Lecture 10. 포인터 (2)

BAE SEKANG(배세강)

Contents

Pointer to Structure

♣ 구조체 포인터

- 구조체 변수의 시작 주소값을 저장하는 포인터

```
#include <stdio.h>

struct User {
    int id;
    int score;
};

int main() {
    struct User user1 = {1, 85};
    struct User *ptr = &user1;

    return 0;
}
```

Pointer to Structure

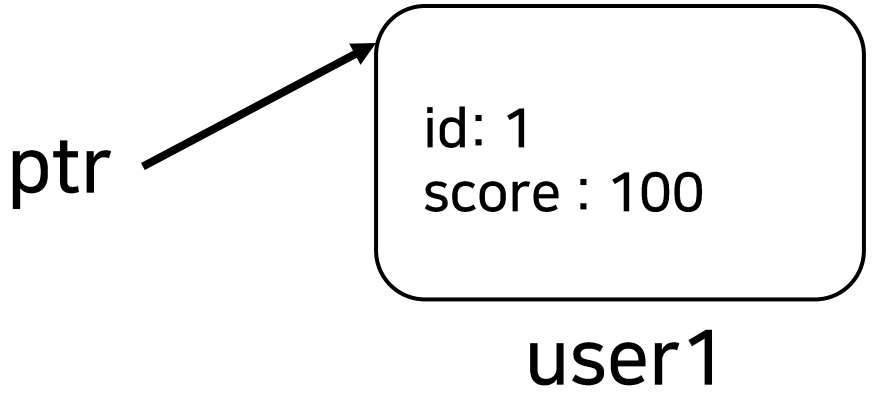
♣ Arrow Operator(화살표 연산자)

- 구조체 포인터를 통해 멤버 변수에 접근하는 방법

```
#include <stdio.h>

typedef struct User {
    int id;
    int score;
} User;

int main() {
    User user1;
    User *ptr = &user1;
    ptr->id = 1;
    ptr->score = 100;
    printf("User ID: %d, Score: %d₩n", ptr->id, user1.score);
    return 0;
}
```



Pointer to Structure

[Quiz] – 다음 프로그램의 출력 결과는?

```
#include <stdio.h>

struct Player {
    int level;
    int hp;
};

void updatePlayer(struct Player *p) {
    p->level += 1;
    p->hp = 100;
}

int main() {
    struct Player p1 = {10, 50};
    struct Player *ptr = &p1;
    updatePlayer(ptr);
    ptr->level += 5;

    printf("%d %d₩n", p1.level, p1.hp);
    return 0;
}
```

Pointer to Structure

[Quiz] – Answer: 16 100

```
#include <stdio.h>

struct Player {
    int level;
    int hp;
};

void updatePlayer(struct Player *p) {
    p->level += 1;
    p->hp = 100;
}

int main() {
    struct Player p1 = {10, 50};
    struct Player *ptr = &p1;
    updatePlayer(ptr); // 함수 호출의 결과 p1이 {11, 100}이 된다.
    ptr->level += 5;

    printf("%d %d₩n", p1.level, p1.hp);
    return 0;
}
```

Operator Precedence

♣ 연산자 우선순위

- 포인터 관련 연산자들이 가지는 서로 다른 우선 순위

순위	연산자	설명
1	(), [], ->	최우선(Postfix)
2	++, -- (후위)	후위 증감
3	!, ~, ++, -- (전위), +, -, *, &	단항 연산자(Unary/ Prefix))

[주의 사항: 같은 우선 순위일 때]

- 1, 2순위: Left-to-Right
- 3순위: Right-to-Left

Operator Precedence

♣ 연산자 우선순위

Example 1) *ptr++
Step 1: ++(후위)가 *보다 우선 순위가 높다
Step 2: ptr++가 먼저 작동하여 주소를 다음으로 옮길 준비를 한다
Step 3: *가 작동하여 증가하기 전의 주소값을 참조한다

Example 2) *++ptr
Step 1: *와 ++(전위)는 우선 순위가 3순위로 동일하다
Step 2: Right-to-Left에 따라 ++ptr이 먼저 실행된다
Step 3: 주소를 옮긴 후, 새로운 주소의 값을 *로 참조한다

Pointer to Structure

[Quiz] – 다음 프로그램의 출력 결과는?

