포인터와 함수

- Call-by-Value, Call-by-Reference, const 키워드 -

성공회대학교 IT융합자율학부
소프트웨어공학전공
홍 성 준

함수의 인자 전달과 매개변수

◎ "함수 호출 시 전달되는 인자의 값은 매개변수에 복사가 된다."

- 함수가 호출되고 나면 전달되는 인자와 매개 변수는 별도로 관리

```
int SimpleFunc(int num) { . . . . }
int main(void)
{
     int age=17;
     SimpleFunc(age);
     . . . .
}
```

- 함수 호출 시 변수를 인자로 전달하면 실제 전달되는 것은 변수가 아닌 변수에 저장된 값

함수의 인자로 배열 전달하기

◎ 함수의 인자로 배열을 전달하는 방법

- 함수의 매개변수로 배열을 선언할 수 없음
- 포인터로 선언된 함수의 매개변수가 배열의 주소값을 인자로 받아 함수 내에서 배열에 접근

```
void SimpleFunc(int *);

int main(void)
{
    int arr[3]={1, 2, 3};
    SimpleFunc(arr);
    . . . .
}

void SimpleFunc(int * param)
{
    printf("%d %d", param[0], param[1]);
}
```

- 주의: 함수 내에서는 인자로 전달된 배열의 길이를 확인할 수 없음. 배열의 길이를 별도의 인자로 전달 받아야 함.

함수의 인자로 배열 전달하기

◎ ArrayParam.c

- int 형 포인터 변수를 사용하여 배열 형태로 값을 참조

```
void ShowArayElem(int * param, int len)
{
    int i;
    for(i=0; i<len; i++)
        printf("%d ", param[i]);
    printf("\n");
}

int main(void)
{
    int arr1[3]={1, 2, 3};
    int arr2[5]={4, 5, 6, 7, 8};
    ShowArayElem(arr1, sizeof(arr1) / sizeof(int));
    ShowArayElem(arr2, sizeof(arr2) / sizeof(int));
    return 0;
}
```

```
1 2 3
4 5 6 7 8
```

함수의 인자로 배열 전달하기

◎ ArrayParamAccess.c

- int 형 포인터 변수로 주소값을 전달하기 때문에, 해당 메모리 공간에 접근하여 값을 변경

```
void ShowArayElem(int * param, int len)
{
    int i;
    for(i=0; i<len; i++)
        printf("%d ", param[i]);
    printf("\n");
}
void AddArayElem(int * param, int len, int add)
{
    int i;
    for(i=0; i<len; i++)
        param[i] += add;
}
```

```
int main(void)
{
    int arr[3]={1, 2, 3};
    AddArayElem(arr, sizeof(arr) / sizeof(int), 1);
    ShowArayElem(arr, sizeof(arr) / sizeof(int));
    AddArayElem(arr, sizeof(arr) / sizeof(int), 2);
    ShowArayElem(arr, sizeof(arr) / sizeof(int));
    AddArayElem(arr, sizeof(arr) / sizeof(int), 3);
    ShowArayElem(arr, sizeof(arr) / sizeof(int));
    return 0;
}
```

```
2 3 4
4 5 6
7 8 9
```

함수의 호출 방식

◎ 함수의 호출 방식

- Call-by-value : 값을 전달하는 형태의 함수 호출
  - 단순히 값을 함수의 매개변수에 전달하는 형태의 함수 호출
- Call-by-reference : 주소값을 전달하는 형태의 함수 호출
  - 메모리 접근에 사용되는 주소값을 전달하는 형태의 함수 호출

```
void NoReturnType(int num)
{
    if(num<0)
        return;
    . . . .
}
```

```
void ShowArayElem(int * param, int len)
{
    int i;
    for(i=0; i<len; i++)
        printf("%d ", param[i]);
    printf("\n");
}
```

함수의 호출 방식

◎ CallByValueSwap.c

- 변수에 저장된 값을 바꾸어주는 Swap() 함수를 만들어 보자

```
void Swap(int n1, int n2)
{
    int temp=n1;
    n1=n2;
    n2=temp;
    printf("n1 n2: %d %d \n", n1, n2);
}

int main(void)
{
    int num1=10;
    int num2=20;
    printf("num1 num2: %d %d \n", num1, num2);

    Swap(num1, num2);   // num1과 num2에 저장된 값이 서로 바뀌길 기대!
    printf("num1 num2: %d %d \n", num1, num2);
    return 0;
}
```

```
num1 num2: 10 20
n1 n2: 20 10
num1 num2: 10 20
```

