CS 3512 - Programming Languages

Programming Project 01

- Group members
    - 200551R - Sahassaka M.A.T.
    - 200708G - Wickramage R.Y.
    - 200405B - Muthumala L.G.P
    - 200360F - Madawa G.C.H.O.

Problem Description

You are required to implement a lexical analyzer and a parser for the RPAL language. Refer RPAL Lex.pdf for the lexical rules and RPAL Grammar.pdf for the grammar details. You should not use 'lex'. 'yacc' or any such tool.

Output of the parser should be the Abstract Syntax Tree (AST) for the given input program. Then you need to implement an algorithm to convert the Abstract Syntax Tree (AST) into a Standardised Tree (ST) and implement a CSE machine.

Your program should be able to read an input file which contains a RPAL program. Output of your program should match the output of"rpal.exe" for the relevant program.

You must use C/C++ or Java for this project.

Scanner: A scanner, often referred to as a lexer, is a fundamental component of a compiler or interpreter. It is responsible for the first phase of the compilation process known as lexical analysis. The primary purpose of the scanner is to read the source code of a program and break it down into a sequence of tokens. Tokens are the smallest meaningful units of the programming language, such as keywords, identifiers, operators, and literals. The scanner ignores whitespace and comments while identifying and categorizing the tokens, which are then passed on to the next phase of compilation, known as the parser.

Parser: The parser is another crucial part of a compiler or interpreter, following the lexical analysis performed by the scanner. It takes the stream of tokens produced by the scanner and organizes them into a hierarchical structure, typically represented as an abstract syntax tree (AST). The AST represents the grammatical structure of the source code according to the rules of the programming language's syntax. The parser checks whether the input code conforms to the language's grammar rules and reports any syntax errors if the code is incorrect. If the parsing process is successful, the resulting abstract syntax tree is used in subsequent compilation steps, such as code optimization and code generation.

CSE Machine Control Stack: The control stack is a data structure that keeps track of the active procedure calls and their corresponding execution contexts. When a function or procedure is called, a new activation record is pushed onto the control stack. This record contains information such as the return address, local variables, and other necessary data. As functions complete their execution, their activation records are popped from the stack, allowing the program to return to the appropriate execution point.

Environment: The environment stores the bindings between identifiers (variables, constants, and function definitions) and their corresponding values in a specific scope. It is responsible for managing variable access and ensuring proper scoping during program execution.

In our project files, the lexer(Scanner) , parser and CSE machine and other related files can be found as follows.

Scanner =>

- lexer.cpp
- lexer.h

Parser =>

- standardizer.cpp
- standardizer.h
- parser.cpp

- parser.h

CSE Machine =>

    - CSEMachine.cpp
    - CSEMachine.h
    - Control.cpp
    - Control.h

Other =>

- Environment.cpp
- Environment.h
- P1.cpp
- token.cpp
- token.h
- treeNode.h

Example Program

```
let Sum(A) = Psum (A,Order A )
          where rec Psum (T,N) = N eq 0 -> 0
                                  | Psum(T,N-1)+T N
in Print ( Sum (1,2,3,4,5) )
```

AST for the Program

```
let
.function_form
..<ID:Sum>
..<ID:A>
..where
...gamma
....<ID:Psum>
....tau
.....<ID:A>
.....gamma
......<ID:Order>
......<ID:A>
...rec
....function_form
.....<ID:Psum>
.....,
......<ID:T>
......<ID:N>
.....->
......eq
.......<ID:N>
.......<INT:0>
......<INT:0>
......+
.......gamma
........<ID:Psum>
........tau
.........<ID:T>
.........-
```

```
..........<ID:N>
..........<INT:1>
.......gamma
........<ID:T>
........<ID:N>
.gamma
..<ID:Print>
..gamma
...<ID:Sum>
...tau
....<INT:1>
....<INT:2>
....<INT:3>
....<INT:4>
....<INT:5>
```

ST for the Program

```
gamma
.lambda
..<ID:Sum>
..gamma
...<ID:Print>
...gamma
....<ID:Sum>
....tau
.....<INT:1>
.....<INT:2>
.....<INT:3>
.....<INT:4>
.....<INT:5>
.lambda
..<ID:A>
..gamma
...lambda
....<ID:Psum>
....gamma
.....<ID:Psum>
.....tau
......<ID:A>
......gamma
.......<ID:Order>
.......<ID:A>
...gamma
....<Y*>
....lambda
.....<ID:Psum>
```

```
.....lambda
......,
.......<ID:T>
.......<ID:N>
......->
.......eq
........<ID:N>
........<INT:0>
.......<INT:0>
.......+
........gamma
.........<ID:Psum>
.........tau
..........<ID:T>
..........-
...........<ID:N>
...........<INT:1>
........gamma
.........<ID:T>
.........<ID:N>
```