System Software and Compiler Design

Lab Assignment 5

Name: Tushar Mittal

PRN: 1032200956

Roll No: PB68

Panel: B

Batch: B2

Code:

app.l

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

#define MAX_IDENTIFIERS 100

// Enum to represent different symbol types
enum SymbolType {
```

```
    USER_VARIABLE,
    USER_FUNCTION
};

// Structure to represent a symbol table entry
typedef struct {
    char name[50];
    enum SymbolType type;
    char dataType[50]; // Added dataType field
} SymbolTableEntry;

SymbolTableEntry symbol_table[MAX_IDENTIFIERS];  // Symbol table array
int symbol_count = 0;  // Counter for symbol table entries
int parsing_complete = 0; // Flag to indicate when parsing is complete

char lastDataType[50] = ""; // Added lastDataType variable

// Function to add an entry to the symbol table
void addToSymbolTable(char* name, enum SymbolType type, char* dataType) {
    if (symbol_count < MAX_IDENTIFIERS) {
        strcpy(symbol_table[symbol_count].name, name);
        symbol_table[symbol_count].type = type;
        strcpy(symbol_table[symbol_count].dataType, dataType);
        symbol_count++;
    } else {
        printf("Error: Symbol table is full.\n");
    }
}

// Function to print the symbol table
void printSymbolTable() {
```

```
    printf("\nSymbol Table:\n");
    printf("--------------\n");
    for (int i = 0; i < symbol_count; i++) {
        printf("Name: %s, Type: %s, DataType: %s\n",
            symbol_table[i].name,
            (symbol_table[i].type == USER_VARIABLE) ? "User Variable" : "User Function",
            symbol_table[i].dataType);
    }
}

// Function to check for duplicate entries in the symbol table
int isDuplicate(char* name, enum SymbolType type, char* dataType) {
    for (int i = 0; i < symbol_count; i++) {
        if (strcmp(symbol_table[i].name, name) == 0 &&
            symbol_table[i].type == type &&
            strcmp(symbol_table[i].dataType, dataType) == 0) {
            return 1; // Entry already exists
        }
    }
    return 0; // Entry does not exist
}

%}

END_CHARACTER \\(0)
REAL_NUMBER [0-9]+"."?[0-9]*
DATATYPE (?i:(int|float|double|char|long|short|signed|unsigned))
KEYWORD (if|else|while|for|return|void|main)
STANDARDIO (?i:(printf|scanf))
PREPROCESSOR \#(include|define|ifdef|endif)
BRACKETS [{}()]
```

```
ID [a-zA-Z_][a-zA-Z0-9_]
DIGIT [0-9]
HEADER <[a-zA-Z0-9_]+\.h>
CLASS [A-Z][a-zA-Z0-9]*
RELATIONAL (==|!=|<=|>=|<|>)
LOGICAL (&&|\|\|)
BITWISE ([&|])
COMMENT "//".*|\n
PUNCTUATION [;,."]
OPERATOR [+*/-=]
USER_FUNCTION [a-zA-Z_][a-zA-Z0-9_]*\(\)
USER_VARIABLE [a-zA-Z_][a-zAZ0-9_]*

%%
{END_CHARACTER} {
    parsing_complete = 1;
    return 0;
}
{DIGIT}+   { printf("INTEGER: %s\n", yytext); }
{ID}       { printf("IDENTIFIER: %s\n", yytext); }
{HEADER}   { printf("HEADER: %s\n", yytext); }
{CLASS}    { printf("CLASS: %s\n", yytext); }
{KEYWORD}  { printf("KEYWORD: %s\n", yytext); }
{RELATIONAL} { printf("RELATIONAL: %s\n", yytext); }
{DATATYPE} {
    printf("DATATYPE: %s\n", yytext);
    strcpy(lastDataType, yytext); // Added lastDataType assignment
}
{LOGICAL}   { printf("LOGICAL: %s\n", yytext); }
{BITWISE}   { printf("BITWISE: %s\n", yytext); }
{COMMENT}  ;  // Skip comments
```

