**Exp 1**

**Lexical Analysis**

Sai Ram Pendyala

RA1911003010696

**AIM**:

To write a program to implement a lexical analyser.

**ALGORITHM:**

1. Start.

2. Get the input program from the file prog.txt.

3. Read the program line by line and check if each word in a line is a keyword, identifier,

constant or an operator.

4. If the word read is an identifier, assign a number to the identifier and make an entry into

the symbol table stored in sybol.txt.

5. For each lexeme read, generate a token as follows:

a. If the lexeme is an identifier, then the token generated is of the form <id, number>

b. If the lexeme is an operator, then the token generated is <op, operator>.

c. If the lexeme is a constant, then the token generated is <const, value>.

d. If the lexeme is a keyword, then the token is the keyword itself.

6. The stream of tokens generated are displayed in the console output.

7. Stop.

**PROGRAM:**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

*bool* isDelimiter(*char* *ch*)

{

    if (*ch* == ' ' || *ch* == '+' || *ch* == '-' || *ch* == '\*' ||

*ch* == '/' || *ch* == ',' || *ch* == ';' || *ch* == '>' ||

*ch* == '<' || *ch* == '=' || *ch* == '(' || *ch* == ')' ||

*ch* == '[' || *ch* == ']' || *ch* == '{' || *ch* == '}')

        return (true);

    return (false);

}

*bool* isSpecialCharacter(*char* *ch*)

{

    if (*ch* == ',' || *ch* == ';' || *ch* == '>' ||

*ch* == '<' || *ch* == '=' || *ch* == '(' || *ch* == ')' ||

*ch* == '[' || *ch* == ']' || *ch* == '{' || *ch* == '}')

        return (true);

    return (false);

}

*bool* isOperator(*char* *ch*)

{

    if (*ch* == '+' || *ch* == '-' || *ch* == '\*' ||

*ch* == '/' || *ch* == '>' || *ch* == '<' ||

*ch* == '=')

        return (true);

    return (false);

}

*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);

    return (true);

}

*bool* isKeyword(*char*\* *str*)

{

    if (!strcmp(*str*, "if") || !strcmp(*str*, "else") ||

        !strcmp(*str*, "while") || !strcmp(*str*, "do") ||

        !strcmp(*str*, "break") ||

        !strcmp(*str*, "continue") || !strcmp(*str*, "int")

        || !strcmp(*str*, "double") || !strcmp(*str*, "float")

        || !strcmp(*str*, "return") || !strcmp(*str*, "char")

        || !strcmp(*str*, "case") || !strcmp(*str*, "char")

        || !strcmp(*str*, "sizeof") || !strcmp(*str*, "long")

        || !strcmp(*str*, "short") || !strcmp(*str*, "typedef")

        || !strcmp(*str*, "switch") || !strcmp(*str*, "unsigned")

        || !strcmp(*str*, "void") || !strcmp(*str*, "static")

        || !strcmp(*str*, "struct") || !strcmp(*str*, "goto"))

        return (true);

    return (false);

}

*bool* isInteger(*char*\* *str*)

{

*int* i, len = strlen(*str*);

    if (len == 0)

        return (false);

    for (i = 0; i < len; i++) {

        if (*str*[i] != '0' && *str*[i] != '1' && *str*[i] != '2'

            && *str*[i] != '3' && *str*[i] != '4' && *str*[i] != '5'

            && *str*[i] != '6' && *str*[i] != '7' && *str*[i] != '8'

            && *str*[i] != '9' || (*str*[i] == '-' && i > 0))

            return (false);

    }

    return (true);

}

*bool* isRealNumber(*char*\* *str*)

{

*int* i, len = strlen(*str*);

*bool* hasDecimal = false;

    if (len == 0)

        return (false);

    for (i = 0; i < len; i++) {

        if (*str*[i] != '0' && *str*[i] != '1' && *str*[i] != '2'

            && *str*[i] != '3' && *str*[i] != '4' && *str*[i] != '5'

            && *str*[i] != '6' && *str*[i] != '7' && *str*[i] != '8'

            && *str*[i] != '9' && *str*[i] != '.' ||

            (*str*[i] == '-' && i > 0))

            return (false);

        if (*str*[i] == '.')

            hasDecimal = true;

    }

    return (hasDecimal);

}

*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);

}

*void* parse(*char*\* *str*)

{

*int* left = 0, right = 0;

*int* len = strlen(*str*);

    while (right <= len && left <= right) {

        if (isDelimiter(*str*[right]) == false)

            right++;

        if (isDelimiter(*str*[right]) == true && left == right) {

            if (isOperator(*str*[right]) == true)

                printf("'%c' IS AN OPERATOR\n", *str*[right]);

            else if (isSpecialCharacter(*str*[right]) == true)

                printf("'%c' IS A SPECIAL CHARACTER\n", *str*[right]);

            right++;

            left = right;

        }

        else if (isDelimiter(*str*[right]) == true && left != right

                || (right == len && left != right)) {

*char*\* subStr = subString(*str*, left, right - 1);

            if (isKeyword(subStr) == true)

                printf("'%s' IS A KEYWORD\n", subStr);

            else if (isInteger(subStr) == true)

                printf("'%s' IS AN INTEGER\n", subStr);

            else if (isRealNumber(subStr) == true)

                printf("'%s' IS A REAL NUMBER\n", subStr);

            else if (validIdentifier(subStr) == true

                    && isDelimiter(*str*[right - 1]) == false)

                printf("'%s' IS A VALID IDENTIFIER\n", subStr);

            else if (validIdentifier(subStr) == false

                    && isDelimiter(*str*[right - 1]) == false)

                printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);

            left = right;

        }

    }

    return;

}

*int* main()

{

*char* str[100] = "float a = (int)b + c)";

    printf("\nLEXICAL ANALYSIS:\n\n");

    parse(str);

    return (0);

}

**Result:**

Input - “float a = (int)b + c”;

Output –

