Lexical Analyzer for F-15 Language

Cabdifatah Mohamed

1 Introduction

In this assignment, the goal was to design and implement a lexical analyzer for the F-15 language using Flex, along with the supporting components such as a C program for testing, a Makefile for compilation, and thorough documentation. The lexer successfully identifies keywords, operators, numbers, strings, and identifiers according to the lexical grammar of the F-15 language. By handling common language features like whitespace and comments, and providing appropriate error handling for unrecognized tokens, the lexer ensures robust and accurate tokenization of input files.

The test input file verifies that all the lexical rules are functioning as expected, ensuring that the lexer can handle various real-world inputs. The automated build process, handled by the Makefile, streamlines the workflow by efficiently compiling and testing the lexer. The documentation explains the design and working of the lexer, offering insight into the rationale behind the choices made during development.

2 Lexer Design

2.1 Lexer File (cabdifatah_mohamed.l)

The lexer is a program that scans through the input file and identifies patterns that match the specified rules. These rules define the tokens that make up the F-15 language, such as keywords (e.g., `PROGRAM`), operators (e.g., `.GT.`), and variables (identifiers).

In the lexer file, each pattern corresponds to a regular expression, and when a match is found, the lexer returns a token representing that match. Below are a few examples:

- `"PROGRAM"` matches the literal word `PROGRAM` and returns a token for the `PROGRAM` keyword.
- `[0-9]+` matches sequences of digits (integers) and converts them into numbers using `atoi()`.
- `[a-z]+` matches lowercase letter sequences, representing identifiers like variable names.

The lexer also handles special characters like `+`, `-`, `*`, and parentheses.

2.1.1 Code Snippet

```
%{
#define INTEGER 1
#define STRING 2
#define IDENTIFIER 3
#define READ 4
#define PRINT 5
#define IF 6
#define THEN 7
#define ELSE 8
#define DO 9
#define PROGRAM 10
#define END 11
#define GT 12
#define LT 13
#define EQ 14
#define PLUS 15
#define MINUS 16
#define TIMES 17
#define DIVIDE 18
#define LPAREN 19
#define RPAREN 20
#define LBRACKET 21
#define RBRACKET 22
#define COMMA 23
#define SEMICOLON 24
#define COLON 25
#define ASSIGN 26

#include <stdlib.h>
#include <string.h>

typedef union {
    int intVal;
    char* strVal;
} YYSTYPE;

YYSTYPE yylval;
%}

%%
"PROGRAM"           { return PROGRAM; }
"INTEGER"           { return INTEGER; }
"READ"              { return READ; }
"PRINT"             { return PRINT; }
"IF"                { return IF; }
"THEN"              { return THEN; }
"ELSE"              { return ELSE; }
"DO"                { return DO; }
"END"               { return END; }
".GT."              { return GT; }
".LT."              { return LT; }
".EQ."              { return EQ; }
[0-9]+          { yylval.intVal = atoi(yytext); return INTEGER; }
```

```
[a-z][a-z]* { yylval.strVal = strdup(yytext); return IDENTIFIER; }
\"[^\"\n]*\"  { yylval.strVal = strdup(yytext); return STRING; }
"+"               { return PLUS; }
"-"               { return MINUS; }
"*"               { return TIMES; }
"/"               { return DIVIDE; }
"("               { return LPAREN; }
")"               { return RPAREN; }
"["               { return LBRACKET; }
"]"               { return RBRACKET; }
","               { return COMMA; }
";"               { return SEMICOLON; }
":"               { return COLON; }
"="               { return ASSIGN; }
[\n]+             { /* Ignore new lines */ }
[\t ]+            { /* Ignore whitespaces */ }
"!".*             { /* Ignore comments */ }
.                 { printf("Unrecognized token: %s\n", yytext); exit(1); }
%%

int yywrap() {
    return 1;
}
```

2.2 Main Program (cabdifatah_mohamed.c)

This is a simple C program that uses the lexer to read through the input file and prints out each token it identifies. The `yylex()` function processes the input and passes tokens to the main program.

2.2.1 Code Snippet

```
#include <stdio.h>

extern int yylex();
extern union {
    int intVal;
    char* strVal;
} yylval;
extern char* yytext;

int main(void) {
    int token;
    while((token = yylex())) {
        switch(token) {
            case INTEGER:
                printf("INTEGER: %d\n", yylval.intVal);
                break;
            case STRING:
                printf("STRING: %s\n", yylval.strVal);
                break;
            case IDENTIFIER:
                printf("IDENTIFIER: %s\n", yylval.strVal);
                break;
            default:
                printf("KEYWORD: Text: %s\n", yytext);
                break;
        }
    }
    return 0;
}
```

3 Makefile

The Makefile automates the process of compiling the lexer, making it easier to build and test the program.

3.0.1 Makefile Code

```
CC=gcc
LEX=flex

all: lexer

lexer: lex.yy.c main.c
    $(CC) -o lexer lex.yy.c main.c -lfl

lex.yy.c: lexer.l
    $(LEX) lexer.l

clean:
    rm -f lexer lex.yy.c
```

4 Test Input File (Cabdifatah_Mohamed_A2.nc)

This file contains test cases that verify the lexer's functionality. For example:

```
PROGRAM
INTEGER
READ
PRINT
IF
THEN
ELSE
DO
```

```
END
123
abc
"string"
.GT.
.LT.
.EQ.
```