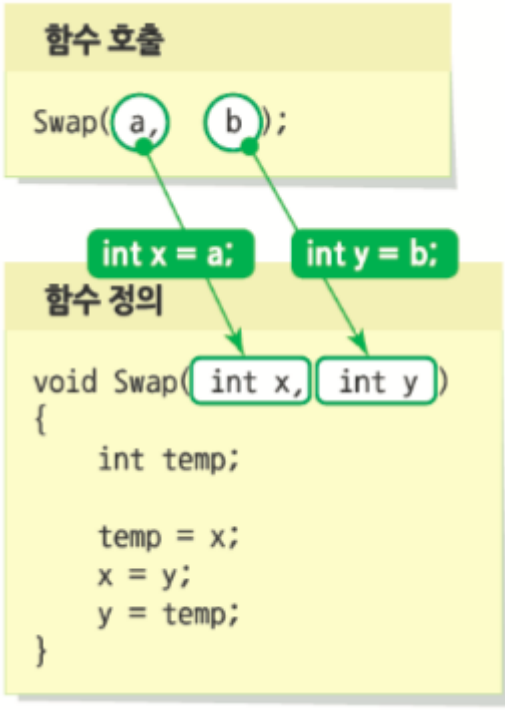
함수의 활용

1. 함수의 전달 방법

▼ 값에 의한 전달

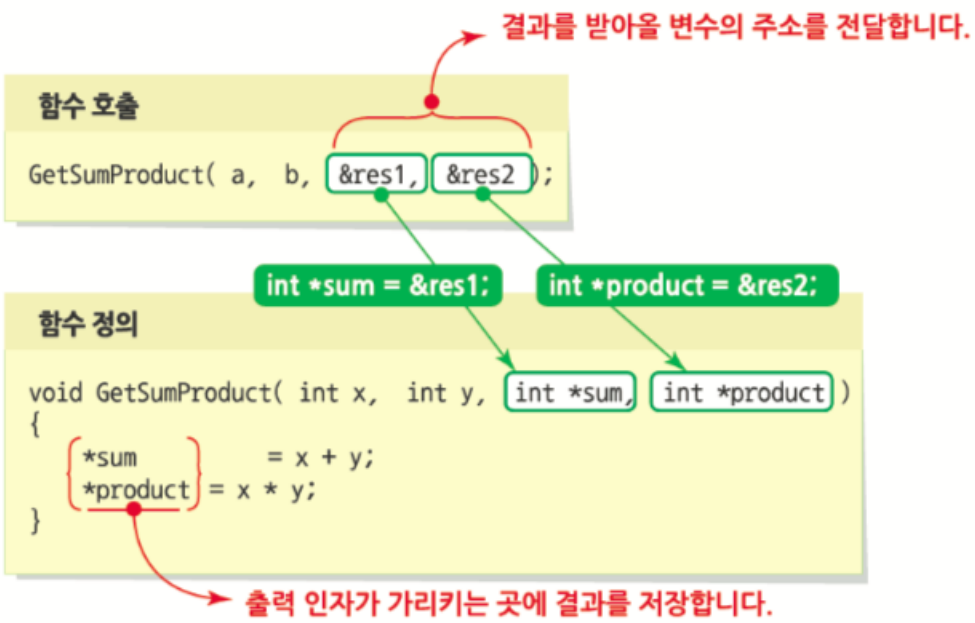
- 함수 호출시 인자의 값을 매개변수로 복사해서 전달하는 방식
- 매개변수는 함수 호출시 인자의 값으로 초기화 된다.
- 인자값을 변경하지 않을때, 값에 의한 전달을 사용 한다. (call by value) (ex1001)



- 값에 의한 전달을 하면 안되는 예가 있다. (ex1002) ⇒ 포인터 사용

▼ 포인터에 의한 전달

- 매개변수로 변수의 주소를 전달한다.
- ex1003.c 의 경우 함수 호출시 넘겨준 주소가 매개변수인 포인터 변수에 저장 된다.
- 주소 안에 있는 값들은 서브 함수의 처리에 의해 변경 된다.
- 포인터에 의한 전달 방법 정리(ex1004)



