

Assignment 2: F-15 Lexer using Flex

Author: Jatin Dangi **Course:** Programming Language Design and Implementation (CS-1319-1)

1. Introduction

This document describes the design and implementation of a lexical analyzer (lexer) for **F-15**, using **Flex**.

The assignment consists of several tasks: writing the Flex specification (`jatin_dangi_A2.l`), creating a `main()` function to test the lexer, preparing a Makefile, and generating a test input file (`jatin_dangi_A2.nc`). This also tells my design choices, the structure of my lexer, and its results.

2. Design of the Lexer

2.1 Token Definitions

The Flex specification (`jatin_dangi_A2.l`) defines regular expressions to recognize various tokens in F-15. These tokens include:

- **Keywords:** Recognized with exact matching, such as `PROGRAM`, `INTEGER`, `READ`, `PRINT`, `IF`, `THEN`, `ELSE`, `DO`, `END`.

Example:

```
PROGRAM      { printf("PROGRAM keyword\n"); }
INTEGER      { printf("INTEGER keyword\n"); }
```

- **Identifiers:** Variable names are represented by lower-case letters and recognized using the pattern `[a-z]+`.

Example:

```
[a-z]+      { printf("Identifier: %s\n", yytext); }
```

- **Numbers:** Integer constants are matched using the pattern `[0-9]+`.

Example:

```
[0-9]+      { printf("Number: %s\n", yytext); }
```

- **Strings:** String constants, which are enclosed in double quotes, are matched by `\"[^\"]*\"`.

Example:

```
\"[^\"]*\"    { printf("String: %s\n", yytext); }
```

- **Arithmetic Operators:** The operators `+` and `-` are recognized as `PLUS` and `MINUS`, respectively.

Example:

```
"+"          { printf("PLUS\n"); }
"-"          { printf("MINUS\n"); }
```

- **Relational Operators:** F-15 uses `.GT.`, `.LT.`, and `.EQ.`, which are mapped to `GREATER THAN`, `LESS THAN`, and `EQUAL TO`.

Example:

```
".GT."        { printf("GREATER THAN\n"); }
".LT."        { printf("LESS THAN\n"); }
".EQ."        { printf("EQUAL TO\n"); }
```

- **Delimiters and Comments:** Commas, parentheses, and comments are handled as well. Comments beginning with `!` are ignored.

Example:

```
",,"          { printf(",\n"); }
"!("[^\n])*    { /* Ignore comments */ }
```

2.2 Handling Newlines and Whitespace

Newlines and whitespace are ignored to prevent unnecessary tokenization:

```
\n          { /* ignore newline */ }  
[ \t]      { /* ignore whitespace */ }
```

2.3 Error Handling

Any characters not fitting the defined patterns are reported as "Unknown characters."

3. Main Function (`jatin_dangi_A2.c`)

The `main()` function serves as the driver to run the lexer on input files. It reads input files provided as command-line arguments and invokes the `yylex()` function generated by Flex.

```
# include <stdio.h>  
  
extern int yylex();  
extern FILE *yyin;  
  
int main(int argc, char **argv) {  
    printf("Starting lexer...\n");  
  
    if (argc > 1) {  
        printf("Opening input file: %s\n", argv[1]);  
        yyin = fopen(argv[1], "r");  
        if (!yyin) {  
            perror("Error opening file");  
            return 1;  
        }  
    }  
  
    printf("Calling lexer...\n");  
    yylex(); // Call the lexer  
  
    if (yyin) {  
        fclose(yyin);  
    }  
  
    printf("Finished lexing.\n");  
  
    return 0;  
}
```

- **yyin**: This variable is used to associate an input file with the lexer.
- **yylex()**: This function is called to start tokenizing the input.

4. Makefile

The following Makefile automates the process of compiling the lexer using Flex and GCC.

```
lexer: jatin_dangi_A2.l jatin_dangi_A2.c
    flex jatin_dangi_A2.l
    gcc lex.yy.c jatin_dangi_A2.c -o lexer
```

- **Command Breakdown:**
 - `flex jatin_dangi_A2.l`: Generates the C file `lex.yy.c` from the Flex specification.
 - `gcc lex.yy.c jatin_dangi_A2.c -o lexer`: Compiles both the generated lexer and the main function into an executable called `lexer`.

5. Test Input File (`jatin_dangi.nc`)

The test input file is designed to exercise all the lexical rules defined in the lexer. It contains two sample F-15 programs: `absdiff` and `sum`.

Content of `jatin_dangi.nc` : that was given in the Fortran-15 language in the assignment

```
PROGRAM absdiff
INTEGER a
INTEGER b
INTEGER c
READ *, a, b
IF (a .GT. b) THEN
    c = a - b
ELSE
    IF (a .LT. b) THEN
        c = b - a
    ELSE
        c = 0
    END IF
END IF
PRINT *, "Absolute Difference of", a, "and", b, "is", c
END PROGRAM absdiff
```

```
PROGRAM sum
INTEGER n
INTEGER i
INTEGER s
READ *, n
s = 0
DO i = 1, n
    s = s + i
END DO
PRINT *, "Sum of first n natural numbers is", s
END PROGRAM sum
```

6. Output

Here's the output produced when running the lexer on the test input file (jatin_dangi_A2.nc):

```
PROGRAM keyword
Identifier: absdiff
INTEGER keyword
Identifier: a
INTEGER keyword
Identifier: b
INTEGER keyword
Identifier: c
READ keyword
*,
Identifier: a
,
Identifier: b
IF keyword
(Identifier: a
GREATER THAN
Identifier: b
)THEN keyword
Identifier: c
=
Identifier: a
MINUS
```

```
Identifier: b
ELSE keyword
IF keyword
(Identifier: a
LESS THAN
Identifier: b
)THEN keyword
Identifier: c
=
Identifier: b
MINUS
Identifier: a
ELSE keyword
Identifier: c
=
Number: 0
END keyword
IF keyword
END keyword
IF keyword
PRINT keyword
*,
String: "Absolute Difference of"
,
Identifier: a
,
String: "and"
,
Identifier: b
,
String: "is"
,
Identifier: c
END keyword
PROGRAM keyword
Identifier: absdiff
PROGRAM keyword
Identifier: sum
INTEGER keyword
Identifier: n
INTEGER keyword
```

```
Identifier: i
INTEGER keyword
Identifier: s
READ keyword
*,
Identifier: n
Identifier: s
=
Number: 0
DO keyword
Identifier: i
=
Number: 1
,
Identifier: n
Identifier: s
=
Identifier: s
PLUS
Identifier: i
END keyword
DO keyword
PRINT keyword
*,
String: "Sum of first n natural numbers is"
,
Identifier: s
END keyword
PROGRAM keyword
Identifier: sum
```

The lexer correctly identifies all tokens, including keywords, identifiers, numbers, relational operators, and string literals. It also handles delimiters such as commas and asterisks.

7. Conclusion

The lexer for F-15 successfully tokenizes the provided input and recognizes all elements of the F-15 grammar. It processes keywords, identifiers, numbers, and operators effectively and outputs the corresponding tokens. The test case demonstrates that the lexer functions as intended.
