R. praveen Ganesh

192011218

CSE

S.no : 33.

Exp-1 :- Develop a lexical Analyzer to identify identifier constants, operators using c program.

Aim :-

To develop lexical Analyzer to identify constants, operators using c program.

Algorithm :-

* start the code and import the package.
* Declare the variable
* Assign the variable
* save and Run the code.

Program :-

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
int main()
{
  int i, c=0, m, cc=0, oc=0, j;
  char b[30], operator[30], identifier[30], constants[30]
  printf("enter string:");
  scanf("%[^\n]s", &b);
  for(i=0; i<strlen(b); i++)
  {
    if(isspace(b[i]))
    {
      continue;
    }
    else if(isalpha(b[i]))
    {
      identifier[ic] = b[i];
      ic++;
    }
    else if(isdigit(b[i]))
```

```
{
    m=(b[i]-'0');
    i=i+1;
    while(isdigit(b[i]))
    {
      m=m*10+(b[i]-'0');
      i++;
    }
    i=i-1;
    constants[cc]=m;
    cc++;
}
else
{
    if(b[i]=='*')
    {
      operators[oc]='*';
      oc++;
    }
    else if(b[i]=='-')
    {
      operators[oc]='+';
      oc++;
    }
    else if(b[i]=='=')
    {
      operators[oc]='=';
      oc++;
    }
}
printf("identifier:");
for(j=0;j<ic;j++)
{
    printf("%c",identifiers[j]);
}
printf("\n constants:");
for(j=0;j<cc;j++)
{
    printf("%d",constants[j]);
}
```

```
printf("\n operators:");
for(i=0;i<oc;i++)
{
  printf("%c",operator[i]);
}
}
```

output:-

```
enter string: a=b+c+e+100
identifiers: a b c e
constants: 100
operators: = + + +
```

Exp - 2 :- Develop a lexical Analyzer to identify whether a given line comment or not using C.

S:No:33

Aim:-

To develop a lexical Analyzer to identify whether given comment or not using C.

Algorithm:-

1. Start the code and import package.
2. Declare the variable.
3. Assign the variable.
4. Save & Run the code.

Program:-

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char com[30]; , i=2, a=0;
  printf("\n Comment:");
  gets(com);
  if(com[0]=='/')
  {
     if (com[1]=='/')
        printf("\n it is Comment");
     else if (com[1]=='*')
     {
        for(i=2; i<=30; i++)
        {
           if(com[i]=='*' && com[i+1]=='/')
           { print("\n is comment");
             a=1;
             break;
           }
           else
             continue;
```

```
    if(d==0)
    printf("\n is not comment");
}
else
    printf("is not comment");
}
```

output:-

Input : Enter

Comment : // hello

output:- It is a

Comment.

Exp-3 :- Design a lexical Analyzer for given langua, should ignore the redundant space, tab and new and ignore comment using c. Sl.No: 33

Aim :-

To develop lexical Analyzer for given language should ignore the redundant space, tab & comment using c.

Algorthim :-

1. Start the code and import pakage.
2. Declare the variable
3. assign the variable
4. save & Run the code

program :-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
int is keyword (char buffer[])
{
char keywords [32][10] = {"main", "auto", "break
"case", "char", "const", "continue", "default",
"do", "double", "else", "enum", "extern", "fl
"for", "goto", "if", "int", "long", "register", "sho
"return", "void", "printf", "while"};
int i, flag =0;
for (i=0; i<32; ++i)
{
```

```
if( strcmp(keywords[i], buffer) == 0) {
    flag = 1;
    break;
  }
}
return flag;
}
int main() {
char ch, buffer[15], operator[] = "+-*/%=";
FILE *fp;
int i, j = 0;
fp = fopen("lex_input.txt", "r");
if (fp == NULL) {
printf("Error while opening\n");
exit(0);
}
while ((ch = fgetc(fp)) != EOF) {
for (i=0; i<6; i++); {
if (ch == operator[i])
printf("%c operator\n", ch);
}
if (isKey(buffer) == 1)
print("key", buffer)
}.
```

S.No: 33

Input:-

