Ex-no:2 Name: M Rohith
24-08-2024 3122 21 5001 085

IMPLEMENTATION OF LEXICAL ANALYSER AND SYMBOL TABLE USING LEX

Develop a scanner that will recognize all the above specified tokens. Test your program for all specified tokens.

Code:

Lexer.l

```
%{
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define MAX_SYMBOLS 100

typedef struct {
    char name[50];
    char type[10];
    int bytes;
    int address;
    int value;
} Symbol;

Symbol symbolTable[MAX_SYMBOLS];
int symbolCount = 0;
int currentAddress = 1000;

void addSymbol(char *name, char *type, int bytes, int value);
void printSymbolTable();
int lookup_keyword(char *s);
int isFunction(char *s);
%}
```

Ex-no:2 Name: M Rohith
24-08-2024 3122 21 5001 085

```
digit [0-9]

letter [A-Za-z_]

id {letter}({letter}|{digit})*

int_const {digit}+

string \"[^\"]*\"

keyword
(auto|break|case|char|const|continue|default|do|double|else|enum|extern|float|for|goto|if|int
|long|register|return|short|signed|sizeof|static|struct|switch|typedef|union|unsigned|void|volat
ile|while)


%%


{keyword} {
   printf("keyword: %s\n", yytext);
}


{id} {
   if (!isFunction(yytext) && !lookup_keyword(yytext)) {
      printf("identifier: %s\n", yytext);
      addSymbol(yytext, "int", 4, 0); // Assuming type int and default value 0 for simplicity
   } else if (isFunction(yytext)) {
      printf("function: %s\n", yytext);
   }
}


{int_const} {
   printf("integer constant: %s\n", yytext);
   // Do not add to symbol table
}


{string} {
   printf("string constant: %s\n", yytext);
```

```
    // Do not add to symbol table
}

"==" | "!=" | "<=" | ">=" | "<" | ">" { printf("relational operator: %s\n", yytext); }
"+" | "-" | "*" | "/" | "%" { printf("arithmetic operator: %s\n", yytext); }
"+=" | "-=" | "*=" | "/=" | "%=" { printf("arithmetic assignment operator: %s\n", yytext); }
"&&" | "||" | "!" { printf("logical operator: %s\n", yytext); }
"&" | "|" | "^" | "<<" | ">>" { printf("bitwise operator: %s\n", yytext); }
"++" | "--" | "=" { printf("assignment operator: %s\n", yytext); }
";" | "," | "." | "[" | "]" | "(" | ")" | "{" | "}" { printf("special character: %s\n", yytext); }
"//".* { /* skip single-line comments */ }
"/*"([^*]|\*+([^*/]))*"*/" { /* skip multi-line comments */ }
[ \t\n]+ { /* skip whitespace */ }
. { printf("other: %s\n", yytext); }

%%

void addSymbol(char *name, char *type, int bytes, int value) {
    for (int i = 0; i < symbolCount; i++) {
        if (strcmp(symbolTable[i].name, name) == 0) {
            return; // Already in the symbol table, don't add again
        }
    }
    if (symbolCount < MAX_SYMBOLS) {
        strcpy(symbolTable[symbolCount].name, name);
        strcpy(symbolTable[symbolCount].type, type);
        symbolTable[symbolCount].bytes = bytes;
        symbolTable[symbolCount].address = currentAddress;
        symbolTable[symbolCount].value = value;
        currentAddress += bytes; // Increment the address for the next symbol
        symbolCount++;
```

```
    }
}

void printSymbolTable() {
    printf("\nContent of Symbol Table\n");
    printf("%-15s%-10s%-15s%-10s%-10s\n", "Identifier", "Type", "No of bytes", "Address", "Value");
    for (int i = 0; i < symbolCount; i++) {
        printf("%-15s%-10s%-15d%-10d%-10d\n", symbolTable[i].name, symbolTable[i].type,
symbolTable[i].bytes, symbolTable[i].address, symbolTable[i].value);
    }
}

int lookup_keyword(char *s) {
    static const char *keywords[] = {
        "auto", "break", "case", "char", "const", "continue", "default", "do", "double",
        "else", "enum", "extern", "float", "for", "goto", "if", "int", "long", "register",
        "return", "short", "signed", "sizeof", "static", "struct", "switch", "typedef",
        "union", "unsigned", "void", "volatile", "while", NULL
    };
    for (int i = 0; keywords[i] != NULL; i++) {
        if (strcmp(s, keywords[i]) == 0)
            return 1;
    }
    return 0;
}

int isFunction(char *s) {
    static const char *functions[] = {
        "printf", "scanf", "main", NULL
    };
    for (int i = 0; functions[i] != NULL; i++) {
```

```
        if (strcmp(s, functions[i]) == 0)
            return 1;
    }
    return 0;
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_file.c>\n", argv[0]);
        exit(1);
    }

    FILE *inputFile = fopen(argv[1], "r");
    if (!inputFile) {
        perror("Could not open file");
        exit(1);
    }

    yyin = inputFile;
    yylex();
    fclose(inputFile);

    printSymbolTable(); // Print the symbol table at the end

    return 0;
}



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

Ex-no:2 Name: M Rohith

24-08-2024 3122 21 5001 085

Input.c

```
{
   int a = 10, b = 20;
   if (a > b)
      printf("a is greater");
   else
      printf("b is greater");
}
```

Output:

```
PS C:\Rohith\Backup\Desktop\SEM 7\UCS2702---Compiler Design(TCP) Lab\Ex-2 Implementation of lexical analyzer using LEX tool> flex Lexer.l
PS C:\Rohith\Backup\Desktop\SEM 7\UCS2702---Compiler Design(TCP) Lab\Ex-2 Implementation of lexical analyzer using LEX tool> gcc lex.yy.c -o la
PS C:\Rohith\Backup\Desktop\SEM 7\UCS2702---Compiler Design(TCP) Lab\Ex-2 Implementation of lexical analyzer using LEX tool> .\la input.c
other: {
keyword: int
identifier: a
other: =
integer constant: 10
other: ,
identifier: b
other: =
integer constant: 20
keyword: if
other: (
identifier: a
other: >
identifier: b
other: )
function: printf
other: (
string constant: "a is greater"
other: )
keyword: else
function: printf
other: (
string constant: "b is greater"
other: )
other: }

Content of Symbol Table
Identifier     Type      No of bytes    Address   Value
a              int       4              1000      0
b              int       4              1004      0
PS C:\Rohith\Backup\Desktop\SEM 7\UCS2702---Compiler Design(TCP) Lab\Ex-2 Implementation of lexical analyzer using LEX tool>
```