1) Introduction to lex and yacc

2) Lexical analyzer using C.

3) Lex program ~~to~~ ~~convert~~ of calculator

4) Counting vowels and consonants using Lex

5) Counting No of words, lines etc.

6) Intermediate code generation

7) NFA to DFA convertor

8) program for constant propagation.

9) yacc specification to recognize a valid arithmetic expression

10) Implement back end of a compiler.

11) First and Follow

12) Shift reduce parser.

EXPERIMENT 5: COUNTING NO OF WORDS

ALGORITHM

Step 1: Take input from keyboard

Step 2: call yylex()

Step 3: check for rule section

Step 4: if \n then lc++

Else check [ ] then sc++

Else if check \t then tc++

Else if characters then ch++

Step 5: call yywrap for wrapping the lex section

Step 6: print the output

Step 7: stop

EXPERIMENT 12: IMPLEMENT BACKEND OF A COMPILER

Algorithm

1. Start the program.
2. Get the three variables from statements and stored in the text file k.txt.
3. Compile the program and give the path of the source file.
4. Execute the program.
5. Target code for the given statement was produced.
6. Stop the program.

CONSTANT PROPAGATION

AIM

Write a proagram to perform constant propagation

ALGORITHM:

1. Start
2. Create a simple C program, test.c
3. Create the source file of the file test.c, test.s
4. Analyse the number of steps in the main function in the source file (before optimization)
5. Optimize the file test.c
6. Analyse the reduction in number of steps in the main function in the source file (after optimization)
7. Stop

EXPERIMENT 7: NFA TO DFA CONVERTOR

Aim:

To write an algorithm for NFA to DFA convertor and execute the program in C

Algorithm

Step 1: Initially Q' = ϕ and input the no of states,

final state and rules as input.

Step 2: Add q0 of NFA to Q'. Then find the transitions from this start state.

Step 3: 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 4: solving according to DFA

Step 5: In DFA, the final state will be all the states which contain F(final states of NFA)

Step 6: print the output state as table.

Step 7: stop

INTERMEDIATE CODE GENERATION

AIM: Implement Intermediate code generation for simple expressions.

ALGORITHM

1. Start
2. Read the expression, str
3. Identify the highest precedent operation in the given string according to DMAS rule
4. Assign a temporary variable to this operation
5. Substitute the operation using this temporary variable in the expression
6. Repeat steps 3 to 5 until the expression is reduced to a single temporary variable.
7. Stop

SHIFT REDUCE PARSER

AIM: Construct a Shift Reduce Parser for a given language

ALGORITHM

1. Start
2. Print the given grammer
3. Read the input symbol, ip_sym
4. Set top element of stack as $
5. Push an element from input string to stack
6. If the pushed element is operator, no action
7. Else, reduce the element in the stack to the given grammar
8. Repeat steps 5 to 7 until input string is $
9. If the input string is $ and stack is reduced to only E, then Accept. Else, Reject
10. Stop

Program : 5

26/10/22

5

TOTAL NUMBER OF VOWELS AND CONSONANTS FROM THE INPUT STRING

Aim:

Write a low program to find out total number of vowels and consonants from the given input string.

Algorithm:

1. Start
2. Define a string
3. Compare string with capital letter (A,E,I,O,U) and lower case letters (a,e,i,o,u)
4. If any character in string matches with small letter vowels then ystext[0]-32 and count the number of vowels.
5. If any characters in string matches with capital letter vowels then ystext[0]+32 and count the number of vowels.
6. If any characters lies between 'a' and 'z' except vowels, then count the number of consonants.
7. print both the counts
8. Stop

Result:

Program executed and output generated successfully.

Program: 6

26 / 10 / 22

6

PROGRAM TO RECOGNIZE A VALID ARITHMETIC EXPRESSION

Aim:

To write a YACC program to recognize valid arithmetic expressions

Algorithm:

1. Start
2. Read an expression
3. check the validity using YACC according to the rules
4. Using expression rule, print the result of the given values
5. Stop

Result:

program verified and output generated successfully

Done 4/11/22

Program 3

20/10/22

3

COUNTING THE NUMBER OF CHARACTERS, WORDS AND LINES

Aim:

To write a program to count the number of characters lines and words in a file.

Algorithm:

1. Start
2. Initialize line = 0, characters = 0 and words = 0
3. If data contains a-z or A-Z then increment characters
4. Whenever a space is encountered, increment words
5. Whenever a newline is executed, increment lines.
6. Display count
7. Stop

Result:

Developed a program in lex to count no of characters, words and lines.

14/11/22

Program 11

22/11/22

CONVERT NFA TO DFA

Aim:

Write a program to convert NFA to DFA

Algorithm:

1. Start the program
2. Input the required array ie, set of alphabets, set of states, initial state, set of final states, transitions.
3. Initially $Q' = \phi$
4. Add q_0 to NFA to Q'. Then find the transitions from the start state
5. In Q', find the possible set of states of each input symbol. If this set of states is not in Q', then add it to Q'.
6. In DFA, the final state will be all states which contains Final States of NFA
7. Stop the program

Result:

program executed and output obtained successfully

Program: 8

01 / 11 / 22

8

IMPLEMENTATION OF CALCULATOR USING LEX & YACC

Aim:

To write a program for implementing a calculator for computing the given expression using semantic rules of the YACC tool and LEX.

Algorithm:

1. A YACC source program has three parts as follows: Declaration %% translation rule %% supporting C routines.
2. Declaration section: This section contains entries that:
   (i) Define standard I/O header file
   (ii) Define global variables
   (iii) Define the list rule as the place to start processing
   (iv) Define the tokens provided by parser
   (v) Define the operators and their precedence.
3. Rules section: Defines the rules that parse the I/p stream. Each rule of a grammar production and the associated semantic action.
4. Program section: Subroutines are included
5. Main- The required main program that calls yyparse subroutine to start program
6. yyerror(s): prints error message.
7. yywrap: The wrap-up subroutines that return a value of 1 when the end of I/p occurs
8. Stop

Result:

Program verified and Output generated successfully.

Program : 15

23 / 11 / 22

CONSTANT PROPAGATION

Aim:

Write a program to perform constant propagation

Algorithm:

1. Start the program
2. Construct a control flow graph (CFG)
3. Associate transfer functions with the edges of the CFG.
4. At every node (program point) we maintain the values of the program's variables at that point. We initialize those to 1.
5. Iterate until the values of the variables stabilize
6. Stop the program.

Result:

program executed and output obtained successfully

Program:14

23/11/22

SHIFT REDUCE PARSER

Aim:

To Construct a shift Reduce parser for a given language

Algorithm:

1. Start the program
2. Get the input expression and store it in the input buffer.
3. Read the data from the input buffer one at the time.
4. Using stack and push and pop operation shift and reduce symbols with respect to production rules available.
5. Continue the process till symbol shift and production rule reduce reaches the start symbol.
6. Display the stack implementation table with corresponding stack actions with input symbols.
7. Stop the program.

Result:

program executed and output obtained successfully

Program : 12

22/11/22

FIND FIRST AND FOLLOW OF ANY GIVEN GRAMMER

Aim:

Write a programs to find first and follow of any given grammar

Algorithm:

1. Start the program
2. Calculating first, $\alpha \rightarrow t\beta$
3. If α is a terminal, then FIRST(α) = $\{\alpha\}$
4. If α is a non-terminal and $\alpha \rightarrow \varepsilon_p$ is a production, then FIRST(α) = $\{\varepsilon_p\}$
5. If α is a non-terminal and $\alpha \rightarrow \gamma_1\gamma_2\gamma_3 \ldots \gamma_m$ and only FIRST(γ) contain t then t is in FIRST(α).
6. Calculating follow
7. If α is a start symbol, then FOLLOW() = \$
8. If α is a non-terminal and has a production $\alpha \rightarrow AB$, then FIRST(B) is in FOLLOW(A) except ε_p
9. If α is a non-terminal and has a production $\alpha \rightarrow AB$, where B ε_p, then FOLLOW(A) is in FOLLOW(α)
10. Stop the program.

Result:

program executed and output obtained successfully.

AIM:

To write a Yacc program to valid arithmetic expression using Yacc .

ALGORITHM:

Step1: Start the program.

Step2: Reading an expression .

Step3: Checking the validating of the given expression according to the rule using yacc.

Step4: Using expression rule print the result of the given values

Step5: Stop the program.

Program : 16

23 / 11 / 22

16

INTERMEDIATE CODE GENERATION

Aim:

Implement intermediate code generation for simple expressions.

Algorithm:

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

Result:

program executed and output generated successfully.