1. 함수를 선언할 때, 처리 결과를 받아올 매개변수를 포인터형으로 선언한다.

```
void GetSumProduct(int x, int y, int *sum, int *product);
```

2. 함수를 호출할 때, 처리 결과를 받아올 변수의 주소를 인자로 전달한다.

```
GetSumProduct(a, b, &sum_result, &product_result);
```

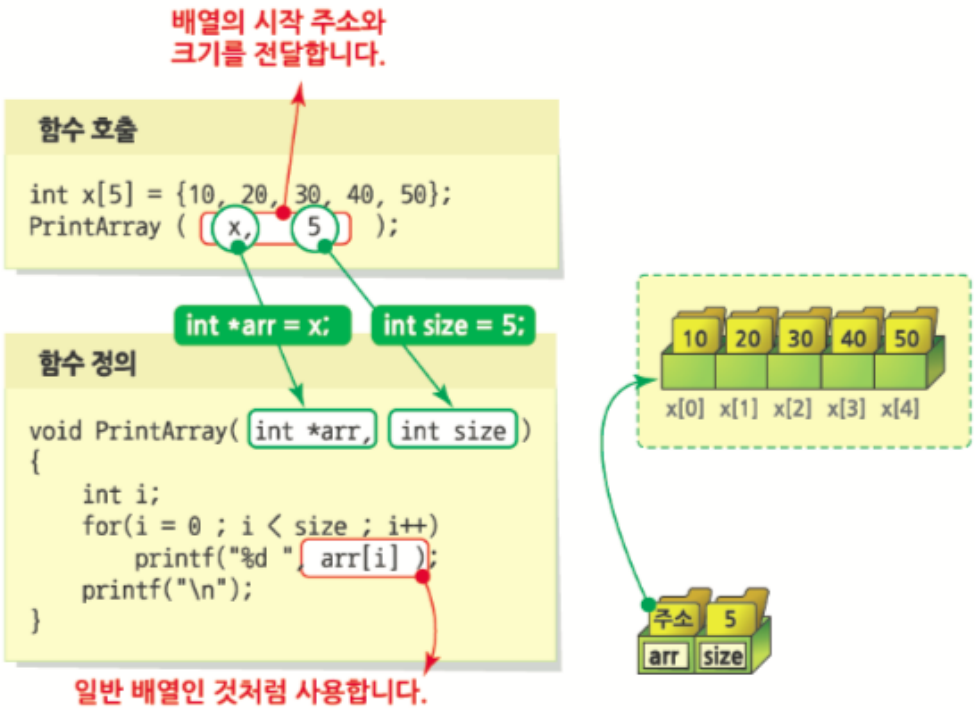
3. 함수를 정의할 때, 매개변수가 가리키는 곳에 함수의 처리 결과를 저장한다.

```
void GetSumProduct(int x, int y, int *sum, int *product){
  *sum = x + y;
  *product = x * y;
}
```

▼ 배열의 전달

- 함수 인자로 배열을 사용할 때는 항상 포인터에 의한 전달 방법을 사용한다.
- 인자로 배열을 넘겨줄 때는 [ ] 없이, 배열명만 넘겨주면 된다.

- 배열을 전달 받는 매개변수는 배열의 원소에 대한 포인터형으로 선언해야 한다.
- 서브 함수 안에서는 전달받은 포인터를 배열처럼 사용하면 된다.
- 배열을 매개변수로 전달 받았을때, 배열의 시작 주소가 넘어오는 것이어서, 서브 함수에서는 배열의 크기를 계산 할 수 없다. ⇒ 배열의 크기도 함께 함수에 전달해야 한다. (ex1005)



- 배열의 일부분을 전달하거나, 배열을 전부 전달후에 일부분만 출력되게 할 수 있다. (ex1006)
- 배열이 서브함수 안에서 변경되지 않을 때(입력인자)는 매개변수의 포인터형에 const를 지정해 변하지 않는 다는 것을 명확히 해준다. (ex1007)
- 배열을 전달하는 방법 정리
  1. 배열을 인자로 갖는 함수를 선언할 때는 매개변수의 데이터형으로 포인터형을 사용

