Friday, 20 October 2023

Project title: Toy Programming Language in Hindi/Gen Z Slang
Assignment: Lexical Analysis of 50 functions
Course: Compiler Design
Instructor: Saurabh Shanu
Student: Vaishnavi Srivastava
SAP ID: 500095279
Batch: B.Tech CSE Blockchain

Github link: https://github.com/Vaishnavi024/Compiler-Design

Screenshot of lexer code:

lexer.c

Functions > lexer.c > isRealNumber(char *)

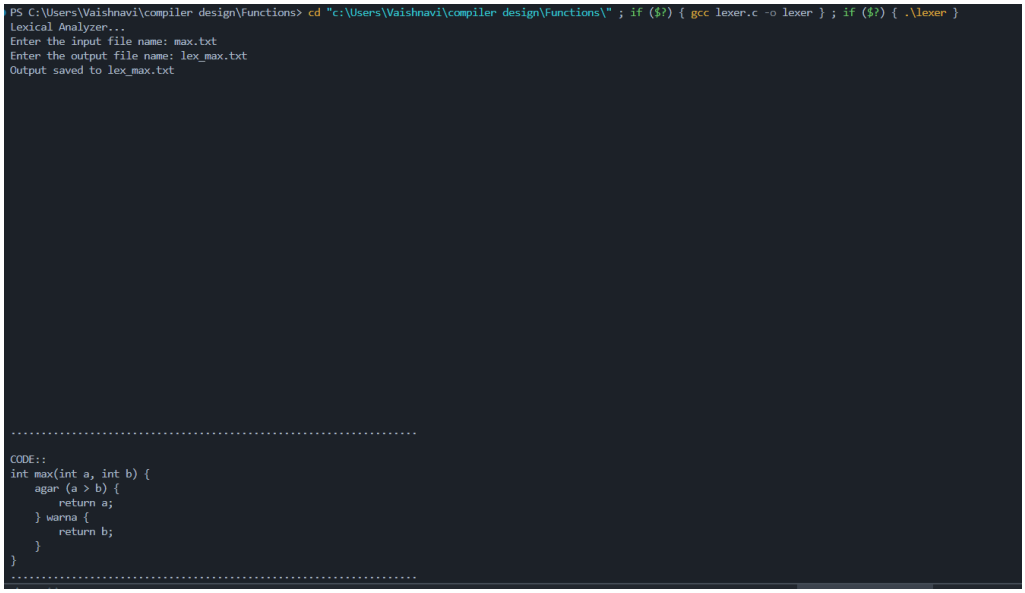
```c
#include <stdbool.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

// Returns 'true' if the character is a DELIMITER.
bool isDelimiter(char ch)
{
    if (ch == ' ' ||ch == '#'|| ch == '$'|| ch == '%' || ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
        ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
        ch == '[' || ch == ']' || ch == '{' || ch == '}'|| ch == '"' )
        return true;
    return false;
}
// Returns 'true' if the string is a VALID IDENTIFIER.
bool validIdentifier(char* str)
{
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isDelimiter(str[0]) == true)
        return false;

    for (int i = 0; str[i] != '\0'; i++) {
        if (!isalnum(str[i]) && str[i] != '_') {
            return false;
        }
    }
    return true;
}
// Returns 'true' if the character is an OPERATOR.
```

```
bool isOperator(char ch)
{
    if (ch == '+' || ch == '!' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '='|| ch == '&'||ch == '%')
        return true;
    return false;
}
// Returns 'true' if the string is a KEYWORD.
bool isKeyword(char* str)
{
    const char* keywords[] = {
        "if", "else", "while", "do", "break",
        "continue", "int", "double", "float",
        "return", "char", "case", "sizeof",
        "long", "short", "typedef", "switch",
        "unsigned", "void", "static", "struct",
        "goto","agar","warna","jabtak","kabtak","karo"
    };
    int numKeywords = sizeof(keywords) / sizeof(keywords[0]);

    for (int i = 0; i < numKeywords; i++) {
        if (strcmp(str, keywords[i]) == 0)
            return true;
    }
    return false;
}
// Returns 'true' if the string is an INTEGER.
bool isInteger(char* str)
{
    int len = strlen(str);
    if (len == 0)
        return false;
    for (int i = 0; i < len; i++) {
        if (str[i] < '0' || str[i] > '9')
            return false;
    }
    return true;
}
// Returns 'true' if the string is a REAL NUMBER.
bool isRealNumber(char* str)
{
    int len = strlen(str);
    bool hasDecimal = false;

    if (len == 0)
        return false;
```

```
    if (len == 0)
        return false;
    for (int i = 0; i < len; i++) {
        if ((str[i] < '0' || str[i] > '9') && str[i] != '.') {
            if (i == 0 && str[i] == '-')
                continue;
            return false;
        }
        if (str[i] == '.')
            hasDecimal = true;
    }
    return hasDecimal;
}
// Extracts the SUBSTRING.
