

Exp No :

Roll.No: 210701295

Date :

DEVELOP A LEXICAL ANALYZER TO RECOGNIZE TOKENS IN C

AIM:

To implement the program to identify C keywords, identifiers, operators, end statements like ; , [], {} using C tool.

ALGORITHM:

Step-1: We identify the basic tokens in c such as keywords, numbers, variables, etc.

Step-2: Declare the required header files.

Step-3: Get the input from the user as a string and it is passed to a function for processing.

Step-4: The functions are written separately for each token and the result is returned in the form of bool either true or false to the main computation function.

Step-5: Functions are issymbol() for checking basic symbols such as () etc , isoperator() to check for operators like +, -, *, / , isidentifier() to check for variables like a,b, iskeyword() to check the 32 keywords like while etc., isInteger() to check for numbers in combinations of 0-9, isnumber() to check for digits and substring().

Step-6: Declare a function detecttokens() that is used for string manipulation and iteration then the result is returned from the functions to the main. If it's an invalid identifier error must be printed.

Step-7: Declare main function get the input from the user and pass to detecttokens() function.

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 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]);

            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()
{
    // maximum length of string is 100 here
    char str[100] = "int a = b + 1c; ";

    parse(str); // calling the parse function

    return (0);
}

```

OUTPUT:

```
UKESHWARAN295 :~$ gcc tok1.c
UKESHWARAN295 :~$ ./a.out
-bash: ./a.out: No such file or directory
UKESHWARAN295 :~$ ./a.out
'int' IS A KEYWORD
'a' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
'b' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'1c' IS NOT A VALID IDENTIFIER
UKESHWARAN295 :~$ |
```

RESULT: