S.No: 2	Exp. Name: ***C program to implement the Lexical Analyzer for 24 operators.***	Date:

Aim:

Complete the following C program to implement the Lexical Analyzer for 24 operators:
>= > <= < == = != ! && & || | ++ += + -- -= - *= * /= / %= %

Source Code:

lexicalnalyzer.c

```
#include<stdio.h>
#include<conio.h>
void main() {
    char op[3];
    printf("Enter an operator: ");
    // gets(op);

    // switch(op[0]) {
    //     case'>': if(op[1]=='=')
    //                 printf("Greater than or equal to");
    //              else if(op[1] == '\0')
    //                 printf("Greater than");
    //              else
    //                 printf("Not an operator");
    //              break;
    scanf("%s",&op);
    switch(op[0]){
        case'>' :
                 if(op[1]=='=')
                  printf("Greater than or equal to\n");
                 else if(op[1]=='\0')
                  printf("Greater than\n");
                 else
                  printf("Not an operator\n");
        case'<' :
                  if(op[1]=='=')
                     printf("Equals to\n");
                  else if(op[1]='\0')
                     printf("Assignment operator\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'=' :
                  if(op[1]=='=')
                     printf("Equals to\n");
                  else if(op[1]=='\0')
                     printf("Assignment operator\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'!' :
                  if(op[1]='=')
                     printf("Not Equals to\n");
                  else if(op[1]=='\0')
                     printf("Logical not\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'&' :
                  if(op[1]=='&')
                     printf("logical and \n");
                  else if(op[1]=='\0')
                     printf("Bitwise and \n");
                  else
                     printf("Not an operator\n");
                  break;
        case'|' :
                  if(op[1]=='|')
                     printf("Logical or\n");
                  else if(op[1]=='\0')
                     printf("Bitwise or\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'+' :
                  if(op[1]=='+')
                     printf("Increment By 1\n");
                  else if(op[1] =='=')
                     printf("Addition and Assignment\n");
                  else if(op[1]=='\0')
                     printf("Addition operator\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'-' :
                  if(op[1]=='-')
                     printf("Decrement By 1\n");
                  else if(op[1]=='=')
                     printf("Subtract and Assgnment\n");
                  else if(op[1]=='\0')
                     printf("Subtraction\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'*' :
                  if(op[1]=='=')
                     printf("Multiplication and Assignment \n");
                  else if(op[1]=='\0')
                     printf("Multiplication\n");
                  else
                     printf("Not an operator\n");
                  break;

        case'/' :
                  if(op[1]=='=')
                     printf("Division and Assignment\n");
                  else if(op[1]=='\0')
                     printf("Division\n");
                  else
                     printf("Not an operator\n");
                  break;
        case'%' :
                  if(op[1]=='=')
                     printf("Modulus and Assignment\n");
                  else if(op[1]=='\0')
```

ID: 1803010180 Page No:

Inderprastha Engineering College

```
                     printf("Modulus\n");
                  else
                     printf("Not an operator\n");
                  break;
            default:printf("Not an operator\n");
         // Complete all cases for all other operators

        // default: printf("Not an operator");
    }
}
```

Page No:

ID: 1803010180

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter an operator: ++
Increment By 1

Test Case - 2
User Output
Enter an operator: #
Not an operator

Inderprastha Engineering College

S.No: 3	Exp. Name: ***C program to identify whether a given grammar is Operator Grammar or not***	**Date:**

Aim:

Complete the following C program to identify whether a given grammar is Operator Grammar or not.

Source Code:

operatorGrammer.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

void f() {
   printf("Not operator grammar\n");
   exit(0);
}
void main() {
   char gramm[20][20],c;
   int i, n, j = 2, flag = 0;
   printf("Enter number of productions: ");
   scanf("%d", &n);
   printf("Enter the grammar:\n");
   for( i=0;i<n;i++) //complete the code in for loop
   scanf("%s", gramm[i]);
   for(i=0;i<n;i++ ) // complete the code in for loop
   {
      c = gramm[i][2];
      while( c!='\0') // complete the condition part
      {
         //Complete code in while loop
            if(gramm[i][3]=='+'||gramm[i][3]=='-'||gramm[i][3]=='*'||gramm[i][3]=='/')
            {              flag=1;
            break;
            }
            else
            {
               f();
               break;
               }

      }
   }
   if(flag == 1)
      printf("Operator grammar\n");

  }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter number of productions: 3

Page No:

ID: 1803010180

Inderprastha Engineering College

Test Case - 1	
Enter the grammar: A=A+A	
A=A-A	A=A-A
A=A/A	A=A/A
Operator grammar	

Test Case - 2	
User Output	
Enter number of productions: 2	
Enter the grammar: A=C+B	
C=$	C=$
Not operator grammar	

Test Case - 3	
User Output	
Enter number of productions: 2	
Enter the grammar: B=C+D	
B=C/D	B=C/D
Operator grammar	

