**COMPILER DESIGN**

**Grammar for the compiler:**

<PROGRAM> -> '{' <BODY> '}’

<BODY>-> {<PRINT>';'|<ASSIGNMENT>';'|<VARIABLE>';’|<WHILE>|<IF>|<SWITCH>|<RETURN>';'}

<ASSIGNMENT> -> identifier '=' <EXPRESSION>

<VARIABLE> -> ('int'|'float'|'boolean'|'char’|'string'|'void')identifier

<WHILE> -> 'while' '(' <EXPRESSION> ')' <PROGRAM>

<IF> -> 'if' '(' <EXPRESSION> ')' <PROGRAM> ['else' <PROGRAM>]

<RETURN> -> 'return'

<PRINT> -> ’print’ ‘(‘ <EXPRESSION> ‘)’

<EXPRESSION> -> <X> {'|' <X>}

<X> -> <Y> {'&' <Y>}

<Y> -> ['!'] <R>

<R> -> <E> {('>'|'<'|'=='|'!=') <E>}

<E> -> <A> {(’+'|'-’) <A>}

<A> -> <B> {('\*'|'/') <B>}

<B> -> ['-'] <C>

<C> -> integer | octal | hexadecimal | binary | true | false | string | char | float | identifier|'(' <EXPRESSION> ')'

<SWITCH> -> 'switch' '(' identifier ')' '{' <CASES> [<DEFAULT>] '}'

<CASES> -> ('case' (integer|octal|hexadecimal|binary) ':' <PROGRAM>)+

<DEFAULT> -> 'default' ':' <PROGRAM>

1. Create Lexer:

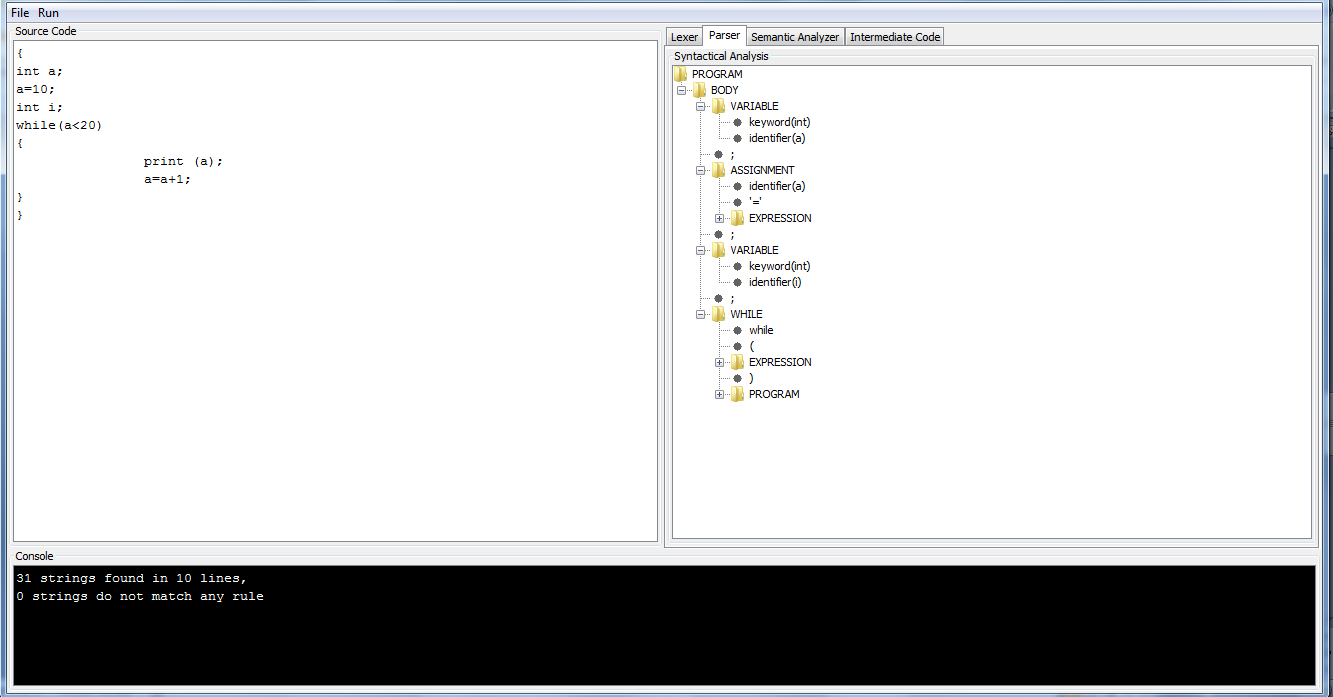
For creating Lexer for above grammar, we need to create DFA for each rule and convert that DFA into tabular representation.



