프로그래밍언어론 HW#3 2017038064김동용

1. 다음과 같은 코드가 주어졌다고 가정하자.

```
function test(int a, int b, int c)
        begin
          a := b + c;
          b := c + 2;
          print a, b, c;
        end

function main
        begin
          int i := 15;
          int j := 5;
          ink k := 10;
          test(i, j, j + k);
          print i, j, k;
        end
```

위 언어의 함수가 각각

i) a, b, c, 모두 call by value

ii) a, b는 call by value result, c는 call by value

iii) a, b는 call by reference, c는 call by value

iv) a, b, c, 모두 call by name

의 parameter 전달 방법을 사용할 때, 결과와 함께 이유를 설명하시오.

프로그래밍언어론 HW#3 2017038064김동용

1) a, b, c 모두 call by value를 했을 때, test(15, 5, 15)로 값이 그대로 전달되어 fuction test 내부에서 a=20, b=17, c=15이며 main에서는 i, j, k값이 바뀌지 않기 때문에

   결과는 **20 17 15 15 5 10** 입니다.

2) a, b는 call by value result, c는 call by value를 했을 때, test(i=15, j=5, 15)로 값과 결과가 전달되어 fuction test 내부에서 20 17 15의 값을 출력하고 call by value result에 의해 fuction test의 a, b가 main의 i, j에 전달되기 때문에 i=20, j=17로 바뀌므로 총 결과는 **20 17 15 20 17 10**입니다.

3) a, b는 call by reference, c는 call by value를 했을 때, test(i, j, 15)로 변수의 주소와 값이 전달됩니다. c는 call by value에 의해 fuction test 내부에서 20 17 15의 값을 출력하고 call by reference에 의해 a, b가 각각 i, j를 각각 가리키고 있기 때문에 a, b가 바뀌면 i, j도 바뀝니다. 따라서 i=20, j=17로 바뀌므로 총 결과는 **20 17 15 20 17 10**입니다.

4) a, b, c, 모두 call by name을 했을 때, i=15, j=5, k=10으로 fuction test에서 a대신 i, b대신 j, c대신 j+k로 바뀌어 다시 함수로 나타내면

```
function test(int a, int b, int c)
    begin
      i := j + j + k;
      j := j + k + 2;
      print i, j, j+k;
    end
```

가 되며 출력하면 20 17 27이고 main에서도 마찬가지로 i, j, k가 적용되기 때문에 총 결과는 **20 17 27 20 17 10**입니다.

2. 다음과 같은 swap subprogram 이 Java에서 올바르게 동작하지 않는 이유를 설명하시오.

```
public static void swap(int x, int y)
{
        int temp = x;
        x = y;
        y = temp;
}
public static void main(String[] args)
{
        int x = 1;
        int y = 2;
        swap(x, y);
        System.out.println(x);
        System.out.println(y);
}
```

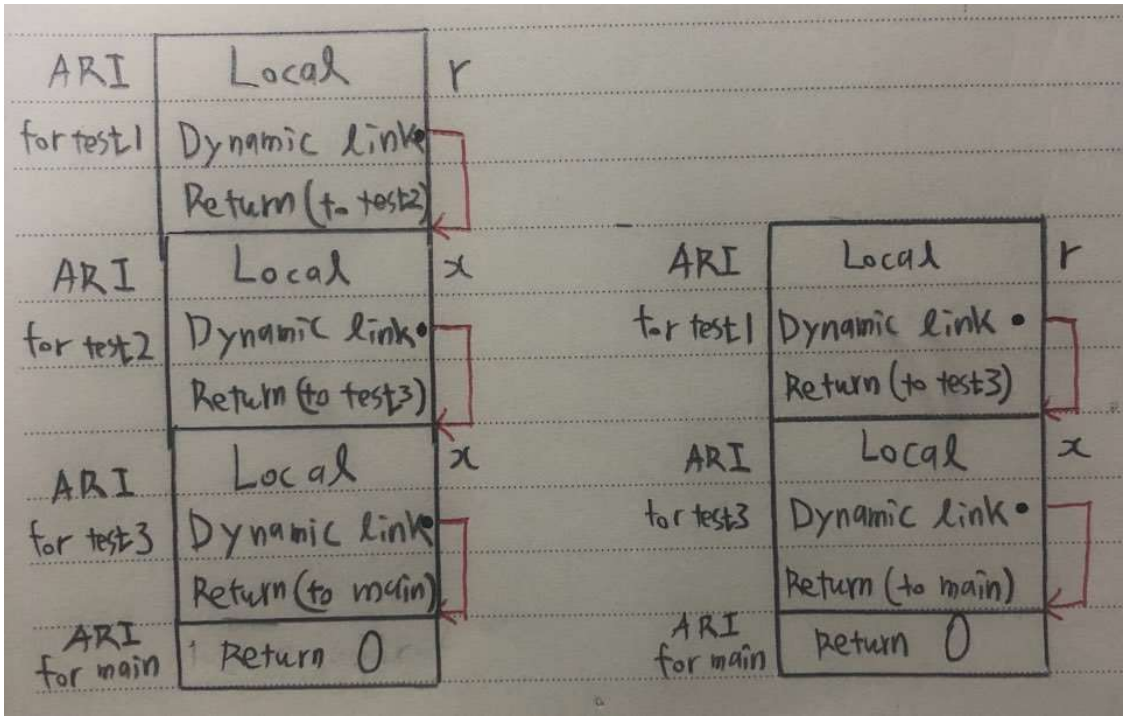
위 swap subprogram은 call by value이며 swap을 호출해 x=1, y=2의 값을 swap함수에 인자로 전달하였습니다. 따라서 **swap함수 내부에서는 x=2, y=1로 바뀌지만** main함수에서 보았을 때, call by value이므로 x, y의 값은 변하지 않기 때문에 **1 2**를 출력합니다. 따라서 swap이라는 의도와는 다르게 x, y 값이 그대로 출력되는 것을 알 수 있으므로 올바르게 동작하지 않는 subprogram입니다.

