10 함수와 변수

10-1. 지역 변수

10-2. 전역 변수

10-3. 함수의 인자 전달 I

10-3-1. 값에 의한 전달 방법

10-3-2. 포인터에 의한 전달 방법

10-4. 기억 부류

10-4-1. 기억 부류 키워드 알아보기

10-4-2. 정적 변수 – 지역 변수와 전역 변수의 종합적 이해

10-4-3. 함수와 기억 부류

\*\*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

지역 변수 : (각각의 작업장) 전용 아이템(도구)

전역 변수 : (모든 작업장) 공용 아이템(도구)

Introduction.

10-1. 지역 변수

이번 장의 소제목을 보고 ‘어? 왜 함수 단원에서 앞에서 다루던 변수가 다시 나왔지? 변수를 그냥 앞에서 다 알려주었으면 되지 않나’ 이렇게 생각하는 사람들이 많을 것이다. 지역 변수와 전역 변수라는 개념은 꼭 함수라는 개념을 알아야 이해가 빠르게 되기에 이 함수 단원에서 다루기로 한다.

우리가 이번 함수 단원에서 계속 사용했던 가구 작업장 이야기로 계속 해보기로 하자. 메인 작업장이 아닌 다른 작업장이 있다고 가정하자. A라는 작업장에서는 나무를 잘라서 저장하는 역할을 하고 B라는 작업장에서는 자른 나무를 가공하는 역할을 한다. A작업장에서는 당연히 전기 톱과 나무를 저장 할 수 있는 저장소가 필요하고, B라는 작업장에서는 나무를 가공 할 수 있는 공장이 필요하다. 작업장마다 쓰이는 도구가 따로따로 있고, 다 다르다. 정리를 해보자면 지역 변수라는 것은 각 작업장에 해당되는 ‘전용 아이템’이라고 말할 수 있다. 작업장 A에서 쓰는 도구를 작업장 B에서 쓸 수 없는 노릇이고, 서로 호환이 될 수 있는지도 의문이다. 마찬가지로 지역변수는 그 작업장(해당 함수)에서만 쓸 수 있는 도구들을 말한다. 다른 함수의 인자로부터 받은 매개변수든지 아니면 함수 자체적으로 재정의한 매개변수든지 딱 그 함수(작업장)에서만 쓸 수 있다. 이제 코드와 함께 지역변수에 대한 이해를 더해보자.

지역 변수의 선언 위치

int sub(void)

{

int x;

x=100;

int y;

}

이를 컴파일해 보면 컴파일 에러가 뜨는데, 이러한 실수는 C언어를 처음 배우는 사람들이 자주 하는 경향이 없지 않아 있다. 함수의 안에서 선언된 변수이기 때문에 int x 나 int y를 지역변수라고 이야기를 한다. 그렇지만, 지역변수는 무조건 함수의 처음에(\*정확히는 블록의 처음에) 선언해야지만 컴파일 에러나 나지 않는다. “왜 꼭 그래야만 하죠?” 라고 물어보면 음.. 이런 질문 하고 비슷할 수 있다. ‘왜 사람의 눈은 2개가 있어야만 하죠?’ ‘왜 혀는 입 안에 있는 거죠?’ 되게 비꼬는 듯이 말하긴 했지만 그만큼 일반적인 것이라는 말이다. 데니스 리치가 C언어를 처음 만들 때 이렇게 하기로 ‘정의’ 한 것이다. 수학에서의 정의처럼, 프로그래밍 언어를 만들 때 그렇게 정의된 것이므로 반드시 이를 따라야만 한다. 가장 기본적인 것부터 잡아 나가도록 하자.

int sub(void)

{

int x;

int y;

x=100;

}

아까 코드를 제대로 고친 예이다. 이렇게 선언을 해야 어느 부분이 변수 선언부이고, 변수에 대입한 값은 무엇인지 확실하게 알 수 있다.

이번에는 특별하게 개념만으로 설명하지 않고, 간단한 문제를 풀면서 지역 변수를 설명하도록 하겠다.

자 여기서 깜짝 문제 하나 갑니다. 아래의 코딩이 이상한 점이 있다면 어떤 점일까요?

void sub1(void)

{

{

int y;

}

y=4;

}

이상한 점을 발견했는가? 발견하지 못했다면 직접 컴파일을 해 보라. 에러가 뜰 것이다. 도대체 왜? 이유는 중괄호 안에 int y 가 선언이 되었다는 점에 있다. 조금 앞에서 변수는 블록의 가장 앞 부분에서 선언해야 한다고 괄호로 묶어 표현했었다. 블록이라는 낯선 말에 당황하는 독자들도 있겠지만, 중괄호 { }를 블록이라고 한다. 따라서 그냥 함수도 하나의 블록이 되고, if문도, for문도, while문도 각각 하나의 블록이 되는 것이다. 그리고 위의 코딩처럼 아무것도 없이 단독으로 블록만 취할 수도 있다. 각각의 블록에서 선언된 변수는 모두 “지역 변수”이고, 이들 지역 변수들은 블록이 끝나면, 즉 }가 나오면 모두 사용하지 못하게 된다. 이를 변수의 영역 규칙(Scope rule)이라고 한다. 이에 관한 부가적인 설명들은 뒤에서 다루기로 한다.

그래서 위의 소스코드를 컴파일 에러가 나지 않는 코드로 고치게 된다면

void sub1(void)

{

int y;

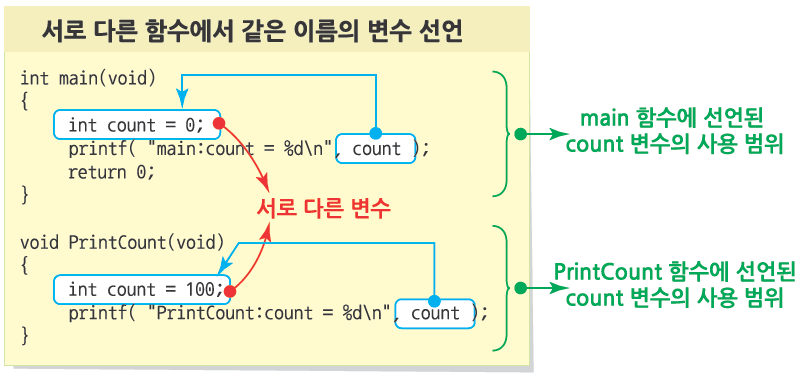
y=4;

}

이런 식으로 고쳐야 지역 변수 y를 사용 할 수 있다.

흥미로운가? 몇 가지를 더 해볼 테니 집중해서 보기 바란다.

(아래와 비슷한 그림을 넣을 것으로 예상 됨)



int main(void)

{

int count=10;

}

int sub(void)

{

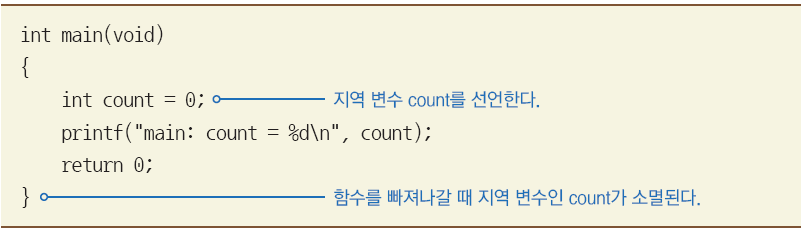
int count=20;

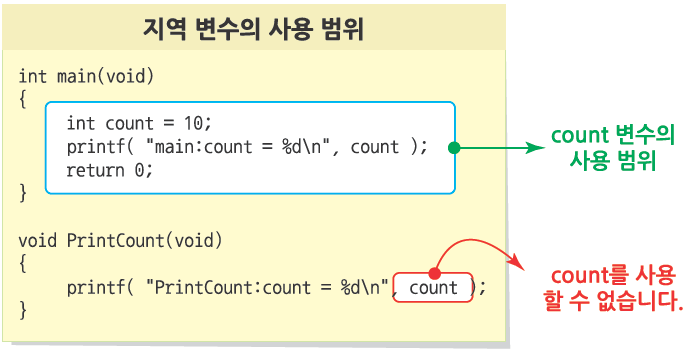
return count;

}

정답 : 어 이거 좀 이상한데? 이름이 같은 변수가 사용 될 수 있나? 할 수도 있지만 이는 아주 지극히 정상적이다. 이름이 같은데 어떻게 다른 값이 나올 수 있을까? 정답은 지역 변수로 선언이 되었기 때문에 아무리 이름이 같아도 나오는 결과 값은 다를 수도 있다. 쉽게 표현 하자면 main이라는 작업장과 sub라는 작업장은 엄연히 다른 작업장이다. 그래서 각각의 작업장에서 쓰는 전용 아이템도 다르기 때문에 이름이 같다고 해서 같은 아이템(변수)는 아닌 것이다.

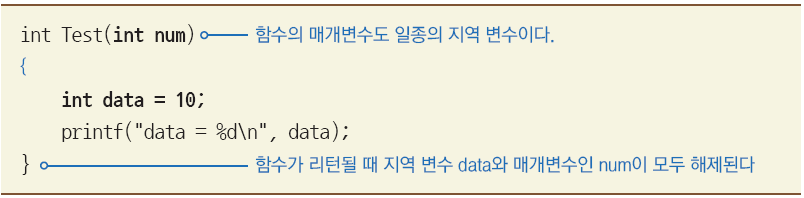
그러면 이런 경우는 어떻게 될까? 만약에 메인 함수(메인 작업장)에서 선언한 변수를 다른 함수(다른 작업장)에서 사용 할 수 있을까?

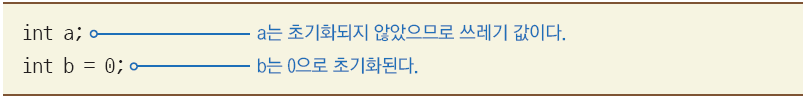




위의 내용에서 보았듯이 그 함수 안에서의 지역 변수(전용 아이템)는 그 함수(작업장)에서만 쓸 수 있다. 다른 함수(다른 작업장)로는 호환이 되지 않는다.

그 외에도 지역 변수는 따로 초기화 하지 않으면 쓰레기 값을 갖는 다는 특징이 있다..

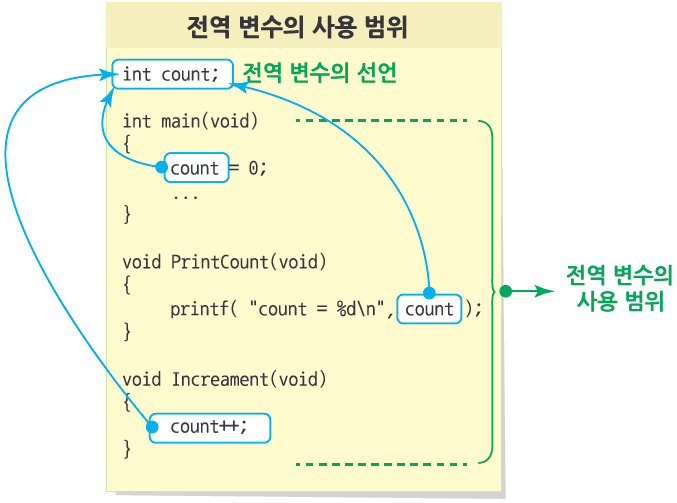




10-2. 전역 변수 🡪 어디서나 쓸 수 있는 공용 재료

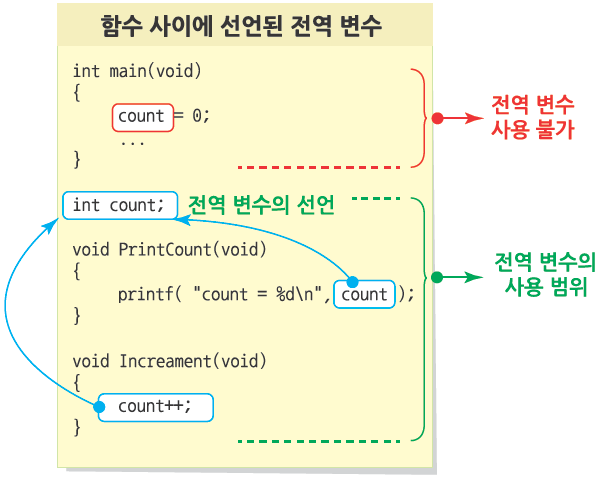
지역 변수를 이해했다면 전역 변수는 정말 빨리 이해할 수 있다. 지역 변수를 한 마디로 정의 하자면 ‘공용재료’라고 정의 할 수 있다. 아까의 가구 작업장의 예를 다시 가지고 와보자. 전역 변수라 함은 모든 작업장에서 공용으로 사용하는 것을 말한다. 예를 들어서 작업장의 디렉터들만 가지고 있는 회사 공용카드가 있다고 가정하자. 작업장A 에서 나무를 자르고 보관하기 위해서 돈이 필요하다. 그래서 회사 공용카드로 그 금액을 충당 했다. 그리고 작업장 B에서도 공장을 운영하기 위해서 인건비나 회사 시설관리비가 들었을 것이다. 그것을 회사 공용카드로 그 금액을 충당 했다. 이 회사 공용카드의 잔액은 계속 줄어들 것이다. 왜냐하면 그 전 작업장에서 쓴 만큼 카드의 돈이 줄어들기 때문이다.

전역 변수의 특징에 대해서 설명 해보자면 전역 변수는 함수 안에서 사용되는 지역변수와는 달리 프로그램 전체에서 실행되는 변수이다. 프로그램이 시작될 때 한 번만 생성되고, 프로그램이 수행되는 동안 여러 함수에서 사용되다가, 프로그램이 종료될 때 비로소 해체된다.



아까의 비유로 돌아가 보면 회사공인카드가 있어야 회사공인카드를 쓸 수 있다. 즉 전역 변수가 먼저 선언 되어야 지만 그 전역 변수를 그 아래에 있는 함수들이 사용 할 수 있다는 뜻이다. 컴퓨터는 코드를 읽을 때 위에서 아래로 읽는 다는 사실은 익히 알고 있을 것이다. 이러한 관점에서 보았을 때 전역 변수를 제일 위에 선언 하는 것이 좋다.

이 때 유의 해야 될 점은 지역 변수와 다르게 초기에 전역 변수를 만들게 되면 자동으로 0으로 초기화가 된다는 점이다.



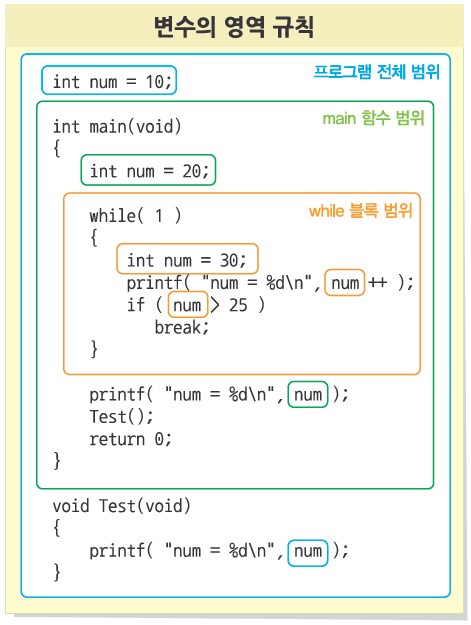
만약에 지역 변수와 전역 변수가 겹치는 어떻게 되겠는가?

정답 : 전역 변수가 아닌 지역변수가 사용되게 된다. 그 이유는 C언어의 성격 때문이다. 위에서 아래로 내려가면서 읽는 방식이기 때문에 아무리 전역 변수가 먼저 나와도 지역변수가 그 다음에 나오게 되면 다시 지역 변수로 초기화가 되어 버린다. 이는 위에서 잠깐 다루었던 변수의 영역 규칙(Scope rule)로 설명할 수 있다. 위에서는 영역 규칙에 대해 제대로 정의하지 않았지만, 여기에서 정의하자면

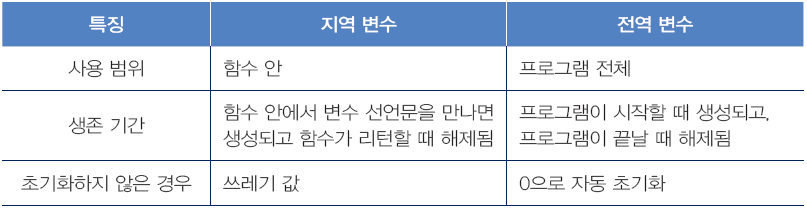
1. 변수는 반드시 블록의 첫 부분에서 정의해야 한다.

2. 해당 블록의 변수가 최우선 순위로 사용된다.

이 두 가지로 변수의 영역 규칙을 정의할 수가 있는데, 아까 전에 다루었던 것은 1번에 해당하는 내용이고 이번에 다루는 것은 2번에 해당하는 내용이다. 전역변수, 심지어 함수 안에 다른 지역변수가 동일한 이름으로 있더라도 현재의 블록에 같은 이름의 지역변수가 따로 선언이 되어 있다고 하면 이번 블록의 지역변수를 최우선으로 사용하는 것이다. 이는 위에서도 말했듯 C언어의 성격 때문인데, 간단히 소개하자면 스택(Stack) 구조라는 것이 있다. 영어 단어에서 알 수 있듯 (여기에서는 선언된 변수를) 하나씩 쌓아두는 건데, 각 블록마다 고유의 스택을 이용하기 때문에 이렇게 중복된 이름으로 변수를 사용하는 것이 굉장히 이상해 보이지만 컴파일과 실행에 아무런 문제가 발생하지 않는다.



마지막으로 변수들을 정리하고 변수를 마치도록 하겠다.



10-3. 함수의 인자 전달 방법

10-3-1. 값에 의한 전달 방법 Call by Value

값에 의함 전달방법을 다른 말로 ‘복사에 의한 전달’ 이라고도 한다. 이제 우리가 전 단원에 배웠던 포인터와 함수를 처음으로 결합해보는 순간이다. 원래 함수(작업장)이라는 개념 자체를 그냥 단순하게 생각 해서는 안 된다. C언어에서 이 부분을 다룰 때 가장 많이 쓰는 예시가 있는데 바로 SWAP 이라는 함수를 이용하는 것이다.

#include <stdio.h>

void Swap(int x, int y);

int main(void)

{

int a = 10;

int b = 20;

printf("Swap 전u의C a = %d, b = %d\n", a, b);

Swap(a, b);

printf("Swap 후A의C a = %d, b = %d\n", a, b);

return 0;

}

void Swap(int x, int y)

{

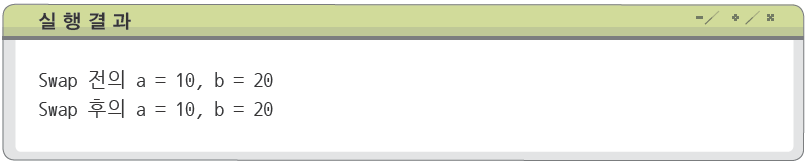
int temp;

temp = x;

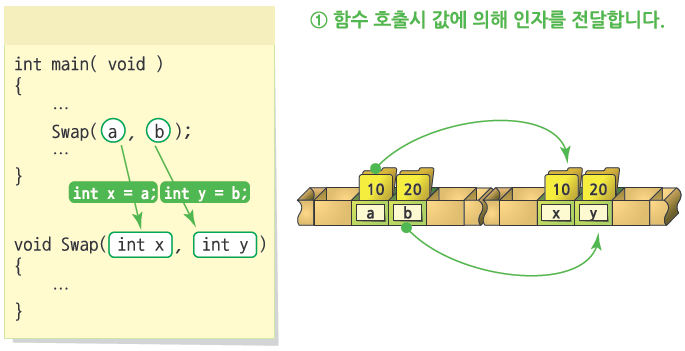
x = y;