According to the final state of DFA of each rule for input string, report the type of Token.

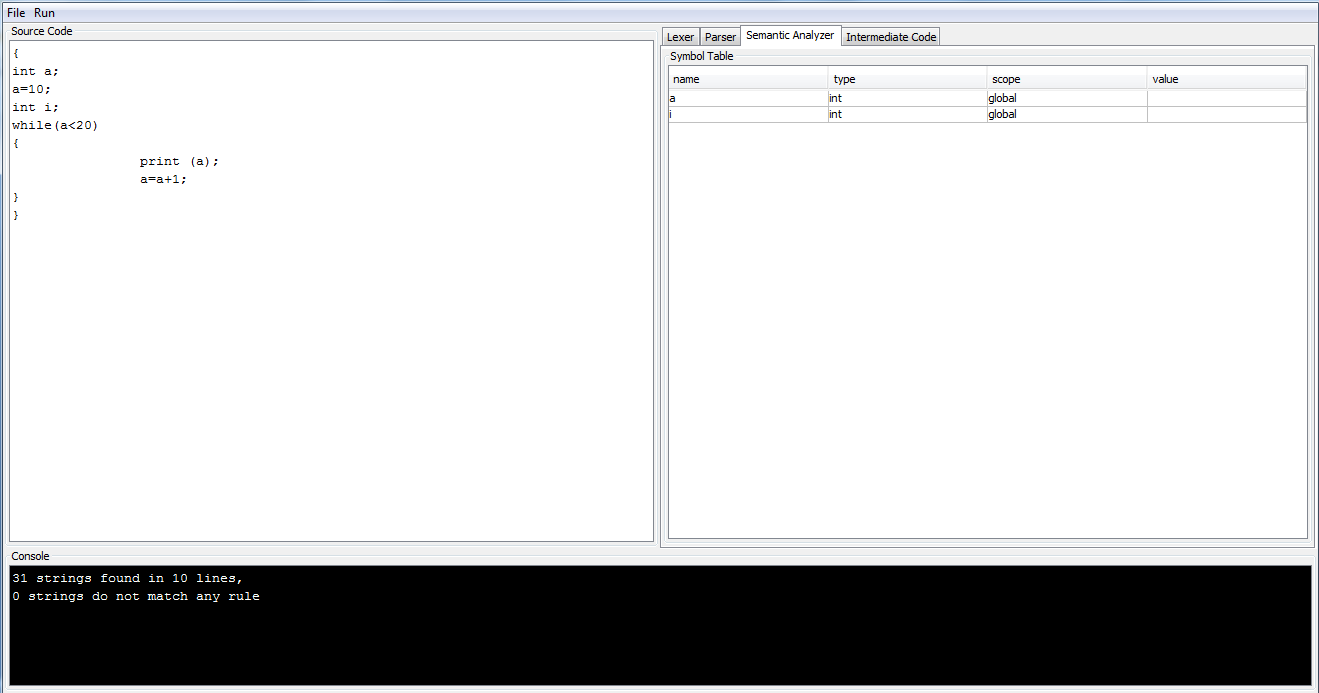
1. Create Parser:

This part is dependent on the first part. Take the identified token from Lexer and generate Parsing tree.



1. Semantic Analysis:

Generate symbol table for the input program by incorporating the rules of semantic analysis.



1. Generate Assembly equivalent code:

Read the input program and convert it into assembly level language program using instructions: LIT, LOD, STO, OPR, JMP and JMC.

LIT:

put a literal (instruction[pc].param1) in the register. pc = pc->next

LOD:

get the value (from the symbol\_table) of the variable instruction[pc].param1 and put the value in the register

pc = pc->next

STO:

get a value from the register and store the value in the variable instruction[pc].param1. pc = pc->next

JMP:

get the value (from the symbol\_table) of the label instruction[pc].param1. pc = value

JMC:

get a value from the register compare the value with instruction[pc].param2 if they are equal get the value (from the symbol\_table) of the label instruction[pc].param1. pc = value

OPR:

get a value A from the register get a value B from the registerevaluate A operation B and store the value in the register

Example:

Input Program:

{

int a;

a=10;

int i;

while(a<20)

{

print (a);

a=a+1;

}

}

Output:

a, int, global, null

i, int, global, null

#e1, int, 14

#e2, int, 3

@

LIT 10, 0

STO a, 0

LOD a, 0

LIT 20, 0

OPR 12, 0

JMC #e1, false

LOD a, 0

OPR 21, 0

LOD a, 0

LIT 1, 0

OPR 2, 0

STO a, 0

JMP #e2, 0

OPR 0, 0

References:

This is academic project for CSE 340: Principles of programming language course. Code snippets were provided by Prof. Gonzalez Sanchez.