Compiler Design Homework 3
Due Before Week 4 & 5 Classes

Compiler Project (Due Week 4)

- The compiler project will use the C– definition from Louden Appendix A. An electronic version called ***BNF-Syntax-for-C-Minus.doc*** is provided in Blackboard.
- You should produce a **Lexical Scanner** and **Parser** for the **Cminus** language using **ANTLR4** tool. The grammar name should be Cminus, and it should be placed in a file called Cminus.g4. I recommend using the java based scanner and parser, since they provide a quicker edit-compile-test cycle. (and please use '_' instead of the '-' characters in the grammar names.
- For Week 4 generate an ANTLR4 **token file** and an ANTLR4 **tree file for each** of the four C-Input-#.txt files.
- In your **zip** file include **readme** file, your **grammar file** and a directory called **CompilerProject** with **all the java files** and **class files** that ANTLR4 produced. Also include your **input** and **output** files. **Submit results to Blackboard before the Week 4 Class**.

Add Recursive Descent Parser to Spreadsheet Project (Due Week 5)

- In Blackboard is an **upgraded** Spreadsheet skeleton. This version includes:
  - **Node class** for building an **abstract syntax tree** (**AST**)
  - The **Node class** will include the **token class** to make the tokens into AST nodes.
  - Each spreadsheet cell will be modified to have an expression root node <$> pointer which when not null must point to an <equation> node.
  - Each cell has been modified to **track** its list of **controllers** and its list of **users**. So the left to right and top to bottom order is no longer a constraint. Cells now maintain their controller and user lists and recalculate immediately when modified.
- The **scanner** must recognize:
  - The symbols: **'+' '–' '*' '/' '(' ')' '#'**
  - digit **=** [**0-9**]
  - **<ID>** = [**A-F**](digit)
  - **<NUM>** = (digit)+
- **Note** that the <**NUM**> token in the EBNF is a **positive whole number** (includes zero) which is not the same as the integer number in a NUM cell because the NUM cell can have a minus sign. The token scanned in an equation line only takes positive numbers. The <equation> does permit subtraction, so the resulting value can be the same. This is to avoid the ambiguity of an equation like **{** B1 = A4–1 **}** which might be parsed as an error if the minus sign was scanned as part of a <NUM> token.

- Build a **recursive descent parser** according to the following EBNF

  <$> => <equation>
  <equation> => <term> **{** <add-op> <term> **}** // **remember rotation**
  <add-op> => '**+**' | '**–**'
  <term> => <factor> **{** < mult-op> <factor> **}** // **remember rotation**
  <mult-op> => '*****' | '/' // **note this is integer division**
  <factor> => <**NUM**> | <**ID**> | <paren-exp> ~~| <func-call>.~~
  <paren-exp> => '**(**' <equation> '**)**'
- Your parser should produce an **AST** from the equation entered and attach the root node to the appropriate cell. (See example in Louden Appendix B)
- If an equation is replaced, the nodes of the old tree must be properly deleted.
- Print out the spreadsheet as you did for last week's homework and add a print out of each cell with its attributes: include at least a list of controllers, a list of users, and the AST Node attributes. (A version of this is available in the skeleton spreadsheet, but if you make any additions to the **SS_Cell**, **Token** or **Node** types, you must also update the code that prints out the cell attributes and AST Node attributes.)
- The spreadsheet skeleton now keeps track of lists of all input cells (**controllers**) and cells that use the equation (**users**) as cell attributes, and it tracks these attributes in every cell type including empty cells. When a cell becomes an equation cell or when an equation changes, the skeleton provides a new controllers list to the update routine. When an equation cell changes kind, the skeleton provides an empty controllers list to the update routine. The update routine in the spreadsheet skeleton will use the old controllers list to remove this cell from the users list in each of the old controller cells and then uses the new controllers list to add this cell to the users list in each of the new controller cells. After the update routine has updated the lists, the equation is calculated and the users list is followed to recalculate the tree of descendant users. If the original cell is found in the tree of descendant users, a loop of calculations exists in the spreadsheet. That cell and all of its descendants will be labeled as **ERROR** cells. (The update routine will stop looping around the error nodes after they are marked **ERROR**. That prevents an infinite loop.)
- In **addition** to the output requested last week, you should output the **users list** and the **controllers list** for each cell (an empty list will specify EMPTY). Also the full **abstract syntax tree** for each equation cell should be displayed in outline form where each child link is indented one level from its parent link. Use a depth first display order starting at the root of the parse tree for each equation cell. Include the name of the node and any attributes (like type and value) that apply to the node. (This is provided by skeleton.)
- The display of output values as a table (last week's output) should precede the output of cells and their users, controllers and parse trees.

- In **summary**, most of your HW3 work will be in the following files (look for **TBD**s):
    - **Scanner.cpp** (mostly a copy of your work for HW2)
    - **Parser.cpp** and its recursive descent buddies**.**
    - **SS_Cell.cpp** (the ostream& << Node& routine if you have any new attributes for a token or an AST node)
    - **Node.cpp** (printCellAttributes routine if you have any new attributes for a cell)
- Use the input file **HW3-Input.txt** to generate the output file to be handed in. Other input files (e.g. HW3-Test-1.txt) are available for your use if you want to use them.
- **Submit** a **zip** file containing a readme file and the directory with **all the code** you have written for the spread sheet, **the executable** (identify the OS in the readme), and the output file your code generated. **Submit your zip file before the Week 5 class.**

Read Louden Chapter 4.2–4.2.4 (LL(1) Parsing), 4.3-4.3.2 (Firsts and Follows) 4.5-4.5.2 (LL(1) Error Recovery) and 5.0-5.2.1 (LR Parsing and Items), 5.3.1–5.3.2 (Conflicts), (5.4-5.4.3 LALR Parsing optional), **5.7.1-5.7.2 (LR Error Recovery)**

Go to the **About The ANTLR Parser Generator** and **read** the **Getting Started** and **Documentation** links referenced on that page.

Read "*The Definitive ANTLR4 Reference*" 2.0-2.2 (Parsing Overview).

Optional Reading "*Language Implementation Patterns*" 1.0-1.1 (Language Patterns Overview)

Submit all results for the Compiler Project to Blackboard before the Week 4 Class.

Submit all results for the Spreadsheet Project within two weeks to Blackboard before the Week 5 Class.