CSE305, Spring 2023    Assignment 4    Due Thu. Apr. 6, 11:59pm

Reading:
First, review the [notes] for the Thursday 3/30 lecture and try the examples in them for yourself. Also read Chapter 15, section 15.1 (only) of the previously-noted OCaml-based book by Stuart Schieber [https://book.cs51.io/pdfs/abstraction.pdf], which includes some compare-contrast of pointers in C/C++ and references in OCaml. (The short page http://wide.land/mut/refs.html may be good to read first, but the examples of ref on the official OCaml site and some others go in directions that are less useful right now.) Then also for next week, read chapters 8 of Sebesta and look ahead to chapters 9 and 10.

—Assignment 4, due Thu. 4/6 “midnight stretchy” only on CSE Autograder (no TopHat)—

(1) (9 + 9 + 12 = 30 pts.)
Write expression trees for the following arithmetical and/or logical expressions. Then convert them into postfix notation. For (b) and (c) you are welcome to use the alternative C/C++/Java notation both in the tree nodes and in the postfix. Do not simplify them.

(a) 4 + (x+y)*(x-y)
(b) (x ∨ y) ↔ ¬(y ∧ z). Or in C notation: (x || y) == !(y && z).
(c) y = if a+b <= 3 then 3 else a+b, or in C notation: y = (a+b <= 3 ? 3 : a+b)

(2) (15 + 15 = 30 pts. total)
Diagram the storage objects and trace their changes in value during the execution of the following C program with pointers. (You can refer to section 6.11.4 of Sebesta; the line p = &z; means that the pointer p gets as its value the binding address of the ordinary variable z. Also you are welcome to compile and run the program to check your work.)

```c
#include<stdio.h>
int main() {
    int w,z;
    int *p; int *q;

    w = z = 15;
p = &z;
    q = p;
    *p = (*q) + w;
    w = 3*z - (w + *q);

    printf("w = %d and z = %d and *p = %d and *q = %d\n", w,z,*p,*q);
    return 0;
}
```
Then translate the five lines with assignments into our rudimentary stack-based language, using store and pop as well as fetch and the postfix operations.
(3) (18 pts.)
Recall the OCaml datatype for which we drew a parse tree in Assignment 2, but now without Parens and extended with Times and Div options, hence simply called exp:

```ocaml
type 'a exp = Const of 'a | Var of string
  | Neg of 'a exp
  | Plus of 'a exp * 'a exp
  | Minus of 'a exp * 'a exp
  | Times of 'a exp * 'a exp
  | Div of 'a exp * 'a exp
```

Our target will be a datatype for the stack commands. For now, we can use

```ocaml
type 'a ptoken = PConst of 'a | PVar of string
  | PNeg | PPlus | PMinus | PTimes | PDiv
  | Fetch | Store | Pop
```

Write a function `pcompile` of type `'a exp -> 'a ptoken list` that executes a postorder traversal on the expression tree that is given and outputs the resulting list of postfix tokens. This is not yet implementing assignments or other mutation, so `Store` and `Pop` will be unused, and every `PVar str` that you output will be an `rvalue` so you can automatically follow it by `Fetch` in the output list.

(We will extend both datatypes later—and will encounter some software engineering issues along the way. One issue is immediately apparent: since we don’t (yet) have encapsulation or namespace guarding, we shy away from using the same constructor names `Const`, `Var`, `Neg` etc. in the `ptoken` type. You can read the extra ‘P; as standing for “push” in the first two cases and for “postfix” on the operators.)

(4) (12 pts.)
The end of the Thu. 3/30 lecture gave the translation of `++x` as

```
x x fetch 1 + store
```

A paragraph marked “Added” to the last page of the posted lecture notes notes at [https://cse.buffalo.edu/~regan/cse305/CSE305Week8Thu.pdf](https://cse.buffalo.edu/~regan/cse305/CSE305Week8Thu.pdf) points out an issue that was also noted in the two translations of `arr[i]`. The translation can change when the element is in an `lvalue` context from when it is an `rvalue`. Consider the following C statements relating to an example earlier in the lecture:

```
int x,y;
x = 3;
y = ++x;
y = ++(++x);
```

Show that `y = ++x;` works fine with the above translation, but `y = ++(++x);` breaks down. Show the expression trees for each with the ‘R’ and ‘L’ labels for `rvalue` and `lvalue` at each node, and explain the issue as best you can. If you wish, you may suggest a possible “switch-hitting” translation for `pre++` that will generalize and work in the latter case, for up to 9 points possible extra credit. (This makes 90 regular credit points total, plus the possible 9.)