```
lex_input.txt
int main() {
    int a, b, c;
}print("%d", c);
```

output:-

```
int is keyword.
a is identifier
b is identifier
c is identifier
= is operator
+ is operator
/ is operator
```

Exp-4:- Design a lexical Analyzer to validate operator to recognize the operator +, -, *, /, using regular arithmetic operator using C.

S:No:33

Aim:-

To develop to design lexical Analyzer to validate operator to recongnize operation +, -, *, /, using regular arithmentic operator using C.

Algorithm:-

1. Start the code and import pakcage.
2. Declare the variable.
3. Assign the variable.
4. Save & Run the code.

Program:-

```
#include <stdio.h>
#include <conio.h>
int main()
{
  char s[5];
  printf("\n Enter any operator");
  gets(s);
  switch(s[0])
  {
     case '>';
        if (s[1] == '=')
           printf("Greater than or equal");
        else
           printf("\n Greaterthan");
           break;
     case '<':
        if (s[1] == '=')
           printf("\n Less than or equa
```

S.No: 33

```
        else
            printf("less than");
        break;
    case '=':
        if (s[1] == '=')
            printf("Equal to");
        else
            printf("Bit Not");
        break;
    case '&':
        if (s[1] == '&')
            printf("AND");
        else
            printf("Bitwise AND");
        break;
    case '+':
        printf("Addition");
        break;
    case '-':
        printf("substraction");
        break;
    case '*':
        printf("multiplication");
        break;
    default:
        printf("Not a operator");
}
```

output:

```
Enter any operator: <=
Less than or Equal
```

Ex-5 :- Design a lexical Analyzer to find the number of whitespaces and newline characters using C.

S:NO: 33

Aim :-

To design lexical Analyzer to find number of whites space and newline character using C.

Algorithm :-

1. Start the code and import package.
2. Declare the variable
3. Assign the variable
4. Save & Run the code.

program :-

```
#include <stdio.h>
int main()
{
char str[100]; //
int word =0 , newline =0, character=0;
Scanf ("%[^~]", &str);
for (int i=0; str[i]!='\0'; i++)
{
    if (str[i]==' ')
    {
     words++;
    }
    else if (str[i]=='\n')
    {
      newline++;
       words++;
    }
    else if (str[i] != ' ' && str[i]!= '\n'){
      characters ++
} }
```

```
if (character==' ' || character==',' || character=='\n' || character=='\t')
{
    word++;
    newline++;
}
printf("Total words : %d", words);
printf("Total lines : %d", nlines);
printf("Total character : %d", character);
return 0;
}
```

S.N : 33

output :-

```
void main()
{
    int a, b;
    a = b + c;
    c = d + e;
}
```

Total words : 18

Total line : 7

Exp-6 :- Develop a lexical Analyzer to test given identifier is valid or not using c.

S:no:33

Aim :-

To develop lexical Analyzer to test given identifier is valid or not using c.

Algorithm :-

1. start the code and import packag.
2. Declare the variable.
3. Assign the variable.
4. Save & Run the code.

program :-

```
#include <stdio.h>
#include <stype.h>
#include <conio.h>
  int main()
  {
    char a[10];
    int flag, i=1;
    Printf ("Enter identifier:");
     get(a);
    if (isalpha (a[10])
        flag =1;
    Else
       Printf ("Not a valid identifier");
         while (a[i]!='\0')
    {
      if(!isdigit (a[i]) && ! isalpha(a[i]))
      {
```

S:No: 33

```
        flag=0;
        break;
    } i++;
}
if(flag==1)
    printf("valid identifier");
}
```

output:-

```
enter an identifier : abc123
valid identifier
```

Exp-7 :- write a c program to find FIRST()- predictive parser for given grammar using C

S → AaAb | BbBa
A → ∈
B → ∈

S:N10:33

Aim :-

To write c program to find FIRST()- predictive parser for given grammar.

Algorithm :-

1. Start the code and import package
2. Declare the variable.
3. Assign the variable.
4. Start & Run the code.

program :-

```
#include <stdio.h>
#include <ctype.h>
void FIRST(char[], char);
void addToResultset(char[],char);
int numofproducts;
char productionset[10][10];
int main()
{
    int i;
    char choice;
    char c;
    char result[20];
    printf("How number of production");
    scanf("%d", &numOfproduction);
```

```
for (i=0; i<numOfproductions; i++)
{ printf ("Enter number productions %d:", i+1);
  scanf ("%s", productionset[i]);
}
do
{ printf ("\n find FIRST of:");
  scanf ("%c", &c);
  FIRST (result, c);
  printf ("FIRST(%c)={", c);
  for (i=0; result[i]!='\0'; i++)
  printf ("%c", result[i]);
  printf ("}\n");
  printf ("press 'y' to continue:");
  scanf ("%c", &choice);
}
while (choice=='y' || choice=='Y');
}
void FIRST(char *Result, char c)
{
  int i, j, k;
  char subResult[20];
  int foundposition;
  subResult[0]='\0';
  {
    addToResultSET(Result, c);
    return;
  }
  for (i=0; i<numofproductions; i++)
{
  if (production set[i][0]==c)
  {
    if (productionset[i][2]=='$')
    addToResults (Result, '$');
  else {
```

```
j=2;
while(production set[i][j]!='\0')
{
  foundE posilon=0;
```

S.NO: 33

```
FIRST(subResult, production set[i][j]);
  for(k=0; subResult[k]!='\0'; k++)
addToResultset(Resut, subResut[k]);
    for(k=0; subResut[k]!='b'; k++)
      if(subResut[k]=='$')
      {
        found posilin=1;
        break;
      }
      j++;
}
}
}
}
return;
}
```

output :-

How many number productions = 4

Enter production 1 : S = AaAb

Enter production 2 : S = BbBa

Enter production 3 : A = $

Enter production 4 : B = $

Find the FIRST : S

FIRST(S) = {$ a b}

press Y continue : Y

Exp - 8 :- write the c program find Follow() - predictive parser for given grammar

S → AaAb | bBBa

A → ∈

B → ∈

S : No = 33

Aim:-

To find Follow() - predictive parser for given grarar using c.

Algorithm :-

1. start and the code & import package.
2. Assign the variable.
3. Decleare the variable.
4. start & Run the code.

Program :-

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int limit, x=0;
char production [10] [10], array [10];
void find-first (char, ch)
void find-follow (char ch);
void Array-manipulation (char ch);
int main()
{
    int count;
    char option, ch;
    printf ("production:\n");
    scanf ("%d", &limit);
    for (count=0; count<limit; count++)
```

```
{
  printf ("Number [%d]: ", count+1);
  scanf ("%s", production [count]);
}
do
{
  x=0;
  printf ("Enter value of Find: ");
  scanf ("%c", &ch);
  find_follow (ch);
  printf ("Follow value of %c: {", ch);
  for (count=0; count<x; count++)
  {
    printf ("%c", array [count]);
  }
  printf ("}\n");
  print ("To continue");
  Scanf ("%c", &option);
} while (option == 'y' || option == 'Y');
  return 0;
}
void find_follow (char ch)
{
  int i, j;
  int length = strlen (products [i]);
  if (production [0][0] == ch)
  {
    Array_manipulation ('$');
  }
  for (i=0; i<limit; i++)
  {
    for (j=2; j<length; j++)
    {
      if (production [i] [j] == ch)
```

S! No: 33

```
{ if(production[i][j+1]!='\0')
  {
    find_follow(production[i][0]);
  }
}
}
void find_first(char ch)
{ int i,k;
  if(!(supperch(ch)))
  {
    Array_manipulation(ch);
  }
  for(k=0;k<limit;k++)
void Array_manipulation(char ch)
{
  int count;
  for(count=0;count<=x;count++)
{ if(array[count]==ch)
  {
    return;
  }
  array[x++]=ch;
```

Sort : 33

output:-

```
Enter production : 2
value of production[1] = S=AaAb
value of production[2] = S=BbBa
Follow value S : {$}
To continue, press Y:N
```

Exp-9 :- Implement c program to eliminate left recursion form given CFC.