S.No: 4	Exp. Name: ***Rewrite and optimize the following code using the variable propagation technique***	Date:

Page No:

ID: 1803010180

Aim:

Rewrite and optimize the following code using the variable propagation technique.

```
#include <stdio.h>
void main() {
         int prod;
         int var1;
         int var2;
         int new_var;
         int final;
         printf("Enter all three variables : ");
         scanf("%d %d %d", &var1, &var2, &new_var);
         prod = var1 * var2;
         new_var = var1;
         final = new_var * var2 + 4 ;
         printf("The Final value is : %d\n",final);
}
```

Example : Original code:

```
a = x + y
temp = x
comp= (temp + y) + 10
```

Optimized code:

```
a = x + y
temp = x
comp= a + 10
```

Source Code:

variablepropagation.c

```
#include<stdio.h>
void main() {
   int prod,var1,var2,new_var,final;
   printf("Enter all three variables : ");
   scanf("%d %d %d",&var1,&var2,&new_var);
   prod = var1*var2;
   new_var = var1;
   final = prod + 4;
   printf("The Final value is : %d\n",final);

}
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter all three variables : 2 4 6

Inderprastha Engineering College

Test Case - 1

```
The Final value is : 12
```

Test Case - 2

User Output

```
Enter all three variables :  1 1 0
The Final value is : 5
```

Test Case - 3

User Output

```
Enter all three variables :  15 9 8
The Final value is : 139
```

Page No:

ID: 1803010180

Inderprastha Engineering College

S.No: 5	Exp. Name: ***C program to implement the Lexical Analyzer for Arithmetic Expression. We have identifiers, constants and operators.***	**Date:**

Page No:

ID: 1803010180

Aim:

Complete the following C program to implement the Lexical Analyzer for Arithmetic Expression. We have identifiers, constants and operators.

Source Code:

arithmeticexpression.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

int main() {
   char expression[50], word[10];
   int i,j=0, flag = 0;
   printf("Enter expression: ");
    gets(expression);      // Take input from the user as an expression.
    for(i=0;expression[i]!='\0';i++ )  // Iterate the given expression with the variabl
e i.
    {
        if(expression[i] == '+' ||  expression[i] == '-' || expression[i] == '*' || exp
ression[i] == '/' || expression[i] == '%')
        {
         word[j] = expression[i];
         j++;
         flag = 1;
        }
        else if (expression[i]>='0'&& expression[i]<='9' )
        {word[j]=expression[i];
        j++;
        flag=2;
         // Write code for checking numbers

        }
        else if (isalpha(expression[i]))
        {
          // Write complete for checking identifiers as done in above 2 parts
         word[j]=expression[i];
         j++;
         flag=3;
      }
        if(expression[i+1] == ' ' || expression[i+1] == '\0')
        {
         if(flag == 1)
              printf("%s is operator\n",word );    // Display message when flag = 1

         else if(flag==2 )       // Write condition for displaying message of numbers /
constants
              printf("%s is constant \n", word);

          else if(flag==3 )    // Write flag condition for identifier
              printf("%s is identifier\n",word );  // Display message for identifier.
```

Inderprastha Engineering College

```
         for(j=0;j<10;j++)
             word[j] = '\0';

          flag = 0;
          j = 0;
      }
    }
   return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter expression:  b + c - 5 % d
b is identifier
+ is operator
c is identifier
- is operator
5 is constant
% is operator
d is identifier
```

Test Case - 2

User Output

```
Enter expression:  5 * 2 + a / 2 - b % c
5 is constant
* is operator
2 is constant
+ is operator
a is identifier
/ is operator
2 is constant
- is operator
b is identifier
% is operator
c is identifier
```

Test Case - 3

User Output

```
Enter expression:  2 - 1 * a / b
2 is constant
- is operator
1 is constant
* is operator
a is identifier
/ is operator
b is identifier
```

Page No:

ID: 1803010180

Inderprastha Engineering College

S.No: 6	Exp. Name: ***C program to identify whether a given word is a keyword or not.***	Date:

Aim:

Complete the following C program to identify whether a given word is a keyword or not.

Source Code:

keyword.c

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {

    char keywords[32][10] = {{"auto"},{"if"},{"else"},{"do"},{"while"},{"switch"},{"fo
r"},{"extern"},{"case"},{"char"},{"const"},
    {"continue"},{"default"},{"double"},{"enum"},{"float"},{"goto"},{"int"},{"long"},
{"return"},{"register"},
    {"signed"},{"short"},{"sizeof"},{"struct"},{"typedef"},{"union"},{"unsigned"},{"voi
d"},{"volatile"},{"break"}};            // Make array of keywords

    char word[50];
   int i, flag = 0;
   printf("Enter a word: ");
    gets(word );  // Complete statement to get input from user
   for (i = 0; i < 32; ++i) {

      // Complete logic for checking the word with each keyword.
      if(strcmp(word,keywords[i])==0)
      flag=1;
   }
   if(flag == 1)
       printf("%s is keyword\n", word);
   else
       printf("%s is not a keyword\n",word ); // Display message that entered word is n
ot keyword.
    return 0;
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a word:  run
run is not a keyword
```

Test Case - 2

User Output

```
Enter a word:  case
case is keyword
```

Test Case - 3

User Output

```
Enter a word:  scanf
scanf is not a keyword
```

ID: 1803010180 Page No:

Inderprastha Engineering College

S.No: 7	Exp. Name: ***Fill the missing code by analyzing the code motion. Bring loop invariant statements out of the loop***	**Date:**

Page No:

ID: 1803010180

Aim:

Rewrite and optimize the following code by analyzing the code motion. Bring loop invariant statements out of the loop.

