CSE505 – Fall 2012
Assignment 1 – Procedural Languages
(Sample Solution)

1 [20%]. Pseudo-random number generator exploiting history-senstive behavior of Fortran 77.

```
INTEGER FUNCTION RAND()
INTEGER SEED/13/, R
R = MOD(13*SEED+1, 65536)
SEED = R
RAND = R
RETURN
END
```

2 [40%]. The sequence of values printed is 125, 100, 200. The contour diagram is shown on the next page.

3 [20%] Translation of:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} (\mathrm{b[i,j]} * \sum_{k=1}^{n} \mathrm{c[i,j,k]})$$

```
int thk1() { return c[i,j,k]; }
int thk2() { return b[i,j] * sigma(thk1,k,1,n); }
int thk3() { return sigma(thk2,j,1,n); }
```

Top-level expression: sigma(thk3,i,1,n)

4 [20%]. C program:

```
#include <stdio.h>
int* foo () {
    int a[3] = {1,10,100};
    return a;
}
int* goo () {
    int b[3] = {2,20,200};
    return b;
}
```

```
int main() {
    int *a, *b
    a = foo();
    b = goo();
    a = printf("%d\n", *a);
    return 0;
}
```

Output produced by program is: 2

C uses assignment-by-sharing and quasi-dynamic object allocation. As explained in the lectures, this combination is not always safe, and the above program illustrates the problem.

The array objects in **foo** and **goo** are allocated by quasi-dynamic allocation, i.e., on the run-time stack. When $\mathbf{foo}^{1}$ returns back to $\mathbf{main}^{1}$ the variable **a in** $\mathbf{main}^{1}$ points to the array object for **[1,10,100]** that has just been deallocated. This is effectively a dangling pointer. When $\mathbf{goo}^{1}$ is called, its stack frame over-writes that of $\mathbf{foo}^{1}$ and thus the array object for **[2, 20, 200]** over-writes the array object for **[1,10,100]** on the stack. Thus, when $\mathbf{goo}^{1}$ returns back to $\mathbf{main}^{1}$ the variable **a in** $\mathbf{main}^{1}$ now points to **[2, 20, 200]**. Hence the print statement outputs 2.

2 [40%]. The sequence of values printed is 125, 100, 200. (The contour diagram below also shows the call to D^1, for completeness, although this was not asked for in the assignment.)