**Problem Statement**

Every identifier declared in the program should be placed in a symbol table and accessed by the symbol table handling procedures. Modify your parser according to the simplified Rat20F and add code to your parser that will produce the assembly code instructions. The instructions should be kept in an array and at the end, the content of the array is printed out to produce the listing of assembly code. Your array should hold at least 1000 assembly instructions. The instruction starts from 1. The listing should include an array index for each entry so that it serves as label to jump to. The compiler should also produce a listing of all the identifiers

- The simplified Rat20F is essentially the same except that the program has NO <Function Definitions>

- No “real” type is tested

- Consider that “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)

**Usage**

1. Open Command Prompt (Windows) with a working directory containing the app and the three test files
2. Run the program with the first argument as the input file in quotation marks and the second argument as the output file with or without quotation marks

**Program Design**

The program is a modified version of the syntax analyzer for the programming language Rat20F. Certain productions now contain additional code that generate assembly instructions and places them in a standard vector called "Instr\_table". A new modified version of the ID function was made in order to distinguish between instances of declaring identifiers and simply using them. Declaring an identifier will make an instance of a "symbol" class containing the record of the identifier and the memory address and put it in the symbol table. Using a undeclared identifier not in the symbol table or redeclaring an identifier already in the symbol table will trigger error functions. In order to enforce the semantics of simplified Rat20F, a stack was made to hold lexer records for arithmetic operations. Lexer records and the lexer function itself have been changed to include a "type" variable for record with the "identifier" token. Records are put in the arithmetic stack when their symbols are used and they are compared during arithmetic operations. An error function will be called when the records' type mismatch or when at least one of the records have a "boolean" type. All error functions print to std::error and exit the program.

**Limitations**

None

**Shortcomings**

None

**Rewritten RAT20F Grammar**

R1. <Rat20F> ::= <Opt Function Definitions> $$ <OptDeclaration List> <Statement List> $$

R2. <Opt Function Definitions> ::= <Function Definitions> | <Empty>

**R3. <Function Definitions> ::= <Function> <Function Definitions>'**

**R4. <Function Definitions>' ::= <Function Definitions> | <Empty>**

R5. <Function> ::= function <Identifier> ( <Opt Parameter List> ) <Opt Declaration List> <Body>

R6. <Opt Parameter List> ::= <Parameter List> | <Empty>

**R7. <Parameter List> ::= <Parameter> <Parameter List>'**

**R8. <Parameter List>' ::= , <Parameter List> | <Empty>**

R9. <Parameter> ::= < IDs > <Qualifier>

R10. <Qualifier> ::= Int | boolean | real

R11. <Body> ::= { <Statement List> }

R12. <Opt Declaration List> ::= <Declaration List> | <Empty>

**R13. <Declaration List> ::= <Declaration> ; <Declaration List>'**

**R14. <Declaration List>' ::= <Declaration List> | <Empty>**

R15. <Declaration> ::= <Qualifier> <IDs>

**R16. <IDs> ::= <Identifier> <IDs>'**

**R17. <IDs>' ::= , <IDs> | <Empty>**

**R18. <Statement List> ::= <Statement> <Statement List>'**

**R19. <Statement List>' ::= <Statement> | <Empty>**

R20. <Statement> ::= <Compound> | <Assign> | <If> | <Return> | <Print> | <Scan> | <While>

R21. <Compound> ::= { <Statement List> }

R22. <Assign> ::= <Identifier> = <Expression> ;

**R23. <If> ::= if ( <Condition> ) <Statement> <If>'**

**R24. <If>' ::= fi | else <Statement> fi**

**R25. <Return> ::= return <Return>'**

**R26. <Return>' ::= ; | <Expression> ;**

R27. <Print> ::= put ( <Expression> );

R28. <Scan> ::= get ( <IDs> );

R29. <While> ::= while ( <Condition> ) <Statement>

R30. <Condition> ::= <Expression> <Relop> <Expression>

R31. <Relop> ::= == | != | > | < | <= | =>

**R32. <Expression> ::= <Term> <Expression>'**

**R33. <Expression>' ::= + <Term> <Expression>' | - <Term> <Expression>' | <Empty>**

**R34. <Term> ::= <Factor> <Term>'**

**R35. <Term>' ::= \* <Factor> <Term>' | / <Factor> <Term>' | <Empty>**

R36. <Factor> ::= - <Primary> | <Primary>

**R37. <Primary> ::= <Identifier> <Primary>' | <Integer> | ( <Expression> ) | <Real> | true |**

**false**

**R38. <Primary>' ::= ( <IDs> ) | <Empty>**

R39. <Empty> ::= ε

Note: <Identifier>, <Integer>, <Real> are token types

**Removal of Left-Recursions**

Original: <Expression> ::= <Expression> + <Term> | <Expression> - <Term> | <Term>

α1 = +<Term>, α2 = -<Term>, δ = <Term>

<Expression> ::= <Term> <Expression>'

<Expression>' ::= + <Term> <Expression>' | - <Term> <Expression>' | ε

<Term> ::= <Term> \* <Factor> | <Term> / <Factor> | <Factor>

α1 = \*<Factor>, α2 = /<Factor>, δ = <Factor>

<Term> ::= <Factor> <Term>'

<Term>' ::= \* <Factor> <Term>' | / <Factor> <Term>' | ε

**Left-Factorizations**

Original: <Function Definitions> ::= <Function> | <Function> <Function Definitions>

<Function Definitions> ::= <Function> <Function Definitions>'

<Function Definitions>' ::= <Function Definitions> | ε

Original: <Parameter List>' ::= <Parameter> | <Parameter> , <Parameter List>

<Parameter List> ::= <Parameter> <Parameter List>'

<Parameter List>' ::= , <Parameter List> | ε

Original: <Declaration List> ::= <Declaration> ; | <Declaration> ; <Declaration List>

<Declaration List> ::= <Declaration> ; <Declaration List>'

<Declaration List>' ::= <Declaration List> |

Original: <IDs> ::= <Identifier> | <Identifier> , <IDs>

<IDs> ::= <Identifier> <IDs>'

<IDs>' ::= , <IDs> | <Empty>

Original: <Statement List> ::= <Statement> | <Statement> <Statement>

<Statement List> ::= <Statement> <Statement List>'

<Statement List>' ::= <Statement> | <Empty>

Original: <If> ::= if ( <Condition> ) <Statement> fi | if ( <Condition> ) <Statement> else

<Statement> fi

<If> ::= if ( <Condition> ) <Statement> <If>'

<If>' ::= fi | else <Statement> fi

Original: <Return> ::= return ; | return <Expression> ;

<Return> ::= return <Return>'

<Return>' ::= ; | <Expression> ;

Original: <Primary> ::= <Identifier> | <Identifier> ( <IDs> ) | <Integer> | ( <Expression> ) | <Real>

| true | false

<Primary> ::= <Identifier> <Primary>' | <Integer> | ( <Expression> ) | <Real> | true |

false

<Primary>' ::= ( <IDs> ) | <Empty>