```
#include
void main() {
        int var1, temp1, temp2;
        printf("Enter the value for var1, temp1 and temp2: ");
        scanf("%d %d %d", &var1, &temp1, &temp2);
        while(var1 > 0) {
        int var2 = temp1 + temp2;
        if (var1 % var2 == 0)
        printf("%d\n", var1);
        var1--;
        }
}
```

Example : Original code:

```
a= 5;
while(a>0) {
sum = x+y ;
printf(“%d , %d”, a, sum);
}
```

Optimized code:

```
a= 5;
sum = x+y ;
while(a>0) {
printf(“%d , %d”, a, sum);
}
```

Source Code:

loopinvariant.c

```
#include<stdio.h>
void main()
{  int var1,temp1,temp2;
printf("Enter the value for var1, temp1 and temp2: ");
scanf("%d %d %d",&var1,&temp1,&temp2);
int var2 = temp1+temp2;
while(var1>0)
{
   if(var1%var2 == 0)
   printf("%d\n",var1);
   var1--;  }

}
```

Execution Results - All test cases have succeeded!

Inderprastha Engineering College

Test Case - 1

User Output

```
Enter the value for var1, temp1 and temp2:  12 2 4
12
6
```

Test Case - 2

User Output

```
Enter the value for var1, temp1 and temp2:  100 5 5
100
90
80
70
60
50
40
30
20
10
```

Test Case - 3

User Output

```
Enter the value for var1, temp1 and temp2:  0 0 0
```

S.No: 8	Exp. Name: ***C program to recognize strings under the regular expression: a*b+***	**Date:**

Page No:

ID: 1803010180

Aim:

Complete the following C program to recognize strings under the regular expression: a*b+

Source Code:

strings.c

```
#include <stdio.h>
void main() {

   char string[20], ch;
   int state = 0, i = 0;
   printf("Enter a string: ");
   gets(string ); // Take input from user in string variable
   for(i=0;string[i]!='\0';i++ ) // Iterate over input using i variable.
   {
       switch (state)
       {
         case 0: ch = string[i];
               if (ch == 'a')
                  state =0  ;    // Assign value to state
               else if (ch == 'b')
                  state = 1 ;    // Assign value to state
               else
                  state = 2 ;    // Assign value to state
               break;
         case 1:  ch=string[i];
         if(ch=='b'){
            state=1;
         }// Write for case 1 just like given for case 0
         else{
            state=2;
         }
         break;


          case 2: printf("%s is not recognised\n", string);
               exit(0);
       }
   }
   if(state == 0 || state == 2)
       printf("%s is not recognized\n",string);    // Display message for invalid state
s
   else
        printf("%s is recognized\n",string);    // Display message for valid states
}
```

Execution Results - All test cases have succeeded!

Inderprastha Engineering College

Test Case - 1

User Output

```
Enter a string:  aa
aa is not recognized
```

Test Case - 2

User Output

```
Enter a string:  ab
ab is recognized
```

Test Case - 3

User Output

```
Enter a string:  abc
abc is not recognized
```

Page No:

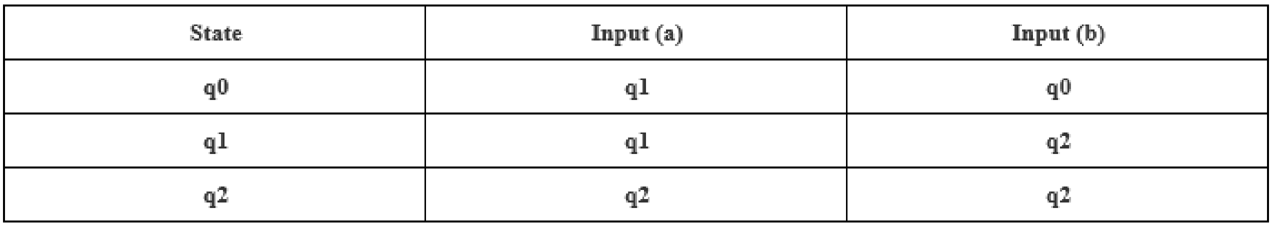
ID: 1803010180

Inderprastha Engineering College

S.No: 9	Exp. Name: ***C program for construction of NFA from Regular Expression: b*a+***	**Date:**

Aim:

Complete the following C program for construction of NFA from Regular Expression: b*a+
Hint: Transition table for the NFA is shown below:



State	Input (a)	Input (b)
q0	q1	q0
q1	q1	q2
q2	q2	q2

Source Code:

NFA.c

