

10장 메모리 관리

숙명여대 창병모

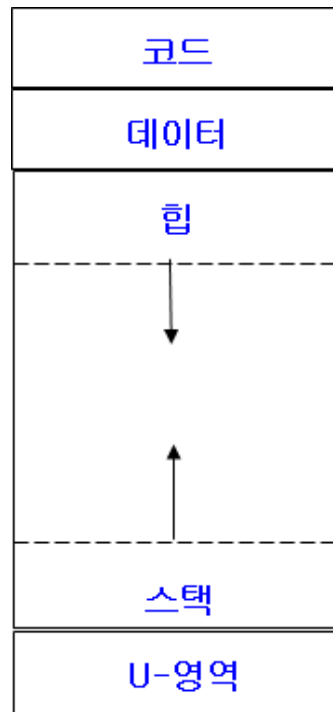
10.1 변수와 메모리

프로세스

- 프로세스는 실행중인 프로그램이다.
- 프로그램 실행을 위해서는
 - 프로그램의 코드, 데이터, 스택, 힙, U-영역 등이 필요하다.
- 프로세스 이미지(구조)는 메모리 내의 프로세스 레이아웃
- 프로그램 자체가 프로세스는 아니다 !

프로세스 구조

- 프로세스 구조



- 코드 세그먼트(code segment)
 - 기계어 명령어
- 데이터 세그먼트(data segment)
 - e.g. `int maxcount = 99;` (initialized)
 - e.g. `long sum[1000];` (uninitialized)
- 스택(stack)
 - 지역 변수, 매개 변수, 반환주소, 반환값, 등
- 힙(heap)
 - 동적 메모리 할당
 - `malloc()` in C,
 - `new class()` in Java

vars.c

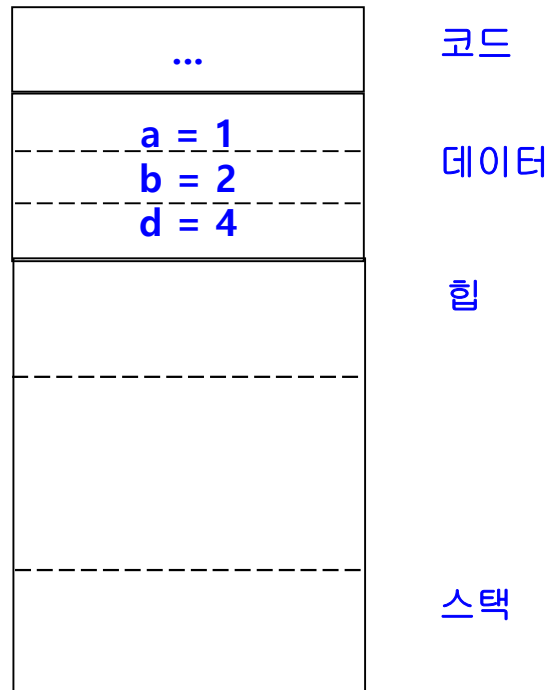
```
#include <stdio.h>
#include <stdlib.h>
int a = 1;
static int b = 2;

int main() {
    int c = 3;
    static int d = 4;
    char *p;

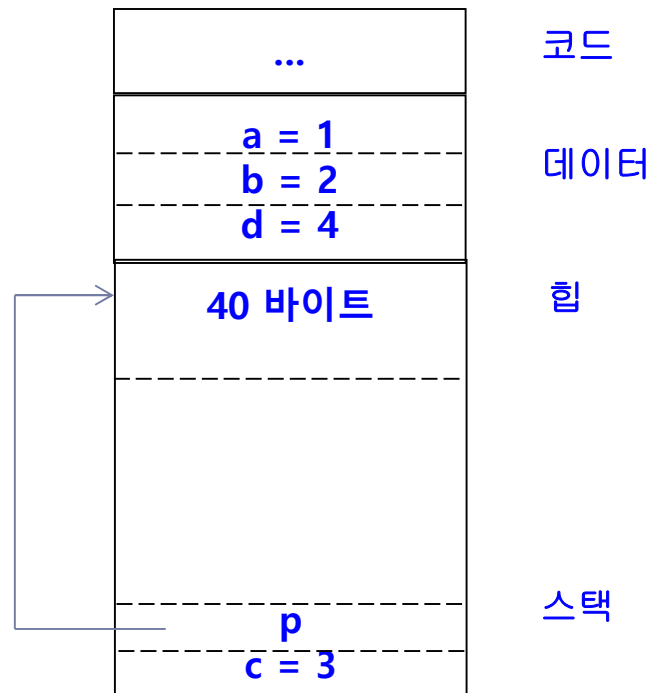
    p = (char *) malloc(40);
    fun(5);
}

void fun(int n)
{
    int m = 6;
    ...
} 5
```

