INF3110

Solutions week 28.9 - 2.10, 2009

Problem 1

This is an example on both visibility and block structure, and that it is important to know the execution model of a language, and especially how initial values are handled.

(a) The reason that this does not work is that the innermost absVal are declared together with setting their values to (-)num. Arriving at the printf statement one has left one of the two blocks above, so the declared absVal variables do not exist anymore. Therefore the value of the absVal variable declared in line 3 is printed, and this has an arbitrary value.
(b) There is a subtle reason why this program works. It is still wrong to declare absVal in the inner blocks, and the printf statement will still write the value of the last declared variable (the two others are not visible), but because the last block (starting line 18) activation record is allocated on the top of the stack and thereby in the same area where one of the inner block activation records were allocated, the last allocated absVal variable will be allocated in the same area as the absVal of one of the inner blocks. And, as there is no default initialization, it will have the value that happens to be there, and this happens to be (-)num.
(c) E.g. {int arbitraryVal = -1 }
(d) This will allocate an integer at the same place as absVal and assign it the value -1, which will never be a correct value.

Problem 2

(a) x+f(x)= 7 + f(7) = 7 + (2+7) = 16

line	2	4	5
x	2	7	7

(b) x+f(x)= 7 + f(7) = 7 + (7 + 7) = 21

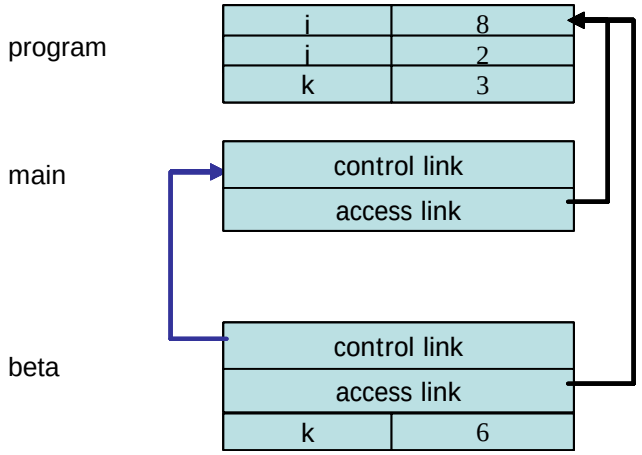
line	2	4	5
x	7	7	7

Problem 3

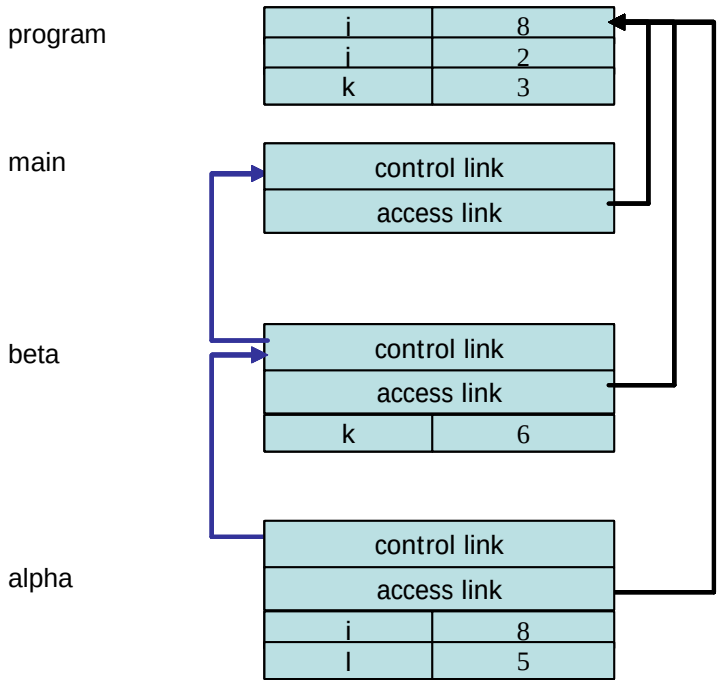
	Scope	**Life-time**
'i' in block 1	block 1 minus block 2 (and thereby also minus block 3 and 4, as these are defined within block 2), and block 5 as this does not define the name 'i'	block 1
'j' in block 1	block 1, block 2 (minus block 3), block 4 and 5	block 1
'k' in block 1	block 1 minus block 2 (and thereby also block 3 and 4), and block 5	block 1
'i' in block 2	block 2 and 3, but not block 4	block 2
'k' in block 2	block 2 and 3, but not block 4	block 2
'j' in block 3	block 3	block 3
'i' in block 4	block 4	block 4
'l' in block 4	block 4	block 4
'a' in block 5	block 5	block 5
'b' in block 5	block 5	block 5
'c' in block 5	block 5	block 5
'd' in block 5	block 5	block 5

