COL226: Programming Languages
II semester 2021-22

Assignment: Abstract Syntax Trees (AST) for WHILE

This assignment is meant to get you to convert programs into abstract syntax trees. **This assignment is compulsory for the next assignment which will require your implmentation of this assignment!**

Here are some preliminary instructions.

- The EBNF of the WHILE language is provided in section 4.6 of the Hyper-notes. You will notice that it has a strange mix of operators from Pascal, C and some disambiguating bracketing symbols borrowed from elsewhere.
- The language of implementation is SML and you may freely use tools like ML-Lex, ML-Yacc and/or ML-Antlr. You may decide to write a recursive descent parser instead of using ML-Yacc or ML-Antlr. Your main file should be called `while_ast.sml`
- Assume that the WHILE language uses *declaration before use* and hence all identifiers need to be declared before they are used.
- The semantics of the language should be clear and self-explanatory from your experiemce of other imperative languages. In particular, the functions "/" and "%" should satisfy the equation and condition[1] for all integers a and $b \neq 0$.
$$a = b * (a/b) + (a\%b), 0 \leq |(a\%b)| < |b|$$
- You need to provide documentation in the form of a `README.md` file which which may be viewed on a browser. (`md` stands for `markdown`. Check out `https://www.markdownguide.org/cheat-sheet/`).
  - You are *not* free to take decisions (such as precedence and associativity of operators) which contradict what has already been specified, or against normal mathematical convention as usually understood in programming languages like C, C++, Java (for the boolean operations), Pascal, ML, Modula etc.
  - This file should contain a clear specification of the context-free grammar that you are using for parsing in a section with the heading "**Context-free grammar**"
  - Define the data-type `AST` of WHILE programs in a separate section with the heading "**AST datatype definition**". For this purpose use the following constructor names (should be obvious what operators/constructors they denote, since the names are very suggestive!).

    Program. `PROG`

    Types. `INT`, `BOOL`

    Boolean operators. `TT`, `FF`, `NOT`, `AND`, `OR`

    Relational operators. `LT`, `LEQ`, `EQ`, `GT`, `GEQ`, `NEQ`. The relational operators are also applicable to comparisons boolean expressions with *false* being strictly *less than true.*

    Integer operators. `PLUS`, `MINUS`, `TIMES`, `DIV`, `MOD`

    Commands. `SET` (for assignment), `SEQ` (for sequencing), `ITE`, `WH`

    Other auxiliary datatypes. If needed you may define other (possibly recursive) auxiliary datatypes (with the given names) for

    Blocks. `BLK`

    Declarations. `DEC` (for list of variable declarations along with their types e.g. `INT(x)`, `BOOL(b)`).

    Commands. `CMD`

    Integer expressions. `IEXP`

    Boolean expressions. `BEXP`

[1] Many programming languages do not consistently follow this definition when $b < 0$; instead they blindly give you a quotient and remainder!

- This file should also contain a clear specification of the semantic rules (in a separate section called **Syntax-directed translation**) that you use in your implementation along with all the extra auxiliary functions and data-types (specified in a separate section called "**Auxiliary functions and Data**") that you use in your implementation.
- Any other design decisions that you take will have to be documented in this file clearly under a separate heading called "**Other Design Decisions**".
- Any other implementation decisions you take will have to be documented in this file clearly under a separate heading called "**Other Implementation Decisions**".
- A final separate section called "**Acknowledgements**" with details of what you copied from where or whom.

Note for all assignments in general

1. Some instructions here may be overridden explicitly in the assignment for specific assignments
2. Upload and submit a single zip file called `<entryno>.zip` where `entryno` is your entry number.
3. You are *not* allowed to change any of the names or types given in the specification/signature. You are not even allowed to change upper-case letters to lower-case letters or vice-versa.
4. The evaluator may use automatic scripts to evaluate the assignments (especially when the number of submissions is large) and penalise you for deviating from the instructions.
5. You may define any new auxiliary functions/predicates if you like in your code besides those mentioned in the specification.
6. Your program should implement the given specifications/signature.
7. You need to think of the *most efficient way* of implementing the various functions given in the specification/signature so that the function results satisfy their definitions and properties.
8. In a large class or in a large assignment, it is not always possible to specify every single design detail and clear each and every doubt. So you are encouraged to also include a README.txt or README.md file containing
   - all the decisions (they could be design decisions or resolution of ambiguities present in the assignment) that you have taken in order to solve the problem. Whether your decision is "reasonable" will be evaluated by the evaluator of the assignment.
   - a description of which code you have "borrowed" from various sources, along with the identity of the source.
   - all sources that you have consulted in solving the assignment.
   - name changes or signature changes if any, along with a full justification of why that was necessary.
9. The evaluator may look at your source code before evaluating it, you must explain your algorithms in the form of comments, so that the evaluator can understand what you have implemented.
10. Do *not* add any more decorations or functions or user-interfaces in order to impress the evaluator of the program. Nobody is going to be impressed by it.
11. There is a serious penalty for code similarity (similarity goes much deeper than variable names, indentation and line numbering). If it is felt that there is too much similarity in the code between any two persons, then both are going to be penalized equally. So please set permissions on your directories, so that others have no access to your programs.
12. To reduce penalties, a clear section called "**Acknowledgements**" giving a detailed list of what you copied from where or whom may be be included.