Dynamic Allocation

C 중급 세미나 - 김혜윤 -

Contents

1. 함수 포인터
   - 정의 / 활용

2. 동적 할당
   - malloc 함수/ 할당 위치
   - 2차원 배열 / 구조체

3. 실습

함수 포인터

함수 포인터 정의

```c
#include <stdio.h>

int func(int x) {
    return 2*x;
}

void main() {
    int i;
    int (*pf)(int a) = func;
    i = pf(2);
    printf("func(2): %d\n", i);
}
```

우선순위	연산자(연산기호)
1	() [] -> .
2	! ~ ++ -- * & sizeof() (자료형)
3	* / %

- 함수 이름: 함수 시작 주소
- int (*pf)(int) : 함수 포인터(괄호 필수)
- Int *pf (int) : int* 리턴하는 함수
- int (*pf)(int a) ⇔ int (*pf)(int)
  인수 생략 가능
- (*pf)(2) ⇔ pf(2)
  (*pf) 괄호 필수

```
1   func(2): 4
```

함수 포인터 활용

```c
#include <stdio.h>

int add(int a, int b) {
   return a + b;
}

int sub(int a, int b) {
   return a - b;
}

void func(int a, int b, int (*ptr)(int, int)) {
   printf("ptr(%d, %d): %d\n", a, b, ptr(a, b));
}

void main() {
   int a = 5, b = 3;
   int (*pf)(int, int);

   if(a % 2)
       pf = add;
   else
       pf = sub;

    func(a, b, pf);
}
```

- typedef와 사용

```c
typedef int (*PF)(int, int);
PF pf;
```

- 함수 포인터를 인수로 사용
  -> 인수로 사용하는 함수를 변경할 수 있음

```
1   ptr(5, 3): 8
```

동적 할당

malloc 함수

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

int main() {
    int *arr;
    arr = (int *)malloc(sizeof(int) * 5); // 크기 5 int형 배열 동적할당

    for (int i = 0; i < 5; ++i)
        printf("arr[%d]: %d\n", i, arr[i] = i);