```
#include <stdio.h>
void main() {

   char string[20], ch;
   int state = 0, i = 0;
   printf("Enter a string: ");
   gets(string );    // Take input from user in string variable
   for(i=0;i<strlen(string);i++ )   // Iterate over input using i variable.
   {
       switch (state)
       {
         case 0: ch = string[i];
               // if (ch == 'a')
                  // state =;
               // else if (ch == 'b')
                  // state =;
                  if(ch=='b'){
                     state=0;
                  }
               // else
                  // state =; // Assign value to state
                  else if(ch=='a'){
                     state=1;
                  }
                  else{
                     continue;
                  }
               printf("q0 [%c] ---> q%d\n",ch,state);
               break;


         case 1: ch=string[i];
         if(ch=='a'){
            state=1;
         }
         else if(ch=='b'){
            state=2;
```

Page No:

ID: 1803010180

Inderprastha Engineering College

```
            // Write case 1 just like Case 0.
         }
         else
         {
            continue;

         }
         printf("q1 [%c] ---> q%d\n",ch,state);
         break;
          case 2: printf("q2 [%c] ---> q%d\n",ch,state);
       }
   }
   if(state == 1)
       printf("%s is terminating at state 1. So, it is an accepted string.\n",string);
// Write the final statement as given in the output when the q1 state has reached.
   else
        printf("%s is not terminating at state 1. So, it is not an accepted string.\n",
string); //write the final statement as given in the output when the q1 state is not re
ached.

}
```

Page No:

ID: 1803010180

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a string:  bba
q0 [b] ---> q0
q0 [b] ---> q0
q0 [a] ---> q1
bba is terminating at state 1. So, it is an accepted string.
```

Test Case - 2

User Output

```
Enter a string:  aab
q0 [a] ---> q1
q1 [a] ---> q1
q1 [b] ---> q2
aab is not terminating at state 1. So, it is not an accepted string.
```

Test Case - 3

User Output

```
Enter a string:  bbaba
q0 [b] ---> q0
q0 [b] ---> q0
q0 [a] ---> q1
q1 [b] ---> q2
q2 [b] ---> q2
bbaba is not terminating at state 1. So, it is not an accepted string.
```

Inderprastha Engineering College

S.No: 10	Exp. Name: ***C program to identify whether a given line is a comment or not***	**Date:**

Page No:

ID: 1803010180

Aim:

Complete the following C program to identify whether a given line is a comment or not.

Source Code:

comment.c

```
#include <stdio.h>
void main() {
   char comment[50];
   int i, flag = 0;
      // Write the complete logic
      printf("Enter comment: ");
      //for(int i=0;i<50;i++)
      gets(comment);
      if(comment[0]=='/'){
         if(comment[1]=='/')
         {

         printf("It is a comment\n");

      }
      else if( comment[1]=='*')
      {for(int i=0;i<=50;i++)
      {
         if(comment[i]=='*'&&comment[i+1]=='/')
         {printf("It is a comment\n");
         flag=1;
         break;

         }
         else continue;
      }
   if(flag==0){
      printf("It is not a comment\n");
   }
      }
      else
      printf("It is not a comment\n");
      }
      else
   printf("It is not a comment\n");


   }
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
Enter comment: // hello //
It is a comment

Inderprastha Engineering College

Test Case - 2

User Output

```
Enter comment:  How are you ?
It is not a comment
```

Test Case - 3

User Output

```
Enter comment:  covid-19
It is not a comment
```

S.No: 11	Exp. Name: ***C program to implement LL(1) parsing algorithm for the following grammar***	**Date:**

Page No:

ID: 1803010180

Aim:

Complete the following C program to implement LL(1) parsing algorithm for the following grammar.

E → TB
T → FC
C → *FC / ϵ
B → +TB / ϵ
F → i / (E)

Source Code:

parsing.c

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
char s[20],stack[20];
void main()
{
   char m[5][6][3] = {"tb"," "," ","tb"," "," "," ","+tb"," "," ","n","n","fc"," ","
","fc"," "," "," ","n","*fc"," ","n","n","i"," "," ","(e)"," "," "}; // This is array o
f productions in reverse order
   int size[5][6] = {2,0,0,2,0,0,0,3,0,0,1,1,2,0,0,2,0,0,0,1,3,0,1,1,1,0,0,3,0,0};   //
This is array of size of each production
   int i, j, k, n, str1, str2;
   printf("Enter the input string: ");
   scanf("%s", s);
   strcat(s, "$");
   n = strlen(s);
   stack[0] = '$'  ; // complete the line with end of input
   stack[1] =  'e' ;    // complete the line with starting production of the grammar
   i = 1;
   j = 0;
   printf("Stack\tInput\n");
   while((stack[i]!='$')&&(s[j]!='$') )      // complete the condition part
   {
      if(stack[i]==s[j] )      //complete the condition part for comparison of string i
nput with stack input
      {
         i--;
         j++;
      }
      switch(stack[i])
      {
            case 'e': str1=0;
            break;
            case 'b': str1=1;
            break;
            case 't': str1=2;
            break;
            case 'c': str1=3;
            break;
            case 'f': str1=4;
            break; // complete the switch-case part for stack input
      }
```

