COMPILER DESIGN ALGORITHM:

EXP: 1 Token Separation using LEX

1.Declare the function yy error which is laid and taking char as input

2. For letter given[a-z A-Z], for digit[0-9] and operation op[- + *].

3. For keyword, it will print it is a keyword

4. .For character as name, it will print it is an identifier.

5. For operators, it will function as intmain()/ print it is an operator.

6..Declare the main function as intmain().

7. In the main function, call the yylex(); function.

8. In the last return(0);

9. Stop the program.

EXP : 2 Evaluation of Arithmetic expression using Ambiguous Grammar(Use Lex and Yacc Tool)

1. Start the program.
2. .Opening the lex compiler and declare the necessary variables
3. Write the corresponding patterns and actions in rule section and return to YACC.
4. .Save the document with “.L” extensions
5. n the declaration part, write the production rules as patterns and call YY parser() in the main function to do
6. .End the program.

EXP : 3 Evaluation of Arithmetic expression using Unmbiguous Grammar(Use Lex and Yacc Tool)

1. Start the program.
2. Opening the lex compiler and declare the necessary variables.

3. Write the corresponding patterns and actions in rule section and return to YACC.

4 Save the document with ".L" extensions and compile LEX.
5.Open YACC compiler, declare the token and associatively the string of the input operators. 6.In the declaration part, write the production rules as patterns and call YY parser() in the main function for parsing

EXP: 4 Use LEX and YACC tool to implement Desktop Calculator

1. 1.Using the file tool create LEX and YACC files.

2. In the c include section define the header file sequence.

3. Declare the race files inside it in the C definition section declare the header file .
4. Required along with a variable valid with value assigned as L

5. .In the YACC declaration declare the format token Num id.

6. .In the grammar rule section if the output string is valid Unambiguous expression then identifier & section the value.

7. .In the user define section if the valid is a string as invalid expression YYerror() and define main function.

EXP 5: Shift Reduce Parser

1. 1.Start the program.
2. Get the expression from the user and call the parser() function.
3. In lexer() get the input symbol and match with the look ahead pointer and then return the token accordingly.
4. In E(), check whether the look ahead pointer is „+" or „-„ else return syntax error.
5. In T(),check whether the look ahead pointer is „*" or „/" else return syntax error.
6. In F(),check whether the look ahead pointer is a member of any identifier.
7. In main(), check if the current look ahead points to the token in a given CFG it doesn"t match the return syntax error

EXP : 6 Shift Reduce Parser

1. 1.Start the program
2. read the variables , stack symbols
3. loop forever:for top-of-stack symbol, s, and next input symbol, a case action of T[s,a]

4. shift x: (x is a STATE number) push a, then x on the top of the stack and advance ip to point to next input symbol
5. reduce y: (y is a PRODUCTION number) Assume that the production is of the form A ==> beta
6. pop 2 * |beta| symbols of the stack. At this point the top of the stack should be a state number, say s'. push A, then goto of T[s',A] (a state number) on the top of the stack.
7. Output the production A ==> beta.
8. accept: return --- a successful parse. 9.default: error --- the input string is not in the language. 10.Stop the program.

EXP: 7 OPERATOR PRECEDENCE PARSING

1. 1. Include the necessary header files.
2. Declare the necessary variables with the operators defined before.
3. Get the input from the user and compare the string for any operators.
4. Find the precedence of the operator in the expression from the predefined operator.
5. Set the operator with the maximum precedence accordingly and give the relational operators for them.
6. Parse the given expression with the operators and values. 7. Display the parsed expression. 8. Exit the program.

EXP : 8 . Implementation of 3-Address Code

1. 1.Start the program.
2. 2.Open the input file.
3. Enter the intermediate code as an input to the program.
4. Apply conditions for checking the keywords in the intermediate code.
5. Analyse each instruction in switch case.
6. After generating machine code, copy it to the output file.
7. Stop the program.

EXP: 9 Symbol Table

1. Start the program.
2. Open the input file.
3. Enter the intermediate code as an input to the program.
4. Apply conditions for checking the keywords in the intermediate code.
5. Analyse each instruction in switch case.
6. After generating machine code, copy it to the output file.
7. Stop the program.

EXP: 10 Implementation of simple code optimization techniques

1. 1.Start
2. Initialize value N
3. Initialise the count value
4. If perform loop unrolling then, i.Perform each operation upto count.
5. Else, loop rolling i.Check the condition ii.perform the operation iii.increment the count
6. Print the result
7. Stop

EXP : 11 SIMPLE CODE GENERATOR

1. 1.Start.
2. Get address code sequence.
3. Determine current location of 3 using address (for 1st operator)
4. If current location not already exist generate move (B,O)
5. Update address of A(for 2nd operand).
6. If current value of B and () is null, exist
7. If they generate operator () A,3 ADPR
8. Store the move instruction in memory 9.Stop