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

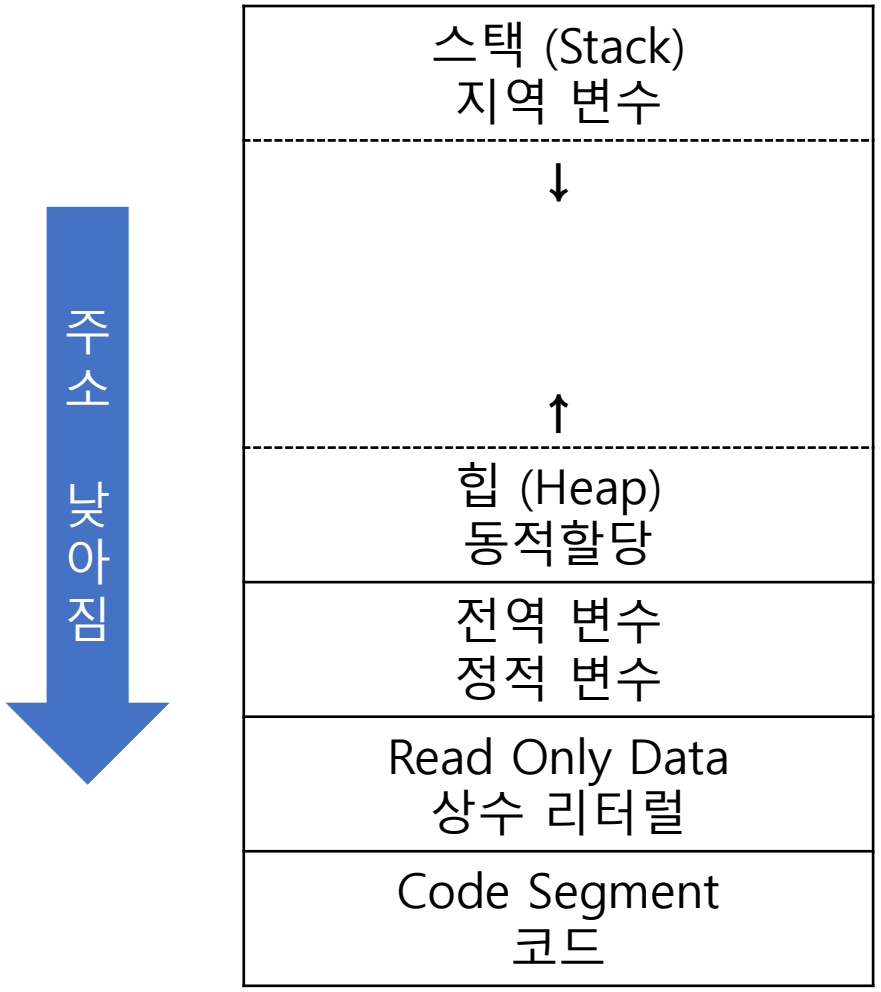
- #include <stdlib.h>

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

- 인자: 할당할 메모리 크기
  반환: 메모리 시작 주소값
- malloc (=memory allocation)
  malloc -> free 필수

```
1  arr[0]: 0
2  arr[1]: 1
3  arr[2]: 2
4  arr[3]: 3
5  arr[4]: 4
```

동적할당의 할당 위치



- 스택(Stack): 아래로 증가
- 힙(Heap): 위로 증가
  사용자가 관리

2차원 배열 동적할당

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

int main() {
    char **names;
    char name[3][7] = {"apple", "banana", "cat"};
    names = (char **)malloc(sizeof(char *) * 3);

    for (int i = 0; i < 3; ++i) {
        names[i] = (char *)malloc(sizeof(char)*strlen(name[i]) + 1);
        strcpy(names[i], name[i]);
    }

    for (int j = 0; j < 3; ++j) {
        printf("names[%d]: %s\n", j, names[j]);
        free(names[j]);
    }

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

```
1  names[0]: apple
2  names[1]: banana
3  names[2]: cat
```

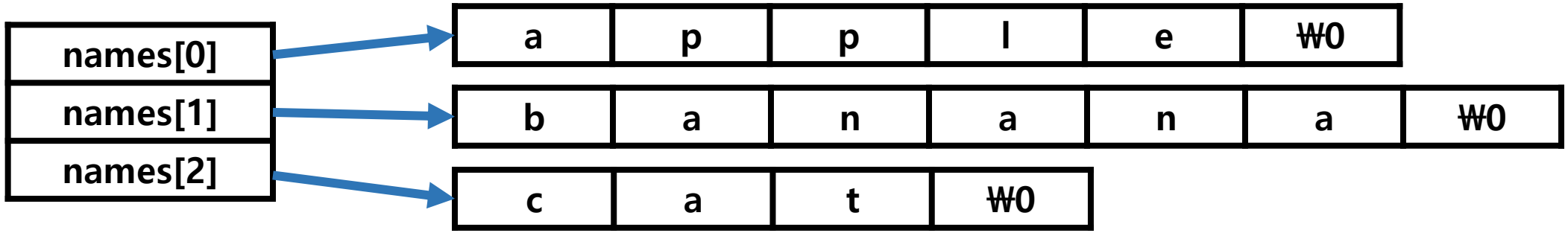
2차원 배열 동적할당

```
char **names;
char name[3][7] = {"apple", "banana", "cat"};
names = (char **)malloc(sizeof(char *) * 3);

for (int i = 0; i < 3; ++i) {
    names[i] = (char *)malloc(sizeof(char)*strlen(name[i]) + 1);
    strcpy(names[i], name[i]);
}
```

a	p	p	l	e	₩0	
b	a	n	a	n	a	₩0
c	a	t	₩0			

2차원 배열 동적할당

```
char **names;
char name[3][7] = {"apple", "banana", "cat"};
names = (char **)malloc(sizeof(char *) * 3);

for (int i = 0; i < 3; ++i) {
    names[i] = (char *)malloc(sizeof(char)*strlen(name[i]) + 1);
    strcpy(names[i], name[i]);
}
```



