Bangladesh University of Engineering & Technology
Department of Computer Science & Engineering
CSE 310: Compiler Sessional
Session January 2017

Assignment 3
Syntax and Semantic Analysis

April 4, 2017

1 Introduction

In the previous assignment, we have constructed a lexical analyzer to generate token stream. In this assignment we will construct the last part of the front end of a compiler for a subset of **C** language. That means we will perform syntax analysis and semantic analysis with a grammar rule containing function implementation in this assignment. To do so, we will build a parser with the help of Lex (Flex) and YACC (Bison).

2 Language

Our chosen subset of **C** language has following characteristics.

- There can be multiple functions. No two function will have the same name. A function need to be defined or declared before it is called. Also a function and a global variable cannot have the same symbol.
- There will be no preprocessing directives like include or define.
- Variables can be declared at suitable places inside a function. Variables can also be declared in global scope.
- All the operators used in previous assignment are included. Precedence and associativity rules are as per standard. Although we will ignore consecutive logical operators or consecutive relational operators like `‘a && b && c’` `‘a < b < c’`.
- No break statement and switch-case.

3 Tasks

You have to complete the following tasks in this assignment.

3.1 Syntax Analysis

For syntax analysis part you have to do the following tasks.

- Incorporate the grammar given in the **grammar.txt** along with this document in your yacc file. When a grammar matches the input from the c code, it should print the matching rule in correct order in an output file (parser.txt).
- Modify your lex file from previous assignment to use it with your yacc file. Remove all symbol table insertion from lex file.
- Use a **SymbolInfo** pointer to pass information from scanner to parser when needed. For example if your scanner detects an identifier, it will return a token named ID and pass it's symbol and type using a SymbolInfo pointer as the attribute of the token. On the other hand in case of semicolon, it will only return the token as the parser does not need any more information.

  You can implement this in two ways: either redefine the type of yylval (YYSTYPE) in parser and associate yylval with new type in scanner, or use %union field in parser.
- Handle any ambiguity in the given grammar (For example: if-else, you can find a solution in page 188-189 of flex-bison manual). Your yacc file should compile with 0 conflict.
- Add a new field **value** in SymbolInfo class. This will contain value of the corresponding symbol if there is any.
- Insert all the identifiers in the symbol table when they are declared in input file. For example if you find `int a,b,c;` then insert a, b and c in the symbol table. You can do this in the grammar rule of declaration.
- Evaluate each expression and update any symbol table entry corresponding to a variable if it gets a new value assigned. For example if the input file contains a line like `x= 2 || 3 < 5 + 6;` you have to update value of the SymbolInfo object corresponding to `x`;
- Print symbol table after each assignment expression (in the log.txt file).
- Print well formed syntax error messages with line number (in a log.txt file).
- Print symbol table after finishing parsing (in the log.txt file).

Bonus Task: Incorporate error recovery in your parser. Go through the bison manual for better understanding of error recovery (you might need to use bison's predefined token **error** for this purpose).

3.2 Semantic Analysis

In this part, you have to perform following tasks:

- **Type Checking:** You have to perform different type checking in this part. For assignment operation, you have to check whether the types of two side are conflicting or not.
  - Generate error message if operands of an assignment operator are not consistent with each other. Note that, the second operand of the assignment operator will be an expression which may contain numbers, variables, function calls etc.
  - A variable cannot be declared as void.
  - Both operand of modulus operator should be integer
  - During a function call all arguments should be consistent with function definition.
- **Undeclared Variables & Multiple Declaration:** You should check whether a variable used in an expression is declared or not and in case of undeclared variable or multiple declaration of same variable generate an error message. Also check whether two variable share the same name.
- **Array Index:** Generate appropriate array index out of bound error.
- **Function Parameter:** Check whether a function is called with appropriate number of parameters with appropriate types.

To implement this task, you can add necessary fields in the SymbolInfo class as required but try to avoid redundant fields.

3.3 Handling Grammar Rules for Functions

For implementing the grammar rules of functions, you will need to add some fields in your SymbolInfo class.

- You will need extra fields to store the return type, parameter list, number of parameters etc. in the SymbolInfo class, for proper handling of functions. You can take another class to hold the above mentioned fields and add a reference of that class in SymbolInfo class for convenience. Note that this is just a guideline, you are free to implement otherwise.
- As a part of semantic analysis you have to match the function declaration and function definition and report an error if there is any mismatch in return type, parameter number, parameter sequence or parameter type as well as invalid scoping of the function.

4 Input

The input will be a C source program in .c extension. File name will be given from command line.

5 Output

In this assignment, there will be two output file. One file, parser.txt, that will contain matching grammar rules and symboltable values as instructed above. Another file, log.txt will contain all actions of the syntax analysis and semantic analysis along with error messages with line number. For any detected error print something like “Line no 5: Corresponding error message". Print the line count and no of errors at the end of log file.

For more clarification about input output Sample I/O files will be provided to you.

6 Submission

- Plagiarism is strongly prohibited.
- No submission after the deadline will be allowed.
- Deadline will not extend in any situation.

7 Submission

All Submission will be taken via moodle. Please follow the steps given below to submit you assignment.

1. In your local machine create a new folder which name is your 7 digit student id.
2. Put the lex file named as <your_student_id>.l and yacc file named as <your_student_id>.y containing your code. Also put additional c file or header file that is necessary to compile your lex file. Do not put the generated lex.yy.c file or executable file in this folder.
3. Compress the folder in a zip file which should be named as your 7 digit student id.
4. Submit the zip file within the Deadline.

8 Deadline

Submission deadline is set at **29 April, 2017**. Please start early if you want to finish the assignment.

HAPPY CODING :)