CS323 Documentation

About 2 pages

1. Problem Statement

The problem to be solved in this assignment is to create a symbol table, error handling and code generation. Rat24S is simplified so that the program will not have functions or “real” type values and will follow the semantics of the language; type must match and arithmetic operations are not allowed for booleans.

2. How to use your program

A. Clone the repository:

```
git clone https://github.com/CodieTamida/compiler.git
```

B. Use the following command to run the compiler:

```
python3 rat24s.py input.txt -o output.txt -v
```

Optional arguments:

The Rat24S compiler has many options for reading input from a file, writing output to a file, extracting the tokens, and checking your code syntax.

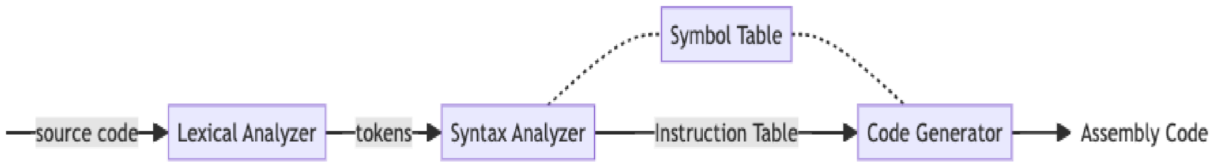
Flag	Full tag	Description
-o	--output	Output file path.
-t	--tokens	Extract tokens only, but don't do anything beyond that.
-s	--syntax	Analyze syntax only, but don't do anything beyond that.
-v	--verbose	Enable verbose mode.

Examples:

```
python3 rat24s.py testcase1.txt -o output1.txt -v
python3 rat24s.py testcase2.txt -o output2.txt -v
python3 rat24s.py testcase3.txt -o output3.txt -v
```

3. Design of your program

The program uses Recursive Descent Parser technique to analyze and parse the syntax of a programming language. During the parsing process, it generates assembly code instructions. Here is the high level diagram of the program:



Syntax Analyzer (Parser): The parser starts at the top of the syntax tree (the root) and works its way down towards the leaves. The parser has recursive functions corresponding to the grammar rules of the language being parsed.

Symbol Table: During parsing, the parser interacts with the symbol table. When encountering identifiers, the parser checks and updates the symbol table for identifier management.

Semantic Rules:

- “true” has an integer value of 1 and “false” has an integer value of 0.
- No arithmetic operations are allowed for booleans.
- The types must match for arithmetic operations (no conversions)

Code Generator: This component is responsible for generating assembly code from parsed instructions. It takes an instruction table generated by the parser and iterates through the instruction table to construct assembly code strings. Additionally, this component provides utility functions to print the contents of both the symbol table and the instruction table.

Error Handling: The parser includes mechanisms for error detection and reporting. It detects and reports syntax and semantic errors during parsing, including issues like undeclared or redeclared identifiers and type mismatches. When an error is encountered, the program throws an exception error with a detailed message.

Grammar rules:

```
R1. <Rat24S> ::=  $ <Opt Function Definitions> $ <Opt Declaration List> $
<Statement List> $
R2. <Opt Function Definitions> ::= <Function Definitions>  | <Empty>
R3. <Function Definitions> ::= <Function> | <Function> <Function
Definitions>
R4. <Function> ::= function <Identifier>  ( <Opt Parameter List> ) <Opt
Declaration List> <Body>
R5. <Opt Parameter List> ::= <Parameter List>  |  <Empty>
R6. <Parameter List> ::= <Parameter>  |  <Parameter> , <Parameter List>
R7. <Parameter> ::= <IDs > <Qualifier>
R8. <Qualifier> ::= integer  |  boolean  | real
R9. <Body> ::= { < Statement List> }
R10. <Opt Declaration List> ::= <Declaration List>  |  <Empty>
R11. <Declaration List> := <Declaration> ;  |  <Declaration> ; <Declaration
List>
R12. <Declaration> ::=  <Qualifier > <IDs>
R13. <IDs> ::=  <Identifier>  | <Identifier>, <IDs>
R14. <Statement List> ::=  <Statement>  | <Statement> <Statement List>
R15. <Statement> ::=  <Compound> | <Assign> |  <If> | <Return>  | <Print>
|  <Scan>  | <While>
R16. <Compound> ::=  { <Statement List> }
R17. <Assign> ::=  <Identifier> = <Expression> ;
R18. <If> ::=  if ( <Condition> ) <Statement>  endif  |  if ( <Condition> )
<Statement>  else <Statement> endif
R19. <Return> ::= return ; | return <Expression> ;
R20. <Print> ::=  print ( <Expression>);
R21. <Scan> ::=  scan ( <IDs> );
R22. <While> ::= while ( <Condition> ) <Statement> endwhile
R23. <Condition> ::=  <Expression> <Relop>  <Expression>
R24. <Relop> ::=  ==  |  !=  |  >  |  <  | <=  |  =>
R25. <Expression> ::=  <Expression> + <Term>  | <Expression> - <Term>  |
<Term>
R26. <Term>  ::=  <Term> * <Factor>  |  <Term> / <Factor>  |  <Factor>
R27. <Factor> ::=  - <Primary>  |  <Primary>
R28. <Primary> ::=  <Identifier> | <Integer> |  <Identifier> ( <IDs> )  |
( <Expression> )  |
<Real> |  true  | false
R29. <Empty>  ::= ε
```

Rules 3, 6, 11, 13, 14, 18, 19, and 28 have left factorization and have to be rewritten.

Rule 3:

```
<Function Definitions> -> <Function> <Function Definitions Prime>
<Function Definitions Prime> -> <Function Definitions> | ε
```

Rule 6:

```
<Parameter List> -> <Parameter> <Parameter List Prime>
<Parameter List Prime> -> , <Parameter List> | ε
```

Rule 11:

```
<Declaration List> -> <Declaration List> ; <Declaration List Prime>
<Declaration List Prime> -> <Declaration List> | ε
```

Rule 13:

```
<IDs> -> <Identifier> <IDs Prime>
<IDs Prime> -> , <IDs> | ε
```

Rule 14:

```
<Statement List> -> <Statement> <Statement List Prime>
<Statement List Prime> -> <Statement List> | ε
```

Rule 18:

```
<If> -> if ( <Condition> ) <Statement> <If Prime>
<If Prime> -> endif | else <Statement> endif | ε
```

Rule 19:

```
<Return> -> return <Return Prime>
<Return Prime> -> ; | <Expression>; | ε
```

Rule 28:

```
<Primary> -> <Identifier> <Primary Prime>
<Primary Prime> -> Integer | ( <IDs> ) | ( <Expression> ) | <Real> |
true | false | ε
```

Rules 25 and 26 have left recursion and have to be rewritten.

Rule 25:

```
<Expression> -> <Term> <Expression Prime>
<Expression Prime> -> + <Term> <Expression Prime> | - <Term>
<Expression Prime> | ε
```

Rule 26:

```
<Term> -> <Factor> <Term Prime>
<Term Prime> -> * <Factor> <Term Prime> | / <Factor> <Term Prime> | ε
```

4. Any Limitation

None

5. Any shortcomings

None.