     ```
     void SortArray(int *arr, int size);  //int *arr == int a[]
     ```

  2. 서브함수에서 배열이 변경되지 않을때, const 포인터로 전달 (크기 포함)

```
void PrintArray(const int *arr, int size);
```

3. 함수 호출시 배열의 시작 주소를 인자로 넘겨준다. (크기 포함)

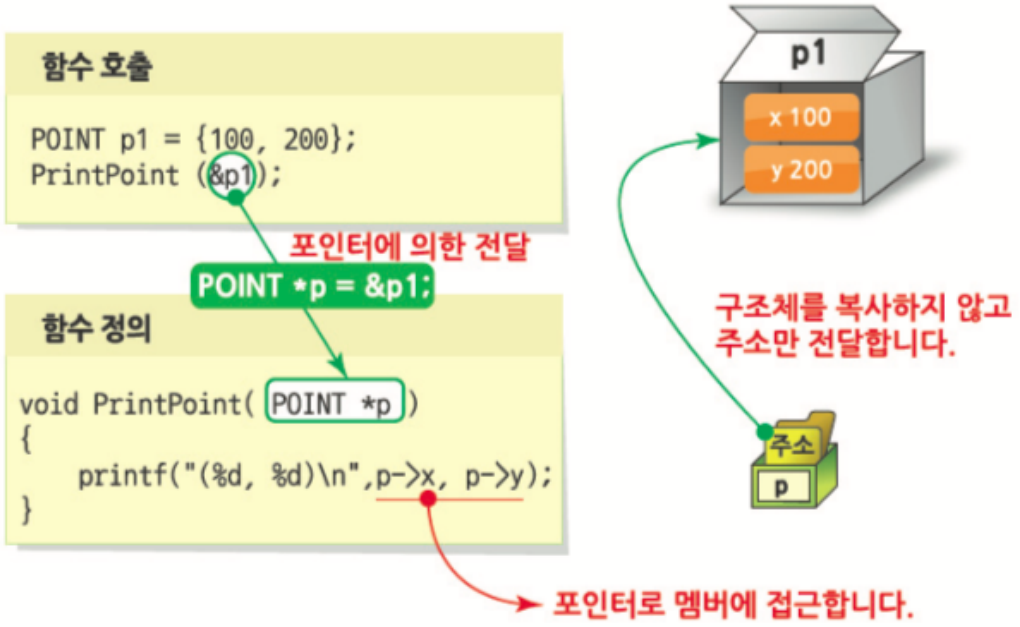
```
int x[5] = {1, 2, 3, 4, 5};
PrintArray(x, 5);
```

4. 함수 정의시 매개변수 포인터를 배열 이름인 것처럼 사용하면 된다.

```
void PrintArray(const int *arr, int size){
  int i;
  for(i=0; i <size; ++i){
    printf("%d ", arr[i]);
  }
  printf("\n");
}
```

▼ 구조체의 전달

- 함수 인자로 구조체를 쓸 경우에는 값에 의한 전달, 포인터에 의한 전달 두가지 모두 가능
- 구조체는 기본형 변수에 비해 크기가 크므로, 일반적으로 포인터로 전달한다. (ex1008.c)



- 값에 의한 전달은 인자와 매개변수 사이에 복사가 일어난다. (ex1008_value.c)
- 서브함수에서 구조체가 변경되지 않을때는 매개변수 포인터에 const를 지정하여 명시적 전달을 한다.
- 매개변수에 const를 선언했을때, 구조체의 멤버 값은 변경할 수 없다. 구조체 포인터에서 멤버에 접근하기 위해서는 → 를 사용한다. (ex1009.c)
- 서브함수에 구조체를 전달할 경우에 대한 정리
  1. 가능하면 포인터로 전달한다. → 매개변수의 데이터형은 구조체 포인터로 한다.

