9 함수의 기초

9-1. 함수의 개념

9-2. 함수의 정의 (사용 방법)

9-2-1. 반환형 / 9-2-2. 함수의 이름 / 9-2-3. 매개 변수 / 9-2-4. 함수 몸체

9-3. 함수 정의 예제

9-3-1. 예제1: 제곱값 구하기

9-3-2. 예제2: 최대값 구하기

9-3-3. 예제3: 정사각형 그리기

9-3-4. 예제4: 속이 빈 정사각형 그리기

9-3-5. 예제5: 팰린드롬(Palindrome)

9-3-6. 예제6: 나눗셈의 몫과 나머지 구하기

9-4. 함수의 호출과 반환

9-4-1. 함수의 호출 / 9-4-2. 인수와 매개 변수 / 9-4-3. 반환 값 /

9-4-4. 호출 과정 예시 / 9-4-5. 함수 호출 시 주의 사항

9-5. 함수 선언의 다른 표현

@ 더 알아보기: 재귀 함수 – 예제를 중심으로

\*\*비유 알아두기\*\*

Int main (void) : 메인 작업장

main 함수 외 함수 : (그 외의 소)작업장

매개 변수 : 재료(작업에 필요한)

return 값 : 작업장 결과물

함수 반환형 : 작업장 결과물의 종류

Introduction.

독자들도 느꼈듯이 우리들은 C언어의 정말 기본적인 내용들을 배웠다. 변수, 연산자, 조건문, 반복문, 배열, 구조체, 포인터 등 C언어에서는 없어서는 안 될 많은 것들을 배우고 달려왔다. 이러한 시점에서 우리는 이 모든 내용을 통합 할 수 있는 것을 배울 차례가 왔다. C언어의 꽃이라고 할 수 있는 ‘함수(Function)’에 대해서 알아보도록 하자.

(예전에 배웠던 내용들이 생각 주머니 속에서 움직이는 듯한 그림을 집어 넣기)

9-1 함수의 개념

함수를 들어가기에 앞서 우리가 1단원 때 배웠던 내용을 다시 되돌아 보기로 하자. 1 단원에서 이런 말이 나왔었다. “, 맨 위의 #include <stdio.h>는 도구상자이다. stdio라는 도구 상자를 이용하여 앞으로 프로그램을 작성할 때 필요한 도구들을 이 맨 위의 도구상자에서 꺼내 사용한다.”, “int main (void)는 메인 작업실이다. 재료를 넣으면 일련의 과정을 통하여 가구가 만들어지는 장소이다.” 이러한 맥락에서 함수를 배워 보도록 하자.

(그림: stdio라는 상자에서 도구를 꺼내는 사람의 모습)/ (그림: 사람이 작업을 할 터전을 정하는 모습)

가구가 만들어 지기 위해서는 정말 많은 가공 과정들이 필요하다. 먼저는

나무를 잘라서 나무를 손질해야 할 것이다. 좋은 가구가 되기 위해서는 습도와 온도를 잘 조절해서 나무를 잘 가공 해야 된다.

두 번째, 나무를 설계 도면에 맞게 톱으로 켜야 한다. 켠 나무 판자에 무늬나 조각을 새길 수도 있다.

세 번째, 잘라진 나무 판자를 가지고 예쁘게 조립한다. 못과 망치로 알맞은 위치에 놓고 못을 박아가며 가구 모양을 잡아간다.

마지막으로 가구의 마무리 작업을 한다. 왁스 칠을 하고 모난 부분들을 점검하며 팔 준비를 한다.

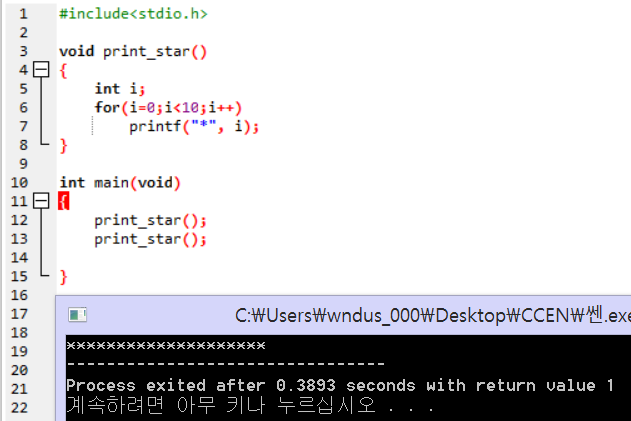
(1, 2, 3, 4 단계 모두 그림으로 표시해서 독자들의 이해를 돕는다.)

함수의 개념도 이와 일맥 상통한다. 여러분이 아까 보았던 1번부터 4번과정 각각 단계를 각 함수의 기능이라고 할 수 있다. 다시 말해 1단계는 나무를 자르는 함수 또는 나무를 가공하는 함수 2단계는 톱으로 켜는 함수 등 가구를 만드는 데에 필요한 일련의 과정을 설명 한 것이다.

정확히 표현을 하자면 메인 함수(메인 작업장)에서 가구의 결과물이 나오게 되지만 다른 작은 작업장에서 나무를 가공하거나 조립을 함으로서 메인 작업장에서 마무리가 되게 된다.

(각각의 작업장에서 메인 작업장으로 이동되는 가구를 표시 한다.)

자 이제 기본 개념은 확실하게 짚고 넘어갔으니 정말 C언어서는 어떻게 쓰이는 지 알아보도록 하자.



이렇게 코드를 쓰게 되면 결과는 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 으로 출력 된다. Main 작업장 안에서 모든 일을 할 수도 있지만 지금처럼 코드가 짧지 않고, 1000줄, 10000줄 이렇게 넘어가게 되면 가독성(코드를 읽는 능력)도 떨어지고, 체계적으로 보이지도 않는다. 또 오류가 났을 때 함수를 따로따로 구분 짓지 않으면 어느 단계에서 잘못 되었는지 모르기 때문에 함수로 처리하는 것이 효율 적이다.

함수의 종류를 살펴보면 ‘라이브러리 함수’와 ‘사용자 정의 함수’ 로 나뉘게 된다. 라이브러리 함수는 우리가 익히 알던 printf, scanf 같은 함수들이다. C언어에서 기본적인 출력함수와 입력 함수(작업장)이다. 1단원 에서도 언급 했듯이 꼭 include <stdio.h>를 써주어야 이러한 기본 함수들을 쓸 수 있다. 그 외에 사용자 정의 함수란 프로그래머 자신이 필요한 함수를 만드는 것이다. 위의 예로 보았을 땐 \*을 출력하는 함수였고, 이름은 print\_star라는 함수였다. 이제 개괄적인 함수의 개념에 대해서 살펴 보았으니 본론으로 들어가보도록 하자.

9-2 함수의 정의(사용 방법)

함수는 함수 헤어(function header) 부분과 함수 몸체(function body)로 크게 나눌 수 있다. 반환형과 함수이름, 매개변수 목록을 합쳐서 함수 헤더라고 한다. 함수 몸체는 대괄호( {} )로 둘러싸인 부분이다. 함수의 몸체 안에서 함수가 수행되는 작업에 필요한 문장들이 들어간다.