char* subString(char* str, int left, int right)
{
    int i;
    char* subStr = (char*)malloc(
        sizeof(char) * (right - left + 2));

    for (i = left; i <= right; i++)
        subStr[i - left] = str[i];
    subStr[right - left + 1] = '\0';
    return subStr;
}
// Parsing the input STRING.
void parseAndWrite(FILE* inputFile, FILE* outputFile) {
  char line[1000];
  while (fgets(line, sizeof(line), inputFile)) {
    int len = strlen(line);
    if (len > 0 && line[len - 1] == '\n') {
      line[len - 1] = '\0'; // Remove the newline character if present
    }
    fprintf(outputFile, "%s\n", line);
    int left = 0, right = 0;
    len = strlen(line);
    while (right <= len && left <= right) {
      if (isDelimiter(line[right]) == false) {
        right++;
      } else if (isDelimiter(line[right]) == true && left == right) {
        if (isOperator(line[right]) == true) {
          fprintf(outputFile, "'%c' IS AN OPERATOR\n", line[right]);
        } else {
          // Print delimiters, including whitespace
          fprintf(outputFile, "'%c' IS A DELIMITER\n", line[right]);
        }
        right++;
        left = right;
      } else if (isDelimiter(line[right]) == true && left != right || (right == len && left != right)) {
```

```
        char* subStr = subString(line, left, right - 1);
        if (isKeyword(subStr) == true) {
          fprintf(outputFile, "'%s' IS A KEYWORD\n", subStr);
        } else if (isInteger(subStr) == true) {
          fprintf(outputFile, "'%s' IS AN INTEGER\n", subStr);
        } else if (isRealNumber(subStr) == true) {
          fprintf(outputFile, "'%s' IS A REAL NUMBER\n", subStr);
        } else if (validIdentifier(subStr) == true && isDelimiter(line[right - 1]) == false) {
          fprintf(outputFile, "'%s' IS A VALID IDENTIFIER\n", subStr);
        } else if (validIdentifier(subStr) == false && isDelimiter(line[right - 1]) == false) {
          fprintf(outputFile, "'%s' IS NOT A VALID IDENTIFIER\n", subStr);
        }
        left = right;
      }
    }
    fprintf(outputFile, "\n"); // Add a line space between lines
  }
  return;
}

int main() {
    printf("Lexical Analyzer...\n");
    printf("Enter the input file name: ");
    char filename[100];
    scanf("%s", filename);
    // Open the input file
    FILE* inputFile = fopen(filename, "r");
    if (inputFile == NULL) {
        printf("Input file not found or unable to open.\n");
        return 1;
    }

    printf("Enter the output file name: ");
    char outputFilename[100];
    scanf("%s", outputFilename);

    // Open the output file for writing
    FILE* outputFile = fopen(outputFilename, "w");
    if (outputFile == NULL) {
        printf("Unable to create or open the output file.\n");
        return 1;
    }

    parseAndWrite(inputFile, outputFile); // Process and write input to the output file

    fclose(inputFile);
```

