

14. 함수와 포인터 활용

12주차

포인터 변수를 통한 반환 (pointerparam.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>

void add(int *, int, int);
int main(void) {
    int m = 0, n = 0, sum = 0;

    printf("두 정수 입력: ");
    scanf("%d %d", &m, &n);
    add( &sum, m, n );
    printf("두 정수 합: %d\n", sum);
    return 0;
}

void add( int *psum, int a, int b) {
    *psum = a + b;
}
```

[결과]
두 정수 입력: 3
7
두 정수 합: 10

함수로 변수의 주소를 전달하고
함수 결과값을 변수에 받기.

주소값을 반환하는 함수 (ptrreturn.c)

```
#define _CRT_SECURE_NO_WARNINGS
```

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

```
int * add( int *psum, int a, int b) {  
    *psum = a + b; return psum;  
}
```

```
int * subtract( int *pdiff, int a, int b) {  
    *pdiff = a - b; return pdiff;  
}
```

```
int * multiply(int a, int b) {  
    int mult = a * b; return &mult;  
}
```

```
int main(void) {
```

```
    int m = 0, n = 0, sum = 0, diff = 0;
```

```
    printf("두 정수 입력: ");    scanf("%d %d", &m, &n);
```

```
    printf("두 정수 합: %d\n", *add(&sum, m, n));
```

```
    printf("두 정수 차: %d\n", *subtract(&diff, m, n));
```

```
    printf("두 정수 곱: %d\n", *multiply(m, n));
```

```
    return 0;
```

```
}
```

[결과]

두 정수 입력: 3

7

두 정수 합: 10

두 정수 차: -4

두 정수 곱: 21

이렇게는 하지 말 것.

결과값이 담긴 주소를 함수 반환값으로 return하면 호출하는 쪽에서 반환값을 저장하지 않고 바로 사용 가능.

상수를 위한 const 사용

- ◆ 포인터를 매개변수로 이용하면 수정된 결과를 받을 수 있어 편리
 - 그러나 수정을 원하지 않는 포인터를 함수의 인자로 넘길 때 앞에 키워드 **const** 사용
 - > **포인터가 참조하는 변수가 수정될 수 없게 함**
- ◆ 인자 기술 방법
 - **const** double *a
 - double **const** *a
 - *a를 대입연산자의 l-value로 사용 불가능
 - 위 두 개가 같은 표현임.

함수 매개변수에 const 사용 (constreference.c)

[결과]

두 실수 입력: 1.5 3.4
두 실수 곱: 5.10, 나눗: 0.44
연산 후 두 실수: 2.50, 4.40

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
```

```
void multiply(double *result, const double *a, const double *b) {
    *result = *a * *b;
}
```

a, b 의 참조값은 수정 못함.

```
void devideandincrement(double *result, double *a, double *b) {
    *result = *a / *b;
    ++*a;
    (*b)++;
}
```

각각 a, b 가 가리키는 값을 1씩 증가.

```
int main(void) {
    double m = 0, n = 0, mult = 0, dev = 0;
    printf("두 실수 입력: "); scanf("%lf %lf", &m, &n);
    multiply( &mult, &m, &n );
    devideandincrement( &dev, &m, &n );
    printf("두 실수 곱: %.2f, 나눗: %.2f\n", mult, dev);
    printf("연산 후 두 실수: %.2f, %.2f\n", m, n);
    return 0;
}
```

함수에 구조체 전달과 반환 (complexnumber.c)

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

```
struct complex {
```

```
    double real; //실수
```

```
    double img; //허수
```

복소수 저장용

```
};
```

```
typedef struct complex complex;
```

```
void printcomplex(complex com) {
```

```
    printf("복소수(a + bi) = %5.1f + %5.1fi \n", com.real, com.img);
```

```
}
```

```
complex paircomplex1(complex com) {
```

```
    com.img = -com.img;
```

```
    return com;
```

```
}
```

```
void paircomplex2(complex *com) { com->img = -com->img; }
```

```
int main(void) {
```

```
    complex comp = { 3.4, 4.8 }, pcomp;
```

```
    printcomplex(comp);
```

```
    pcomp = paircomplex1(comp);
```

```
    printcomplex(pcomp);
```

```
    paircomplex2(&pcomp);
```

```
    printcomplex(pcomp);
```

```
    return 0;
```

[결과]

```
복소수(a + bi) = 3.4 + 4.8i
```

```
복소수(a + bi) = 3.4 + -4.8i
```

```
복소수(a + bi) = 3.4 + 4.8i
```

구조체도 user-defined type이므로
pre-defined type처럼 인자나 반환 타
입이 될 수 있다.