반환형 함수이름(매개변수1, 매개변수2)

{함수의 시작

함수 몸체

return ?;

}함수의 종료

int add (int x, int y)

{

int result;

result=x+y;

return result;

}

갑자기 생소한 단어들이 팍팍 튀어져 나오니 갑자기 머리가 복잡하고 질끈질끈 할 것이다. 그러나 책을 정독 하면서 따라오기만 하면 함수를 기본부터 정확하게 이해할 수 있을 것이다. 만약에 함수를 공부하다가 변수가 기억이 나지 않거나 앞부분의 내용이 기억이 나지 않는다면 반드시 모르는 단원을 복습하고 다시 되돌아오기를 바란다.

9-2-1 반환형

함수의 반환형은 함수의 성격을 나타낸다. 작업장의 비유로 다시 돌아가 보자면 작업장에서 작업을 하고 나온 결과물이 나무 판자인지, 조각된 나무 판자인지, 마무리 작업이 덜 된 가구인지, 완벽한 가구인지 그 함수(단계)에 따라서 그 결과물이 다를 것이다. 함수의 반환형도 마찬 가지이다. int, char, long, double, void등 차례대로 정수, 문자, 실수, 실수, 반환 값 없음 과 같이 결과물을 다른 작업장에 전달 해준다.

Q) 여기서 잠깐!! 독자들 중에는 분명히 이런 궁금증을 가진 사람이 있을 것이다. “메인 작업장의 개념과 작은 작업장의 개념은 이해가 되었어요. 하지만 아직 이해가 덜된 부분은 작은 작업장에서의 결과물은 항상 메인 작업 장으로 가야 하는 것인가요?” 이렇게 물을 수도 있다.

A) 정답은 정말 간단하다. 나무를 자르고 손질하는 과정을 세분화 시켜 보면

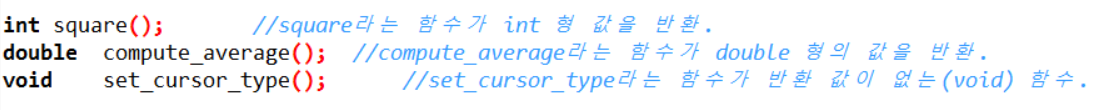
1. 숲에 가서 나무를 톱으로 켜는 과정

2. 자른 나무를 작은 작업 장으로 옮기는 과정

3. 나무를 소금물에 절여서 말리는 과정

등으로 세분화 될 수 있다. 따라서 정답은 꼭 메인 작업장으로 가지 않아도 된다는 점이다. 작은 작업장에서 작업한 결과물을 다른 작은 작업장으로 옮길 수도 있다. 아주 좋은 질문!

자 다시 본론으로 돌아와서 코딩 예를 들어보면

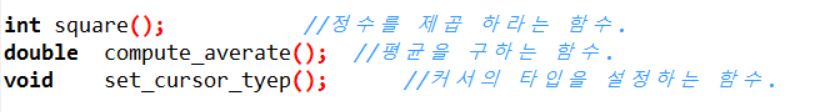


예전의 컴파일러에서는 반환형을 명시하지 않으면 int 형이라고 가정하곤 했었다. 그러나 이는 좋은 습관이 아니므로 반환형을 명시하는 것이 현명하다.

반환은 함수 안에서 return 문장을 사용하는 것을 통해 이루어진다. return 다음에 수식을 써주면 수식의 값이 반환된다. 예를 들어 return 0; 하면 0값이 반환된다. return result;라고 하면 변수 result에 저장된 값이 반환된다. 그렇게 된다면 int add(int x, int y) 라는 값은 result라는 어떤 정수의 값을 갖게 된다. 한 마디로 작업장에서 결과물이 나온 것이다.

9-2-2 함수의 이름

함수의 이름은 식별자에 대한 규칙만 따른다면 어떤 이름이라도 가능하다. 변수 이름을 설정할 때와 규칙이 같다. 다만 소스코드를 읽기 쉽게 하기 위하여 함수의 기능을 암시하는 이름을 부여하는 것이 좋다. 일반적으로 함수의 목적을 설명하기 위해서 동사, 동사+명사로 사용하면 좋다. 다음은 함수의 예이다. 언더바(\_) 문자도 사용 가능하다.

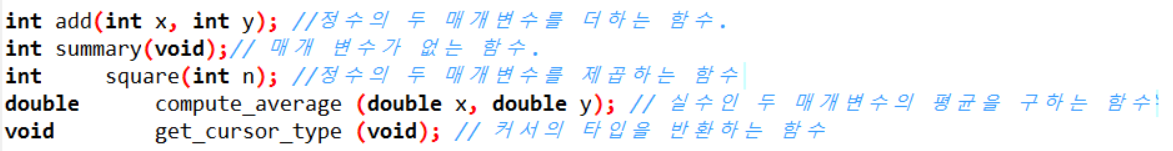


9-2-3 매개변수

매개 변수를 간단히 설명하자면 어떤 작은 작업장에서 작업을 해야 되는데 그 때 필요한 재료를 나타낸다. 이 재료는 다른 작업장으로부터 받아들여진 재료이다.

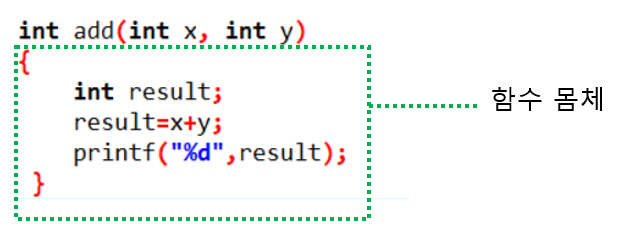
외부에서 함수를 호출 할 때는 작업에 필요한 데이터 값들을 보낼 수 있다. 매개 변수는 이러한 데이터를 받는 변수이다. 매개변수는 여러 개가 될 수 있으며 각 매개 변수는 쉼표로 분리된다.

각 매개 변수에 대하여 자료형과 이름을 지정한다. 예를 들어 add() 함수의 경우, 첫 번째 매개 변수는 int형으로서 이름은 x이다. 두 번째 매개 변수도 int 형으로서 이름은 y 가 된다. 만약 매개 변수가 없는 경우에 int summary(void) 와 같이 매개 변수 위치에 void를 써주거나 그냥 비워 놓으면 된다. 매개 변수에 관한 내용은 뒷 부분에 자세하게 나오니 걱정할 필요 없다.



9-2-4 함수 몸체

함수 몸체는 중괄호로 둘러싸여 있다. 함수 몸체에는 함수가 수행되는 작업에 필요한 문장이 쓰이는 곳이다. 여기가 바로 실질적인 작업이 이루어지는 곳이다. 이 문장들은 함수가 호출 되는 처음부터 순차적으로 하나씩 실행되며 return 문장을 만나게 되면 문장의 실행한 곳으로 돌아가게 된다.



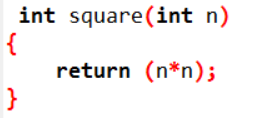
9-3. 함수 정의 예제

지금까지 우리들은 함수의 개념과 문법에 대해서만 알아보았다. 반환형, 함수의 이름, 함수의 몸체, 매개변수 이런 것들이 어떻게 함께 어우러져 사용되는지 궁금 할 것이다. 바로 예시 코드로 들어가서 함수가 어떻게 짜여지고 쓰여지는지 보도록 하자.

