CSE305, Fall 2007 **Assignment 5** **Due Wed. Oct. 3**

Includes the revised directions for Assignment 4

Reading: For Friday, also read Sebesta Chapter 10, except section 10.6 on implementing dynamic scoping. This is a short chapter, and actually fits hand-in-hand with the chapter 5 material. For next week, read the "Stack Language" handout and begin reading Chapter 6.

(1) For Friday 5pm, you must submit online a working C# translation `Dispatching.cs` of `Dispatching.java`. You should compile it both without `virtual` and `override` used on the `Foo` method in class `Base` and the version in class `Derived` that can override it, and with them. You should submit the version in which `override` is present. (Both versions must have `override` applied to the `ToString()` methods, since they are already `virtual` in the C# library.) This is worth 45 pts. The report questions, worth 45 pts. and now due **in hardcopy only** on `Wed. 10/3`, are still TBA (sorry!), but will follow the outline given before.

(2) Diagram the four storage objects `p`, `q`, `x`, and `y` declared in the following C program, and show the changes to them during its execution. Use `2310,2312` as the binding addresses of the pointers `p` and `q`, and `3418,3420` for the binding addresses of the integers `x` and `y`, respectively. Give the final values of all four storage objects, not just `y`, and show your work clearly. (18 pts.)

```
#include "stdio.h"
void main() {
        int x, y, *p, **q;
        x = 5;
        y = x+3;
        p = &x;
        q = &p;
        (**q) = y + x;
        (*q) = &y;
        (**q) = y - x;
        printf("Final value of y is %d\n",y);
}
```

(3) Consider the Ada program at left, *or* if you prefer, consider the C program at right, which is *completely equivalent* for this question.[1]

```
with Ada.integer_text_io; use Ada.integer_text_io;
with text_io; use text_io;
procedure G is
   x: integer := 2;
   z: integer := 4;

   function A(y: integer) return integer is
      x: integer := 10;
   begin
      return x + y + z;
   end A;
```

```
#include <stdio.h>
/*Global variables*/
int x = 2;
int z = 4;

int A(int y)
{
   int x = 10;
   return x + y + z;
}
```

[1] Nowadays, not saying `int main()` in C gives a warning, but let's ignore that.

```
   function B(y,z: integer) return integer is
   begin
      x := 8;
      return A(x + y + z);
   end B;

   procedure M is
      y: integer := 1;
      z: integer := 3;
   begin
      put("B(x+y,z) = "); put(B(x+y,z));
      put(" and x = "); put(x); new_line;
   end M;
begin  --of G
   M;
end G;
```

```
int B(int y, int z)
{
   x = 8;
   return A(x + y + z);
}

void main()
{
   int y = 1;
   int z = 3;
   printf("B(x+y,z) = %d ",
           B(x+y,z));
   printf("and x = %d\n",x);
}

/* end-of-file */
```

(a) There are three referencing environments in which the identifiers `x`, `y`, and `z` *occur* in expressions, namely `A`, `B`, and `M` (or "`main`" in the C code). Those three blocks are nested inside the outer block `G` (which corresponds to "global file scope" in the C code). For each of the 9 *occurrences* of `x`, `y`, and `z`, say *which* block has the declaration that binds that occurrence.[2] (9 pts. total)

(b) Trace the execution of the program, showing the sequence of stack frames and activities including assignments going on inside them. What final values of `B(x+y,z)` and `x` are printed? (18 pts., for 27 on the problem and 90 on the set, including the 45 pts. for the Assignment 4 report.)

[2]For instance, if `w` were declared in `G` and in `A`, and occurred in `A` and `M`, then the occurrence in `A` would be called "`A.w`" as wour answer, while that in `M` would be "`G.w`." You should have 9 such answers, preferably in a 3×3 grid with rows labeled `A`, `B`, `M` and columns labeled `x`, `y`, `z`.