S.N: 33

Aim :-

To implement c program to eliminate left recursion for given CFG.

Algorithm :-

1. Start the code and import packag.
2. Assign the variable.
3. declare the variable.
4. Start & Run the code.

Program :-

```
#include <stdio.h>
#define size 10
  int main () {
  Char non-terminal;
  Char beta, alpha;
  int num;
  Char production [10] [size];
int index =3
printf ("Enter Number production \n");
Scanf ("%d \n", &num);
printf ("Enter grammar as E=>E-A:");
for (int i=0; i<num; i++)
{
  scanf ("%s", production [i]);
}
for (int i=0; i<num; i++) {
```

```
printf("Grammar : : production[i]);
non_terminal = production[i][0];
if (non_terminal == production[i][index]) {
alpha = product[i][index+1];
printf("left recursive");
while (production[i][index] != 0 && production
                                [i][index] != '|')
    index++;
if (production[i][index] != 0) {
    printf("non_terminal, alpha, non_termi");
}
else
    printf("cant reduced");
}
else
    printf("is not left recursive");
    index;
}
```

S : N = 33

output:—

Enter Number of production : 2

Enter the grammar as E → E+A;

S → (L) | a

L → L,S | S

GRAMMAR : : S→(L) | a is not left recursive

GRAMMAR : : L→L,S | S is left recursive.

Exp-10 :- Implement at c program to Eliminate left factoring from given CFG.

S. No-33

Aim :-

to implement c program to Eliminate left factoring from given CFG.

Algorithm :-

1. Starte the code and import packags.
2. Assign the variable.
3. Declare the variable.
4. Start & run the code.

program :-

```
#include <stdio.h>
#include <string.h>
  int main()
  {
    char gram[20], part1[20], part2[20], modif
    Gram[20], newGram[20], temGram[20];
    int i, j=0, k=0, l=0, pos;
    printf("enter productions");
    gets(gram);
    for(i=0; gram[i]!='\0'; i++, j++)
        part1[j] = gram[i];
        part1[j] = '\0';
        for(j=++i; i=0; gram[j]!='\0';
                        j++, i++)
```

```
part2[i]=gram[j];
part2[i]='\0';
for(i=0;i<strlen(part1)||i<strlen(part2);i++)
{
if(part1[i]==part2[i])
  {
    modifiedGram[k]=part1[i];
    k++;
    pos=i+1;
  }
}
for(i=pos,j=0;part1[i]!='\0';i++,j++){
    newGram[j]=part1[i];
}
modifiedGram[k]='X';
modifiedGram[++k]='\0';
newGram[j]='\0';
printf("\n S->%s",modifiedGram);
printf("\n X->%s\n",newGram);
}
```

Output :-

Enter prodution = S → iEtS | iEtSeS | a

S → iEtSX

X → | eS | a

Exp - 11 :- Implement a program to perform Symbol table operations.

S:no = 33

Aim :-

To implement a program to perform Symbol table operations.

algorithm :-

1. Start the code and import packag.
2. Assign the variable.
3. Declare the variable
4. start the Run the code.

program :-

```
#include <stdio.h>
#include <string.h>
 int cnt=0;
 struct symtab
 {
   char label [20];
    int addr;
 }
 sy [60];
 void insert ();
  int search (char *);
   void display ();
   void modify();
   {
     int ch, val;
     char lab [10];
      do
      {
```

```
printf("1: insert 2: display 3: search 4: modify
        5: exit");
scanf("%d", &ch);
switch (ch)
{
case 1:
        insert();
        break;
case 2:
        display();
        break;
case 3:
        print("Enter label");
        scanf("%s", lab);
        val = search(lab);
        if (val == -1)
        printf("label found");
        else
          printf("not label found");
        break.
case 4:
        modify();
        break;
Case 5:
        exit(0);
        break
}
void insert()
{
  int val;
      char lab[20];
       int symbol;
       printf("Enter label:")
```

