

CPSC 213

Midterm Practice Questions

Part I

Do all the exercises (2.1 to 2.8) which are listed at the end of Chapter 2 of the CPSC 213 Companion (pages 52 and 53)

Part II

Question 1

Consider the following C snippet. All three variables are global variables.

```
int i;
int a[10];
int * b = (int *) malloc( 10* sizeof(int) );
void main () {
    a[i] = *(b+2);
}
```

Give the Gold Assembly instructions that implement the statement `a[4] = *(b+2)`, assuming that `i`, `a`, `b` have been allocated at locations 1000, 2000 and 3000 respectively.

Question 2

Consider the following Java code:

```
public class Foo {
    static int i;
    static void foo () {
        int j;
        int k;
        bar();
        i=0;
        j=0;
        k=0;
    }
}
```

Give the Gold Assembly instruction(s) for assigning each of the variables `i`, `j` and `k` the value 0. Assume `r5` stores the stack pointer, `r6` stores the return address and `r7` stores the address of the object. Also assume that any static variables allocated by the compiler start at address `0x1000`. Clearly state any other assumptions you make.

Question 3 (*Uses arguments which will be discussed on Monday*)

Consider the following Java code:

```
public class Foo {
    static int i;
    static void foo (int m) {
        int j;
        int k;
        i=0;
        m=0;
        j=0;
        k=0;
    }
}
```

Give the Gold Assembly instruction(s) for assigning each of the variables i, j, k, m the value 0. Assume r5 stores the stack pointer, r6 stores the return address and r7 stores the address of the object. Also assume that any static variables allocated by the compiler start at address 0x1000. Clearly state any other assumptions you make.

Question 4

Consider the following C statements:

```
int x = 5;
int y = 3;
int *p, *q;
p = &x;
q = &y;
y = *p;
p = q;
*p *= *q;
```

Draw a diagram showing the values of x,y, p, and q, after the execution of the above statements.

Question 5 (*Uses arguments which will be discussed on Monday*)

Here is a Gold ISA implementation of a procedure that takes two arguments (we call them arg1 and arg2) and returns a value. Your goal is to write this procedure in C. The procedure has a loop in it and it computes a simple, useful function on its inputs, which it returns. First you should place some useful comments on the side of the assembly statements and then start the translation into C.

```

        add $-4, r5           # create stack frame for 2 local
        add $-4, r5           # variables (local1 and local2)
        ld $0x0, r0
        st r0, 0x0(r5)        # local1 = 0
        st r0, 0x4(r5)        # local2 = 0
loc1:   ld 0x0(r5), r0         # r0 = local1
        ld 0xc(r5), r1        # r1 = arg2
        not r1
        inc r1
        add r0, r1
        beq r1, loc2
        ld 0x8(r5), r1        # r1 = arg1
        ld (4*r0,r1), r1      # r1 = arg1[local1]
        ld 0x4(r5), r2
        add r1, r2
        st r2, 0x4(r5)
        inc r0
        st r0, 0x0(r5)        # local1++
        br loc1
loc2:   ld 0x4(r5), r0
        add $4, r5            # teardown stack frame
        add $4, r5
        jmp 0x0(r6)          # return

```

Question6

Consider the following Java code:

```

Class Polygon {
    int  noSides;           //number of sides
    float[]  side;         // a dynamic array with the length of each side

    Polygon( int n ) {
        noSides = n;
        side = new float[n];
    }

    setSide(int s, float length) {
        if ( 0 <= s && s < noSides )
            side[s] = length;
    }

    . . .
}

```

Translate this code to C following the ideas we have discussed in the class. The code of a constructor for a Java class is represented in C by a regular function which creates a dynamic object and returns the address of that object. We usually name these functions with a name like `creat<object type>`, where `<object type>` is the data structure name that defines the object type. For instance the C function for the polygon's constructor may be called `createPolygon`.