```
{PREPROCESSOR} { printf("PREPROCESSOR: %s\n", yytext); }
{REAL_NUMBER} { printf("REAL_NUMBER: %s\n", yytext); }
{BRACKETS} { printf("BRACKETS: %s\n", yytext); }
{PUNCTUATION} { printf("PUNCTUATION: %s\n", yytext); }
{OPERATOR} { printf("OPERATOR: %s\n", yytext); }
{STANDARDIO} { printf("STANDARDIO: %s\n", yytext); }


{USER_FUNCTION} {
    printf("USER_FUNCTION: %s\n", yytext);
    if (!isDuplicate(yytext, USER_FUNCTION, lastDataType)) {
        addToSymbolTable(yytext, USER_FUNCTION, lastDataType);
    }
}

{USER_VARIABLE} {
    printf("USER_VARIABLE: %s\n", yytext);
    if (!isDuplicate(yytext, USER_VARIABLE, lastDataType)) {
        addToSymbolTable(yytext, USER_VARIABLE, lastDataType);
    }
}


%%

int main(){
    yylex();
    yywrap();
    if (parsing_complete) {
        printSymbolTable();  // Print the symbol table at the end of parsing
    }
```

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

Input:

sample.c

```
#include <stdio.h>

int print()
{
    printf("");

    return 0;
}

int main ()
{
    int a, b;
    int c = a + b;
    printf(a, b, c);

    return 0;
}
```

Output:

```
tusharmittal@LAPTOP-ONVQFQKH:/mnt/c/Users/Tushar Mittal/Desktop/ssc/Assignment5$ flex app.l
tusharmittal@LAPTOP-ONVQFQKH:/mnt/c/Users/Tushar Mittal/Desktop/ssc/Assignment5$ gcc lex.yy.c -o a.out
tusharmittal@LAPTOP-ONVQFQKH:/mnt/c/Users/Tushar Mittal/Desktop/ssc/Assignment5$ ./a.out
#include <stdio.h>
PREPROCESSOR: #include
 HEADER: <stdio.h>
int print()
DATATYPE: int
 USER_FUNCTION: print()
{
BRACKETS: {
printf("");
STANDARDIO: printf
BRACKETS: (
PUNCTUATION: "
PUNCTUATION: "
BRACKETS: )
PUNCTUATION: ;
 return 0;
 KEYWORD: return
 INTEGER: 0
PUNCTUATION: ;
}
BRACKETS: }
int main ()
DATATYPE: int
 KEYWORD: main
 BRACKETS: (
BRACKETS: )
{
BRACKETS: {
int a, b;
DATATYPE: int
 USER_VARIABLE: a
PUNCTUATION: ,
 USER_VARIABLE: b
PUNCTUATION: ;
int c = a + b;
DATATYPE: int
 USER_VARIABLE: c
 OPERATOR: =
 USER_VARIABLE: a
 OPERATOR: +
 USER_VARIABLE: b
PUNCTUATION: ;
```

```
printf(a, b, c);
STANDARDIO: printf
BRACKETS: (
USER_VARIABLE: a
PUNCTUATION: ,
 USER_VARIABLE: b
PUNCTUATION: ,
 USER_VARIABLE: c
BRACKETS: )
PUNCTUATION: ;
return 0;
KEYWORD: return
 INTEGER: 0
PUNCTUATION: ;
}
BRACKETS: }
\0

Symbol Table:
--------------
Name: print(), Type: User Function, DataType: int
Name: a, Type: User Variable, DataType: int
Name: b, Type: User Variable, DataType: int
Name: c, Type: User Variable, DataType: int
tusharmittal@LAPTOP-ONVQFQKH:/mnt/c/Users/Tushar Mittal/Desktop/ssc/Assignment5$
```