     ```
     void PrintPoint(POINT *p1, POINT *p2);
     //서브함수에서 구조체 값이 변경되지 않는다면 cont POINT *p2 와 같이 한다.
     ```

  2. 서브함수 호출시에는 구조체 변수의 주소를 인자로 넘겨준다.

     ```
     POINT pt1 = {100, 200};
     PrintPoint(&pt1);
     ```

  3. 서부함수 정의시에는 매개변수 포인터를 이용해 구조체 멤버에 접근한다.

```c
void PrintPoint(const POINT *p){
    printf("%d  %d \n", p->x, p->y);
}
```

▼ 함수정의 방법 정리

1. 함수의 입력인자가 기본형일 때는 값으로 전달하지만, 배열이나 구조체일 때는 포인터로 전달한다.

```c
int GetFactorial(int n);

typedef struct point{
    int x;
    int y;
}POINT;
void PrintPoint(const POINT *p); //(ex1009.c

void ReverseString(const char *str); //(ex1011.c)
```

2. 함수의 입력인자가 많을 때는 입력 인자를 모아서 구조체를 정의하여 전달한다.

```c
void SetFont(const char *facename, int height, int weight, int color);

typdef struct font_info{
  const char *facename;
  int height;
  int weight;
  int color;
}FONT_INFO;
void SetFont(const FONT_INFO *pfi);
```

3. 함수의 결괏값이 구조체나 배열인 경우에는 return 대신 포인터로 전달한다.

```c
void InitializeArray(int *arr, int size);
void ReverseString(const char *str, char *reverse, int size); //ex1011-2
void SetPoint(POINT *p, int x, int y);
```

4. 함수 결괏값이 둘 이상일때도 return 대신 포인터를 사용

```c
void GetSumProduct(int x, int y, int *sum, int *product); //ex1004.c
```

5. 함수의 입력부분과 출력 부분을 담당하는 배열 하나 또는 두 개를 만들어 사용할 수도 있다.

```
//출력 부분이 1개
void SortArray(int *arr, int size);  //ex1007.c
int data[5] = {10, 20, 34, 3, 9};
SortArray(data, 5);

//출력 부분이 2개
void SortArray(const int *arr, int *result, int size);
int data[5] = {10, 20, 34, 3, 9};
int result[5] = {0};
SortArray(data,result,5);
```