Inderprastha Engineering College

```
      switch(s[j])
      {
         case 'i': str2=0;
         break;
         case '+': str2=1;
         break;
         case '*': str2=2;
         break;
         case '(': str2=3;
         break;
         case ')': str2=4;
         break;
         case '$': str2=5;
         break;
         // complete the switch -case part for string input
      }
      if(m[str1][str2][0]=='\0' )      // complete the condition part
      {
      printf("Error\n");
      exit(0);
      }
      else if( m[str1][str2][0]=='n')  // complete the condition part
         i--;
      else if(m[str1][str2][0]=='i' )  // complete the condition part
         stack[i] = 'i';
      else
      {
         for(k=size[str1][str2]-1;k>=0;k-- ) // complete code in for loop
         {
               stack[i] = m[str1][str2][k];
               i++;
         }
         i--;
      }
      for(k=0;k<=i;k++ )   // complete code in for loop
      printf("%c", stack[k]);
      printf("\t");
      // for( ) { // complete code in for loop
      for(k=j;k<n;k++){
         printf("%c", s[k]);
      }
      printf("\n");
   }
   printf("Success\n");
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

Test Case - 1

User Output

```
Enter the input string:  i*i+i
Stack   Input
$bt     i*i+i$
$bcf    i*i+i$
$bci    i*i+i$
$bcf*   *i+i$
$bci    i+i$
$b      +i$
$bt+    +i$
$bcf    i$
$bci    i$
$b      $
Success
```

Page No:

ID: 1803010180

Inderprastha Engineering College

S.No: 12	Exp. Name: ***C program to test whether a given identifier is valid or not valid.***	**Date:**

Aim:

Complete the following C program to test whether a given identifier is valid or not valid.

Source Code:

identifier.c

```
#include <stdio.h>
#include <ctype.h>
void main() {
   char identifier[20];
   int flag, i;

   // Write the complete logic
   printf("Enter an identifier: ");
   gets(identifier);
   flag=1;
   if(!((identifier[0]>='a'&& identifier[0]<='z')||(identifier[0]>='A' && identifier[0]
<='Z')||identifier[0]=='_')){
      flag=0;
   }
    i=1;
   for(int i=0;identifier[i]!='\0';i++)
   {
      char c=identifier[i];
      if(!((c>='a'&& c<='z')|| (c>='A' && c<='Z')|| (c>='0'&& c<='9')|| c=='_')){
         flag=0;
      }
   }
   if(flag==1)
   {
      printf("It is a valid identifier\n");
   }
   else{
   printf("It is not a valid identifier\n");
   }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter an identifier:  first
It is a valid identifier
```

Test Case - 2

User Output

```
Enter an identifier:  1aqw
It is not a valid identifier
```

Page No:

ID: 1803010180

Inderprastha Engineering College

Test Case - 3

User Output

```
Enter an identifier:  int
It is a valid identifier
```

Page No:

ID: 1803010180

Inderprastha Engineering College

S.No: 13	Exp. Name: ***C program to implement Recursive Descent Parser***	**Date:**

Page No:

ID: 1803010180

Aim:

Complete the following C program to implement Recursive Descent Parser for the Grammar
E -> TE'
E' -> +TE' | ε
T -> FT'
T' -> *FT' | ε
F -> (E) | id

Source Code:

descentparser.c

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>

void Tp();
void Ep();
void CheckExp();
void check();
void T();


char expr[10];
int count, flag;

int main() {
   count = 0;
   flag = 0;
      printf("Enter an Algebraic Expression: ");
      scanf("%s", expr);
   CheckExp();
      if((strlen(expr) == count) && (flag == 0)) {
            printf("The Expression %s is Valid\n", expr);
   }
   else {
            printf("The Expression %s is Invalid\n", expr);
   }
}

void CheckExp () {
   T();
   Ep();
}
 void T() {
   check();
   Tp();
}
void Tp() {
      if(expr[count] == '*') {

        //Write code when expr[count] == '+
         count++;
         check();
         Tp();
```

Inderprastha Engineering College

Page No:

ID: 1803010180

```
   }
}

void check() {
      if(isalnum(expr[count])) {

        // Write code if expr[count] is alphabet or number
        count++;
   }
   else if(expr[count] == '(') {

        // Write code if expr[count] is ‘(‘
      count++;
         CheckExp ();
   if(expr[count]==')')
   {
      count++;
      }
      else
      flag=1;

   }
   else {
         flag = 1;
   }
}

void Ep() {
      if(expr[count] == '+') {

         //Write code when expr[count] == '+
          count++;
          T();
          Ep();

   }
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter an Algebraic Expression:  a+a*a
The Expression a+a*a is Valid
```

Test Case - 2

User Output

```
Enter an Algebraic Expression:  (a+b)*c+d
The Expression (a+b)*c+d is Valid
```

Test Case - 3

Inderprastha Engineering College

Test Case - 3

User Output

```
Enter an Algebraic Expression:  (a+(b/c)
The Expression (a+(b/c) is Invalid
```

Page No:

ID: 1803010180

Inderprastha Engineering College

S.No: 14	Exp. Name: ***C program to implement Code Generator. The name of the input file is supplied as a command-line argument. Use printf to print the generated code to the standard output***	**Date:**

Aim:

Complete the following C program to implement Code Generator. The name of the input file is supplied as a command-line argument. Use printf to print the generated code to the standard output.

Source Code:

codegenerator.c

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
char op[2], arg1[5], arg2[5], result[5];
void main(int argc, char *argv[]) {
   FILE *fp1 = fopen(argv[1], "r");
   while (!feof(fp1)) {
      fscanf(fp1,"%s%s%s%s", op, arg1, arg2, result);

   //Write the code snippets for division, multiplication, addition,subtraction operato
rs using if conditions for comparing the ‘op’ variable
      if(strcmp(op,"+")==0){
         printf("MOV R0,%s\n",arg1);
         printf("ADD R0,%s\n",arg2);
         printf("MOV %s,R0\n",result);

      }
         if(strcmp(op,"*")==0){
            printf("MOV R0,%s\n",arg1);
            printf("MUL R0,%s\n",arg2);
            printf("MOV %s,R0\n",result);
            }

            if(strcmp(op,"-")==0){
               printf("MOV R0,%s\n",arg1);
               printf("SUB R0,%s\n",arg2);
               printf("MOV %s,R0\n",result);

            }

            if(strcmp(op,"/")==0){
               printf("MOV R0,%s\n",arg1);
               printf("DIV R0,%s\n",arg2);
               printf("MOV %s,R0\n",result);
               }

               if(strcmp(op,"=")==0){
                  printf("MOV R0,%s\n",arg1);
                  printf("MOV %s,R0\n",result);
                  }

   }
   fclose(fp1);
}
```

Page No:

ID: 1803010180

input.txt

```
+ a b t1
* c d t2
- t1 t2 t
= t ? x
```

input1.txt

```
* z y t1
/ x w t2
- t1 t2 t
= t ? d
```

Execution Results - All test cases have succeeded!

Test Case - 1
User Output
MOV R0,a
ADD R0,b
MOV t1,R0
MOV R0,c
MUL R0,d
MOV t2,R0
MOV R0,t1
SUB R0,t2
MOV t,R0
MOV R0,t
MOV x,R0

Inderprastha Engineering College

S.No: 15	Exp. Name: ***C program to implement the Shift Reduce Parsing algorithm***	**Date:**

Aim:

Complete the following C program to implement the Shift Reduce Parsing algorithm.

Source Code:

shiftparsing.c

```
#include <stdio.h>
#include <string.h>
int k = 0, z = 0, i = 0, j1 = 0, c = 0;
char s[16], ac1[20], stack[15], act1[10];
void check();
int main() {
    puts("Grammar is E->E+E \n E->E*E \n E->(E) \n E->id");
    puts("Enter input string: ");
    gets(s);
    c = strlen(s);
    strcpy(act1,"Shift->");
    puts("stack \t input \t action");
    for(k = 0, i = 0; j1 < c; k++, i++, j1++) {
   //Code to insert “id” on Stack
      if(s[j1] == 'i' && s[j1+1] == 'd') {
         stack[i] = s[j1];
         stack[i+1] = s[j1+1];
         stack[i+2] = '\0';
         s[j1] = ' ';
         s[j1+1] = ' ';
         printf("$%s\t%s$\t%sid\n",stack, s, act1);
         check();
      }
      else {
      stack[i]=s[j1];
      stack[i+1]='\0';
      s[j1]=' ';
      printf("$%s\t%s$\t%s symbols\n",stack,s,act1);
      check();

         //Write Code to insert Symbol on Stack

      }
   }
}
void check() {
      // Code to reduce 'id' to 'E'
strcpy(ac1,"Reduce To E");
   for (z = 0; z < c; z++)
        if (stack[z] == 'i' && stack[z+1] == 'd') {
            stack[z] = 'E';
            stack[z+1] = '\0';
            printf("$%s\t%s$\t%s\n", stack, s, ac1);
            j1++;
        }

   for (z = 0; z < c; z++)
        if(stack[z] == 'E' && stack[z+1] == '+' && stack[z+2] == 'E') {
            // Write Code to reduce 'E+E' to 'E'
            stack[z]='E';
            stack[z+1]='\0';
            stack[z+2]='\0';
            printf("$%s\t%s$\t%s\n",stack,s,ac1);


         i = i - 2;
        }

    for (z = 0; z < c; z++)