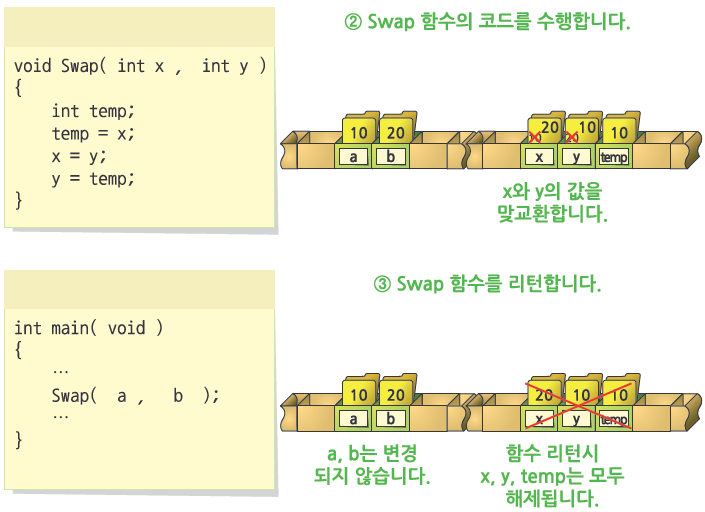
y = temp;

}



이상하게도 실행결과 자체에는 아무런 변화가 없다. 어떤 일이 있었길래 바뀌지 않았던 것일까? 당연히 SWAP 함수에서는 A와 B의 값이 바뀐 것이 맞다. 하지만 여기에서 중요하게 생각해야 될 점은 바로 PRINT를 하는 함수가 바로 메인 함수라는 점이다. 아까 우리가 배웠던 지역 변수의 개념을 다시 떠올려 보자. 아무리 SWAP 함수에서는 값이 바뀌었을지 몰라도 다시 메인 함수로 돌아오니깐 원래 있던 지역변수가 자리잡고 있었기 때문에 아무런 변화도 있지 않았던 것이다.





위의 그림이 함수의 과정을 정확히 설명해주는 그림이다.

.

10-3-2. 포인터에 의한 전달 방법 (참조에 의한 전달 방법) Call by Reference

포인터에 의한 전달방법에 의하면 진짜 swap 함수를 구연할 수 있다. 포인터에 의한 전달 방법이라 함은 변수의 값을 전달하는 대신 변수의 주소를 전달하는 방식이다. 함수의 인자로 변수의 주소를 전달한다. 이게 무슨 이야기 인지는 예시를 보면서 이야기 해보도록 하자

#include <stdio.h>

void Swap(int \*x, int \*y);

int main(void)

{

int a=10;

int b=20;

printf("Swap 전u의C a = %d, b = %d\n", a, b);

Swap(&a, &b);

printf("Swap 후A의C a = %d, b = %d\n", a, b);

return 0;

}

void Swap(int \*x, int \*y)

{

int temp=\*x;

\*x = \*y;

\*y = temp;

}

우리가 이 예시를 보면서 알고 있어야 될 점은 &a, &b라는 것은 a,b 변수의 주소(위치)를 구한다는 점하고, \*x, \*y 은 변수 x, y에 저장된 값이라는 것이다. 메인 함수에서 &a, &b를 인자로 해서 보내주게 된다. 이 뜻은 10, 20 을 보내 준다는 뜻이 아니라, 10, 20을 가지고 있는 변수(상자)의 위치를 나타내는 주소를 인자로 보낸 다는 것이다. 그러면 바로 swap 함수로 이동하게 된다. &a라는 것은 변수 x와 같게 되고, &b는 변수 y와 같게 된다. x라는 상자에 &a라는 내용물이 들어있게 된다. 그래서 \*x라고 하게 되면 a의 주소가 가리키는 값이 되게 된다. 값을 직접 보내는 것이 아니라 주소를 전달해 줌으로써 swap 함수의 문제가 해결된다.

10-4. 기억 부류

앞의 포인터 단원에서 const 키워드를 변수 앞에 선언하였던 것을 기억할 것이다. 이렇게 변수 선언 시 자료형 앞에 오는 키워드들에는 여러 종류가 있는데, 그 중에서도 변수의 ‘저장 위치’와 ‘이용 범위’를 결정하는 키워드, 즉 기억 부류들에 대해 알아볼 것이다.

포인터 단원에서 변수를 동적 할당하여 이용하던 것을 기억해보자. 독자들은 아마도 변수들이 모두 메모리(RAM)에 저장되는데, 왜 저장 위치를 지정하는 것일까 하는 의문을 품을 것이다. 이에 대한 대답은, ‘모든 변수가 메모리에 저장되어야만 하는 것은 아니다’이다.

10-4-1. 기억 부류 키워드 알아보기: register, extern, static

모든 변수가 메모리에 저장되어야만 하는 것이 아니라고? 이해가 되지 않을 것이다. C언어에서 변수는 꼭 메모리에 저장할 필요가 있는 것이 아니라는 말은 변수를 다른 곳에 저장할 수도 있다는 뜻이 된다. 그렇다면 어디에 저장할 수 있을까?

컴퓨터 본체를 열어보면, 메모리(RAM) 말고 소위 컴퓨터의 두뇌라고 불리우는 정사각형 모양의 중앙처리장치(CPU)가 있다. 이 CPU 내부에는 자료를 보관하는 아주 빠른 기억 장소인 register라는 것이 있는데, C언어에서 변수를 메모리에 저장하는 대신 이 register에 저장할 수도 있다. 메모리에 있는 변수를 이용하려면 매번 그 변수의 데이터를 CPU에서 불러오고, 저장하는 별도의 명령 체계를 이용하기 때문에 속도가 느리다. 그러나 register에 있는 변수는 CPU 내부에 저장된 것이므로 이런 별도의 과정이 필요하지 않다. 그래서 변수를 접근하는 속도가 매우 빠르다. 그러면 이렇게 register에 변수를 저장하기 위해서는 어떤 키워드를 이용해야 할까?

답은 말 그대로 register이다. register int a; 라고 선언을 해 주면, 이 a라는 변수는 메모리가 아닌 register에 저장되고 이용된다.

register 변수는 이용할 때 여러 유의점들이 있다. 우선 메모리에 저장되는 것이 아니므로 주소 구하기 연산자(&)를 이용할 수도 없고, 포인터로서 이용할 수도 없다. 또한 이런 register 변수는 지역변수로만 선언이 가능하다는 것을 염두에 두어야 한다. register 변수는 메모리를 직접 불러오지 않아도 되어 처리 속도가 매우 빠르고, 따라서 for문과 같은 반복문에서 제어 변수(i)로 이용되는 경우가 많다.

register 키워드는 변수의 ‘저장 위치’를 지정하는 기억 부류이다. 그러면 이번에는 ‘이용 범위’를 지정하는 키워드를 알아보도록 하자. ‘이용 범위’를 지정하는 키워드에는 extern 과 static 두 가지가 있다. 먼저 extern 부터 알아보도록 한다.

extern 키워드는 external linkage의 약자로, 변수가 전역 변수일 때 이용하는 것인데 전역 변수의 경우 해당하는 변수가 다른 곳에 선언되어 있다는 것을 명시해주는 키워드이다. 그래서 전역 변수의 선언 위치와 관계없이 이용 가능하다. (예시 그림) extern 키워드를 이용하는 경우 메모리 할당은 하지 않으며, 같은 프로그램 내(같은 프로젝트 내) 다른 소스 파일이나 헤더 파일에서도 그 변수를 이용할 수 있다. (지역변수 extern은?)

