Assignment on YACC

(Individual submission)

Submission Deadline: 14/09/2019

1. Apply YACC to construct CFG that will validate the language $L=\{a^i b^n c^n \mid n>0, i>=0\}$. [Also for, $L=\{a^k b^n c^k \mid n>0, k>=0\}$].

2. Apply YACC to construct CFG to validate the language $L=\{a^n b^m \mid m < n$ and $m>0, n>0\}$.

3. Write a YACC program with semantic actions to recognize a valid arithmetic expression and calculate the corresponding value of the expression. If there is any syntax error in the source code, the program will show the error with the line number. Parentheses should be considered.

4. Write a YACC program to check the syntax of "if-else", nested if-else, "if-elseif-else" block in C language. While there is any error, the program displays the error with the line number.

5. Write a YACC program to check the syntax of "for" loop and nested "for" loop block in C language.

6. Imagine the syntax of a programming language construct such as while-loop --

```
while (
condition )
begin
        stateme
nt ;  end
```

where *while*, *begin*, *end* are keywords; condition can be a single comparison expression (such as x == 20, etc.); and statement is the assignment to a location the result of a single arithmetic operation (eg., a = 10 * b). Write a context free grammar for the above construct, and write a YACC program that can parse the above syntax and generate the result.

7. Write a CFG for Simple-C programming language.

Given

The Simple-C language is a simplified and modified subset of C programming language. Consider a simple C-like language with the following specifications:

1. Data Type : integer (INT/int), floating point (FLOAT/float)
2. Condition constructs: if

3. Loop Constructs: for, while, do-while
4. Input / Output Constructs:
    1. read(x) - Read into variable x
    2. print(x) - Write variable x to output
5. Relational operators, assignment and arithmetic operators
6. Only function is main(), there is no other function.

Example

```
main()

{       INT  i=0;       INT  sum=0;    INT  count;

        read(count);
        for(i=0;i<10;i++)
        {

            read(
            x);
            sum+=x;

        }

        print(sum)
;

}
```

To be done:

i) Implement a Lexical Analyzer/Scanner for the simple C-like programming language using 'lex' tools.

ii) Write a CFG for the Simple-C language specifications.

Example Input Programs

Example 1: Correct Simple-C Program

```
int main()
{

    int fn1   =1;
    int fn2   =1;

    int i=0;
    int n;
    int fn=0;
```

```
        read(n);  if(n < 1)
                print(0);
        e
        l
        s
        e
        {
                while(i < n)
                {
                        fn = fn1 + fn2;
                        print(fn);
                        fn2 = fn1;
                        fn1 = fn;
                        i
                ++;  }

        }
}
```

The above program should be accepted by the parser. Example 2: Wrong Simple-C Program

```
int
main()
{
        int r;
        int p                   /* Syntactic error here, missing
        semicolon */
        int int j;              /* Parse error here, keyword int
        occurs two
times */
        int
        total=0
        ; int
        i;

        read(r);
        read(p);
        scanf (j;               /* unrecognized token here: scanf and
        mismatched
parentheses*/
        for(i = r; i < p; i+= j)
                total = total + j;
        print(total);
}
```

The input to your program should be given from the command line. Like a.out < testinput.txt.

Output Expected

If input is wrong, i.e. the parser was not able to parse the input or lexer was not able to recognize a token, then the output along with diagnosed reasons should be presented to the user.

Sample Output

If input is not correct

FAIL
Line number: 7 Parse Error

If input is correct

OUTPUT should be simply PASS.