3. 다음 C 프로그램이 있다 가정하자.

```
void test1(void)
{
        double r = 2;
        printf("%g₩n",r);
        printf("%d₩n",x);
}
void test2(void)
{
        int x = 1;
        test1();
}
void test3(void)
{
        double x = 3;
        test2(); ---------- (1)
        test1(); ----------(2)
}
Int main()
{
        test3();
        return 0;
}
```

프로그래밍언어론 HW#3 **2017038064김동용**

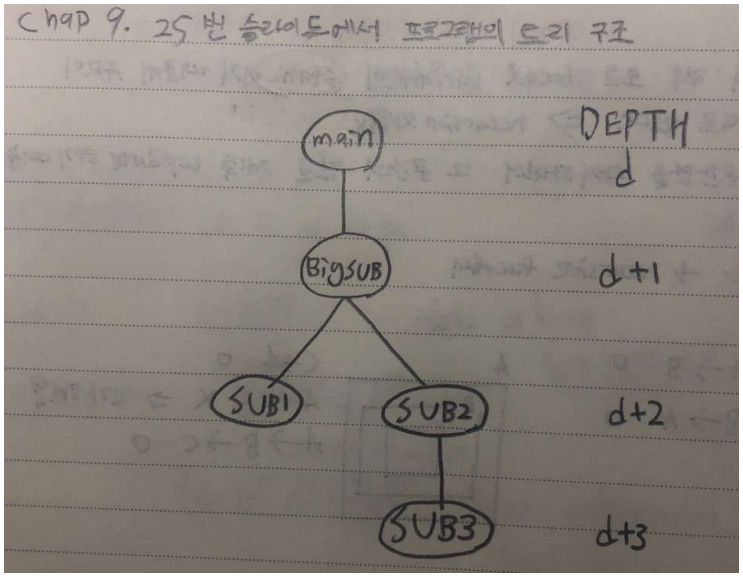
(1), (2) 부분이 실행된 **직후**, run-time stack 안에서 각 subprogram 들의 activation record의 상태를 각각 표현 하시오. 이 때 각 subprogram 들은 non-nested 하고 stack-dynamic local variable이 존재하는 있는 subprogram이라 가정 하시오. 또한 dynamic link는 항상 해당 AR의 맨 위를 가리키게 하시오.



첫 번째 그림은 main에서 test3호출, test3에서 (1) 즉, test2를 호출했을 때 그림입니다. test2를 호출하면 local variable이 AR에 들어가고 그 다음 test1을 호출하므로 이것을 그림으로 나타내면 위와 같습니다. 또한, 마지막에 main함수가 종료되었을 때, 0을 반환하므로 return 0이라는 것을 명시하였습니다.

두 번째 그림은 main에서 test3호출, (1)번 과정을 마치고 (2)번 즉, test1을 호출했을 때 그림입니다. (1)번 과정을 마치고 test3가 아직 종료되지 않았으므로 AR에는 main, test3에 대한 상태가 들어가 있고 test1을 호출하므로 test1의 상태가 AR에 들어갑니다. 첫 번째 경우와 마찬가지로, 마지막에 main함수가 종료되었을 때, 0을 반환하므로 return 0이라는 것을 명시하였습니다.

4. Chap 9 slide 25 page에 나오는 activation record 상에 있는 각 subprogram의 static link를 어떻게 결정하는지 28 page에 나오는 방법 2를 이용하여 (static chain을 이용하는 방법) 설명하시오.



MAIN_2->BIGSUB : d-(d+1)+1=0

메인함수에서 BIGSUB를 호출했을 때, 트리 상에서 0번 타고 올라가므로 BIGSUB의 LCA는 MAIN_2이다.

BIGSUB->SUB2 : d+1-(d+2)+1=0

BIGSUB가 SUB2를 호출했을 때, 트리 상에서 0번 타고 올라가므로 SUB2의 LCA는 BIGSUB이다.

SUB2->SUB3 : d+2-(d+3)+1=0

SUB2가 SUB3를 호출했을 때, 트리 상에서 0번 타고 올라가므로 SUB3의 LCA는 SUB2이다.

SUB3->SUB1 : d+3-(d+2)+1=2

SUB3가 SUB1을 호출했을 때, 트리 상에서 2번 타고 올라가므로 SUB1과 SUB3의 LCA는 BIGSUB이다.