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40};
    int *p = arr;
    ++*p;
    *p++;
    (*p)++;
    *++p = 50;

    printf("%d₩n", *(arr + 1));
    return 0;
}
```

Pointer to Structure

[Quiz] – Answer: 21

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40};
    int *p = arr;
    ++*p;                    // arr[0]이 11이 된다
    *p++;                    // p가 &arr[1]을 가리킨다
    (*p)++;                  // arr[1]이 21이 된다
    *++p = 50;               // arr[2]가 50이 된다

    printf("%d₩n", *(arr + 1));
    return 0;
}
```

Double Pointer

♣ 이중 포인터

- 포인터의 주소를 저장하는 포인터

```
#include <stdio.h>

int main() {
    int data = 100;
    int *ptr = &data;
    int **dptr = &ptr;

    **dptr = 200;

    printf("%d₩n", data);
    return 0;
}
```

Example)

Address	Variable	Value
0x1000	data	100
0x2000	ptr	0x1000
0x3000	dptr	0x2000

Double Pointer

♣ 2차원 배열

- A[i][j]를 desugaring하면 *(*(A + i) + j)과 같다.

```
#include <stdio.h>

int main() {
    int arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
    printf("%d₩n", arr[1][1]);
    printf("%d₩n", *(*(arr + 1) + 1));
    printf("%p₩n", arr[0]);
    printf("%p₩n", arr + 1);

    return 0;
}
```

```
5
5
0x7ffffaf01a70
0x7ffffaf01a7c
```

Double Pointer

♣ Multi-dimensional Array

- A[i][j][k] = *(*(*(A + i) + j) + k)
- target_addr = base_addr + (i * J * K + j * K + k) * sizeof(type)
  (array가 type arr[I][J][K]일 때; I, J, K는 각 차원별 배열의 크기)

```
#include <stdio.h>

int main() {
    int arr[2][2][3] = {
        { {1, 2, 3}, {4, 5, 6} },
        { {7, 8, 9}, {10, 11, 12} }
    };
    printf("Value: %d₩n", *(*(*(arr + 1) + 0) + 2));
    return 0;
}
```

- 실제 Memory Layout은 1차원 구조

Double Pointer

♣ N-dimensional Pointer

- 포인터 변수의 주소를 저장하는 포인터를 여러 겹 쌓아 올린 것
- 동적 할당에서 중요하게 사용됨

```
#include <stdio.h>

int main() {
    int data = 7;
    int *p1 = &data;
    int **p2 = &p1;
    int ***p3 = &p2;

    ***p3 = 77;
    printf("%d₩n", data);
    return 0;
}
```

Advanced Pointer

♣ NULL Pointer

- 주소값이 0인 포인터; 가리키는 대상이 없음을 명시

```
#include <stdio.h>

int main() {
    int *ptr = NULL;
    int data = 100;
    if (ptr == NULL) {
        printf("ptr은 현재 비어있습니다. 주소를 할당합니다.₩n");
        ptr = &data;
    }

    if (ptr != NULL) printf("데이터 접근 성공: %d₩n", *ptr);
    return 0;
}
```

Advanced Pointer

♣ NULL Pointer

- 초기화의 중요성: pointer를 선언만 하고 초기화하지 않으면 메모리의 garbage 주소를 가리킨다. 이 때 *로 접근 시 문제가 발생한다.

➔ NULL로 초기화하여 if문으로 체크할 수 있다.

[Quiz] 다음 코드의 문제점은?

```
int *p = NULL;
*p = 50;
printf("%d₩n", *p);
```

: 접근이 금지된 0번지에 간접 참조를 시도함 ➔ Segmentation Fault 발생

Advanced Pointer

♣ const pointer

- 상수 성질을 부여 받은 포인터
- 가리키는 대상의 값이나 포인터 자신의 주소값을 변경할 수 없도록 제약을 건 포인터
- const: 바로 왼쪽 대상을 상수화(왼쪽 대상이 없다면 오른쪽 대상 상수화)
- const int x;와 int const x;는 동일한 문법

1. 가리키는 대상이 상수가 되는 경우: const int *ptr;
2. 포인터 자체가 상수가 되는 경우: int * const ptr;
3. 두 대상 모두 상수로 고정하는 경우: const int * const ptr;

Advanced Pointer

♣ const pointer

1. 가리키는 대상이 상수가 되는 경우: const int *ptr

```
int a = 0;
int b = 0;

const int *ptr = &a; // *ptr이 상수
*ptr = 10;  // 불가능
ptr = &b;   // 가능
a = 10;     // 가능
```

Advanced Pointer

♣ const pointer

2. 포인터 자체가 상수가 되는 경우: int * const ptr

```
int a = 0;
int b = 0;

int * const ptr = &a; // ptr이 상수

*ptr = 10;  // 가능
ptr = &b;   // 불가능
a = 10;     // 가능
```

Advanced Pointer

♣ const pointer

3. 두 대상 모두 상수로 고정하는 경우: const int * const ptr

```
int a = 0;
int b = 0;

const int * const ptr = &a; // ptr, *ptr 모두 상수

*ptr = 10;  // 불가능
ptr = &b;   // 불가능
a = 10;     // 가능
```

Advanced Pointer

[Quiz] – 다음 중 오류가 발생하는 code line은?

```
1  #include <stdio.h>
2
3  int main() {
4      int *p = NULL;
5      int num = 10;
6      p = &num;
7      const int **pp = &p;
8      num *= 2;
9      *pp = &num;
10     **pp += 50;
11     return 0;
12 }
```

Advanced Pointer

[Quiz] – Answer: line 10

```
1   #include <stdio.h>
2
3   int main() {
4       int *p = NULL;
5       int num = 10;
6       p = &num;
7       const int **pp = &p;
8       num *= 2;
9       *pp = &num;
10      **pp += 50; // const에 의해 **pp를 통해 값을 직접 수정할 수 없다!
11      return 0;
12  }
```

Advanced Pointer

♣ void Pointer

- 가리키는 대상의 type이 결정되지 않은 포인터
- 오직 메모리 주소 값만 저장할 수 있다
- 역참조와 pointer arithmetic이 불가능하다
- 어떤 타입의 포인터라도 void *에 대입할 수 있다

```
// …
int data1 = 0x12345678;
void *ptr = &data1;
char data2 = ‘A’;
ptr = &data2;
```

- 만약 ptr이 data2를 가리키는 상태에서 *(int *)ptr로 읽으면? ➔ 4Byte 읽음(OOB)

Advanced Pointer

♣ void Pointer

```
#include <stdio.h>

void my_memcpy(void *dest, void *src, int size) {
    char *d = (char *)dest;
    char *s = (char *)src;

    for (int i = 0; i < size; i++) {
        d[i] = s[i];
    }
}

int main() {
    int a = 10, b = 0;
    double x = 5.5, y = 0.0;
    my_memcpy(&b, &a, sizeof(int));
    my_memcpy(&y, &x, sizeof(double));

    printf("b: %d, y: %.1f₩n", b, y);
    return 0;
}
```

Advanced Pointer

♣ 배열 포인터

- 배열 전체를 가리키는 포인터
- 선언 방식: int (*ptr)[3];

```
#include <stdio.h>

int main() {
    int arr[3] = {1, 2, 3};
    int *p = arr;
    int (*ap)[3] = &arr;
    return 0;
}
```

연산자 우선 순위: [] > *

- int *ptr[3];과 차이점이 무엇일까? ➔ int * 3개를 모아놓은 배열 선언
  ➔ i.e., int *ptr[3] = {&a, &b, &c};

Advanced Pointer

♣ 배열 포인터

- 배열 전체를 가리키는 포인터
- 선언 방식: type (*ptr)[3];
- 위 경우 type이 int라면, ptr += 1;의 결과는 12의 증가(3 * sizeof(int))

```
#include <stdio.h>

int main() {
    int arr[3] = {1, 2, 3};
    int *p = arr;
    int (*ap)[3] = &arr;
    return 0;
}
```

※ 연산자 우선 순위: [] > *

- int *ptr[3];과 차이점이 무엇일까? ➔ 포인터 3개를 모아놓은 배열 선언

Advanced Pointer

♣ 함수 포인터

- 함수의 실행 코드 시작 주소를 가리키는 포인터
- 선언 방식: type (*fptr)(parameter);

```
#include <stdio.h>

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

int main() {
    int (*ptr)(int, int);
    ptr = add;
    printf("더하기: %d₩n", ptr(10, 5));
    ptr = sub;
    printf("빼기: %d₩n", ptr(10, 5));
    return 0;
}
```

Advanced Pointer

♣ 함수 포인터

- Callback 함수: 함수 포인터가 인자로 넘어가 다른 함수 내부에서 호출되는 함수
- calculate 입장에서 add, mul의 구체적인 구조를 알 필요 없음(add, mul이 callback)

```
#include <stdio.h>
int add(int a, int b) {return a + b;}
int mul(int a, int b) {return a * b;}

void calculate(int x, int y, int (*op)(int, int)) {
    printf("%d₩n", op(x, y));
}

int main(void) {
    calculate(3, 4, add);
    calculate(3, 4, mul);
    return 0;
}
```

Advanced Pointer

♣ 함수 포인터 Example: qsort

- qsort(배열 주소, 원소의 개수, 원소 하나의 크기, **비교 함수**):

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

typedef struct {
    char name[20];
    int score;
} Student;

int compare_score_desc(const void *a, const void *b) {
    const Student *s1 = (const Student *)a;
    const Student *s2 = (const Student *)b;

    return s2->score - s1->score;
}

int main(void) {
    Student arr[] = {
        {"Kim", 80},
        {"Lee", 95},
        {"Park", 70},
        {"Choi", 95}
    };

    int n = sizeof(arr) / sizeof(arr[0]);
    qsort(arr, n, sizeof(Student), compare_score_desc);
    return 0;
}
```

Exercise

[Quiz] – 다음 프로그램의 출력 결과는?

```
#include <stdio.h>

int func(int x, int y, int (*arr)[3]) {
    x += (unsigned int)y * arr[1][2];
    return x;
}

int main(void) {
    int arr[2][3] = {
        {4, 7, 2},
        {9, 5, 6}
    };
    int (*p)[3] = arr;
    int (*fp)(int, int, int (*)[3]) = func;
    printf("%d₩n", fp(*(*(p + 1) + 0), (*p)[1], arr));
    return 0;
}
```

Exercise

[Quiz] – Answer: 51

```
#include <stdio.h>

int func(int x, int y, int (*arr)[3]) {
    x += (unsigned int)y * arr[1][2]; // x += 9 + 7 * 6;
    return x; // 51 반환
}

int main(void) {
    int arr[2][3] = {
        {4, 7, 2},
        {9, 5, 6}
    };
    int (*p)[3] = arr;
    int (*fp)(int, int, int (*)[3]) = func;
    printf("%d₩n", fp(*(*(p + 1) + 0), (*p)[1], arr)); // fp(9, 7, func);
    return 0;
}
```

THANK YOU!

Any Question?