    //Write  Code to reduce 'E*E' to 'E'
      if(stack[z]=='E' && stack[z+1]=='*' && stack[z+2]=='E')
      {     stack[z]='E';
      stack[z+1]='\0';
      stack[z+2]='\0';
      printf("$%s\t%s$\t%s\n",stack,s,ac1);
      i=i-2;

      }

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Grammar is E->E+E  id+id*id
 E->E*E  id+id*id
 E->(E)  id+id*id
 E->id id+id*id
Enter input string:  id+id*id
stack    input   action
$id       +id*id$       Shift->id
$E        +id*id$       Reduce To E
$E+        id*id$       Shift-> symbols
$E+id        *id$       Shift->id
$E+E         *id$       Reduce To E
$E           *id$       Reduce To E
$E*           id$       Shift-> symbols
$E*id           $       Shift->id
$E*E            $       Reduce To E
$E              $       Reduce To E
```

Test Case - 2

User Output

```
Grammar is E->E+E  5a*b/c
 E->E*E  5a*b/c
 E->(E)  5a*b/c
 E->id 5a*b/c
```

Test Case - 2

```
Enter input string:  5a*b/c
stack    input   action
$5       a*b/c$ Shift-> symbols
$5a       *b/c$ Shift-> symbols
$5a*       b/c$ Shift-> symbols
$5a*b       /c$ Shift-> symbols
$5a*b/       c$ Shift-> symbols
$5a*b/c       $ Shift-> symbols
```

Test Case - 3

User Output

```
Grammar is E->E+E  b+a+cccccc/dddddddddddd
 E->E*E  b+a+cccccc/dddddddddddd
 E->(E)  b+a+cccccc/dddddddddddd
 E->id b+a+cccccc/dddddddddddd
Enter input string:  b+a+cccccc/dddddddddddd
stack    input   action
$b       +a+cccccc/dddddShift->$        Shift-> symbols
$b+       a+cccccc/dddddShift->$        Shift-> symbols
$b+a       +cccccc/dddddShift->$        Shift-> symbols
$b+a+       cccccc/dddddShift->$        Shift-> symbols
$b+a+c       ccccc/dddddShift->$        Shift-> symbols
$b+a+cc       cccc/dddddShift->$        Shift-> symbols
$b+a+ccc               ccc/dddddShift->$        Shift-> symbols
$b+a+cccc               cc/dddddShift->$        Shift-> symbols
$b+a+ccccc               c/dddddShift->$        Shift-> symbols
$b+a+cccccc               /dddddShift->$        Shift-> symbols
$b+a+cccccc/               dddddShift->$        Shift-> symbols
$b+a+cccccc/d               ddddShift->$        Shift-> symbols
$b+a+cccccc/dd               dddShift->$        Shift-> symbols
$b+a+cccccc/ddd               ddShift->$        Shift-> symbols
$b+a+cccccc/dddd                       dShift->$        Shift-> symbols
$b+a+cccccc/ddddd                       Shift->$        Shift-> symbols
$b+a+cccccc/dddddS                       hift->$         hift-> symbols
$b+a+cccccc/dddddSh                       ift->$          ift-> symbols
$b+a+cccccc/dddddShi                       ft->$           ft-> symbols
$b+a+cccccc/dddddShif                       t->$            t-> symbols
$b+a+cccccc/dddddShift                       ->$             -> symbols
$b+a+cccccc/dddddShift-                       >$              > symbols
$b+a+cccccc/dddddShift->                               $                symbols
```

S.No: 16	Exp. Name: ***C program for the construction of DFA from NFA***	Date:

Page No:

ID: 1803010180

Aim:

Complete the following C program for the construction of DFA from NFA given below. Transition table for the NFA is shown below

State	Input (a)	Input (b)
1	{1,2}	{1}
2	-	{3}
3	-	{4}
4	-	-

Source Code:

DFA.c

```
#include <stdio.h>
void main() {

    char string[20],ch;
    int state = 1, row = 0, col = 0, i;
    printf("Enter a string: ");
    gets(string);    // Take input from user in string variable

               //Here, in NFA we have taken NULL state as 0.
    int nfa[4][3] = {
                     {1,12,1},
                     {2,0,3},
                     {3,0,4},
                     {4,0,0}
                  };
    printf("Transition table for NFA is:\n");
    for(row=0;row<4;row++)
    {       for(col=0;col<3;col++){
      printf("%d  ",nfa[row][col]);
      }
      printf("\n");   }


    // Iterate over nfa array and display its content using row and col variables

    int dfa[4][3] = {      {1,12,1},      {12,12,13},       {13,12,14},       {14,12,1}

      //Fill dfa array with its correct values.