함수의 호출 방식

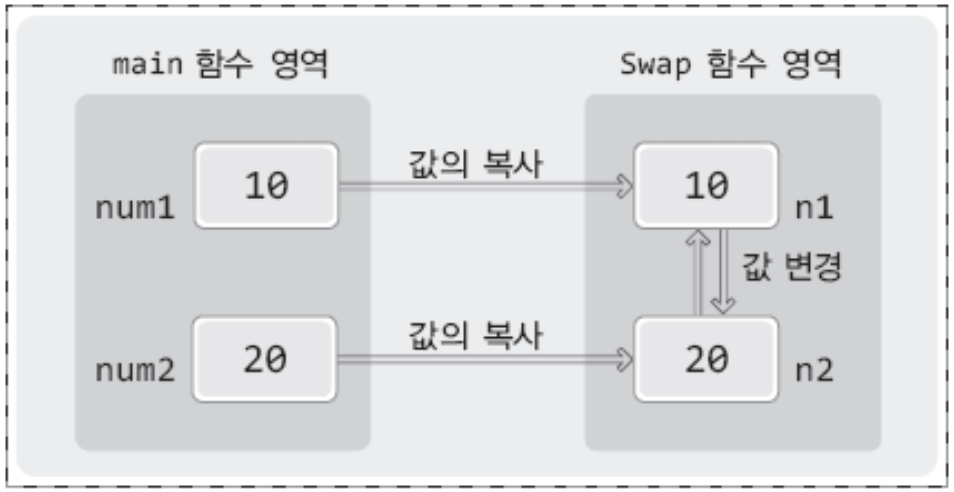
◎ CallByValueSwap.c

- 변수에 저장된 값을 바꾸어주는 Swap() 함수를 만들어 보자

```
void Swap(int n1, int n2)
{
    int temp=n1;
    n1=n2;
    n2=temp;
    printf("n1 n2: %d %d \n", n1, n2);
}

int main(void)
{
    int num1=10;
    int num2=20;
    printf("num1 num2: %d %d \n", num1, num2);

    Swap(num1, num2);   // num1과 num2에 저장된 값이 서로 바뀌길 기대!
    printf("num1 num2: %d %d \n", num1, num2);
    return 0;
}
```




```
num1 num2: 10 20
n1 n2: 20 10
num1 num2: 10 20
```

함수의 호출 방식

◎ CallByReferenceSwap.c

- 변수에 저장된 값을 바꾸어주는 Swap() 함수를 만들어 보자

```
void Swap(int * ptr1, int * ptr2)
{
    int temp = *ptr1;
    *ptr1 = *ptr2;
    *ptr2 = temp;
}

int main(void)
{
    int num1=10;
    int num2=20;
    printf("num1 num2: %d %d \n", num1, num2);
    Swap(&num1, &num2);
    printf("num1 num2: %d %d \n", num1, num2);
    return 0;
}
```

```
num1 num2: 10 20
num1 num2: 20 10
```

함수의 호출 방식

◎ scanf 함수 다시보기

- 변수 num 앞에 & 연산자를 붙이는 이유
  - scanf 함수 내부에서 외부에서 선언된 변수 num에 접근하여 값을 기록하기 위해 주소값을 전달

```
int main(void)
{
     int num;
     scanf("%d", &num);
     . . . .
}
```

- 배열 이름 str 앞에 & 연산자를 사용하지 않는 이유
  - 배열의 이름 str은 그 자체가 배열 str의 시작 위치를 나타내는 주소값이기 때문에 & 사용하지 않음

```
int main(void)
{
     char str[30];
     scanf("%s", str);
     . . . .
}
```

포인터에서의 const 사용

◎ 포인터 자료 형 앞에 붙는 경우 : 상수를 가리키는 포인터 (pointer to a constant value)

```
int main(void)
{
    int num=20;
    const int * ptr=&num;
    *ptr=30;    // 컴파일 에러!
    num=40;     // 컴파일 성공!
    . . . .
}
```

- 포인터 변수가 가리키는 대상은 변경할 수 있지만, 참조하는 대상의 변경은 허용하지 않음
- 포인터를 사용한 값의 변경에 제한을 둔 것으로 가리키는 변수를 상수로 만드는 선언이 아님
- 포인터 변수를 통해 접근할 때 변수를 상수로 취급
- 함수가 전달된 인수를 실수로 변경하지 않기 위해 사용 (배열의 함수 전달)

포인터에서의 const 사용

◎ 포인터 자료형 뒤에 붙는 경우 : 상수 포인터 (constant pointer)

- 상수 포인터 : 초기화 이후에 가리키는 주소를 변경할 수 없는 포인터 상수

```
int main(void)
{
    int num1=20;
    int num2=30;
    int * const ptr=&num1;
    ptr=&num2;      // 컴파일 에러!
    *ptr=40;        // 컴파일 성공!
    . . . .
}
```

- 포인터가 가리키는 대상은 변경할 수 없지만 참조하는 대상의 값은 변경 가능
- 즉, 한 번 주소 값이 저장되면 항상 같은 주소를 가리킴

포인터에서의 const 사용

◎ 포인터 자료형 앞/뒤에 모두 붙는 경우 : 상수를 가리키는 상수 포인터

```
const int * ptr=&num;
 int * const ptr=&num;
```

→

```
const int * const ptr=&num;
```

- 상수를 가리키는 상수 포인터는 다른 주소를 가리키도록 수정할 수도 없으며, 참조를 통해 값을 수정할 수도 없음

포인터에서의 const 사용

◎ const 키워드를 사용하는 이유?

- const는 원래 C++에서 제정된 키워드
- const 선언을 많이 하면 프로그램 코드의 안정성을 높일 수 있음

```
int main(void)
{
     double PI=3.1415;
     double rad;
     PI=3.07;   // 실수로 잘못 삽입된 문장, 컴파일 시 발견 안됨
     scanf("%lf", &rad);
     printf("circle area %f \n", rad*rad*PI);
     return 0;
}
```

```
int main(void)
{
     const double PI=3.1415;
     double rad;
     PI=3.07;   // 컴파일 시 발견되는 오류상황
     scanf("%lf", &rad);
     printf("circle area %f \n", rad*rad*PI);
     return 0;
}
```