S.NO = 33

```
int search (char *s)
{
   int flag=0, i; for (i=0; i<cnt; i++)
   {
     if (strcmp(sy[i].label, s)==0)
        flag=1;
   }
return flag;
}
void display()
{
   int i;
   for (i=0; i<cnt; i++)
   printf("%s", sy[i].label, sy[i].addr);
}
```

output:-

1. insert
2. display
3. search
4. modify
5. Exit

1

Enter the label a

Enter address 100

5

exit

Exp-12 :- write c program to construct the recursive descent for the grammar.

S : A|a = 33

Aim :-

To write c program to Construct the recursive descent for grammar.

Algorithm :-

1. start the code and import packag
2. Declare the variable
3. Assin the variable
4. start & Run the code.

program :-

```
#include <stdio.h>
#include <string.h>
    char input[100];
    int i, l;
    void main()
{
    printf("Recursive pasing :");
    printf("E->TE'|E'->+TE'@nt->E+T|");
    printf("enter string:");
     gets(input);
    if(E())
    {
      if(input[i+1]=='\0')
      Printf("string accepted");
      Else
```

```
printf("\n string not accepted");
}
else
printf("string not accepted");
getch();
}
```

E: n10 = 33

```
if(T())
{
if (EP())
return(1);
else
return(0);
}
else
return(0);
}
EP()
{
if (input[i]=='+')
{
i++;
if (T()) {
if (EP())
return(1);
else
return(0);
}
else
return(0);
}
else if (input[i]>='a' && input[i])
{i++
return(0);
}}
```

output :-

Recursive descent parsing

$E \rightarrow TE'$, $E' \rightarrow +TE'$ @ string not accepted

Exp-13 :- Write C program to implement either top down parsing technique or Bottom up parsing technique to check whether the given input string grammar or not

S.No =33

Aim :-

Write C program to check grammar or not for top down parsing or Bottom up parsing.

Algorithm :-

1. Start the code and import packag.
2. Declare the variable
3. Assin the variable
4. Start & Run the code.

Program :-

```
#include <stdio.h>
#include <string.h>
void main()
{
  char string[50];
  int flag, count=0;
  printf("the grammar is → S→aS, S→ab,
                          S→ab");
  printf("enter string:");
  gets(string);
  if (string[0]=='a') {
                flag=0;
                for (count=1; string[count-1]!=
                     count++) {
```

S.No: 33

```
if (string[count] == 'b')
{
    flag = 1;
    continue;
}
else if (flag == 1) && (string[count] == 'a')
{
    printf("the string");
    break;
}
else {
    printf("string accepted.");
}
}
}
```

output:-

the grammar is : S → aS, S → Sb, S → ab

enter the string : abb

string accepted.

Exp-14 :- Implement the Concept of shift reduce parsing in c programming.

S.NO: 23

Aim:-

To implement the shift reduce parsing using c program.

Algorithm:-

1. Start the code and import packag.
2. Declare the variable
3. Assign the variable
4. Start & Run the code.

program:-

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char ip_sym[15], stack[15]; int ip_ptr=0, st_pto
char temp[2], temp2[2];
char act[15];
void check(); int main()
{
printf("u. shift Reduce parser");
printf("u Grammar u");
printf("u E->E+E"); printf( E->E*E E->a|b)
printf("u enter the input symbol");
get(ip_sym);
Printf("u stack table"); printf("stack");
```

