From: Vandana Agarwal . (via Nalanda)
Sent: 16 February 2023 22:00
To: RAMAKANT PANDURANG TALANKAR .
Subject: 11551001318_CS F363_JAN_2023: Stage 1: Interface details

11551001318_CS F363_JAN_2023 » Forums » Announcements » Stage 1: Interface details

V

Stage 1: Interface details
by Vandana Agarwal . - Thursday, 16 February 2023, 9:36 PM

Complete compiler project must be developed using C programming language. Ensure that your code is compatible to **GCC version 9.4.0 on Ubuntu 20.04.4.**

Teams are advised to design data structures for token info, grammar, parse table, parse tree, first and follow sets etc. and use names in self explanatory form. Following are the suggested prototypes for better understanding of the implementation needs and are provided as support. **However, the teams will have the flexibility to select the prototypes, parameters etc. appropriately**.

1. File lexer.c : This file contains following functions

FILE *getStream(FILE *fp): This function takes the input from the file pointed to by file pointer 'fp'. This file is the source code written in the given language. The function uses efficient technique to bring the fixed sized piece of source code into the memory for processing so as to avoid intensive I/O operations mixed with CPU intensive tasks.

The function also maintains the file pointer after every access so that it can get more data into the memory on demand. The implementation can also be combined with getNextToken() implementation as per the convenience of the team.

getNextToken( ): This function reads the input character stream and uses efficient mechanism to recognize lexemes. The function tokenizes the lexeme appropriately and returns all relevant information it collects in this phase (lexical analysis phase) encapsulated as tokenInfo and passes to the parser on demand. The function also displays *all* lexical errors appropriately.

removeComments(char *testcaseFile, char *cleanFile): This function is an additional plug in to clean the source code by removal of comments. The function takes as input the source code and writes the clean code in the file appropriately. Ensure that the line numbers of code in the cleanFile are same as original line numbers of the same code in testcaseFile. [Note: The function is invoked only once through your driver file to showcase the comment removal. However, your lexer does not really pick inputs from comment removed file. For showcasing your lexer's ability, directly take input from user source code]

2. File parser.c : This file contains following functions

ComputeFirstAndFollowSets (grammar G, FirstAndFollow F): This function takes as input the grammar G, computes FIRST and FOLLOW information and populates the appropriate data structure F. First and Follow set automation must be attempted, keeping in view the

programming confidence of the team members and the available time with the teams. The credit for the above is only 4 marks out of 45 marks reserved for stage 1 module. If teams opt not to develop the module for computation of First and follow sets, the same can be computed manually and information be populated in the data structure appropriately. However, all members of the team must understand that any new grammar rule for any new construct will then require their expertise in computing FIRST and FOLLOW sets manually (especially during online exam). Note: While First and Follow computation from grammar can be skipped at the cost of 4 marks, and data structure F can be populated manually, it is yet mandatory to populate the parse table automatically using the following function.

createParseTable(FirstAndFollow F, table T): This function takes as input the FIRST and FOLLOW information in F to populate the table T appropriately.

parseInputSourceCode(char *testcaseFile, table T): This function takes as input the source code file and parses using the rules as per the predictive parse table T and returns a parse tree. The function gets the tokens using lexical analysis interface and establishes the syntactic structure of the input source code using rules in T. The function must report *all* errors appropriately (with line numbers) if the source code is syntactically incorrect. If the source code is correct then the token and all its relevant information is added to the parse tree. The start symbol of the grammar is the root of the parse tree and the tree grows as the syntax analysis moves in top down way.

The function must display a message "Input source code is syntactically correct....." for successful parsing.

printParseTree(parseTree PT, char *outfile): This function provides an interface for observing the correctness of the creation of parse tree. The function prints the parse tree in inorder in the file outfile.

The output is such that each line of the file outfile must contain the information corresponding to the currently visited node of the parse tree in the following format

lexeme CurrentNode lineno tokenName valueIfNumber parentNodeSymbol isLeafNode(yes/no) NodeSymbol

The lexeme of the current node is printed when it is the leaf node else a dummy string of characters "----" is printed. The line number is one of the information collected by the lexical analyzer during single pass of the source code. The token name corresponding to the current node is printed third. If the lexeme is an integer or real number, then its value computed by the lexical analyzer should be printed at the fourth place. Print the grammar symbol (non-terminal symbol) of the parent node of the currently visited node appropriately at fifth place (for the root node print ROOT for parent symbol) . The sixth column is for printing yes or no appropriately. Print the non-terminal symbol of the node being currently visited at the 7th place, if the node is not the leaf node [Print the actual non-terminal symbol and not the enumerated values for the non-terminal]. Ensure appropriate justification so that the columns appear neat and straight.

Description of other files

lexerDef.h : Contains all data definitions used in lexer.c

lexer.h : Contains function prototype declarations of functions in lexer.c

parserDef.h : Contains all definitions for data types such as grammar, table, parseTree etc. used in parser.c

parser.h : Contains function prototype declarations of functions in parser.c

driver.c : As usual, drives the flow of execution to solve the given problem. (more details, if needed, will be uploaded soon). Take the input file name and buffer size at command line.

makefile : This file uses GNU make utility, which determines automatically which pieces of a large program need to be recompiled, and issues the commands to recompile them. The correctness of your make file depends on file dependencies used correctly.

NOTE:

1. A file using definitions and functions from other files must include interface files appropriately. For example parser.c uses functions of lexer.c, so lexer.h should be included in parser.c. Do not include lexer.h in lexer.c, as lexer.c already has its own function details. Also keep data definitions in files separate from the files containing function prototypes. In case of doubts, meet me and clarify your doubts. It is essential to place the contents in appropriate files and have correct set of files.

2. Use of any high level library other than standard C library is strictly not allowed.

Please feel free to discuss with me any queries regarding implementation

See this post in context
