

1) (((a \* b)1 - 1)2 + c)3

2) (((a \* (b – 1)1)2 / c)3 mod d)4

3) (((a – b)1 / c)2 & (((d \* e)3 / a)4 – 3)5)6

4) ((–a)1 or ((c = d)2 and e)3)4

5) (((a > b)1 xor c)3 or (d <= 17)2)4

6) (–(a + b)1)2



1) (a \* (b – (1 + c)1)2)3

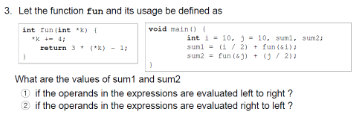
2) (a \* ((b – 1)2 / (c mod d)1)3)4

3) ((a – b)5 / (c & (d \* (e / (a – 3)1)2)3)4)6

4) (–(a or (c = (d and e)1)2)3)4

5) (a > (b xor (c or (d <= 17)1)2)3)4

6) (–(a + b)1)2



1) left -> right 이기 때문에

sum1 = ((i / 2)1 + fun(&i)2)3

sum2 = (fun(&j)1 + (j / 2)2)3

순으로 진행된다.

이 때 sum2의 경우 함수 호출 시 j에 대한 포인터 연산을 진행하므로 main에서 j값이 바뀐다.

따라서 sum1 = 10/2 + 3\*(10+4)-1 = 46이고, sum2 = 3\*(10+4)-1 + 14/2 = 48이다.

Sum1 = 46, sum2 = 48

2) right -> left이기 때문에

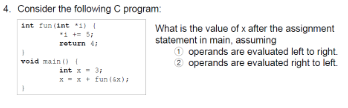
sum1 = ((i / 2)2 + fun(&i)1)3

sum2 = (fun(&j)2 + (j / 2)1)3

순으로 진행된다.

마찬가지로 sum1의 경우 포인터연산을 진행하므로 main에서 함수 호출 후 i의 값이 바뀐다. 따라서 sum1 = 3\*(10+4)-1 + 14/2 = 48이고, sum2 = 10/2 + 3\*(10+4)-1 = 46이다.

Sum1 = 48, sum2 = 46

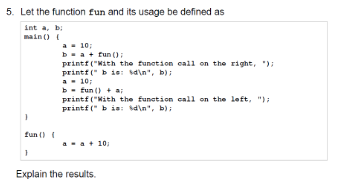


1) left -> right 이므로 x값인 3과 함수의 리턴값인 4가 더해져 x = 3+4 = 7이 된다.

X = 7

2) right -> left 이므로 함수호출이 된 후 더하기 연산 순으로 진행된다. 이 때 함수는 x에 대한 포인터 연산으로 함수 호출 시 x의 값이 바뀌게 된다. 함수의 리턴값은 4이지만 함수 호출 과정에서 x = x+5가 되므로 x = 8이 된다. 따라서 x = 8+4 = 12가 된다.

X = 12



* fun함수 int fun(){ a=a+10; return(a);}로 수정

C언어의 경우 operand간의 우선순위가 있고 연산자가 많을 때 연산자 우선순위에 따라 식이 바인딩 되고 그 바인딩된 식을 연산해준다. 그리고 C언어에서는 함수호출의 연산자는 우선순위가 가장 높은것들 중 하나이다.

With the function call on the right 부분에서 함수의 우선순위가 +보다 높으므로 함수를 호출한 후 +연산이 진행된다. 이 때 a는 전역변수로 fun함수 진행 후 값이 바뀌는데, fun 함수 내부에서 a=a+10이 되므로 a=20이 된다. 따라서 b=20+20=40이 된다. 결과로는

With the function call on the right, b is: 40

이 출력 된다.

With the function call on the left 부분에서 역시 함수 호출 후 +가 진행되고, 함수에서 사용되는 변수들 역시 전역변수 이므로 fun함수 수행 결과 a값이 바뀌게 된다. 따라서 b=20+20=40이 되고

With the function call on the left, b is: 40

이 출력된다.