Program : 02

30/09/22

Lexical Analyzer Using C

Aim:

To develop a lexical analyzer to identify identifiers, constants, comments, operators etc. using C program.

Algorithm:

Step 1: Start the Program
Step 2: Declare all the variables and file Pointers.
Step 3: Display the input program
Step 4: Separate the keyword in the program and display it.
Step 5: Display the header files of the input program
Step 6: Seperate the operators of the input program and display it.
Step 7: Print the Punctuation marks.
Step 8: Print the constants that are present in input program.
Step 9: Print the identifiers of the input program
Step 10: Stop the program.

Experiment - 3

Calculator Using LEX

Aim:

To write an algorithm for an calculator and execute program using lexical analyzer tool

Algorithm:

Step 1: Take the input string

Step 2: Call the function yylex(). The controll will given to rule section.

Step 3: Define the rules.

dig [0-9]+|([0-9]*)"."([0-9]+)

add "+"

sub "-"

mul "*"

div "/"

pow "^"

Step 4: Check the rules as

{dig} {digi();}

{add} {op=1;}

{sub} {op=2;}

{mul} {op=3;}

{div} {op=4;}

{pow} {op=5;}

{\n} {printf("\n The Answer: %of\n\n", a);}

Switch(op)

```
{
case 1 : a = a+b;
        break;
case 2 : a = a-b;
        break;
case 3 : a = a*b;
        break;
case 4 : a = a/b;
        break;
case 5 : for (i=a; b>1; b--)
        a = a*i;
        break;
} op = 0; }}
```

Step 5: Call yywrap() to wind up the session

Step 6: Stop.

Result:

Program run successfully and output verified

Experiment 4 21/10/22

Counting Vowels And Consonants

Aim:

To write an algorithm for counting vowels and execute the program using lexical analyzer tool.

Algorithm:

Step 1: Start

Step 2: Take the input string

Step 3: Call the function yylex(). The control will given to rule section

Step 4: Check the rule if input is [aeiouAEIOU] then increment count as v++

Else

Step 5: Increment count as c++

Step 6: Output the no of vowels and consonants

Step 7: Stop.

Result:

Program run successfully and output verified.

Experiment - 5 28/10/2022

Counting Number of Words, Lines and Character, etc

Aim:

To write an algorithm for counting number of words, lines and characters and execute the program using lexical analyzer tool.

Algorithm:

Step 1: Start

Step 2: Take a string as input

Step 3: Call yylex() , Step 4: Check for rule section

Step 5: if \n then lc ++

Else check[ ] then sc++

Else if check \t then tc++

Else if characters then ch++

Step 6: Call yywrap for wrapping the lex section

Step 7: Print the output

Step 8: stop

Result:

Program run successfully and output verified

Experiment - 6

Intermediate Code Generation

Aim:

To Implement Intermediate Code generation for simple expression using Program in C

Algorithm:

Step 1: Start the Program
Step 2: Open the input file in read mode
Step 3: Open the output file in write mode
Step 4: In Input file scan for operator, argument 1, argument 2 and result
Step 5: If the operator is '+', move arg1 to R0, Add arg2 and R0, Move R0 to result
Step 6: If the operator is '-', Move arg1 to R0, Subtract arg2 and R0, Mov R0 to result
Step 7: If the operator is '*' move arg1 to R0, Multiply arg2 and R0, Move R0 to result
Step 8: If the operator is '/', move arg1 to R0, Divide arg2 and R0, mov R0 to result
Step 9: If the operator is '=', move arg1 to R0 mov R0 to result
Step 10: Close both the files
Step 11: Stop the Program

Experiment - 7

NFA to DFA Convertion

Aim:

To write an algorithm for NFA to DFA Convertor and execute the Program in C.

Algorithm:

Step 1: Start the Program
Step 2: Input the required array ie, set of alphabets, set of states, instral state, final state,
Step 3: Instrally $Q' = \phi$ ~~and input~~
Step 4: Input the no. of states, final state and rules as input
Step 5: Add q_0 of NFA to Q'. Then find the transitions from this start state.
Step 6: In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.
Step 7: Solving according to DFA
Step 8: In DFA, the final state will be all the states which contain final states of NFA.
Step 9: Print the output state as table
Step 10: Stop

Experiment - 8

Constant Propagation

Aim:

To write an algorithm for Constant Propagation and execute the Program in C.

Algorithm:

Step 1: Start the Program
Step 2: Input the maximum no. of expressions and the inputs
Step 3: Set arr[i].flag = 0
Step 4: Construct a control flow graph (CFG)
Step 5: Associate transfer functions with the edges of CFG
Step 6: At every node, maintain the values of the Program's Variable at that point. Then initialize those to 1
Step 7: Iterate until the values of the variables stabilize
Step 8: Stop

Experiment - 9

YACC Specification to Recognize a Valid Arithmetic Expression

Aim:

To write a Yacc Program to Valid arithmetic expression using Yacc

Algorithm:

Step 1: Start the Program
Step 2: Enter the expressing (Reading)
Step 3: Checking the Validating of the given expression according to the rule using Yacc
Step 4: Using expression rule, Print the result of the given values
Step 5: Stop the Program

Experiment - 10

Back-end of a Compiler

Aim:

To implement the back end of the compiler using C

Algorithm:

Step 1: Start the Program
Step 2: Get / Read the three address code
Step 3: Check the operation in it and copy to Varrable
Step 4: Print MOV and Varrable R
Step 5: Print Operation Second Varrable R
Step 6: Print MOV R, , result Varrable
Step 7: Stop

Experiment - 11

First and Follow

Aim:

To Write a program to find first and follow of any given grammar.

Algorithm

Step 1: Start the program

Step 2: Calculating first, $\alpha \rightarrow t\beta$

Step 3: If α is a terminal, then FIRST(α) = {α}

Step 4: If α is a non-terminal and $\alpha \rightarrow \epsilon$ is a production, then FIRST(α) = {ϵ}

Step 5: If α is a non-terminal and $\alpha \rightarrow r_1 r_2 r_3 \ldots r_n$ and any FIRST(r) contain t then t is in FIRST(α)

Step 6: Calculating Follow

Step 7: If α is a start symbol, then FOLLOW() = $

Step 8: If α is a non-terminal and has a production $\alpha \rightarrow AB$ then FIRST(B) is in FOLLOW(A) except ϵ

Step 9: If α is a non-terminal and has a production $\alpha \rightarrow AB$, where $B \epsilon$, then FOLLOW(A) is in FOLLOW(α)

Step 10: STOP the program.

Experiment - 12

Shift Reduce Parser

Aim:

To construct a shift reduce parser for a given language in C.

Algorithm:

Step 1: Start the program

Step 2: Assume the grammar be:

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow (E)$

$E \rightarrow id$

Step 3: Read input variables

Step 4: ~~Stack~~ Set top element of stack as $

Step 5: Push an element from input string to stack

Step 6: If the pushed element is operator, no action

Step 7: Else, reduce the element in the stack to the given grammar

Step 8: Repeat steps 5 to 7 until input string is $

Step 9: If the input string is $ and stack is reduced to only E, then Accept, Else, Reject

Step 10: STOP.