CPSC 323 Project Documentation

About 2-3 pages

1. Problem Statement

The programming assignments are based on a language called "Rat18S" which is described as follows. The Rat18S language is designed to be an easy to understand. It has a short grammar and relatively clean semantics.

1) Lexical Conventions:

The lexical units of a program are identifiers, keywords, integers, reals, operators and other separators. Blanks, tabs and newlines (collectively, "white space") as described below are ignored except as they serve to separate tokens.

Some white space is required to separate otherwise adjacent identifiers, keywords, reals and integers.

<*Identifier*> is a sequence of letters or digits, however, the first character must be a letter and last char must be either $ or letter. Upper and lower cases are same.

<*Integer*> is an unsigned decimal integer i.e., a sequence of decimal digits.

<*Real*> is integer followed by "." and Integer, e.g., 123.00

Some identifiers are reserved for use as ***keywords***, and may not be used otherwise:

e.g., int, if, else, endif, while, return, get, put etc.

Comments are enclosed in **!** **!**

2) Syntax rules : The following BNF describes the Rat18S.

R1. <Rat18S> ::= <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> ::= int | 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> ::= put (<Expression>);
R21. <Scan> ::= get (<IDs>);
R22. <While> ::= while (<Condition>) <Statement>
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> ::=

2. How to use your program

- *Open cmd/terminal*
- *goto extracted directory*
- *type java lexicalAnalysis*

3. Design of your program

I have chosen to use Enum and couple it with a switch case in java because enum gives me the facility to write all the keywords together and switch case avoids nasty if else if loops which get confusing often. To handle file io I have used bufferedreader from java.io package.

4. Any Limitation

My code does not implement the required solution completely.

5. Any shortcomings

I could not implement the distinction between keywords and identifier because my approach was initially to use a Map object and run it through the array of String . But that was not catching other words which is why I had to change the logic at the very end moment and now it is implemented using a char by char reading of the string literals.

This is why the keywords like for if return are being treated as identifiers. I am sorry for not being able to complete it as required.
-Yours Truely,
Tejas