9-3-1. 예제 1: 제곱값 구하기

먼저 정수의 제곱 값을 구하는 함수를 만들어 보자. 먼저 함수 이름(작업장의 이름)을 결정하여야 한다. 함수가 하는 기능이 정수를 제곱 하는 것이므로 함수 이름을 square(작업장의 이름) 라고 하자. 다음에는 함수의 반환값(작업장의 결과물)과 매개변수(작업에 필요한 재료)를 결정하여야 한다. 정수의 제곱을 하여야 하므로 외부에서 정수를 받아야 한다. 따라서 매개변수는 한 개의 정수가 된다. 매개변수 이름도 지어야 하는데 그냥 n(재료의 이름)이라고 하자. 제곱한 값도 역시 정수이므로 함수의 반환값은 int 형이 된다. 지금까지 결정한 것을 적어보자

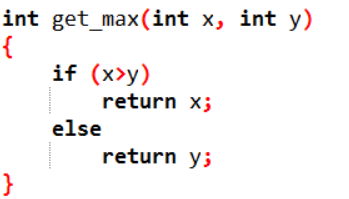
(이름에 따라 작업장에서 어떻게 움직이는 지 그림으로 표시해보기)



이제 함수를 정의 할 수 있다. 반환형(결과물의 종류)을 먼저 쓰고 함수 이름(작업장의 이름), 매개변수(재료) 순으로 쓰면 된다. 그 다음은 함수 몸체인데 먼저 중괄호를 적고 그 안에 문장들을 넣으면 된다. 어떤 문장을 넣어야 할까? 매개변수 n(재료의 이름) 을 통하여 받은 값을 제곱하여 반환하면 된다. C에서는 제곱 연산자는 없고 n\*n으로 표현하면 된다. 이제 제곱한 값이 계산되었으므로 값을 반환할 차례이다. 값을 반환하는 방법은 return(결과물) 다음에 반환하려는 값을 적어주면 된다.

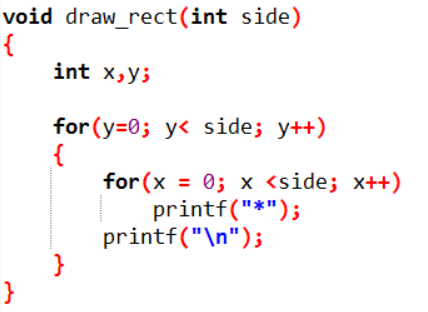
9-3-2. 예제 2: 최대값 구하기

이번에는 조금 더 복잡한 작업을 하는 함수를 작성하여 보자. 두 개의 정수가 주어지면 두 수 중에서 더 큰 수를 찾아서 이것을 반환하는 함수를 만들어 보자. 함수의 이름은 get\_max(작업장의 이름)이라고 하자. 반환되는 값은 역시 정수형(결과물의 종류)이다. 이번에는 매개 변수가 두 개가 된다.(재료가 2개임) 각 각 x, y라고 하자. 두 수 중에서 더 큰 수를 찾으려면 if 문을 사용한다. 두 수를 비교해서 더 큰 값으로 반환하면 된다. return 문(결과물)이 두 번 사용 된 것에 유의해야 한다. 수식 “x>y”가 함이면 x가 최대값이 되고, 그렇지 않으면 당연히 y가 최대값이 된다. return 다음에는 괄호가 있어도 된다.



9-3-3. 예제 3: 정사각형 그리기

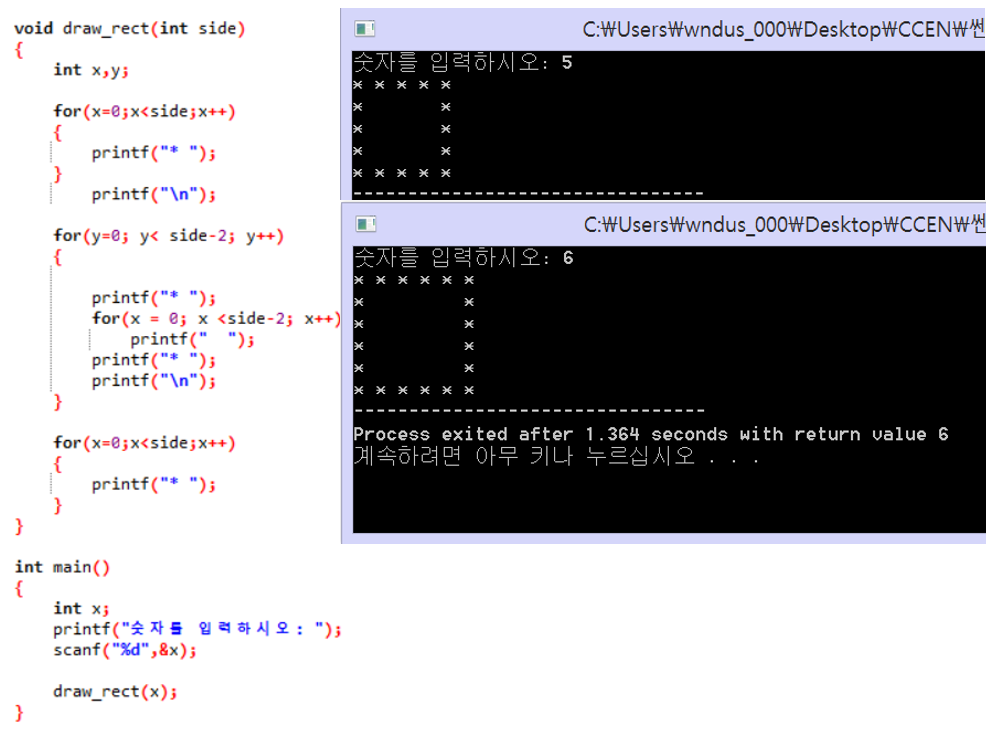
별표(\*) 기호로 채워진 정사각형을 그리는 함수를 작성하여 보자. 정사각형의 한 변의 길이를 고정하지 않고 매개 변수(재료)를 통하여 외부에서 변경 할 수 있도록 해보자. 먼저 함수의 이름은 동사+명사 규칙을 적용하여 draw\_rect()(작업장의 이름)라고 하자. 정사각형을 출력하고 끝이 나는 함수를 만들 것이므로 반환할 값을 지정할 필요가 없다. 따라서 반환형은 void(결과물의 종류)가 된다. 매개 변수의 이름을 side(재료의 이름)라고 하고 정사각형의 한 변의 길이를 전달 받는다. 별표 반복 구조를 이용하여서 그리면 될 것이다. 반복해야 하는 횟수가 고정되어 있으므로 for 루프를 사용하도록 하자



더 알아보기)) draw\_rect() 와 같은 함수는 아무것도 반환하지 않는다. 하지만 호출되면 화면에 별표를 그리는 작업을 한다. 이것을 함수의 부수 작용이라고 하는데 이러한 종류의 함수도 상당히 많이 사용된다.