```
temp[0] = ip_sym[ip_ptr]; temp[1] = '\0';
strcat(act, temp);
{
stack[st_ptr] = ip_sym[ip_ptr];
stack[str_ptr] = '\0'; ip_sym[ip_ptr] =
printf("%d\t%s\t%s", ip, smp, act);
temp[0] = ip_str[ip_ptr];
st_ptr++;
}
st_ptr++; check();
}
void check()
{
int flag = 0; temp2[0] = stack[st_ptr];
if (!strcmpi(temp2, "a") || (!strcmpi
{
stack[st_ptr] = 'E'; if (!strcmpi(temp,
printf("\nE->a", stack, ip_sym);
flag = 1;
}
if ((!strcmpi(stack, "E+E") || (!strcmpi
(stack, "E\E"))
{
stcpy(stack, "E");
printf("\nE->E+E", stack, ip_sym);
exit();
printf("\nE+E", stack, ip_sym);
```

S.No 33

S: No: 33

```
if (flag==0)
{
  printf ("reject", stack, ip-sym);
  Exit (0)
}
return;
}
```

output:-

Shift Reduce parser

GRAMMER

$E \rightarrow E+E$

$E \rightarrow E/E$

$E \rightarrow E*E$

$E \rightarrow a|b$

input : a+b

stack	input	action
$	a+b$	
$a	+b$	shift a
$E	+b$	$E \rightarrow a$
$E+	b$	shift +
$E+b	$	shift b
$E+E	$	$E \rightarrow b$
$E	$	$E \rightarrow E+E$
$ E	$	ACCEPT

EXP-15 :- write a C program to implement the operator precedence parsing.

3:ab=33

Aim :-

To implement the operator precedence parsing using C program.

Algorithm :-

1. Start the code and impot packag
2. Declare the variable
3. Assin the variable.
4. Start & Run the code.

Program :-

```
#include <stdio.h>
#include<string.h>
  char *input;
 int i=0;
  char lasthandle[6],stack[50],handles[][
  =[ 5:(" ,"E*E","E+E","i",5^E),
 int top=0,l;
  char prec[9][9]={
 int getindex(char c)
 {
  switch(c)
 {
  case '+':return 0;
  case '-':return 1;
```

```
case '+' return 3;
case '*' return 2;
case '^' return 4;
case '(' return 5;
case ')' return 6;
case '#' return 7;
case '$' return 8;
}
}
int shift()
{
stack[t+top] = *(input+i+1);
stack[top+1] = '\0';
}
int reduce()
{
int i, len, found, t;
for(i=0; i<5; i++)
{
len = strlen(handles[i]);
if(stack[top] == handles[i][0] && top+1 == len)
{
found = 1;
for(t=0; t<len; t++)
{
if(stack[top-t] != handles[i][t])
{
found = 0;
break;
}
}
if(found == 1)
{
stack[top+1] = 'E';
```

5/10/22

S:No: 83

```
top = top-(+1)
Stack [top+1] = '\0'
return 1;
}
}
ret 0;
}

void main()
{
  int i;
  input = (char *) malloc (50 * size (char));
  printf("enter string");
  scanf ("%s", input);
  Input = strcat (input, "$");
  l = strlen (input);
}}
  if (strcmp (stack, "$E$") == 0)
     printf ("Accepted");
  else
     printf ("Not Accepted");
}
```

output :-

Stack	Input	Action
$i	*(i+i)*i$	shift
$E	*(i+i)i$	Red E→i
$E*	(i+i) i$	shift.
$E*(	i+i)i$	shift
$E*(E+i)	i)i$	R E→i
$E*(E+i)E	$	shift
$E*	$	Red E→i
$E	$	shift.

Ex-16 :- Write a program to generate three address code representation for given input statement.

S.No = 33

Aim :-

To Generate three address code representation for given statement using C program.

Algorithm :-

1. Start the code and import packag
2. Declare the variable.
3. Assign the variable.
4. Start & Run the code

program :-

```
#include <stdio.h>
#include <string.h>
 struct three
{
 char data[10], temp[7];
} s[30];
 int main()
{
  char d1[7], d2[7] = 't';
  int i=0, j=1, len=0;
  FILE *f1, *f2;
  f1 = fopen("sum.txt", "r");
  f2 = fopen("out.txt", "w");
  while (fscanf(f1, "%s", s[len].data) != EOF)
  len++;
  itoa(j, d1, 7);
```