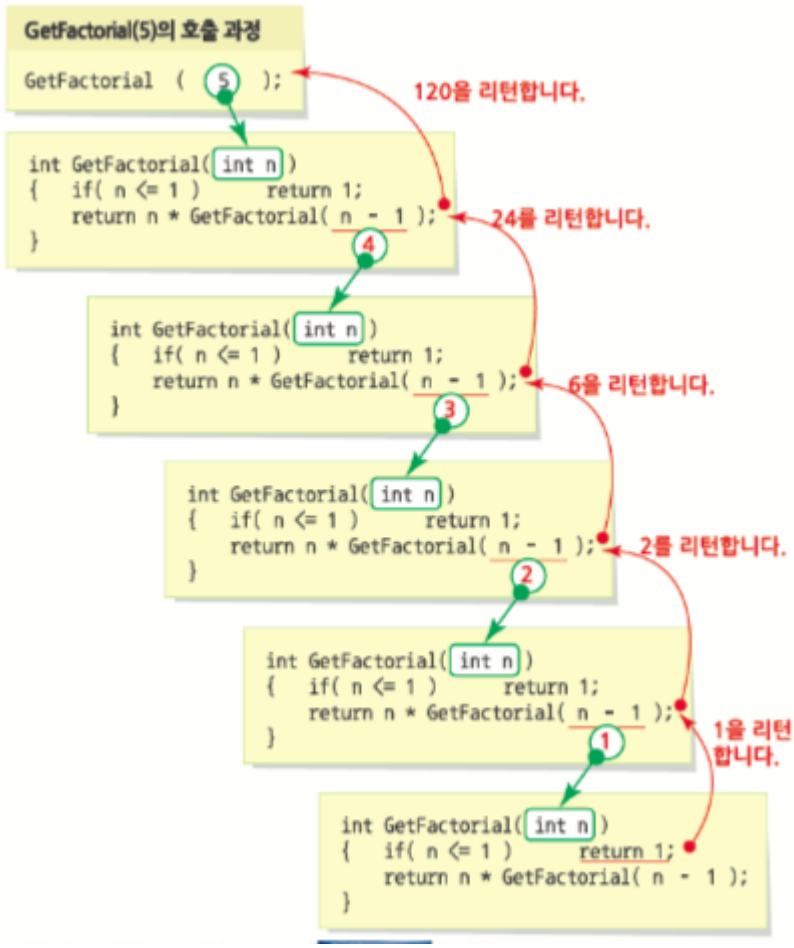
2. 재귀함수

▼ 자기 자신을 다시 호출하는 함수

▼ 재귀함수의 예

```
#include <stdio.h>
int addNumbers(int n);
int main() {
    int num;
    printf("Enter a positive integer: ");
    scanf("%d", &num);
    printf("Sum = %d", addNumbers(num));
    return 0;
}

int addNumbers(int n) {
    if (n != 0)
        return n + addNumbers(n - 1);
    else
        return n;
}
```



3. Storage class

▼ 정의와 형식

- 변수나 함수 선언할 때 사용하는 키워드로 저장 위치와 사용 범위를 결정
- 형식

```
auto int num = 0;
register int i;
static int count;
extern int global;
```

- 함수에는 static 과 extern만 사용할 수 있다.
- auto와 register 는 지역변수에만 쓴다.

▼ auto 변수

- auto : 모든 지역변수는 auto 변수로, 기본적으로 생략되어 있다.

```c
int main(){
  auto int x =10;
  int y = 20;  //x, y 둘다 auto 변수
}
```

▼ register 변수

- register : 변수를 CPU 레지스트리에 할당한다.

```c
register int i;
for (i=0; i<100000; ++i)
    sum += i;
```

- 좀더 빠른 계산을 원할때, register 를 사용 한다.
- register 변수의 주소는 구할 수 없기 때문에 &i 와 같이 사용하면 컴파일 에러가 발생한다.

▼ extern 변수

- 변수에 extern 을 선언하면, 해당 변수가 다른 곳에 선언 되어 있다고 컴파일러에 알려 준다.
- 즉, 변수에 대하여 별도의 메모리 할당 없이 해당 변수의 이름을 사용할 수 있게 해준다.
- 함수의 디폴트 Storage class 가 extern 이다.
- 예> 전역변수가 함수 중간에 선언되어 있을 경우, extern 문을 사용하면 함수의 첫 부분에서도 전역변수를 사용 할 수 있다. (이 경우 extern 문을 맨 위에 써주지만 메모리는 할당 되지 않고 사용 범위만 알려 주기 때문에 메모리 절약 효과가 있다. extern int global;)

▼ static 변수

- 지역변수에 static을 선언하면 정적 지역변수가 된다.
- 정적 지역 변수 전역변수처럼 static 변수는 프로그램 시작시 할당 되고, 프로그램 종료시 해제 된다. (0 으로 초기화도 된다.)
- 그러나, 전역변수와 달리 선언된 함수 안에서만 사용할 수 있다.
- 즉, static 변수는 함수가 리턴되도 사라지지 않아서 함수가 재 호출 될 경우에 예전 값을 기억하고 재 사용 할 수 있다. (ex1010.c)
- 문자열을 뒤집 는 프로그램에 사용(ex1011.c) (ex1011-2.c : 다른 풀이)

- 정적 전역 변수로 사용 : 소스파일이 여러개 일때, 전역변수에 대해 static 을 선언 하면, static이 지정된 전역 변수는 해당 소스파일에서만 사용이 가능하고 다른 소스 파일에서는 사용이 불가능 하다.
- 함수에 static 을 선언하면(static void func(void)) 해당 함수는 정의된 소스 파일에서만 호출할 수 있고, 다른 소스파일에서는 호출이 불가능해진다.

4. 실습