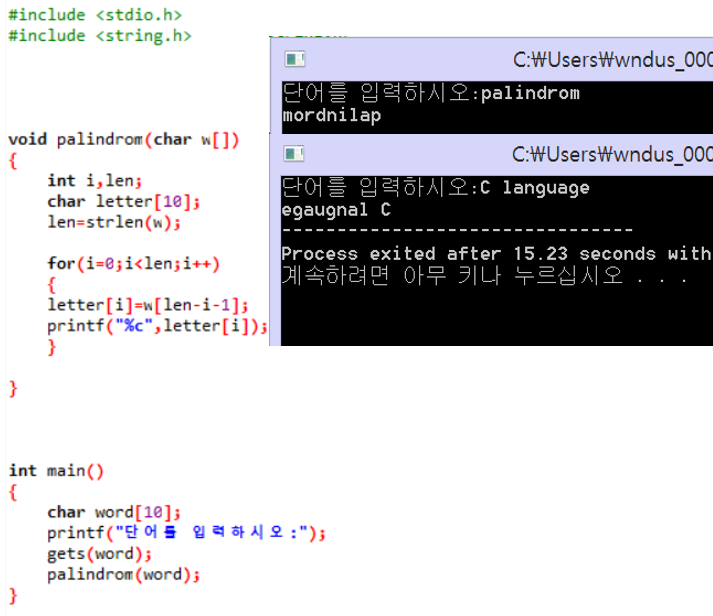
9-3-4. 예제4: 속이 빈 정사각형 그리기

별표(\*) 기호로 가운데가 비워진 정사각형을 그리는 함수를 작성하여 보자. 채워진 함수에서 조금 변형하면 쉽게 만들 수 있다. 첫번째 행, 중간 행, 마지막 행 세 부분으로 나누어서 반복문을 작성하면 된다.



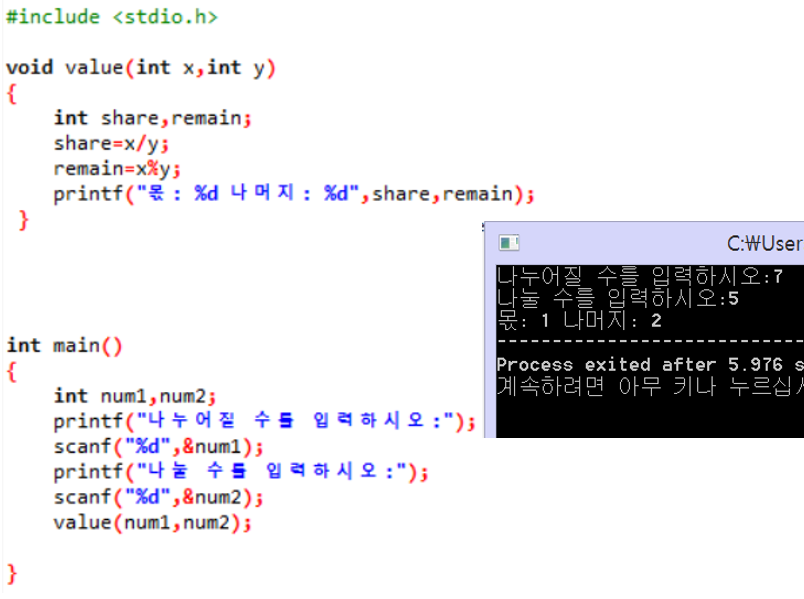
9-3-5. 예제5: 팰린드롬(Palindrome)

10자 이하의 문자열을 입력 받아 palindrom함수(문자열을 넘겨주면 거꾸로 만들어 반환하는 함수)를 작성해보도록 하자. 문자열의 길이를 구해 변수 len에 저장한다. 문자열의 길이를 구하기위한 strlen함수는 <string.h>헤더에 저장되어있다. 출력될 배열의 첫자리에 입력된 배열의 마지막자리를 대입하고, 두번째자리에 배열의 마지막에서 두번째 자리를 대입하도록 함수를 작성한다.



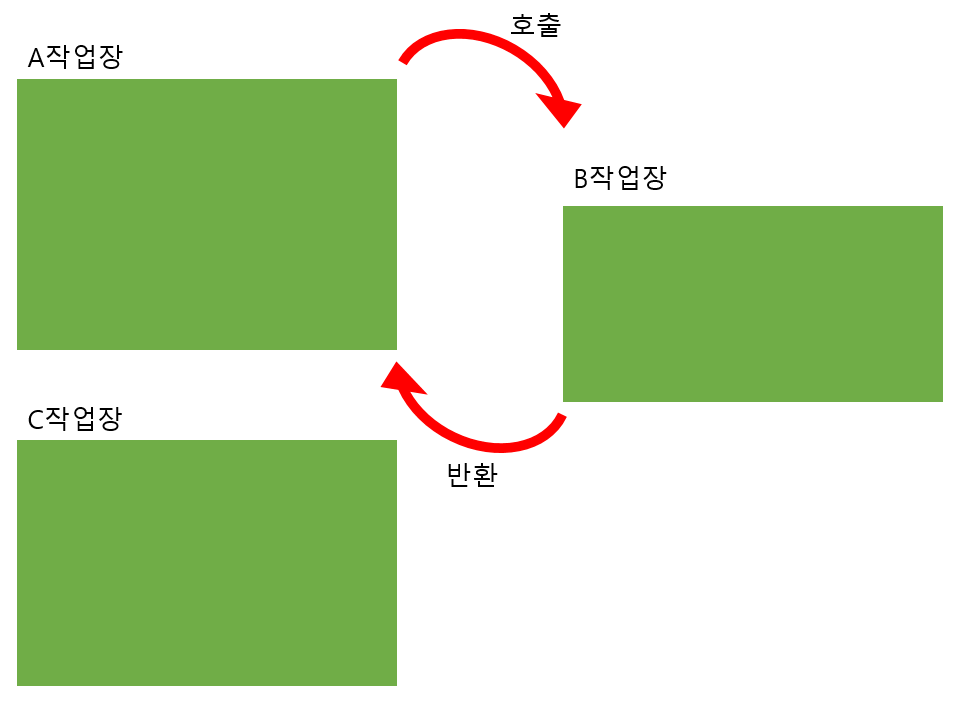
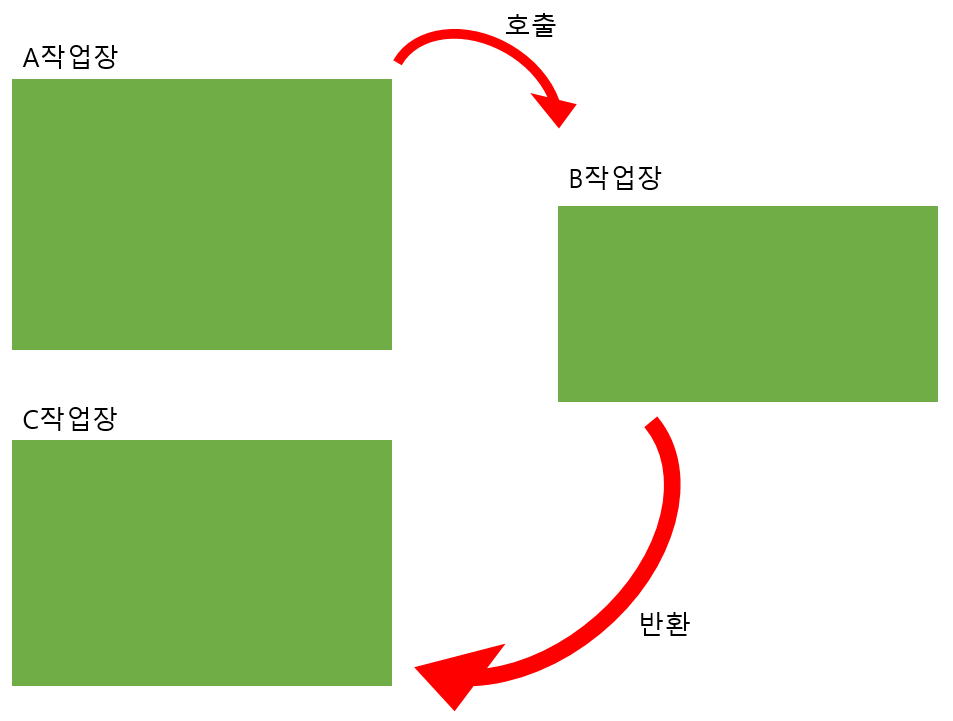
9-3-6. 예제6: 나눗셈의 몫과 나머지 구하기

