End Practical Examination

Reg No: 39110636	Name: Mohnish Devaraj
Branch: CSE	semester: VI Sem
Subject Code: SCSA2601	Subject Name: Compiler Design Lab
Date of Exam: 27-04-2022	Batch-Id: CD-7
Duration: 3 Hours	Total no. pages: 8

① Evaluate any given arithematic expression using Ambigous grammar. Use Lex and Yacc Tool.

Aim:

To write the program using LEX and YACC to implement parser on ambigous grammar.

Algorithm:

File.l

step1: Start

step 2: Include the necessary header files and declare the necessary variables

step 3: initialize the digits, operators, parenthesis and return the value else print syntax error

step 4: Call the function & return 1

step 5: Stop

File.y

step 1: Start

step 2: Include the necessary header files and declare the necessary variables

step 3: Substitute the values and calculate respective for Addition, Subtraction, Multiplication and division and return the result.

step 4: Call the main function and print the result

step 5: Stop

Program:

File.l

```
%option noyywrap
%{
    #include <stdio.h>
    #include "y.tab.h"
    void yyerror(char *s)
    extern int yylval;
%}
%%
[0-9]+      {yylval = atoi (yytext);
return NUM;
[-+*/\n]    {return *yytext;}
"("         {return *yytext;}
")"         {return *yytext;}
```

```
[\t];
.        {yyerror ("Syntax Error");}
%%
int yywrap()
{
return 1;
}
```

File.y

```
%{
    #include<stdio.h>
    extern int yylex(void);
    void yyerror(char*);
%}
% token NUM
%%
S:
S expr '\n' {printf("%d\n", $2);}
|
;
expr:
expr '+' expr     {$$= $1 + $3;}
|expr '-' expr    {$$ = $1 - $3;}
|expr '*' expr    {$$ = $1 * $3;}
|expr '/' expr    {$$ = $1 / $3;}
| NUM             {$$ = $1;}
| '('expr')'      {$$ = $2;}
```

```
}
%%
void yyerror(char *s)
{
printf("%s\n", s);
 }
int main()
{
yyparse();
return 0; }
```

Output:

The output is attached below

Result:

The use of LEX and YACC to implement parser for ambigous is executed successfully.

(2) Write a C program to parse the given string using Operator precedence parser.

Aim:

To write a c program implementing Operator Precedence parser. ~~algor~~

Algorithm:

step 1: start

step 2: Input the no. of terminals, terminals and table values

step 3: Using the inputs, construct the operator precedence table

step 4: get an expression as input string step 5: Push the expression into the stack

step 6: Pop the top most operands and operator from the stack

step 7: check the validity of the expression by checking the precedence from the table constructed.

step 8: Return error message if the expression does not matches

step 9: stop

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
    char stack[20], opt[10][10];
    int i,j,k, n=4, top=0, col, row;
   char ter[]={'a','+','*','$'};
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%c",&opt[i][j]);
      }
   }
   stack[top]='$';
   Char ip[]={'a','+','a', , '*','a','$'};
   i=0;
  while(i<strlen(ip))
  {
```

```
for(k=0;k<n;k++)
{
    if(stack[top]==ter[k]]
        row=k;
    if(ip[i]==ter[k])
        col=k;
}
if((stack[top]=='$')&&(ip[i]=='$'))
{
    printf("String is accepted");
    break;
}
else if((opt[row][col]=='<')||(opt[row][col]=='='))
{
    stack[++top]=opt[row][col];
    stack[++top]=ip[i];
    printf("Shift %c",ip[i]);
    i++;
}
else
{
    if(opt[row][col]=='>')
    {
      while(stack[top]!='<')
      --top;
      top=top-1;
      printf("Reduce");
    }
```

```
        else
        {
         printf ("string is not accepted");
         break;
        }
      }
    }
    ~~retur~~
    return 0;
}
```

Output:

The output is attached below

Result:

The above program is executed successfully.