‘이용 범위’를 지정하는 두 번째 키워드는 static이다. static 키워드는 기억 부류 중 가장 많이 이용한다. 다음 소단원에서 더 자세히 알아보자.

10-4-2. 정적 변수 – 지역 변수와 전역 변수의 종합적 이해

static 키워드를 변수 앞에 지정하면 정적 변수가 되는데, static 키워드의 경우는 지역 변수와 전역 변수 모두에 붙여 쓸 수 있다. 그러나 두 가지의 기능이 매우 다르다. static 키워드를 지역 변수에 붙이면 정적 지역 변수가 되고, 전역 변수에 붙이면 정적 전역 변수가 된다. 하나씩 기능을 알아보자.

먼저 정적 지역 변수에 대해 알아보자.

(static 넣은 것과 안 넣은 것 비교하는 간단한 예제)

이 예제를 통해 알 수 있는 점은, 함수 내의 지역변수에 static 키워드를 지정할 경우 함수가 종료되어도 그 변수는 살아남는다는 사실이다. 이는 이 함수의 호출 횟수를 판별하는 프로그램에서 유용하게 사용될 수 있는 기능이다. 정적 변수, 그리고 이를 위한 static 키워드를 가장 많이 이용한다고 하였는데 고작 이런 기능만을 위해 존재하는 것일까? 아니다. 정적 지역 변수는 앞 장에서 포인터를 배우고, 또 함수의 인자 전달 방법을 학습한 여러분들에게 꼭 필요한 기능을 제공한다. 다음 예제를 통해 더 알아보도록 하자.

(함수 안에서 선언한 지역 변수의 주소를 리턴하는 예제)

이 소스 코드는 아무 문제가 없는 것처럼 보인다. 독자 여러분들이 직접 컴파일을 해 보라! 어..? 무언가 이상한 점이 있다. 눈썰미가 좋은 독자들은 알아챘을지도 모르겠다. 변수의 영역 규칙 덕분에 함수의 블록이 종료되는 동시에 그 변수를 담았던 스택이 초기화되고, 따라서 리턴할 주소 값이 증발해버린다! 참 총체적 난국이다. 어떻게 하면 이 문제를 해결할 수 있을까?

(위의 예제에서 static 키워드를 지정해준 예제)

이렇게 static 키워드를 지정해 주기만 하면, 함수가 종료되어도 그 변수만큼은 살아남게 되고, 이러한 변수의 경우에만 주소 값을 리턴할 수 있다! 이는 정말 유용하게 이용되니 독자 여러분들은 제대로 익히도록 하자.

그럼 지금까지 정적 지역 변수, static 지역 변수에 대해 알아보았다. 지역 변수를 알아보았으니, 이젠 전역 변수가 남았지 않은가! 전역 변수의 경우 지역 변수와는 그 성질이 매우 다르다. 전역 변수는 디폴트 값이 extern 으로 지정되어 있는데, 앞에서 배웠듯 extern 의 경우 다른 소스코드에서 이용할 수가 있다! 그렇다면 다른 소스 코드에서 이용하지 못하게 하도록 하는 방법은 무엇일까?

그렇다. 이럴 때 이용하는 것이 바로 static 키워드를 이용한 정적 전역 변수이다. 처음 선언한 변수에 한번 static이 키워드가 달리면, 다른 소스코드에서 extern 선언을 다시 하더라도 이용할 수가 없게 제한된다.

10-4-3. 함수와 기억 부류

지금까지 변수 앞에 기억 부류 키워드를 붙이는 경우에 대해 알아보았다. 함수에도 이와 같이 기억 부류를 사용할 수 있다. C언어에서 **하나의 함수는 하나의 전역 변수로 봐도 무방**한데, 따라서 함수에 사용할 수 있는 키워드는 extern과 static 뿐이다. 전역 변수와 마찬가지로 extern을 함수 앞에 붙이면 같은 프로그램 내 다른 소스 코드나 헤더 파일에서 그 함수를 이용가능하고, static을 붙일 경우 다른 소스 코드에서 접근하지 못하게 된다.