몫과 나머지를 구하는 함수를 작성해보자. 나누어질 숫자와, 나눌 숫자를 메인 함수에서 입력 받는다. 매개변수에 두 수를 인자로 받으면, 함수에서는 두 수를 나눈 몫과 나머지를 계산한다. 결과를 출력하기만 하면 되므로 반환값은 없다. 따라서 반환형은 void형을 사용한다.



9-4 함수 호출과 반환

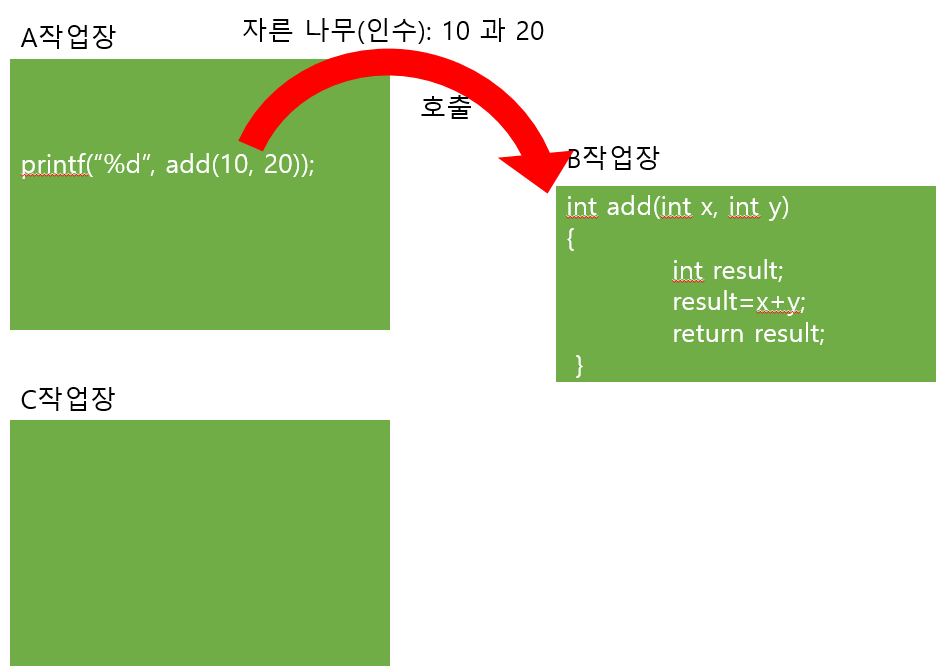
지금부터는 함수의 ‘호출’ 과 ‘반환’에 대해서 알아볼 것이다. 비유를 통해서 먼저 설명 해보자면 A라는 작업장은 나무를 자르는 작업장이고, B라는 작업 장은 자른 나무를 가공하는 작업장이라고 가정을 해보자. 먼저 A라는 작업장에서 나무를 다 자른 다음에 B라는 작업장으로 다 자른 나무를 줄 것이다. B작업장에서 A작업장에서 자른 나무를 가져오는 것을 함수의 ‘호출’ 이라고 한다. 다 자른 나무를 받은 작업장 B에서는 나무를 가공 하고 난 다음에 다시 작업장 A로 가공 된 나무를 보낼 것이다.( 또는 가구를 조립하는 작업장 C이나 다시 작업장 B에도 보낼 수 있음) . 이것을 ‘반환’이라고 한다.

9-4-1. 함수의 호출

컴퓨터의 관점에도 봐보자면 위에서부터 아래로 당신이 코딩한 코드들을 읽어 나가면서 한 줄 한 줄 처리를 할 것이다. 그런 의미에서 함수를 호출 하기 위해서는 컴퓨터가 그 함수를 먼저 알아야 호출 할 수 있다. 더 개념적으로 접근을 해보도록 하자 함수의 호출(function call)이란 add(10, 20)과 같이 함수의 이름(add)를 써 주고 ()안에 함수가 필요로 하는 데이터를 나열하면 된다. 함수 안에 들어있는 문장들은 함수가 호출되기 전까지는 전혀 실행되지 않는다. 함수가 호출되면 함수 안에 있는 문장들이 순차적으로 실행되며 문장들의 실행이 끝나면 호출한 위치로 되돌아간다. 이때 작업의 결과값을 호출한 곳으로 전달 할 수 있다. 함수의 호출 형식은 다음과 같다.

함수( 데이터1, 데이터2…)

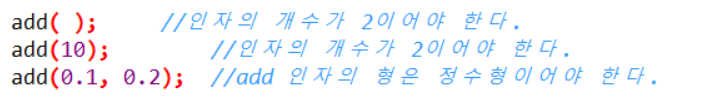
 (몇몇의 작업장이 나와서 A라는 작업장이 실행되고 있을 때는 B라는 작업장이 쉬고 있고, B라는 작업장이 실행되고 있을 때는 A라는 작업장이 쉬고있음을 나타내는데 순서도를 이용해서 표시하기)

9-4-2. 인수와 매개 변수 ->> 재료가 충분하지 않으면 가공 불가

먼저 인수의 개념에 대해서 짚고 넘어가도록 하자. 아까 예시에서 A작업장과 B작업장 이야기가 나왔다. A작업장에서 B작업장으로 넘기는 재료의 이름을 인자 라고 하고, B작업장에서 A작업장으로부터 받은 재료를 매개변수라고 한다. 인자와 매개변수의 값은 똑같지만 작업장마다(함수마다) 부르는 호칭은 다를 것이다. 따라서 인수와 매개변수는 함수 호출시에 데이터를 주고 받기 위하여 필요하다.

Sum = add (2,3); // 2 와 3 을 인수로 갖고 있다.

가구를 만드는 과정 중에서 만약에 B작업장으로 가공 할 나무가 전달되지 않았다고 가정을 해보자. 그렇게 되면 B작업장에서는 제대로 작업을 할 수 없게 된다.(함수를 실행 할 수 없게 된다.) 즉, 여기서 주의할 점은 매개 변수의 개수와 인자의 개수는 정확히 일치하여야 한다는 점이다. 즉, 매개 변수가 두 개이면 인자도 두 개를 전달하여야 한다. 매개 변수의 개수와 인자의 개수가 일치하지 않으면 아주 찾기 어려운 오류가 발생하게 된다. 또한 인자의 매개변수의 타입은 서로 일치하여야 한다. 예를 들어서 매개변수는 int 형으로 정의 되었는데 float 형의 데이터를 전달하면 오류가 발생한다. 다은은 잘못된 함수 호출이다.



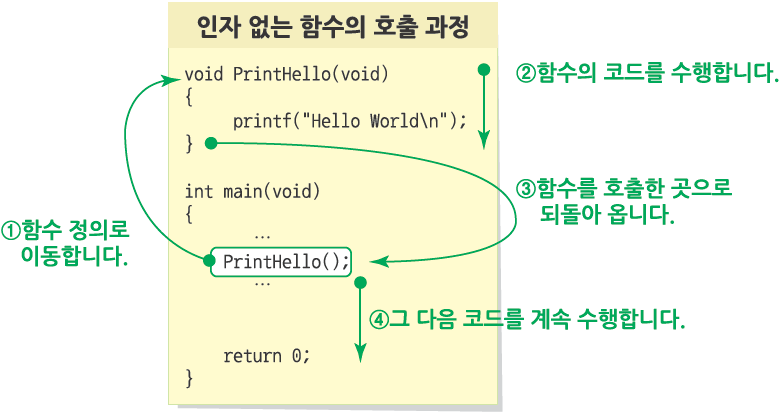
9-4-3. 반환 값

익히 들었듯이 반환 값이라 함은 결과물이라고 비유로 표현 했었다. 그렇다고 무턱대고 모든 값(결과물)을 반환 할 수 없다. 반환 값의 형(결과물)은 함수를 정의 할 때 썼던 함수의 형(결과물의 종류를 나타냄)과 일치 하여야 한다. 그리고 만약 함수의 형(결과물의 종류가) void 였다면 return은 어떻게 하여야 할까? 위의 예제 3(9-3-3. 예제 3)처럼 그냥 쓰지 않아도 된다. 또는 return; 와 같이 입력하여 아무것도 반환하지 않는 다고 나타낼 수도 있다.

9-4-4. 호출 과정 예시

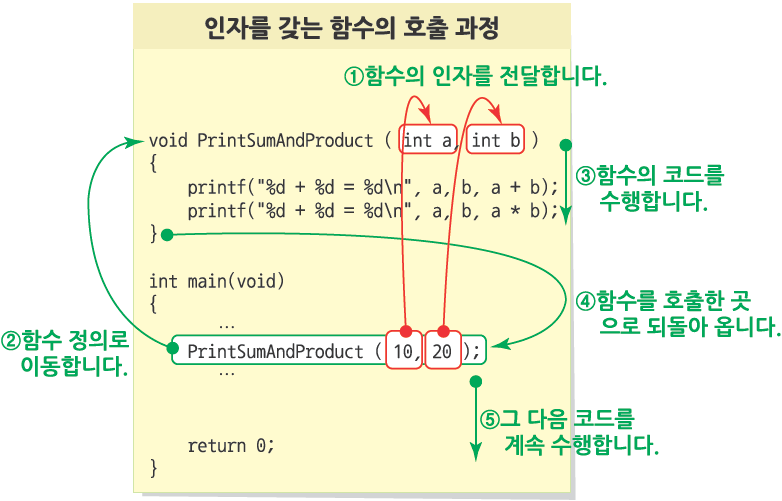
ㄱ. 인자를 포함한 함수의 호출 과정을 알아보도록 하자. 🡪언제 쓰이는지

간단한 예시코드와 그에 상응하는 순서도가 존재하고 있다. 우리가 항상 기억해야 할 점은 코드는 항상 위에서부터 아래로 읽고, main 함수(main 작업장) 부터 읽어낸다는 점이다. 처음에 메인(main) 함수(작업장)에서 읽다 보면 PrintHello()라는 함수가 등장하게 된다. 당연히 PrintHello()라는 함수가 메인함수 위에 정의가 되었기 때문에 PrintHello()함수로 이동하게 된다. PrintHello() 라는 함수는 반환형(작업장에서 만들어내는 결과물의 종류)이 void(반환하는 값 없음)이다. return이 따로 없기 때문에 printf(“Hello World\n”);을 불러오게 되면서 Hello World 라는 값을 보여주게 된다.



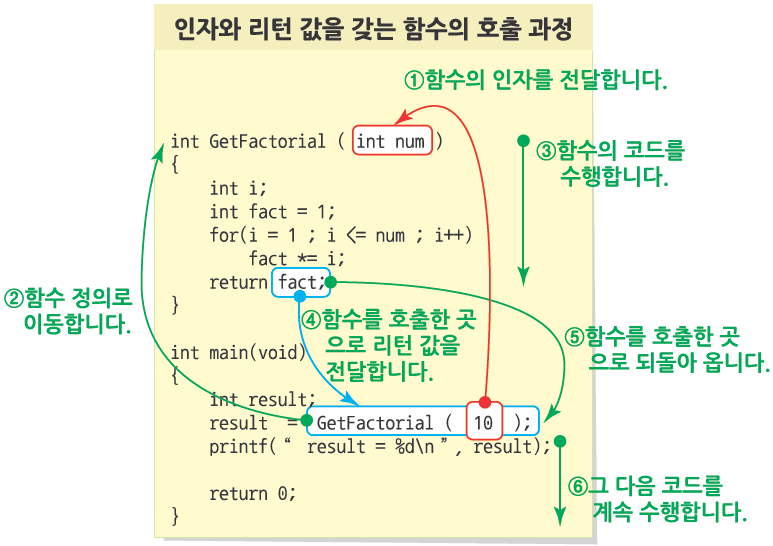
ㄴ. 인자를 갖는 함수의 호출 과정

인자(재료)를 갖는 함수의 호출 과정은 인자가 없는 함수의 호출 과정과 정말 유사하다. 단지 달라진 점은 인자의 유무이다. 메인 함수에서의 PrintSumAndProduct(10,20);의 10 과 20은 void PrintSumAndProduct( int a, int b)의 int a 와 int b 로 차례대로 값이 들어가게 된다. int a = 10 으로 정의 되고, int b = 20 으로 정의된다. 이제 PrintSumAndProduct 함수에서는 int a 와 int b를 마음대로 사용 할 수 있다. 위의 예제와 비슷하게 PrintSumAndProduct 함수는 반환형(작업장에서 만들어내는 결과물의 종류)이 void(반환하는 값 없음.)이다. Return이 따로 없기 때문에 printf(“%d + %d = %d\n, a, b, a+b”); printf(“%d \* %d = %d\n, a, b, a\*b”);을 불러오게 되면서 10 + 20 = 30, 10 \* 20 = 200 이라는 값을 보여주게 된다.



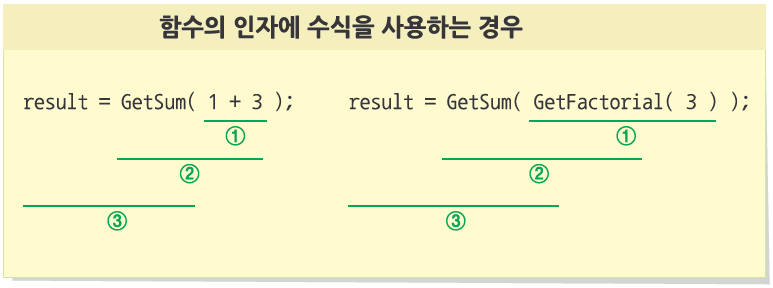
ㄷ. 인자와 리턴 값을 갖는 함수의 호출 과정

마지막으로 인자(작업장에서 쓸 재료)와 리턴 값(작업장의 결과물)을 다 갖는 경우가 나올 수 있다. 이 경우에는 위의 두 경우와 다르게 반환형(작업장의 결과물의 종류)이 void 형이 아니다. 더 정확하게 말하자면 void 형 일 수가 없다. 그 이유는 return값(작업장의 결과물)을 갖는 다는 것 자체가 int 형이든 char 형이든 어떻게든 변수의 형태가 존재하기 때문이다. 그래서 결과물 자체는 비교적 간단하게 나올 수 밖에 없다. 만약에 반환형과 리턴형의 자료형이 다르게되면 올바른 결과가 출력 되지 않는다. (아래는 오류예시)

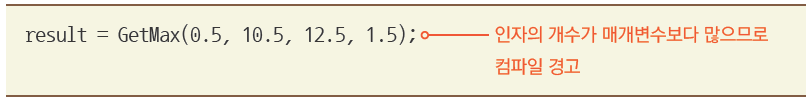
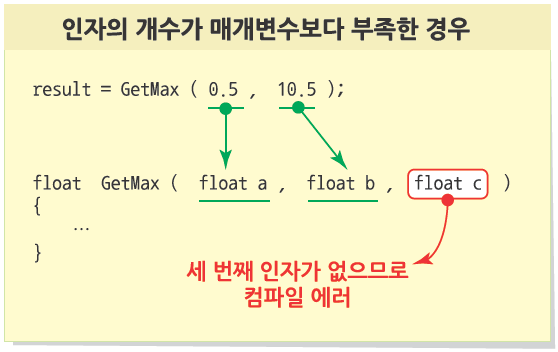
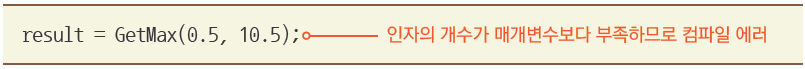


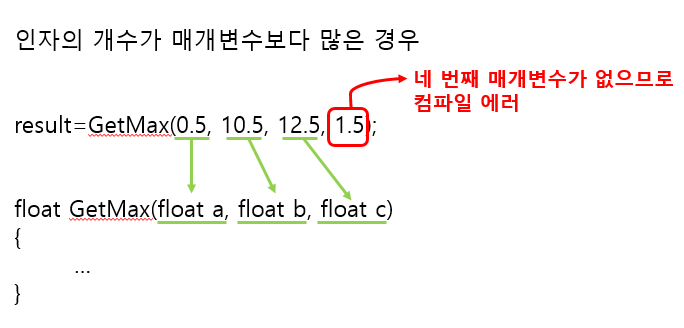
9-4-5. 함수 호출 시 주의 사항

1). 괄호 안에 있는 함수부터 풀어서 함수를 호출 한다.

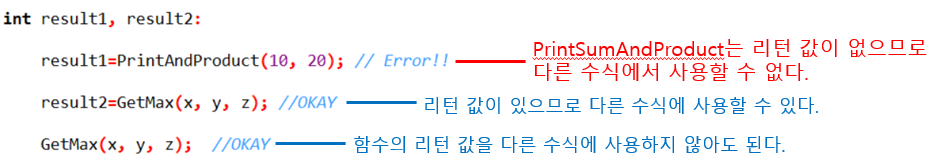


2). 인자의 개수와 매개변수의 개수가 일치해야 한다.



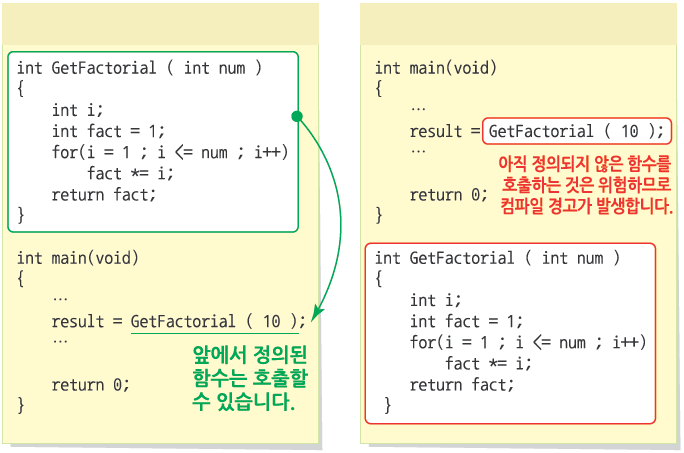


3). 인자는 반드시 필요하지만 리턴 값은 꼭 필요한 것은 아니다.

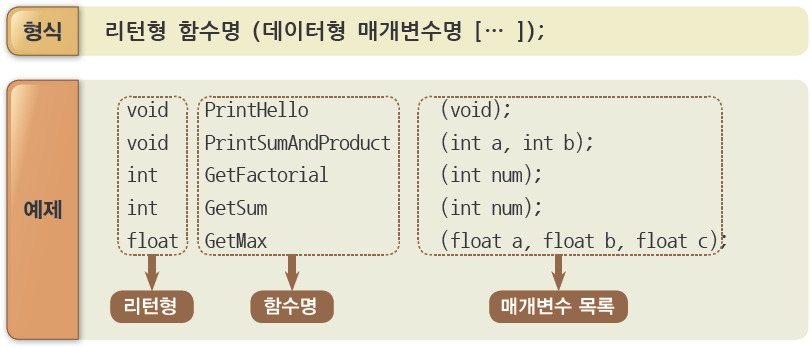


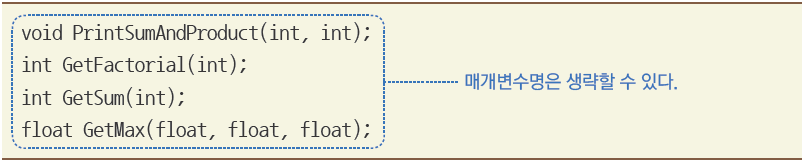
9-5. 함수 선언의 다른 표현

C언어에서는 코드의 순서가 매우 중요하다. C언어는 위에서 아래로 읽기 때문에 순서가 잘못 되어서 코딩을 잘못 하는 경우가 다반사이다. 그러한 맥락에서 보았을 때 함수를 ‘선언’하는 것은 정말 중요하다. 비유로 풀어서 설명을 해보겠다. 아까 나왔던 작업장의 예시를 들자면, A작업장에서 작업을 마치고 B작업장으로 보내고 싶다. 하지만 B작업장 자체가 존재 하지 않으면 보낼 수 조차 없다. 따라서 B라는 작업장이 존재 해야지만 A에서 작업이 끝난 다음에 B작업장으로 재료를 보낼 수 있는 것이다. 정리 하자면 함수를 호출 할 때는 반드시 호출 함수보다 앞쪽에 함수를 정의 해주어야 한다는 것이다.

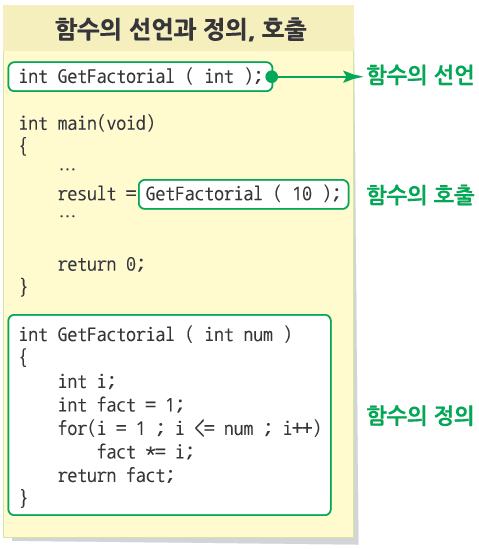


함수를 선언 하는 방법은 간단하다. 꼭 메인 함수(메인 작업장) 위에 선언을 해주어야 한다. 먼저 리턴형 (결과물의 종류), 함수명 (함수에 대한 설명), 매개변수 목록(인자로부터 어떤 형의 값을 받을 것인지 알 수 있다.) 매개변수 목록을 쓸 때 매개변수 이름을 선언하지 않을 수도 있으나, 너무 복잡해 지기 때문에 쓸 수 있는 것들은 다 쓰도록 하자.





이 마지막 그림이 우리가 지금까지 함수 단원에서 배웠던 것을 총 정리 해주는 그림이다. 컴퓨터의 입장에서는 함수가 선언이 되어있어야만(작업장이 만들어져 있어야지만) 그 함수(작업장)을 쓸 수 있다. 따라서 함수의 선언을 메인 함수 앞에 하고, 메인 함수 다음에 선언한 함수들의 나머지 정의를 배치하면 된다. 함수의 선언이란 함수의 내용을 알려주지는 않지만 함수 호출에 필요한 리턴형, 함수의 이름, 매개변수 정보를 미리 알려주는 것이다. 이렇게 하는 이유는 메인 함수가 가장 앞에 있을 때 코드의 전체적인 논리를 알 수 있는데 메인 함수에서 함수를 쓰기 위해서는 사용될 함수를 앞에서 선언해야 하기 때문이다.



\*\*함수의 기본 적인 정리\*\*

함수의 정의(definition) : 함수의 리턴형, 함수의 이름, 함수의 매개변수를 써준 다음 { } 안에 실제로 함수가 처리할 내용을 기술

함수의 호출(call) : 앞에서 선언되거나 정의된 함수를 이용할 수 있고, 인자를 넘겨주고 리턴 값을 받아올 수 있다.

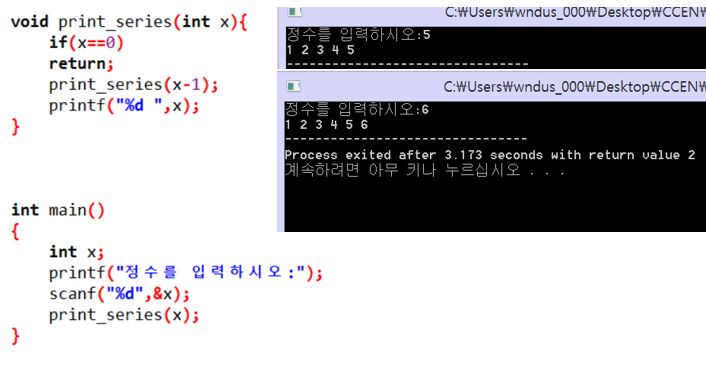
함수의 선언(declaration) : 함수의 내용을 알려주지는 않지만 함수 호출에 필요한 리턴형, 함수의 이름, 매개변수 정보를 미리 알려준다.

@더 알아보기: 재귀함수

지금까지 A작업장에서 B작업장을 호출한 뒤 다시 A작업장으로 반환하거나, C작업장으로 반환하는 방법에 대해 배웠다. 재귀함수는 A작업장에서 호출한 B작업장을 B작업장에서 또다시 B작업장을 호출하고 반환하는 것을 연속적으로 하는 함수이다. 재귀함수를 활용하면 효율적으로 코드를 작성할 수 있는 경우가 있다. 그러나 항상 효율적인 코드를 작성할 수 있는 것은 아니므로, 상황에 따라 적절히 사용할 수 있어야 한다. 몇 가지 예시를 통해 재귀함수를 사용하는 방법을 알아보도록 하자.

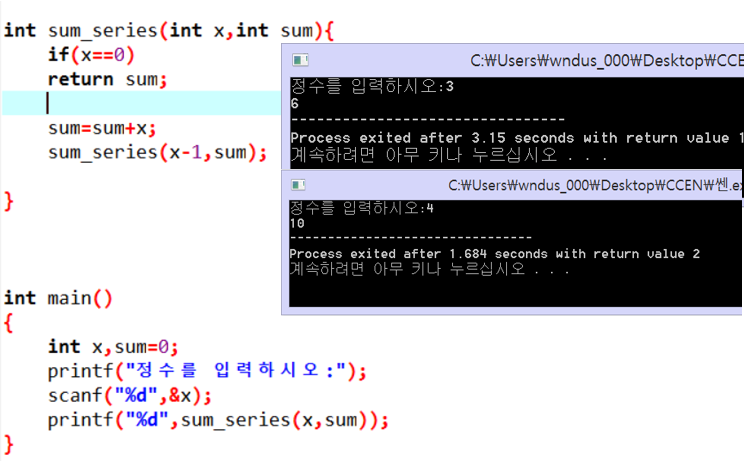
1부터 지정한 수까지 출력.

scanf로 숫자 x값을 정하면 x x-1 x-2 x-3 … 1까지의 값을 출력한다. print\_series(x)의 x의 값이 0을 만족할 때까지 함수를 호출하고 print\_series(x)의 조건문이 x==0인 경우에 return이 되고부터 print\_series(1) print\_series(2) … print\_series(x-1) print\_series(x) 순서로 반환이 된다.



1부터 지정한 수까지 합

scanf로 숫자 x값을 정하면 x + x-1 + x-2 + x-3 + … + 1까지의 값을 합한 결과를 출력한다. sum\_series(x)의 x의 값이 0을 만족할 때까지 함수를 호출하고 sum\_series(x)의 조건문이 x==0인 경우에 return이 되고부터 sum\_series(1) sum\_series(2) … sum\_series(x-1) sum\_series(x) 순서로 반환이 된다. 다시 말해 x 값이 4인 경우 4, 4 + 3, 4 + 3 + 2, 4 + 3 + 2 + 1 순서로 값이 sum에 저장된다.



팩토리얼 함수

scanf로 숫자 x값을 정하면 x \* x-1 \* x-2 \* x-3 \* … \* 1까지의 값을 곱한 결과를 출력한다. Factorial함수의 x의 값이 0을 만족할 때까지 함수를 호출하고 Factorial함수의 조건문이 x==0인 경우에 return이 되고부터 Factorial(1) Factorial(2) … Factorial(x-1) Factorial(x) 순서로 반환이 된다. 다시 말해 x 값이 4인 경우 4, 4\*3, 4\*3\*2, 4\*3\*2\*1 순서로 값이 Factorial함수에 반환된다.

