프로그래밍 1
Lecture Note #14

백윤철
ybaek@smu.ac.kr

내용

- 함수의 인자 배열 전달
- 배열의 주소값 전달
- call by value
- const 선언

인자전달의 기본방식은 값의 복사

```
int SimpleFunc(int num) { … }

int main(void) {
  int age = 17;
  SimpleFunc(age);
  return 0;
}
```

age에 저장된 값이
매개변수 num에 복사됨

실제 전달되는 것은 age가 아니라
age에 저장된 값

배열을 함수의 인자로 전달하는 방식

```
int main(void) {
   int arr[3] = { 1, 2, 3 }
   int *ptr = arr;
   …
}
```

배열의 이름은 int형 포인터

배열의 이름은 int형 포인터! 따라서 int형 포인터 변수에 배열의 이름이 지니는 주소 값을 저장할 수 있다.

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

배열 이름 arr이 지니는 주소 값 전달

배열 이름 arr이 int형 포인터.
매개변수는 int형 포인터 변수

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

포인터 변수를 이용해도 배열 형태로 접근 가능!

배열을 함수의 인자로 전달하는 예제

```
void ShowArrayElem(int *p, int len) {
  int i;
  for (i = 0; i < len; i++)
    printf("%d ", p[i]);
  printf("\n");
}
int main() {
  int arr1[3] = { 1, 2, 3 };
  int arr2[5] = { 4, 5, 6, 7, 8 };
  ShowArrayElem(arr1, sizeof(arr1) / sizeof(int));
  ShowArrayElem(arr2, sizeof(arr2) / sizeof(int));
  return 0;
}
```

실행결과

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

배열을 함수의 인자로 전달하는 예제

```
void ShowArrayElem(int *p, int len) { … }
void AddArrayElem(int *p, int len, int add) {
  int i;
  for (i = 0; i < len; i++)
    p[i] += add;
}
int main() {
  int arr1[3] = { 1, 2, 3 };
  int size = sizeof(arr1) / sizeof(int);
  AddArrayElem(arr1, size, 1);
  ShowArrayElem(arr1, size);
  AddArrayElem(arr1, size, 2);
```

배열을 함수의 인자로 전달하는 예제

```
    ShowArrayElem(arr1, size);
    AddArrayElem(arr1, size, 3);
    ShowArrayElem(arr1, size);
    return 0;
}
```

실행결과

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

배열을 함수의 인자로 전달받는 함수의 또 다른 선언

```
void ShowArrayElem(int *p, int len) { … }
void AddArrayElem(int *p, int len, int add) { … }
```

동일한 선언

```
void ShowArrayElem(int p[], int len) { … }
void AddArrayElem(int p[], int len, int add) { … }
```

- 매개변수의 선언에서는 int* p와 int p[]가 동일함

배열을 함수의 인자로 전달받는 함수의 또 다른 선언

```
int main(void) {
  int arr[3] = { 1, 2, 3 };
  int *ptr = arr; /*int ptr[] = arr; 로 대체 불가*/
```

- 하지만 매개변수 이외의 영역에서는 int *p의 선언을 int p[]로 대체할 수 없음

값을 전달하는 형태의 함수호출: Call by value

- 함수를 호출할 때 단순히 값을 전달하는 형태의 함수호출을 가리켜 call-by-value라 하고, 메모리의 접근에 사용되는 참조(reference) 값을 전달하는 형태의 함수호출을 call-by-reference라 함 (일반론)

```
void f(int num) {
  if (num < 0)
    return ;
  …
}
```

call-by-value

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

~~call-by-reference~~

값을 전달하는 형태의 함수호출: Call by value

- call-by-value와 call-by-reference라는 용어를 기준으로 구분하는 것이 중요하지는 않음
- C 언어는 기본적으로 call-by-value 방식을 지원하고 call-by-reference를 포인터를 통해 simulate 한다고 보아야 함
- call-by-value 형태의 함수에서는 함수 외부에 정의된 변수에 접근 불가
- call-by-reference 형태의 함수에서는 외부에 정의된 변수에 접근 가능

