# CSE305, Spring 2023 Assignment 2 Due Sun. Mar. 5, 11:59pm

**Reading:**

For Thursday (March 2) into next week, read Sebesta Chapter 5. Pay special attention from subsection 5.4.3 onward. Thursday will also mark a pause in the emphasis on OCaml alone, as these concepts apply to multiple programming languages. Accordingly, the emphasis on the text itself will be technical again, as it was with grammars in section 3.3.

Looking ahead, chapters 6 and 7 will feel half like a review since we've focused on list types already in OCaml and on *expressions* fairly generally, not just arithmetical. They have other material too, but the quasi-review apsect will be helpful in the days leading up to the first prelim exam. Thus although the last assignment (maybe in two parts) before the exam will be a mix of Chapter 5 material and some programming, material from chapters 6 and maybe 7 may be deemed relevant to the exam too.

—-**Assignment 2**, due Sun. 3/5 with separate parts on *TopHat* and *CSE Autograder*—-

**(1)** The *TopHat portion*: 14 multiple-choice questions worth 1 or 2 points each, totaling **20 pts.** They are organized into one assigned document, and answers are automatically recorded and revealed as you work through it. Scores from it will eventually be ported to *Autograder* and later everything to *UBLearns*, which is being used only as a gradebook.

The rest is to be submitted as **a single PDF file** via *CSE Autograder*, plus `submit_cse305 CSE305ps2NN.ml` with your answers to problem (3), where you substitute your initials for the NN. Problem (3) will thus be submitted both ways.

**(2)** (a) Draw a parse tree from the derivation on pages 3–4 of the posted Week 4 course notes (https://cse.buffalo.edu/˜regan/cse305/CSE305Week4.pdf) of the OCaml program

```
let rec sumList ell = match ell with
     [] -> 0
   | x :: rest -> x + sumList rest
```

Use the nonterminals given in the derivation as the interior nodes in your tree, with `DEF` at the root. Follow the rules as given in my notes, which shortcut the way rules are given on official OCaml sites.

(b) Then draw a parse tree for the additive expression type on page 18 of the same notes:

```
type 'a addExp = Const of 'a | Var of string | Parens of 'a addExp
  | Neg of 'a addExp
  | Plus of 'a addExp * 'a addExp
  | Minus of 'a addExp * 'a addExp;;
```

You will need to reference the rules for `TYPEXP` as given on page 13 of the week 3 notes, noting also `TYCON ::=` *the-basic-types* | `list` | *lots-of-other-stuff.*

```
TYPEXP ::= '<ident> | _ | (TYPEXP)   ( 'a is like template <A> in C++/Java )
 | TYPEXP -> TYPEXP                  (function type, associates to the right)
 | TYPEXP { * TYPEXP }+              (tuple type, no left/right handedness)
 | [TYPEXP] TYCON                    (simple example: int list)
 | (TYPEXP {, TYPEXP}) TYCON
```

You are welcome to derive `<ident>`, `<alphaid>`, `<lcalphaid>`, and `<Ucalphaid>` straight to an appropriate identifier in one step. Notice the literal apostrophe in the first rule for `TYPEXP`, besides its use in `GENID`. Again, the root will have `DEF`. In the rule for `TYPEREP` you will use the BNF braces at "warp 5" to make 6-fold overall branching, which may force you to be creative in fitting the tree on one page (but the shortness of what follows may help). (12 + 18 = 30 pts.)

**(3)** *Two functions in OCaml.*
We ask you to submit them both "on paper" in this PDF document and by online submission using `submit_cse305` on `timberlake`. (The former can even be handwritten; of course you will have to type the latter.) Your file for the latter should be named `CSE305ps2`NN`.ml` where you substitute your initials for the NN. It should also have a `(** ...` "header comment" at the top giving the filename, your name, and some brief usage notes.

(a) Write a function `uniq` that when applied to a list `ell` of any type removes duplicates of consecutive equal elements and outputs the resulting (often shorter) list. Equal elements that are not consecutive can be left as-is. A highly representative test case is:

```
uniq [1;7;7;7;3;2;2;3;7;7;1;7] = [1;7;3;2;3;7;1;7]
```

Of 15 pts., 3 pts. are for making this work with lists of any type, not just integers.

(b) Consider lists `ell` of pairs $(a, b)$ of nonnegative floats where always $a < b$. Think of $a$ and $b$ as scheduled start and end times of an appointment or processor job or whatnot. Say that two scheduled items $(a_1, b_1)$ and $(a_2, b_2)$ **conflict** if $a_1 \leq a_2$ but $a_2 < b_1$, so that the latter starts before the former has ended.

Write a function `hasConflict` that determines whether `ell` has any conflicts. You may assume that `ell` is already sorted by the first components $a$. (Later labs will show how to hook this up with `mergesort` from this week's labs to do without this assumption.) If none, return `None`, else return `Some` together with a pair of items that conflict—notice this will be a nested tuple of tuples. On your hardcopy submission, write down the type of the `hasConflict` function you got. (15 pts., making 30 on the problem and 80 pts. on the whole assignment)