CSE305, Spring 2023 Assignment 5 Due Sun. Apr. 16, 11:59pm

The **Second Prelim Exam** will be on **Thursday, April 27** in class period. It will cover material cumulatively through chapter 12 of the text and assignments 1–6 (esp. 4–6).

Reading:

For this week, read Sebesta chapters 9 and 10 as one unit. Pay closest attention to sections 9.4–9.6 and the diagrams in sections 10.3–10.6 (can read 10.4–10.6 later). Also read section 11.7 on namespaces—because we are using this benefit of modules in OCaml. The previous parts of chapter 11 should be largely review for you coming out of CSE250 or equivalent course where the value of implementing data structures via ADTs was emphasized.

———-**Assignment 5**, due **Sun. 4/16** "midnight stretchy"————

(1) The *TopHat portion*: 13 questions worth 1 or 2 points each, totaling **20 pts.** They are organized into one assigned document **CSE305S23A5**, and answers are automatically recorded and revealed as you work through it.

The rest is to be submitted as **a single PDF file** via *CSE Autograder*, with the code in problem (2) also submitted on `timberlake` via `submit_cse305 CSE305ps5NN.ml`.

(2) (36 pts. total)

Augment the `'a exp` datatype (inside modules—here you may keep the names `CE` and `SL` from recitations or use your own names) to include the following variants:

- `Assign` to represent assignments inside expressions.
- `PreInc` and `PostInc` for pre-increment and post-increment, respectively. (Adding support for pre-decrement and post-decrement via `PreDec` and `PostDec` is optional.)
- `ArrayEntry`. Whereas the above will take two, respectively one, `'a exp` arguments, this one is OK to limit to a `string` argument for the name of the array and an `int exp` argument for the index. *Note the fix to the translation of* `arr[i]` *now shown on the last page of the Thu. 3/31 notes and first page of the Tue. 4/4 notes.*

Then rewrite your translation code to handle them. Note that all three will require some way to distinguish *lvalue* from *rvalue* in the tree—but the final stack code output does *not* have these labels. You may find it expedient to add a variant `LVar` to distinguish *lvalue* uses of variables, and maybe do similarly for array entries (and `PreInc`?). Or you may find that bumping up `pcompile` to use a two-level match or extra recursion parameter may do the trick.

Using modules around the extensible datatypes is required, and the type of the stack code entries can just be `'a token`, not `'a ptoken`. Please include an augmented version of the function that converts from the `'a token list` output into a simple string of the stack code.

(3) (18 pts. total)

Compile the following loop code into our rudimentary stack language, using the scheme with four stack entries and a "`jmpifeq`" operation (which you can use more than once) sketched toward the end of the Thu. 4/6 lecture. State and/or diagram the semantics of how `jmpifeq` changes the stack and instruction counter (IC) when the test fails, and when it succeeds.

```
double sum = 0;
for (int i = 0; i < n; i++) { sum += arr[i]; }
```

(4) (27 pts. total)

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;

   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;
```

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

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

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 101 on the set.)

---

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

[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`.