26th Sep 2016

AST– Clite

Compilers 2016 Assignment 3 | **Deadline: Friday, 07-10-2016, 11:59 p.m.**

C-lite Grammar

The primary task in this module is to build the abstract syntax tree (AST) for the parsed program. Once we have the AST, we can do a variety of operations on it, including traversals(DFS/BFS), evaluations and conversions.

You have complete freedom on how you build the AST, but **your output must match the given output** (which can easily be ensured using a proper traversal of your AST).

Output to stdout:

- "Success" on a successful parse
- "Syntax Error" in case of an error

On a successful parse, make an output files XML_visitor.txt.

The table below contains a mapping between rules of the grammar and the output to be printed by their corresponding AST nodes. You need to implement the AST only for the subset of the C-lite grammar mentioned in Assignment 2.

Output file: XML_visitor.txt

Output	**Rule Found**
<program> <declarations count="n"> ... </declarations> <statements count="m"> … </statements> </program> n : number of declarations m : number of statements	Program -> int main() { Declarations Statements }
Normal <declaration name="x" type="t" /> x : Name of the variable t : type ("integer" / "boolean")	Declaration -> Type Identifier
Array <declaration name="x" type="t" size="n" />	Declaration -> Type Identifier [Integer]

<table>
<tr><td>n : Number of elements</td><td></td></tr>
<tr><td><assignment>
<LHS name="x">
</LHS>
<RHS>
..expr..
</RHS>.
</assignment></td><td>Assignment-> Identifier = Expression ;</td></tr>
<tr><td><assignment>
<LHS name="x">
..expr..
</LHS>
<RHS>
..expr..
</RHS>.
</assignment></td><td>Assignment-> Identifier [Expression]= Expression ;</td></tr>
<tr><td><binary_expression type=”x”>
… left hand expr …
… right hand expr …
</binary_expression></td><td>Expression -> Expression bin_op Expression
<table><tr><th>Bin_op</th><th>x</th></tr><tr><td>+</td><td>Addition</td></tr><tr><td>-</td><td>Subtraction</td></tr><tr><td>*</td><td>Multiplication</td></tr><tr><td>/</td><td>Division</td></tr><tr><td>%</td><td>Modulus</td></tr></table></td></tr>
<tr><td><integer value=”n” /></td><td>Factor -> IntegerLiteral</td></tr>
<tr><td><boolean value=”true/false” /></td><td>Factor -> BooleanLiteral</td></tr>
<tr><td><identifier name="x" /></td><td>Factor -> Identifier</td></tr>
<tr><td><identifier name="x">
..expr..
</identifier></td><td>Factor -> Identifier [Expression]</td></tr>
<tr><td></td><td></td></tr>
</table>

Submission Format:

Compress a) the flex code (named Assignment3.l), b) the bison code (named Assignment3.y), c) other files, d) a readme file, e) a executable (named Assignment3), and f) a makefile and upload the zip file. The output file generated must be as specified above.

The zip file should be named rollno_Assignment3.zip.