[2018254-김도현] PL-HW4.

1. Assume the following rules of associativity and precedence for expressions

Precedence Highest	*, /, **not**
	+, –, &, **mod**
	– (unary)
	=, /=, < , <=, >=, >
	and
Lowest	**or**, **xor**
Associativity Left to right	

Show the order of evaluation of the following expressions by parenthesizing all subexpressions and placing a superscript on the right parenthesis to indicate order. For example, for the expression

`a + b * c + d` ⇨ `((a + (b * c)1)2 + d)3`

① `a * b - 1 + c`
② `a * (b - 1) / c mod d`
③ `(a - b) / c & (d * e / a - 3)`
④ `-a or c = d and e`
⑤ `a > b xor c or d <= 17`
⑥ `-a + b`

---

① *가 가장 Precedence가 높고 +,−는 동일하다. 그리고 associativity가 left to right.

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

② *, / 가 precedence가 동일하게 높고 mod는 그 다음 순위이다.

$\therefore ((((a*(b-1)^1)^2/c)^3 \bmod d)^4$

③. 양 옆의 ( ) . ( ) 들은 왼쪽부터 연산하고 모든쌍 ( )는 *부터 /, − 순으로 진행한다. &가 마지막이 된다.

$\therefore \left(\left(\left((a-b)^1/c\right)^2 \,\&\, \left(\left(\left((d*e)^3/a\right)^4-3\right)^5\right)^6\right.\right.$.

④. 연산 순서는 −(unary), =, and, or 순이다.

$\therefore \left(\left(-a\right)^1 \text{ or } \left(\left(c=d\right)^2 \text{ and } e\right)^3\right)^4$

⑤. 연산순서는 (precedence 동일시 왼쪽→오른쪽). >, <=, xor, or 순이다.

$\therefore \left(\left(\left((a>b)^1 \text{ xor } c\right)^3 \text{ or } \left(d<=17\right)^2\right)^4\right.$

⑥ 연산순서는 +, −(unary) 이다.

$\therefore \left(-(a+b)^1\right)^2$

2. Show the order of evaluation of the expressions of Problem 1, assuming that there are no precedence rules and all operators associate right to left.

precedence rule이 없기 때문에 우선하는 순서를 associate 또는 right→left로 설정해주면 된다.

① $(a * (b - (1+c)^1)^2)^3$

② $(a * ((b-1)^2 / (c \text{ mod } d)^1)^3)^4$

③ $((a-b)^5 / (c \,\&\, (d * (e / (a-3)^1)^2)^3)^4)^6$.

④ $(- (a \text{ or } (c = (d \text{ and } e)^1)^2)^3)^4$

⑤ $(a > (b \text{ xor } (c \text{ or } (d <= 17)^1)^2)^3)^4$

⑥ $(- (a+b)^1)^2$

3. Let the function **fun** and its usage be defined as

```
int fun(int *k) {
   *k += 4;
     return 3 * (*k) - 1;
}
```

```
void main() {
         int i = 10, j = 10, sum1, sum2;
         sum1 = (i / 2) + fun(&i);
         sum2 = fun(&j) + (j / 2);
}
```

What are the values of sum1 and sum2

① if the operands in the expressions are evaluated left to right ?

② if the operands in the expressions are evaluated right to left ?

①. 왼쪽에서 오른쪽으로 계산되기 때문에

$Sum1 = ((i/2)^1 + fun(\&i)^2)^3 = ((10/2)^1 + (3*(10+4)-1)^2)^3 = (5+41) = 46$

∴ Sum1 = 46.

$Sum2 = (fun(\&j)^1 + (j/2)^2)^3 = ((3*(10+4)-1)^1 + (14/2)^2)^3 = (41+7) = 48$

(j값이 바뀐다. 먼저 fun이 호출되기 때문에.)

∴ Sum2 = 48.

② 오른쪽에서 왼쪽으로 계산되기 때문에

$Sum1 = ((i/2)^2 + fun(\&i)^1)^3 = ((14/2)^2 + (3*(10+4)-1)^1)^3 = 7+41 = 48.$

(fun이 먼저 호출되기 때문에 i값이 10→14로 변경.)

∴ Sum1 = 48

$Sum2 = (fun(\&j)^2 + (j/2)^1)^3 = ((3*(10+4)-1)^2 + (10/2)^1)^3 = 41+5 = 46.$

∴ Sum2 = 46.

4. Consider the following C program:

```
int fun(int *i) {
        *i += 5;
        return 4;
}
void main() {
        int x = 3;
        x = x + fun(&x);
}
```

What is the value of x after the assignment statement in main, assuming
① operands are evaluated left to right.
② operands are evaluated right to left.

① 왼쪽에서 오른쪽으로 계산되기 때문에
x = x + fun(&x) = 3 + 4 = 7. (x값이 이때 3으로 변경되버림).

② 오른쪽에서 왼쪽으로 계산되기 때문에
fun(&x) 먼저 호출이 되서 x가 3+5=8로 바뀐다. (fun(&x)의 return값은 그대로 4)
∴ x = x + fun(&x) = 8 + 4 = 12.

5. Let the function **fun** and its usage be defined as

```
int a, b;
main() {
        a = 10;
        b = a + fun();
        printf("With the function call on the right, ");
        printf(" b is: %d\n", b);
        a = 10;
        b = fun() + a;
        printf("With the function call on the left, ");
        printf(" b is: %d\n", b);
}

fun() {
        a = a + 10;
}
```
return a?

Explain the results.

- C compiler에서 연산과 우선순위 등 함수호출이 가장 높은 순위를 차지하기 때문에 계산할때 제일 먼저 호출된다.

그래서 b=a+fun() 에서 fun이 먼저 호출되어 a=a+10, 즉 a=10+10=20이 된다. (a=10, global 변수)

a가 global 변수이기 때문에 20으로 변경된다.

그래서 b=a+fun()=20+20=40 이다.

결과: with the function call on the right, b is: 40.

이후 다시 a=10으로 변경하고 b=fun()+a 를 계산한다. 이 때도 함수호출이 우선이기 때문에 fun()에서 a=a+10, 즉 a=10+10=20 으로 global 변수 a가 20으로 바뀌었다.

그래서 b=fun()+a= 20+20=40 이 된다.

결과: with the function call on the left, b is: 40.

(실제 gcc compiler를 이용했을 때 예상과 같은 결과가 나왔음을 알 수 있었다.)