구조체 동적할당

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

typedef struct Food {
    char name[10]; // 이름
    int expiration_date[3]; // 유통기한
} Fd;

int main() {
    Fd *pFood;
    pFood = (Fd *)malloc(sizeof(Fd));

    strcpy(pFood->name,"apple");
    pFood->expiration_date[0] = 2020;
    pFood->expiration_date[1] = 5;
    pFood->expiration_date[2] = 28;

    printf("%s\n%d년\t%d월\t%d일", pFood->name, pFood->expiration_date[0],
                    pFood->expiration_date[1], pFood->expiration_date[2]);

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

sizeof(구조체) 사용 필수

```
1   apple
2   2020년  5월  28일
```

실습

실습 코드 설명 기본 변수 및 구조체

```
char kind_str[5][7] = { "Meat", "Dairy", "Fruit", "Bread", "Sweets" };

typedef struct Food {
    char *name;
    int expiration_date[3];
    int num;
}Food;

typedef struct Foods {
    int Foodnum, capacity;
    Food* pfood;
}Foods;
```

실습 코드 설명 main 함수

```
void main() {
    int choice; // 유저가 선택한 메뉴
    int food_num = 0; // 현재 식재료 개수
    int date[3] = { 2020, 3, 15 }; // 오늘 날짜
    Foods foods[5];
    for (int i = 0; i < 5; ++i) {
        foods[i].pfood = (Foods*) malloc(sizeof(Food) * 2);
        foods[i].capacity = 2;
        foods[i].Foodnum = 0;
    }
```

```
for (int i = 0; i < 5; i++)
    free(foods[i].pfood);
```

실습 구현 내용 #1

```
번호 입력 :1

식재료 종류 (0: Meat, 1: Dairy, 2: Fruit, 3: Bread, 4: Sweets):
 Dairy capacity를 4개로 늘렸습니다.
식재료 이름:Yogurt
 Yogurt를 냉장고에 넣었습니다.
식재료 유통기한(ex. 2020 1 23):2020 3 30
 Yogurt의 유통기한은 2020년  3월 30일입니다. 상하기 전에 빨리 먹으세요.^^
식재료 개수:3
```

```
번호 입력 :1

식재료 종류 (0: Meat, 1: Dairy, 2: Fruit, 3: Bread, 4: Sweets):3
식재료 이름:Pizza Bread
 Pizza Bread를 냉장고에 넣었습니다.
식재료 유통기한(ex. 2020 1 23):2020 1 15
 Pizza Bread의 유통기한은 2020년  1월 15일입니다. 상하기 전에 빨리 먹으세요.^^
식재료 개수:3
```

```c
void add_food(Foods* pfoods, int* food_num) {
    /*** Your Code Here ***/
    printf("add food\n"); // delete
}
```

- 식재료 종류 입력받기
- pfood가 가리키는 food배열 크기 조정(realloc)
- 띄어쓰기 포함해서 문자열 입력받기(선택)

실습 구현 내용 #2, 3, 4

```
void show_food(Foods* pfoods, int food_number) {
    /*** Your Code Here ***/
    printf("show food\n"); // delete
}

void show_expired_food(Foods* pfoods, int food_number, int date[]) {
    /*** Your Code Here ***/
    printf("show expired food\n"); // delete
}

void change_food_num(Foods* pfoods, int* food_number) {
    /*** Your Code Here ***/
    printf("change_food_num\n"); // delete
}
```

```
Dairy
1: Cheese      | 2020년  5월  6일 |  5개
Bread
2: Pizza Bread | 2020년  1월 15일 |  3개
```

- 식재료 종류를 포함해서 함수 코드를 바꾸세요
- 각 함수의 내용은 이전의 실습과 동일합니다