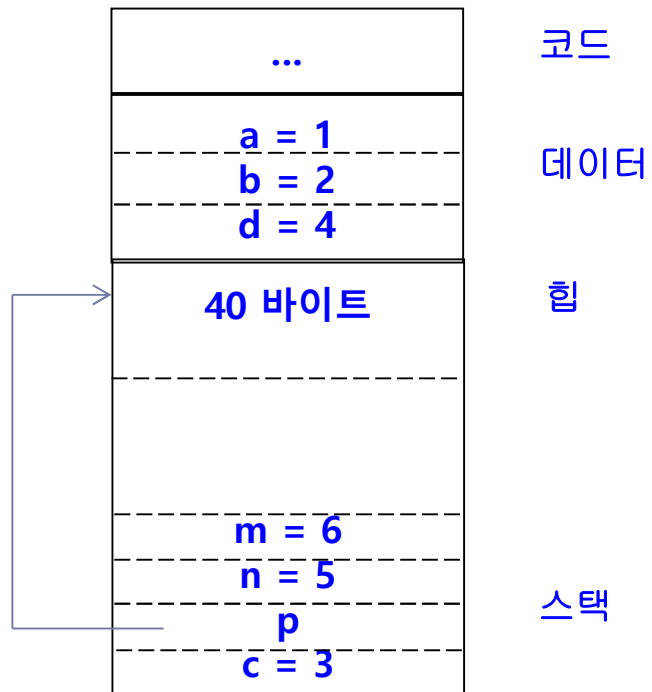
프로그램 시작할 때 메모리 영역



main() 함수 실행할 때 메모리 영역



함수 fun() 실행할 때 메모리 영역



할당 방법에 따른 변수들의 분류

변수 구분	변수 종류
정적 변수	전역변수, static 변수
자동 변수	지역변수, 매개변수
동적 변수	힙 할당 변수

10.2 동적 할당

동적 메모리 할당

- 동적 할당을 사용하는 이유
 - 필요할 때 필요한 만큼만 메모리를 요청해서 사용하여
 - 메모리를 절약한다.
- malloc()
- calloc()
- realloc()
- free()

메모리 할당

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

```
void *malloc(size_t size);
```

size 바이트만큼의 메모리 공간을 할당하며 그 시작주소를 void* 형으로 반환한다.

```
void free(void *ptr);
```

포인터 p가 가리키는 메모리 공간을 해제한다.

- 힙에 동적 메모리 할당
- 라이브러리가 메모리 풀을 관리한다
- malloc() 함수는 메모리를 할당할 때 사용하고 free()는 할당한 메모리를 해제할 때 사용한다.

