# CSE 310-July 2016

# Assignment 3

Department of CSE, BUET November 7, 2016

## 1 Introduction

In the previous assignment, we have constructed a lexical analyzer to generate token stream. In this assignment we will construct a full front end of a compiler for a subset of 'C' language. That means we will perform syntax analysis, semantic analysis and intermediate code generation in this assignment. To do so, we will build a parser with the help of Lex (Flex) and YACC (Bison). As an intermediate language, we choose 8086 assembly language.

## 2 Language

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

- Only one function per program which is the main function.
- No global variables and preprocessing directives like include or define.
- Variables are declared at the very beginning of main function.
- A special function **println(ID)** which will print the value of the ID in the console in a line.
- 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.
- Modify your lex file from previous assignment to use it with your yacc 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.
- Print well formed syntax error messages with line number.
- Print symbol table after finishing parsing.
- **Bonus:**
  - Add some rules so that your parser can detect global variables, function declaration and function definition. You may chose some strict rule such that all global variables to be declared before main function if you wish.
  - Incorporate error recovery in your parser.

## 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. For example if a[10] is an array, you should report an error if there is a line containing a=10; in the input file. On the other hand you should allow assignment of an integer valued constant to a float variable but generate a warning in the reverse case. One more point is both operand of a modulus operation should be integer.
- **Undeclared Variables & Multiple Declaration:** In our assignment we consider all the local variables should be declared in the very beginning of a function. You should check whether a variable used in an expression is declared or not and in case of undeclared variable generate an error message. Also check whether two variable share the same name.
- **Array Index:** Generate appropriate array index out of bound error.
- **Bonus:** Add semantic rules for handling scope, in case you complete the bonus task of writing grammar for function as described in previous section.

To implement this task, you can add some field in the SymbolInfo class.

## 3.3 Intermediate Code Generation

You have to generate 8086 assembly language program from the input file after the input file successfully pass all the previous steps (lexical, syntax, semantics). To generate assembly code you may find the following instruction helpful.

- Add a field ‘code’ in SymbolInfo Class.As each of your non-terminal has a SymbolInfo pointer as attribute, you can propagate code for different portion using this newly added field.
- To generate assembly code for conditional statements and loops, define two functions named **newLabel()** and **newTemp()** where newLabel will generate a new label on each call and newTemp will generate a new temporary variable.
- Write a procedure for println(ID) function and call this procedure in your assembly code whenever you reduce a rule containing println.
- Do not forget some initialization part in assembly program like initializing the data segment register.

  You may also find the given sample code helpful in this case. This subsection may be updated. Notification will be given via moodle in such case.

### 3.4 Optimization

You have to do some **peephole optimization** works after intermediate code is generated. To do this you will take pair of consecutive instructions in the generated code and remove redundancy like following:

- Redundant mov instruction like

  ```
  mov ax, a
  mov a, ax
  ```

  In this case the second mov instruction can be omitted.

- Redundant addition or multiplication like a=a+0 and x=x*1

This subsection may be updated. Notification will be given via moodle in such case.

## 4 Input

The input will be a text file containing a C source program. File name will be given from command line.

## 5 Output

In this assignment, there will be two output file. One is a file containing generated code. You will output the optimized generated code after in this file.

The other file is a log file named as <YourStudentID>_log.txt. In this file you will output all the grammar rules matched by your parser. Print the symbol table entry after inserting any item into the symbol table and after any assignment operation. Also print the symbol table after finishing parsing. For any detected error print “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 please refer to the sample input output file given in moodle. You are highly encouraged to produce output exactly like the sample one.

## 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. This assignment is evaluated in two phases. In first phase you have to submit the task of syntax analysis and semantic analysis and in the second phase you will submit the rest. 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.

In case you don't understand the above mentioned steps, please check out the sample submission in moodle.

## 8 Deadline

- **Phase 1:** Deadline is set at 9:00 am, November 20, 2016 for all lab groups.
- **Phase 2:** Deadline is set at 9:00 am, December 4, 2016 for all lab groups.

Start early if you want to complete this offline. For any confusion feel free to contact at shareef.tamal@gmail.com. Best of Luck!