   };
    printf("Transition table for DFA is:\n");
       for(row=0;row<4;row++){
         for(col=0;col<3;col++){
            printf("%d  ",dfa[row][col]);       }
            printf("\n");
            }

    // Iterate over dfa array and display its content using row and col variables
```

Inderprastha Engineering College

```
    printf("Transitions for the given string are as follows:\n");
    for(i=0;i<strlen(string);i++ )           // Iterate over input using i variable.
    {
        switch (state)
        {
            case 1: ch = string[i];
                    if (ch == 'a')
                            state =12 ; // Assign value to state
                    else if (ch == 'b')
                            state =1 ; // Assign value to state
                    else
                            state = 5 ; // Assign value to dead state
                  printf("q1 [%c] ---> q%d\n",ch,state);
                    break;
            case 12:    ch=string[i];
            if(ch=='a')
            {                    state=12;
            }                 else if(ch=='b')
            {                    state=13;                  }
            else
            {                    state=5; }
            printf("q12 [%c] ---> q%d\n",ch,state);
            break;

                  // Write case 12 just like Case 1.

         case 13:    ch=string[i];
         if(ch=='a')
         {                 state=12;               }
         else if(ch=='b')
         {                 state=14;               }
         else{
            state=5;             }
            printf("q13 [%c] ---> q%d\n",ch,state);
            break;

               // Write case 13 just like Case 1.

            case 14: ch=string[i];
            if(ch=='a'){
               state=12;                  }
               else if(ch=='b'){
                  state=1;                }
                  else
                  {                    state=5;                }
                  printf("q14 [%c] ---> q%d\n",ch,state);
                  break;

                  // Write case 14 just like Case 1.

         case 5: printf("q5 [%c] ---> q%d\n", ch, state); // state 5 is dead state.
      }
   }
   if(state==14 )  // Write condition for final state
      printf("%s is terminating at state {1,4}. So, it is an accepted string.\n",strin
g); // Write the final statement as given in the output when the q14 state has reached.
   else
      printf("%s is not terminating at state {1,4}. So, it is not an accepted strin
g.\n",string);  // Write the final statement as given in the output when the q14 state
is not reached.
}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter a string:  aabb
Transition table for NFA is:
1  12  1
2  0  3
3  0  4
4  0  0
Transition table for DFA is:
1  12  1
12  12  13
13  12  14
14  12  1
Transitions for the given string are as follows:
q1 [a] ---> q12
q12 [a] ---> q12
q12 [b] ---> q13
q13 [b] ---> q14
aabb is terminating at state {1,4}. So, it is an accepted string.
```

Test Case - 2

User Output

```
Enter a string:  aab
Transition table for NFA is:
1  12  1
2  0  3
3  0  4
4  0  0
Transition table for DFA is:
1  12  1
12  12  13
13  12  14
14  12  1
Transitions for the given string are as follows:
q1 [a] ---> q12
q12 [a] ---> q12
q12 [b] ---> q13
aab is not terminating at state {1,4}. So, it is not an accepted string.
```

Test Case - 3

User Output

```
Enter a string:  abaabb
```

Test Case - 3

```
Transition table for NFA is:
1  12  1
2  0  3
3  0  4
4  0  0
Transition table for DFA is:
1  12  1
12  12  13
13  12  14
14  12  1
Transitions for the given string are as follows:
q1 [a] ---> q12
q12 [b] ---> q13
q13 [a] ---> q12
q12 [a] ---> q12
q12 [b] ---> q13
q13 [b] ---> q14
abaabb is terminating at state {1,4}. So, it is an accepted string.
```

S.No: 1	Exp. Name: ***Fill the missing code by analyzing the code motion. Avoid any type of computation within the loop***	**Date:**

Aim:

Fill the missing code by analyzing the code motion. Avoid any type of computation within the loop.

```
#include <stdio.h>
void main() {
        int a;
        int i,counter = 0;
        printf("Enter the value of a and i: ");
        scanf("%d %d", &a, &i);
        while(i>5) {
                counter = (a / 10) +i;
                i--;
        }
        printf("%d\n",counter);
}
```

Original:

```
while(i<100) {
        a = Sin(x)/Cos(x) + i;
        i++;
}
```

Optimized:

```
t= Sin(x)/Cos(x) + i;
while(i<100) {
        a = t + i;
        i++;
}
```

Source Code:

computation.c

```
#include<stdio.h>
void main(){
   int a,i,counter=0;
   printf("Enter the value of a and i: ");
   scanf("%d %d",&a,&i);
   a /= 10;
   while(i>5){
      counter=a+i;
      i--;
      }
      printf("%d\n",counter);

}
```

Execution Results - All test cases have succeeded!

Test Case - 1

User Output

```
Enter the value of a and i:  100 10
16
```

Test Case - 2

User Output

```
Enter the value of a and i:  200 10
26
```

Test Case - 3

User Output

```
Enter the value of a and i:  1500 30
156
```