함수 포인터

- ◆ 함수 포인터(pointer to function)
 - 함수의 주소값을 저장하는 포인터 변수
 - **반환형, 인자목록이 일치하는** 함수의 주소를 저장할 수 있음.
- ◆ 함수 포인터 선언 방법
 - 함수원형에서 함수이름을 제외한 반환형과 인자목록의 정보가 필요
`void add(double *, double, double);`
`void (*pf1) (double *z, double x, double y) ;`
 - 함수 포인터 선언 시에는 반드시 * 를 앞에 붙이고 괄호를 사용해야 함. (괄호 없으면 함수 원형 선언이 됨!)
- ◆ 함수 포인터에 값 대입
 - `pf1 = add;` (O)
 - `pf1 = &add;` (O)
 - `pf1 = add();` (X : 함수 호출이 됨!)
- ◆ 함수 포인터를 이용한 함수 호출 (원래 함수를 호출하는 것과 같음)
 - `pf1(&d1, 1.2, 3.4);`

함수 포인터 변수 사용 (funptr.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
void add(double *z, double x, double y) { *z = x + y; }
void subtract(double *z, double x, double y) { *z = x - y; }
int main(void) {
```

```
    void(*pf)(double*, double, double) = NULL;
    double m, n, result = 0;
    printf("+, -를 수행할 실수 2개를 입력하세요. >> ");
    scanf("%lf %lf", &m, &n);
    pf = add; pf(&result, m, n);
    printf("pf = %p, 함수 add() 주소 = %p\n", pf, add);
    printf("더하기 수행: %lf + %lf == %lf\n\n", m, n, result);
    pf = subtract; pf(&result, m, n);
    printf("pf = %p, 함수 subtract() 주소 = %p\n", pf, subtract);
    printf("빼기 수행: %lf - %lf == %lf\n\n", m, n, result);
```

```
    return
```

[결과]
+, -를 수행할 실수 2개를 입력하세요. >> 5.3

1.1

pf = 00007FF6991913AC, 함수 add() 주소 = 00007FF6991913AC
더하기 수행: 5.300000 + 1.100000 == 6.400000

pf = 00007FF6991913BB, 함수 subtract() 주소 = 00007FF6991913BB

함수 포인터 배열 사용 (fparray.c)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
void add(double *z, double x, double y) { *z = x + y; }
void subtract(double *z, double x, double y) { *z = x - y; }
void multiply(double *z, double x, double y) { *z = x * y; }
void devide(double *z, double x, double y) { *z = x / y; }
int main(void) {
    char op[4] = { '+', '-', '*', '/' };
    void(*fpary[4])(double*, double, double) =
        { add, subtract, multiply, devide }; 함수 포인터 선언하면서 초기화
    double m, n, result;
    printf("사칙연산을 수행할 실수 2개를 입력하세요. >> ");
    scanf("%lf %lf", &m, &n);
    for (int i = 0; i < 4; i++) {
        fpary[i](&result, m, n); 함수 포인터 이용한 호출
        printf("%.2lf %c %.2lf == %.2lf\n", m, op[i], n, result);
    }
    return 0;
}
```

[결과]

사칙연산을 수행할 실수 2개를 입력하세요. >> 3.4 1.1

3.40 + 1.10 == 4.50

3.40 - 1.10 == 2.30

void 포인터 사용 (voidptr.c)

```
#include <stdio.h>
void myprint(void) { printf("void 포인터, 신기하네요!\n"); }
```

```
int main(void) {
    int m = 10;
    double d = 3.98;
    void *p = &m;
    printf("p 참조 정수: %d\n", *(int *)p);
```

```
    p = &d;
    printf("p 참조 실수: %.2f\n", *(double *)p);
```

```
    p = myprint;
    printf("p 참조 함수 실행 : ");
    ((void*)(void)) p();
```

```
    return 0;
```

```
}
```

void 포인터에는 (함수 포함) 어떤 타입의 주소
든지 저장 가능.

단, 타입이 없으므로 참조가 안됨. 참조하기 위
해서는 void가 아닌 타입으로 형변환해야 함!

[결과]

p 참조 정수: 10

p 참조 실수: 3.98

p 참조 함수 실행 : void 포인터, 신기하네요!