```
if(!strcmp(s[3].data, "+"))
{
fprintf(f2, "%s = %s + %s", s[j].temp,
                s[j+2].data);
  j++;
}
else if(!strcmp(s[3].data, "-"))
{
 fprintf(f2, "%s = %s - %s", s[j].temp);
 j+1;
}
 for(i=4; i<len; i+=2)
{
 itoa(j, d1, 7)
 strcat(d2, d1);
 strcat(s[j].temp, d2);
 if(!strcmp(s[i+1].data)
 fprintf(f2, "%s = %s + %s");
 else
   strcpy(d1, "");
   strcpy(d2, "+");
   j+1
}
 getch();
}
```

s:no = 33;

output :-

Input : int1, in2, in3

$t_1 = int1 + in2$

$t_2 = t_1 - in3$

out = t_2 //.

Exp-17 :- write a c. program for implementing a lexical Analyzer to scan and count number of character, words and line in file.

S.No.:33

Aim :-

To implement the lexical Analyzer to scan & count number of character, word using c.

Algorithm :-

1. start the code and import packag.
2. declare the variable.
3. Assign the variable.
4. Start & Run the code.

program :-

```
#include <stdio.h>
 int main()
{
  char str[100];
  int words = 0, newline=0, character=0;
  Scanf("%[^\n]", &str);
  for(int i=0; str[i] != '\0'; i++)
  {
    if (str[i] == ' ')
     { words++;
     }
     else if (str[i] == '\n')
     {
       newline++;
```

```
}
else if (str[i] != ' ' && str[i] != '\n'){
    character++;
}
}
if (character > 0)
{
    word++;
    newline++;
}
printf("\ntotal words %d", words);
printf("\ntotal lines : %d", lines);
printf("\ntotal character %d", character);
return 0;
}
```

S.NO = 33

output :-

```
void main()
{
    int a, b, c;
    c = a + b;
}
```

total number word = 15

Total lines = 5

Exp-18 :- write a program to implement the back end of compiler.

S:NO:33

Aim :-

To implement the back end of compiler using C program.

Algorithm :-

1. Start the code and import packag.
2. Declare the variable.
3. Assign the variable.
4. Start the code and Run program.

program :-

```
#include <stdio.h>
#include<string.h>
 void main()
{
   int n,i,j;
   char a[50][50];
   printf("enter no: intermediate:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
  {
    printf("enter 3 address code:%d",i+1);
    for(j=0;j<6;j++)
    {
      scanf("%c",&a[i][j]);
  }  }
   printf("the generated is:");
```

```
for (i=0; i<n; i++)
{
printf ("\n mov %c, R%d", a[i][3], i);
 if (a[i][2]=='-')
  {
   printf ("\n sub %c, R%d", a[i][3], i);
  }
 if (a[i][2]=='+')
  {
   printf ("\n add %c, R%d", a[i][3], i);
  }
 if (a[i][2]=='*')
  {
   printf ("\n multi %c, R%d", a[i][3], i);
  }
 printf ("\n move %d, %c, i, a[i][1]);
 printf ("\n");
 }
  return 0;
}
```

Output :-

```
enter no: intermediate code : 2
enter 3 address 1 : a=b+c
enter 3 address 2 : d=n*d

generated code :
  mov b, R0
  add c, R0
  mov R0, a
  mov n, R1
  mov d, R1
  mov R1, d
```

EXP.-19 :- write c program to computer LEADING operator precedence parser for given grammar.

S.NO = 33

Aim :-

To write c program to computer LEADING() operator precedence parser for grammar.

Algorithm :-

1. start the code and import packag.
2. Declare the variable
3. Assign the variable.
4. Start & Run the code.

program :-

