**Stage-1**

Note: This file contains the requirement for the stage-1 of the assignment. You may download other files (tokens.docx/pdf, natural.docx/pdf, LL1.docx/pdf, errata.docx/pdf) before you proceed to read. Also note that you can report (by email) if any discrepancies found with the subject line “discrepancy – stage# - PLCC assignment”. End of note.

Date of posting : 18 – Jan – 2011

Deadline : 13 – Feb – 2011

Deadline with lifeline : 14 – Feb – 2011

Requirement for Scanner

The list of tokens that the scanner recognizes is given in the file “tokens.doc” file. Our language reserves all the key words that can appear in the language.

* Requirements Specification:
  + Input: Program File
  + Output: Token Stream
  + Side Effects: White spaces removed
  + Exceptions: Invalid tokens
* Interface Requirements:
  + void initializeScanner(char \*filename); // initializes the scanner
  + Token nextToken(); // should scan and return the next token
  + void printTokenStream(FILE f); // prints out the token stream for a given program
* Files
  + Interface file : lexer.h
  + Implementation : lexer.c

Requirement for Parser

The natural grammar and the LL(1) grammar can be found from files natural.docx/pdf and LL(1).docx/pdf. The outline of the language features are as described in the classroom

* Requirements Specification:
  + Input: Program File
  + Output: Parse Tree
  + Side Effects: none
  + Exceptions: Syntax Errors (Halt at the first error, printing the error description. Error recovery mechanism is desired but not a part of requirement)
* Interface Requirements:
  + void initializeParser(char \*filename); // initializes the Parser
  + ParseTree Parse(); //An LL(1) parser, generating the parse tree for the program
  + void printParseTree(FILE f);
* Files
  + Interface file : parser.h
  + Implementation : parser.c

**Stage-1 Deliverables**

* **C Code** for scanner and parser. Adhere to the interface requirements, and file name conventions. Place your main function in a file driver.c file.
* **makefile** for compilation and execution
* Test cases and output (parse tree & symbol table) files for test cases
* Brief **readme** file for directory/program structure, invocation command etc.

**Stage-1 Test Suite**

Formulate your own test set / programs from the uploaded token list, natural grammar, LL(1) grammar. The test cases which we use to evaluate the assignment will be uploaded shortly.

Stage-1 How-to

1. Read the language Specification : the overview, the grammar (natural form) and the tokens to gain an over-all understanding.
2. Apply your understanding to the given examples and work out the details by deriving the program text using the rules given in the grammar.
3. Understand the LL(1) form of the grammar.

a) Compare the grammar in both forms to verify the conversion to LL(1) form - in particular understand how disambiguation, left-recursion elimination and left factoring have been done.

b) Verify the FIRST and FOLLOW information given inside the comments on LL(1) version of the grammar.

 c) Parse - by hand - a small example or a few individual constructs using the LL(1) version of the grammar.

1. Implement the scanner using techniques discussed in the classroom
2. Test the scanner with the (to be)given test cases.
3. Design the data representation for the parse tree.
4. Construct FIRST and FOLLOW sets.
5. Construct a Boolean parser first i.e. write the parser to verify whether the program is valid or not. Alternate approaches for the parser structure:

a) Implement the parser as a custom recursive descent parser with a single token lookahead using the FIRST and FOLLOW sets.

b) Implement a generic parsing engine and construct the parsing table from FIRST and FOLLOW sets for the given grammar.

1. Test your parser with the given test cases .
2. Modify the parser to generate the parsing tree. Pay attention to memory allocation. Use macros/wrappers for allocation and data structure assignments instead of littering your code with pointer manipulations.
3. Write a procedure to traverse and print the parse tree.
4. Test your parser with the given test cases.
5. Write your own test cases and document your code