Problem 4

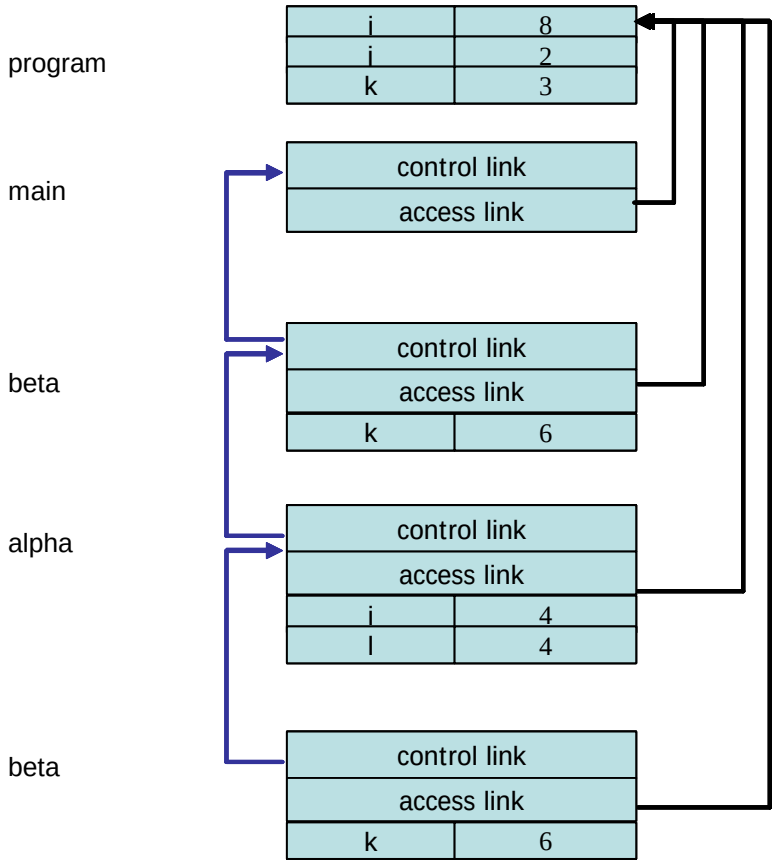
1)


program
i
8
j
2
k
3
main
control link
access link
beta
control link
access link
k
6

2)


program
i
8
j
2
k
3
main
control link
access link
beta
control link
access link
k
6
alpha
control link
access link
i
8
l
5