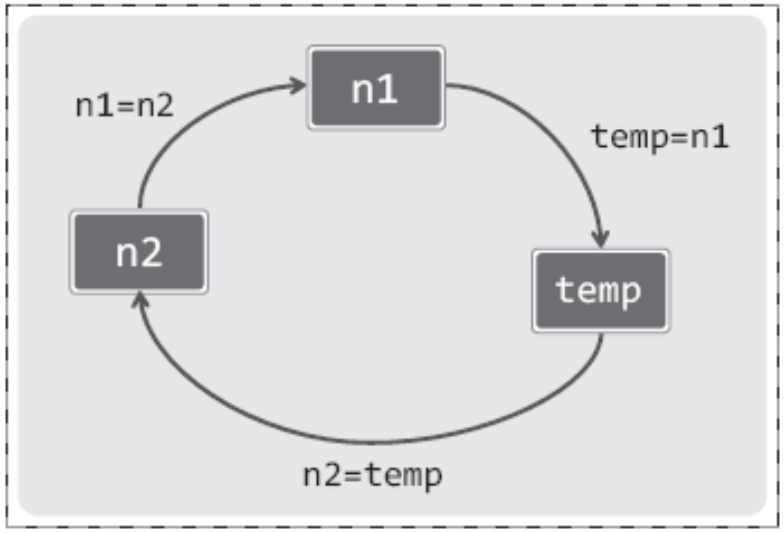
잘못 정의된 call-by-value

```
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 num2가 바뀜? */
  printf("num1 num2: %d %d\n", num1, num2);
  return 0; }
```

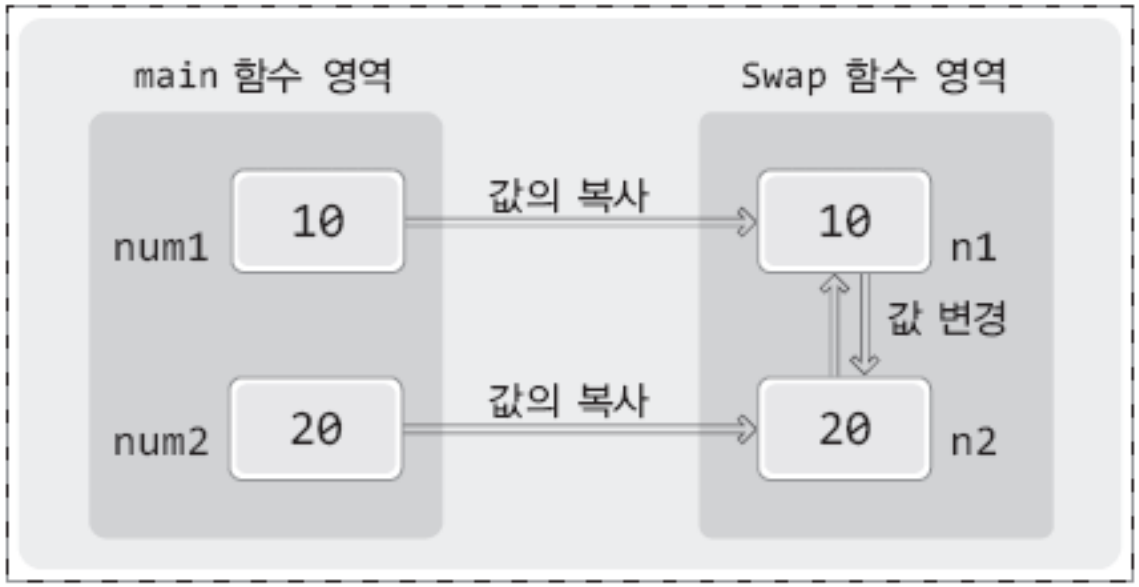
실행결과

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

잘못 정의된 call-by-value



Swap 함수 내에서의 값의 교환



Swap 함수 내에서의 값의 교환은 외부에 영향을 주지 않는다.

