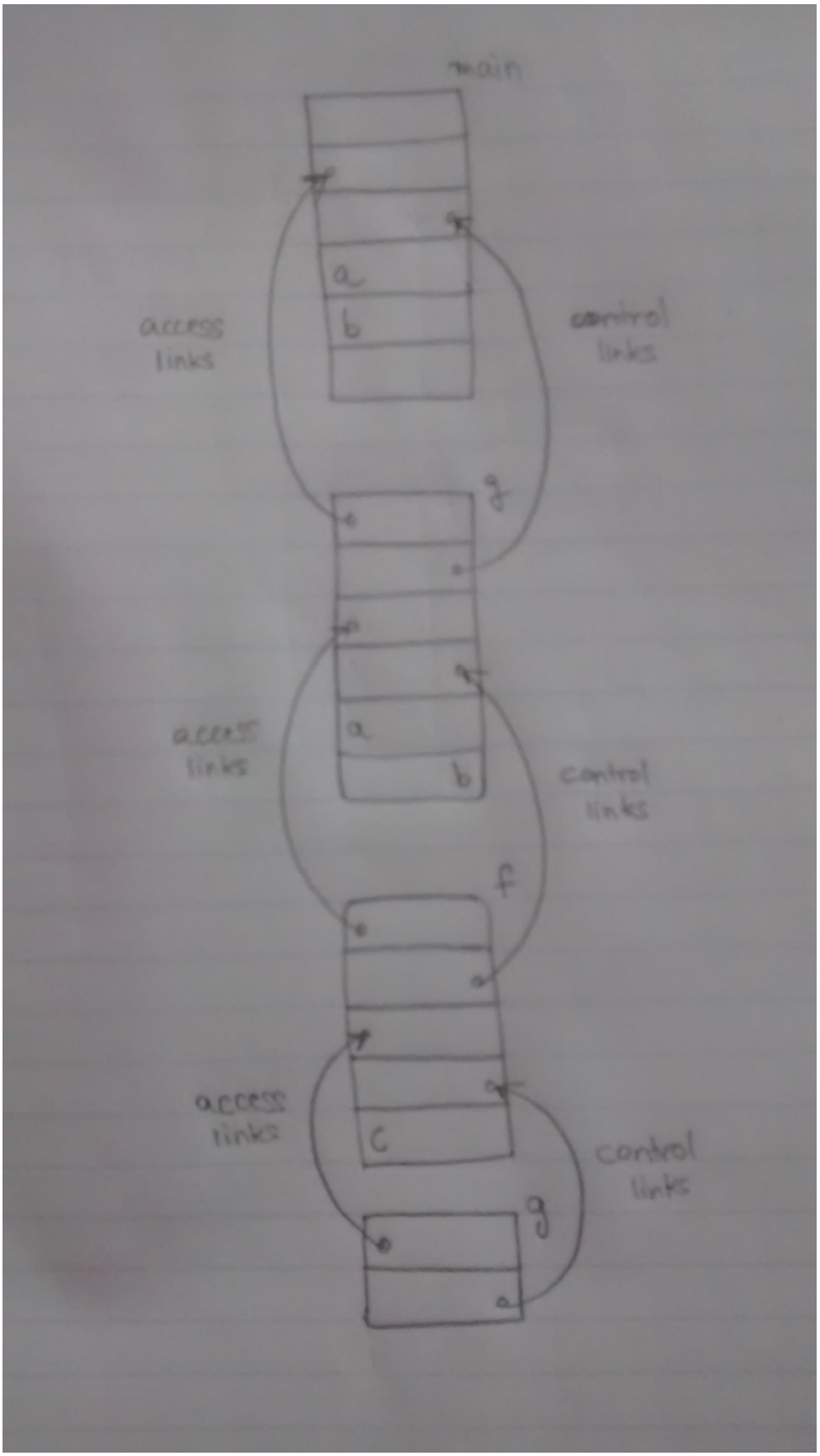
Jessie George
Section 1
HW 5

Problem 1 part 1

See runtime stack on next page


main
a
b
access links
control links
g
a
b
access links
control links
f
C
access links
control links
g

Problem 1 part 2

Explanation because of lexical scoping:
(*A*) store 3+1 = 4 because b=3 and c=1
(*B*) store 3+1 = 4 because b=3 and c=1

RISC code for (*A*):

```
loadI 3 → r1          //because the value of b in the main function is 3
loadI 1 → r2          //because the value of c in the f function is 1
add r1, r2 → r3       // 3 + 1 = 4
```

RISC code for (*B*)

```
loadAI r0,4 → r4      //value of c which is 1
add r1, r4 → r5       //value of r1 is 3 as shown above. So r5 = 3 + 1
```

Problem 2

```
loadI 3 → r1          //constant value 3
storeAI r1 → r0, 4    // z = 3
loadAI r0, -12 → r2   // value of x
loadAI r0, -8 → r3    // value of y
loadAI r0, 4 → r4     // value of z
add r2, r3 → r5       // x + y
add r5, r4 → r6       // (x+y) + z
storeAI r6 → r0, -12  // x = x + y + z
loadI 1 → r7          // constant value 1
storeAI r7 → r0, -8   // y = 1
```

Output is 10, 5.
Explanation:
a = x = x + y + z = 2 + 5 + 3 = 10 (this is because x is an alias for a)
b = 5 (this is because change to y has no affect on b)

Problem 3 part 1

```
procedure foo(integer x)
{
        x = x + 1;
        x = x + a;
}
```

Problem 3 part 2

Call-by-value output is 1 because foo only operates on the local ‘x’ variable but does not change the argument ‘a’ variable.

Call-by-reference output is 4 because ‘x’ is an alias for ‘a’.

Call-by-value-result output is 3 because it does not use aliasing but the result of ‘x’ is copied back to ‘a’ at the end of the function.