```
# include <stdio.h>
# include <conio.h>
char arr[18][3] = {{'E','+','F'}, {'E','*','F'},
{'E','(','F'}, {'E'}, {'E',')','F'}, {'E','$','n'},
{'T','+','F'}, {'T','(','F'}, {'T',')','F'},
{'T','T','F'}, {'T','$','F'}}.
char prod [] = "EETTFF";
char res [6][3] = {{'E','+','T'}, {'T','10'},
{'T','*','F'}, {'F','10'}, {'(','i','y'}
char stack [5][2];
int top = -1;
void install (char pro, char re) {
```

S.No = 33

```
int r;
for (i=0; i<18; i++) {
    if (arr[i][0] == pro && arr[i][1] == re)
    {
        arr[i][2] = 'T';
        break;
    }
}
++top;
stack[top][0] = pro;
stack[top][1] = re;
}
int main() {
    int i=0, j;
    char pro, re, pri='1';
    for (i=0; i<6; i++) {
        for (j=0; j<3 && res[i][j]!='0'; j++) {
            if (res[i][j]=='+' || res[i][j]=='*' ||
                res[i][j]=='(' || res[i][j]==')') {
                install (prod[i], res[i][j]);
                break;
            }
        }
    }
    for (i=0; i<18; i++)
    {
        printf ("\n");
        for (j=0; j<3; ++j)
            printf ("%c", arr[i][j]);
    }
    getch();
```

S:No=33

```
printf("\n");
for(i=0; i<18; i++)
{ if (pr != arr[i][0])
  {
    pri = arr[i][0];
    printf("\n %c -> ", pri);
  }
  getch();
}
```

output:-

E + T
E * T
E C T
E) F
E i T
E $ F
F + F
F * F
F C F
F) F
F i T
T * T
T C T
T) F
T $ F
E → + * Ci
F → Ci
T → * Ci

Exp -20 :- write c program to computer TRAILING() - operator parser to given :

S:NO = 33

Aim :-

To write c program to computer TRAILING() - operator parser for grammar :

Algorithm :-

1. start the code and import packag
2. declare the variable.
3. Assign the variable.
4. Start & Run the code.

Program :-

```
#include <stdio.h>
#include <conio.h>
char arr [18] [3] = {{'E', '+', 'F'},
{'E', '*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}
{'E', '+', 'F'}, {'F', '*', 'F'}, {'F', '(', 'F'},
{'F', 'i', 'F'}, {'F', '$', 'F'},
{'T', '+', 'F'}, {'T', '*', 'F'},
{'T', 'i', 'F'};

Char prod [6] = "EETTFF";
char res [6] [5] = {{'E', '+', 'T'},
{'T', '0', '0', '0', '0'}, {'E', '0', '0'}
{'T', '0', '0'};
```

S.N: 33

```
char stack[5][2];
int top=-1;
void install(char pro, char re){
  int i;
  for(i=0; i<18; i++)
  { if(arr[i][0]==pro && arr[i][1]==re)
  {
  }
  }
  top++;
  arr[2]=i+1;
  stack[top][0]=pro;
  stack[top][1]=re;
int main(){
  int i=0,j;
  char pro,re,pri=' ';
  for(i=0;i<6;++i)
  {
    for(j=2;j>=0;--j)
  {
  if(res[i][j]=='+'||res[i][j]=='*'||
     res[i][j]=='('||res[i][j]==')'||
     res[i][j]=='$')
    {
    install(prod[i],res[i][j]);
    break;
```

```
Else if (res[i][j] == 'E' || res[i][j] == 'F'
    || res[i][j] == 'T' || res[i][j] == 'i')
    break;
```

S.NO = 33

```
    }
}
  for (i=0; i<18; i++)
  {
    if (pri != arr[i][0])
    {
      pri = arr[i][0];
      printf ("\n\t c-> %c", pri);
    }
    if (arr[i][2] == 'T')
      printf ("%c", arr[i][1]);
  }
}
```

output :-

E + E
E * E
E (E
E) E
E i E
E $ E
F + F
F * F
F (E
E) E
E $ E
T + E
T * E
T (E

E →
F →
T →