Screenshot of code output as a sample output:

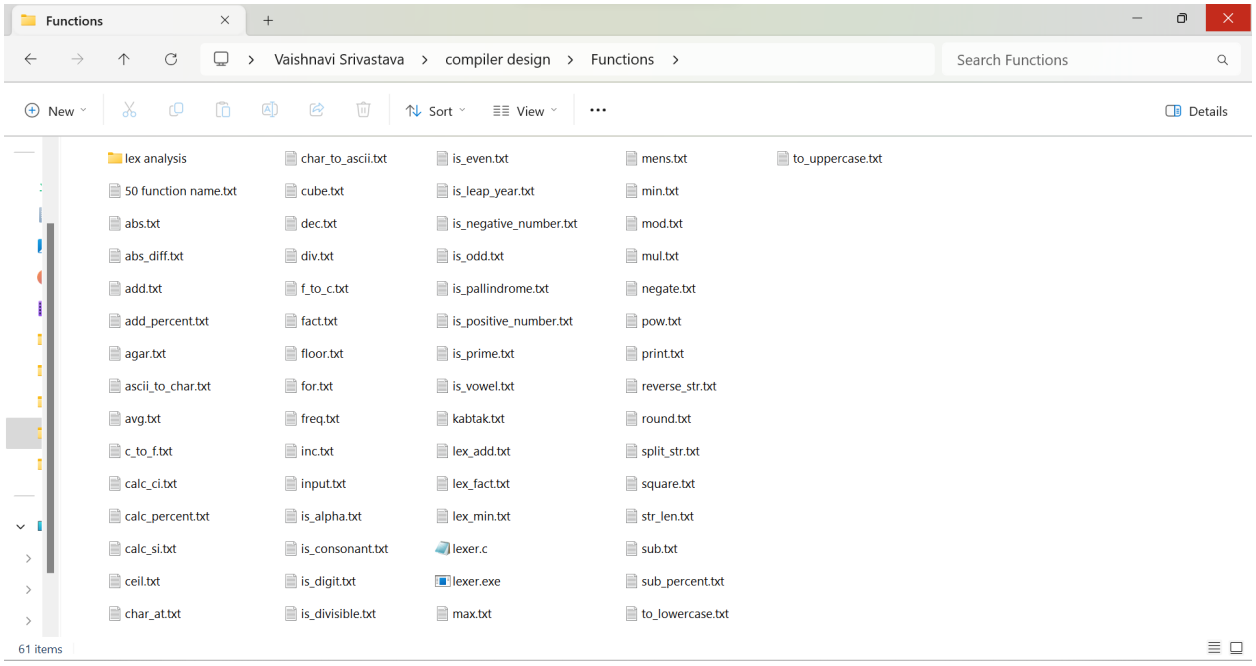


```
PS C:\Users\Vaishnavi\compiler design\Functions> cd "c:\Users\Vaishnavi\compiler design\Functions\" ; if ($?) { gcc lexer.c -o lexer } ; if ($?) { .\lexer }
Lexical Analyzer...
Enter the input file name: max.txt
Enter the output file name: lex_max.txt
Output saved to lex_max.txt

.....

CODE::
int max(int a, int b) {
    agar (a > b) {
        return a;
    } warna {
        return b;
    }
}
.....
```

```
CODE::
int max(int a, int b) {
    agar (a > b) {
        return a;
    } warna {
        return b;
    }
}
.....

LEXICAL ANALYSIS::
int max(int a, int b) {
'int' IS A KEYWORD
' ' IS A DELIMITER
'max' IS A VALID IDENTIFIER
'(' IS A DELIMITER
'int' IS A KEYWORD
' ' IS A DELIMITER
'a' IS A VALID IDENTIFIER
',' IS A DELIMITER
' ' IS A DELIMITER
'int' IS A KEYWORD
' ' IS A DELIMITER
'b' IS A VALID IDENTIFIER
')' IS A DELIMITER
' ' IS A DELIMITER
'{' IS A DELIMITER

    agar (a > b) {
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
'agar' IS A KEYWORD
' ' IS A DELIMITER
'(' IS A DELIMITER
'a' IS A VALID IDENTIFIER
' ' IS A DELIMITER
'>' IS AN OPERATOR
' ' IS A DELIMITER
```

```
' ' IS A DELIMITER
'a' IS A VALID IDENTIFIER
';' IS A DELIMITER

    } warna {
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
'}' IS A DELIMITER
' ' IS A DELIMITER
'warna' IS A KEYWORD
' ' IS A DELIMITER
'{' IS A DELIMITER

        return b;
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
'return' IS A KEYWORD
' ' IS A DELIMITER
'b' IS A VALID IDENTIFIER
';' IS A DELIMITER

    }
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
' ' IS A DELIMITER
'}' IS A DELIMITER

}
'}' IS A DELIMITER
```

Screenshot of folder containing 53 functions written in text file:



Screenshot of folder containing the saved output of the codes in text files: