malloc(COLUMN * sizeof(int));
ary[0][0] = 10;
.....
```