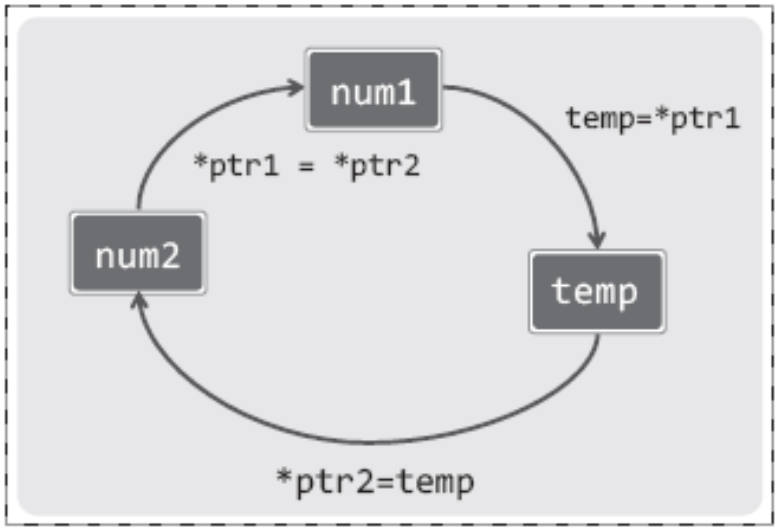
주소 값을 전달하는 형태의 함수 호출

```
void Swap(int *p1, int *p2) {
  int temp = *p1;
  *p1 = *p2;
  *p2 = temp
  printf(“*p1 *p2: %d %d\n", *p1, *p2);
}
int main(void) {
  int num1 = 10;
  int num2 = 20;
  printf("num1 num2: %d %d\n", num1, num2);
  Swap(&num1, &num2); /* num1과 num2가 바뀜? */
  printf("num1 num2: %d %d\n", num1, num2);
  return 0; }
```

실행결과

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

주소 값을 전달하는 형태의 함수 호출



Swap 함수 내에서
함수 외부에 있는
변수간 값의 교환

- Swap함수 내에서의 *ptr1 = main함수의 num1
- Swap함수 내에서의 *ptr2 = main함수의 num2

scanf 함수 호출 시 &연산자를 붙이는 이유

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

변수 num 앞에 & 연산자를 붙이는 이유는?

scanf 함수 내에서 외부에 선언된 변수 num에 접근 하기 위해서는 num의 주소 값을 알아야 한다. 그래서 scanf 함수는 변수의 주소 값을 요구함

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

변수 str 앞에 & 연산자를 붙이지 않는 이유는?

str은 배열의 이름이고 그 자체가 주소 값이기 때문에 & 연산자를 붙이지 않는다. str을 전달함은 scanf 함수 내부로 배열 str의 주소 값을 전달하는 것이다.

포인터 변수의 참조대상에 대한 const 선언

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

왼편의 const 선언이 갖는 의미

포인터 변수 ptr을 이용해서 ptr이 가리키는 변수에 저장된 값을 변경하는 것을 허용하지 않음

- 그러나 변수 num에 저장된 값 자체의 변경이 불가능한 것은 아님
- 다만 ptr을 통한 변경을 허용하지 않을 뿐임

포인터 변수의 상수화

```
int main(void) {
   int num1 = 20;
   int num2 = 30;
   int * const ptr = &num1;
   ptr = &num2; /*컴파일 에러*/
   *ptr = 40; /* 컴파일 성공 */
   printf("%d\n", num1);

}
```

왼편의 const 선언이 갖는 의미

포인터 변수 ptr에 저장된 값을 상수화

포인터 변수의 상수화

- 두 가지 const 선언을 동시에 할 수 있음

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

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

const 선언이 갖는 의미

- 코드 원본 (수정 전)

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

const 선언이 갖는 의미

- 코드 원본 (수정 후)
  - 안전성이 높아짐

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

const 선언은 추가적인 기능을 제공하기 위한 것이 아니라, 코드의 안전성을 높이기 위한 것이다.
따라서 이러한 const의 선언을 소홀히하기 쉬운데, const의 선언과 같이 코드의 안전성을 높이는 선언은 가치가 매우 높은 선언이다.

정리

- 함수의 인자 배열 전달
- 배열의 주소값 전달
- call by value
- const 선언