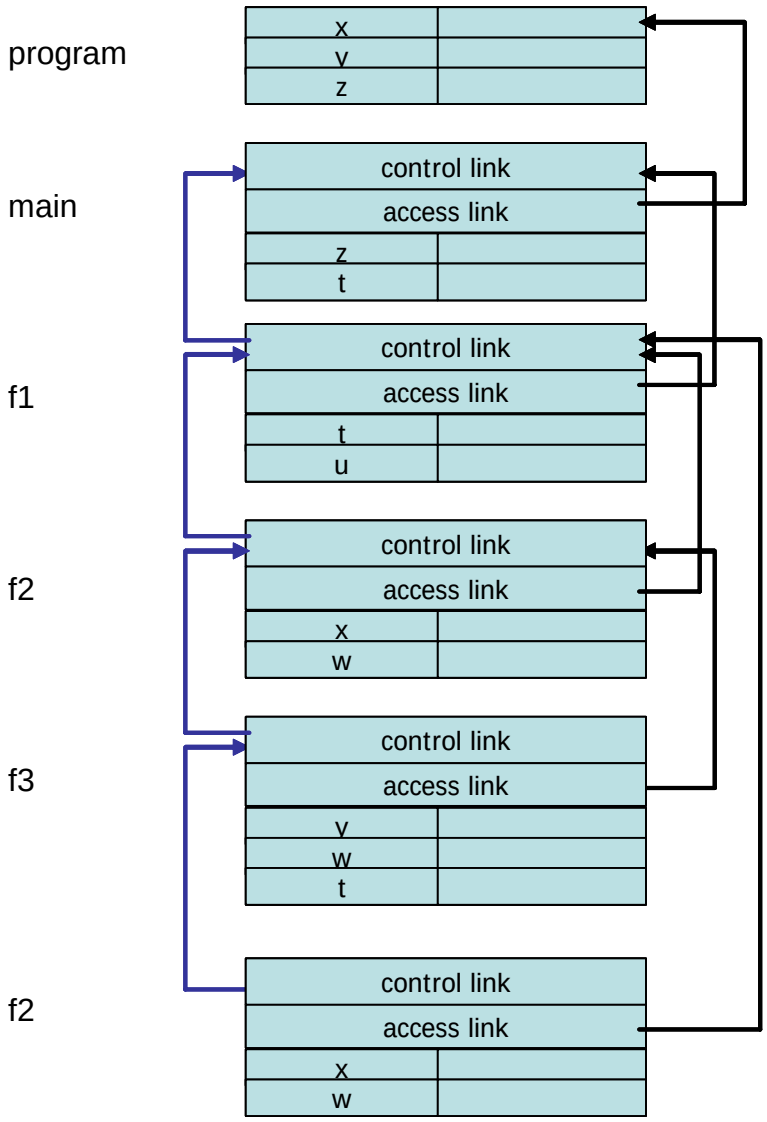
3)



Problem 5

```
f1()in main: ’i’ is bound to ’i’ in main, ’k’ is bound to ’k’ in main
f2()in f1: ’i’ is bound to ’i’ in f2, ’j’ is bound to ’j’ in f1
f2()in main: ’i’ is bound to ’i’ in f2, ’j’ is bound to ’j’ in main
```

Problem 6



Problem 7

a)

The following three cases for proc power (x, y, z: int) will work:

power(x by-value, y by value, z by-reference)

power(x by-value, y by reference, z by-reference)

power(x by-reference, y by value, z by-reference)

while the cases where z is by-value will not work.

Obviously z must be by-reference, otherwise c would not be changed. The power procedure is not a function that delivers a value, so the only way of changing c in power(a,a,c) is by having z by reference. If z is by-value, then z corresponds to a local variable, and power will just changes the value of this local variable and not the c.

power(x by-value, y by value, z by-reference): For this alternative the code in the while-loop computes the value of z based upon local x and y.

power(x by-value, y by reference, z by-reference): For this alternative it may look dangerous that changing y in the loop really changes the value of a, however, with x by-value, the value of a at the time of the call is taken care of in the local x. The x used in the loops will therefore give the right value (the value of a).

power(x by-reference, y by value, z by-reference): This alternative is ok since x is only used on the right hand side of an assignment within the loop. Decrementing y in the loop will just decrement the local y.

Problem 8

	by-value	by-reference	by-value-result
x:= x + 1;	x=2	i=2	x=2
y:= x + 1;	y=3	i=3	y=3
x:= y;	x=3	i=3	x=3
i:= i + 1	i=2	i=4	i=2 result 3

Problem 9

As an example we assume a(i)=1 and a(j)=2. In the following table x and y are only used in the call-by-result case, while for call by reference the addresses are used. For i=j we just use i and thereby a(i)=1.

by reference, not(i=j)	**by value-result, not(i=j)**
a(i) = a(i) + a(j) = 1 + 2 = 3 a(j) = a(i) - a(j) = 3 - 2 = 1 a(i) = a(i) - a(j) = 3 - 1 = 2	x = a(i) = 1 y = a(j) = 2 x = x + y = 1 + 2 = 3 y = x - y = 3 - 2 = 1 x = x - y = 3 - 1 = 2 a(i) = x = 2 a(j) = y = 1
by reference, i=j	**by value-result, i=j**
a(i) = a(i) + a(i) = 1 + 1 = 2 a(j) = a(i) - a(i) = 2 - 2 = 0 a(i) = a(i) - a(i) = 0 - 0 = 0	x = a(i) = 1 y = a(i) = 1 x = x + y = 1 + 1 = 2 y = x - y = 2 - 1 = 1 x = x - y = 1 - 1 = 0 a(i) = x = 0