CS3005D Compiler Design - Winter 2024-25
Course Project: Details

Objective: Design and implement the front end of a compiler for a simple example language. The compiler will take the source code written in this language as input and produce an Intermediate Representation (IR). The project requires understanding of the theoretical concepts related to lexical analysis, parsing, semantic analysis and intermediate code generation. The project can be completed in three stages as given below:
Stage 1: Learning to use tools Lex and Yacc.
Stage 2: Building Lexical Analyzer and Parser using Lex and Yacc.
Stage 3: Implementing Semantic Analysis and Intermediate Code Generation.

Students must work in teams of 3. There will be two submissions - one after Stage 2 and the other after Stage 3. Evaluation will be based on these two submissions. Assessments may include tests involving implementing small extensions to the work done and viva-voce.

Stage 1: Learning to use tools Lex and Yacc. Please see the following documents:
Using Lex: `https://silcnitc.github.io/lex.html`
Using Yacc: `https://silcnitc.github.io/yacc.html`
Using Lex with Yacc: `https://silcnitc.github.io/ywl.html`

Complete the given examples and exercises before 31.01.2025

Stage 2: Building Lexical Analyzer and Parser. Using Lex and Yacc, build a parser for the example language named *Bcs*22 whose specification is given below:

A *Bcs*22 program starts with the keyword **BcsMain**, followed by a declaration list and a statement list enclosed within a pair of curly braces. All variables are declared in the beginning of the program. The two primitive data types in the language are **int** and **bool**. The syntax and semantics of statements and expressions are almost similar to that of the *C* programming language.

Syntax:

program → **BcsMain**{ *declist stmtlist* }
declist → *declist decl* | *decl*
decl → *type* **id** ;
type → **int** | **bool**
stmtlist → *stmtlist* ; *stmt* | *stmt*
stmt → **id** = *aexpr*
| **if** (*expr*) {*stmtlist*} **else** {*stmtlist*}
| **while** (*expr*) {*stmtlist*}
expr → *aexpr* **relop** *aexpr* | *aexpr*
aexpr → *aexpr* + *aexpr* | *term*
term → *term* * *factor* | *factor*
factor → **id** | **num**

Token Specifications:

letter → [a-zA-Z]
digit → [0-9]
id → *letter* (*letter*|*digit*)*
num → $digit^+$
relop → < | > | <= | >= | == |! =

Keywords: BcsMain, if, else, while, int, bool

A sample *Bcs22* program is given below:

```
BcsMain
{
   int sum; int i; int n;
   n=10;
   i=1; sum=0;
   while(i<n)
      {sum=sum+i; i=i+1};
   sum=sum*10
}
```

Input: Name of the program file to be parsed (as command-line argument)
Output: "Parsing Successful", if program is syntactically valid, "Syntax Error", otherwise.

The parser should detect any kind of syntax error in the program. The exact nature of the error or line number need not be printed. Note that checking if a variable is declared before use is not to be done in this stage.

Submission: On or before **02.02.2025, 10.00 p.m.** (extended to **16.02.2025, 10.00 p.m.**)
Submit the following as a single gzipped file (*.gz*) through the submission link in the course page:

- The lex program
- The yacc program
- A program for which your compiler prints "Parsing Successful" (not the sample program)
- A program for which your compiler prints "Syntax Error"

Name of the *zip* file and the name of each file in the folder should be prefixed by
$CS < 01/02/03/04 > \_ < GroupNumber >$ (e.g. $CS01\_1.gz$)

Stage 3: Intermediate Code Generation.

Generate 3-address code for a program consisting of one or more assignment statements (flow of control statements are not included in this stage). You can add semantic actions to the Yacc program, similar to the SDD for intermediate code generation for assignment given in class. SDD should be rewritten for the expression grammar for $Bcs22$. Instead of attribute *id.entry*, use the name of the identifier itself. Write the generated code to a text file.

Submission: On or before **16.03.2025, 10.00 p.m.**
Submit the following as a single gzipped file (*.gz*) through the submission link in the course page:

- The lex program.
- The yacc program.
- A sample $Bcs22$ input program with at least three assignment statements in which at least one of the assignments has a right and side expression with more than one operators.
- The generated 3-address code.

Name of the *zip* file and the name of each file in the folder should be prefixed by
$CS < 01/02/03/04 > \_ < GroupNumber >$ (e.g. $CS01\_1.gz$)