메모리 할당 예

- `char *ptr;`
- `ptr = (char *) malloc(40);`

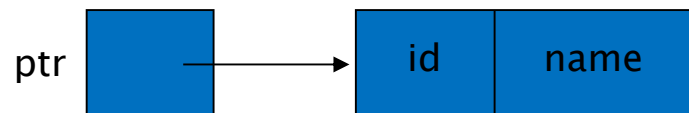


- `int *ptr;`
- `ptr = (int *) malloc(10 * sizeof(int));`



구조체를 위한 메모리 할당 예

```
struct student {  
    int id;  
    char name[10];  
};  
struct student *ptr;  
ptr = (struct student *) malloc(sizeof(struct student));
```



구조체 배열을 위한 메모리 할당 예

```
struct student *ptr;  
ptr = (struct student *) malloc(n * sizeof(struct student));
```



stud1.c

```
#include <stdio.h>
#include <stdlib.h>
struct student {
    int id;
    char name[20];
};

/* 입력받을 학생 수를 미리 입력받고 이어서 학생 정보를 입력받은 후,
이들 학생 정보를 역순으로 출력하는 프로그램 */
int main()
{
    struct student *ptr; // 동적 할당된 블록을 가리킬 포인터
    int n, i;
    printf("몇 명의 학생을 입력하겠습니까? ");
    scanf("%d", &n);
    if (n <= 0) {
        fprintf(stderr, "오류: 학생 수를 잘못 입력했습니다.\n");
        fprintf(stderr, "프로그램을 종료합니다.\n");
        exit(1);
    }
}
```


stud1.c

```
ptr = (struct student *) malloc(n * sizeof(struct student));
if (ptr == NULL) {
    perror("malloc");
    exit(2);
}

printf("%d 명의 학번과 이름을 입력하세요.\n", n);
for (i = 0; i < n; i++)
    scanf("%d %s\n", &ptr[i].id, ptr[i].name);

printf("\n* 학생 정보(역순) *\n");
for (i = n-1; i >= 0; i--)
    printf("%d %s\n", ptr[i].id, ptr[i].name);

printf("\n");
exit(0);
}
```

배열 할당

- 같은 크기의 메모리를 여러 개를 할당할 경우

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

```
void *calloc(size_t n, size_t size);
```

크기가 size인 메모리 공간을 n개 할당한다. 값을 모두 0으로 초기화한다. 실패하면 NULL를 반환한다.

- 이미 할당된 메모리의 크기 변경

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

```
void *realloc(void *ptr, size_t newsize);
```

ptr이 가리키는 이미 할당된 메모리의 크기를 newsize로 변경한다.

calloc() 예

```
int *p,*q;  
p = malloc(10*sizeof(int));  
if (p == NULL)  
    perror("malloc");  
  
q = calloc(10, sizeof(int));  
if (q == NULL)  
    perror("calloc");
```

10.3 연결 리스트

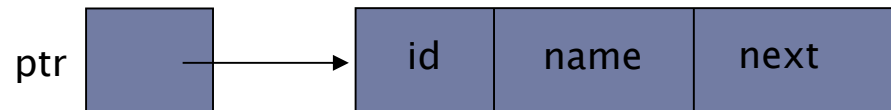
연결 리스트의 필요성

- 예: 여러 학생들의 데이터를 저장해야 한다고 생각해보자.
 - 가장 간단한 방법은 구조체 배열을 선언하여 사용하는 것이다.
 - 이 방법은 배열의 크기를 미리 결정해야 하는 문제점이 있다.
 - 배열의 크기보다 많은 학생들은 처리할 수 없으며 이보다 적은 학생들의 경우에는 배열의 기억공간은 낭비된다.
- **연결리스트(linked list)**를 사용하여 해결할 수 있다.



자기 참조 구조체를 위한 메모리 할당

```
struct student {  
    int id;  
    char name[20];  
    struct student *next;  
};  
struct student *ptr;  
ptr = (struct student *) malloc(sizeof(struct student));
```



stud2.c

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

```
/* 학생 정보를 입력받아 연결 리스트에 저장하고 학생 정보를 역순으로
   출력한다. */
```

```
int main()
{
    int count = 0, id;
    char name[20];
    struct student *p, *head = NULL;

    printf("학번과 이름을 입력하세요\n");
```

