CPSC 213, Winter 2010, Term 1 — Quiz 2 **Solution**

Date: Oct 20, 2010; Instructor: Tamara Munzner

1 Consider the following C code.

```
int a[10] = { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55};
int j = *(a+(&a[4]-&a[1]) + *(&a[5]-3));
```

Does the execution of this code generate a runtime error? If not, what is the value of j after the code executes? **Show your work/calculations to justify your answer.**

```
j = *(a+(&a[4]-&a[1]) + *(&a[5]-3));
  = *(a+((a+4)-(a+1)) + *((a+5)-3));
  = *(a+(3)+*(a+2));
  = *(a+3+ a[2]);
  = *(a+3+2);
  = *(a+5);
  = a[5];
  = 8;
```

10 marks total, one mark per transformation:

```
&a[4] -> (a+4)
&a[1] -> (a+1)
&a[5] -> (a+5)
(a+4)-(a-1) -> 3
((a+5)-3) -> a+2
*(a+2) -> a[2]
a[2] ->2
(a+3+2) -> (a+5)
*(a+5) -> a[5]
a[5] -> 8
```

Remember: x[j] = *(x+j), &x[i] = (x+j), pointer arithmetic takes into account the size of the type that is pointed to.

2 Give the SM213 assembly code that a compiler might generate for the function foo in the following C code; include comments. Assume that the location in memory for the variable a is 0x1000. You do not need to write down any .address/.pos specifications, just the assembly language commands.

```
struct A {
    int i;
    int j;
};
```

```
struct A* a;
void foo () {
    a->i = a->j + 3;
}
```

SM213 assembly code with comments.

```
ld $0x1000, r0      # r0 = &a
ld (r0), r0     # r0 = a
ld 4(r0), r1        # r1 = a->j
add $3, r1      # r1 = a->j+3
st r1, 0(r0)        # a->i = a->j+3
```

2 marks per instruction. Half credit (one mark) for an instruction that's close but not quite right.

3 Compare the following two alternatives implementing a similar computation. Recall that "`strncpy (s1, s2, n)`" copies string s2 into strings s1.

```
char* foo () {
    char* x = (char*) malloc (11);
    strncpy (x, "Tra la la!", 11);
    return x;
}

void bar () {
    char* y = foo ();
}
```

```
char* foo () {
    char* x = (char*) malloc (11);
    strncpy (x, "Tra la la!", 11);
    free (x);
    return x;
}
void bar () {
    char* y = foo ();
}
```

Both versions of this code have a different bug. Carefully describe each bug (give its name if you can) and then fix the code.

3a Describe the bug on left-hand side.

The failure to free memory allocated at line 1 of `foo` results in a memory leak.

2 marks total: 1 mark for general idea correct, 1 more mark for specific phrase 'memory leak'.

3b Describe the bug on right-hand side.

Returning the value of `x` in `foo` after the memory it points to has been freed results in the use of a dangling pointer in `bar`.

2 marks total: 1 mark for general idea correct, 1 more mark for specific phrase 'dangling pointer'.

3c Re-write the code so that neither bug is present.

```
void foo (char *x, int n) {
    strncpy (x, "Tra la la!", n);
}
void bar () {
    char y[11];
    foo (y, sizeof(y));
    /* full credit for foo(y,11) or foo(y,10) */
}
```

6 marks total:
+ 1 mark for any attempt to change to memory allocation/deallocation
+ 2 marks for any attempt to allocate memory in bar caller and pass in string to foo callee
+ 2 marks for correct memory management in bar, so that memory for y is available for full scope of bar. best way is static allocation as above, so happens automatically with local variables on the stack. full credit for if dynamic allocation and then explicit free are both in bar.
+ 1 marks for not also allocating memory in foo callee