stud2.c

```
while (scanf("%d %s", &id, name) == 2) {  
    p = (struct student *) malloc(sizeof(struct student));  
    if (p == NULL) {  
        perror("malloc");  
        exit(1);  
    }  
    p->id = id;  
    strcpy(p->name, name);  
    p->next = head;  
    head = p;  
}
```


stud2.c

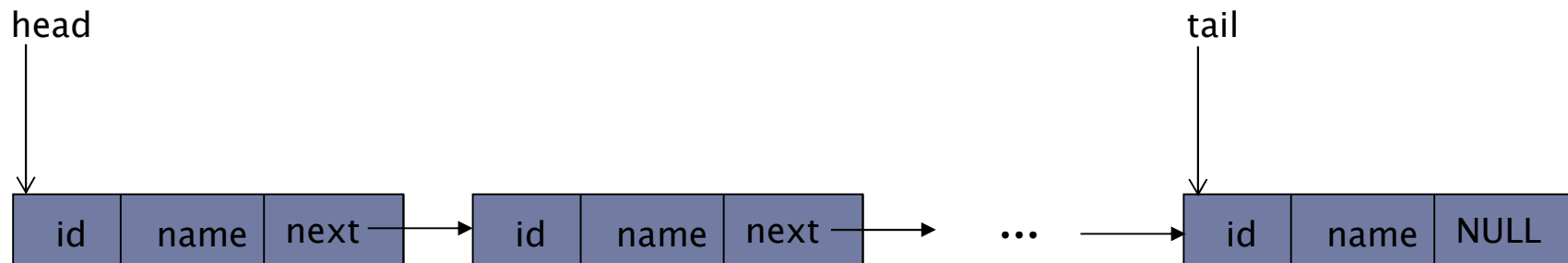
```
printf("\n* 학생 정보(역순) *\n");
p = head;
while (p != NULL) {
    count++;
    printf("학번: %d 이름: %s \n", p->id, p->name);
    p = p->next;
}
printf("총 %d 명입니다.\n", count);
exit(0);
}
```

큐 형태의 연결 리스트

```
struct student *ptr, *head = NULL, *tail = NULL;  
ptr = (struct student *) malloc(sizeof(struct student));
```

...

```
tail->next = ptr;      // 큐의 끝에 연결  
tail = ptr;           // 큐의 끝을 가리킴
```



10.4 메모리 관리 함수

메모리 관리 함수

```
# include <string.h>
```

```
void *memset(void *s, int c, size_t n);
```

s에서 시작하여 n 바이트만큼 바이트 c로 설정한 다음에 s를 반환한다.

```
int memcmp(const void *s1, const void *s2, size_t n);
```

s1과 s2에서 첫 n 바이트를 비교해서, 메모리 블록 내용이 동일하면 0을 반환하고 s1이 s2보다 작으면 음수를 반환하고, s1이 s2보다 크다면 양수를 반환한다.

```
void *memchr(const void *s, int c, size_t n);
```

s가 가리키는 메모리의 n 바이트 범위에서 문자 c를 탐색한다. c와 일치하는 첫 바이트에 대한 포인터를 반환하거나, c를 찾지 못하면 NULL을 반환한다.

```
void *memmove(void *dst, const void *src, size_t n);
```

src에서 dst로 n 바이트를 복사하고, dst를 반환한다.

```
void *memcpy(void *dst, const void *src, size_t n);
```

src에서 dst로 n 바이트를 복사한다. 두 메모리 영역은 겹쳐지지 않는다. 만일 메모리 영

역을 겹쳐서 쓰길 원한다면 memmove() 함수를 사용해라. dst를 반환한다.

mem.c

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

void main()
{
    char str[32]="Do you like Linux?";
    char *ptr,*p;

    ptr = (char *) malloc(32);
    memcpy(ptr, str, strlen(str));
    puts(ptr);
    memset(ptr+12,'l',1);
    puts(ptr);

    p = (char *) memchr(ptr,'l',18);
    puts(p);
    memmove(str+12,str+7,10);
    puts(str);
}
```