

R. praveen Ganesh

192011218

CSE

S.no : 83.

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

Objetive :-

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("%s", b);
    for (i=0; i<strlen(b); i++)
    {
        if (isspace(b[i]))
            continue;
        else if (isalpha(b[i]))
        {
            identifier[cc] = b[i];
            cc++;
        }
        else if (isdigit(b[i]))
            constants[m] = b[i];
            m++;
        else if (operator[i])
            operator[oc] = b[i];
            oc++;
    }
}
```

if ( $b[i] == '0'$ ) { constant or identifier }  $\rightarrow f = f + 1$

$f = f + 1;$   $\rightarrow$  incrementing value of f by 1

loop until  $b[i] \neq '0'$  (add numbers until f is 1)

while ( $b[i] \neq '0'$ )

{  $m = m * 10 + (b[i] - '0');$

statement of if block is executed because of f

$i = i - 1;$

constants [cc] = m;

cc++;

}  $\rightarrow$  another while loop branch with f=0

else

{ if ( $b[i] == '*'$ )

{ operators [oc] = '\*';  $\rightarrow$  operators initialized

oc++;

}

else if ( $b[i] == '-'$ )  $\rightarrow$  digit or character

{ operators [oc] = '-';  $\rightarrow$  operators initialized

oc++;

}

else if ( $b[i] == '='$ )

{ operators [oc] = '=';  $\rightarrow$  operators initialized

oc++;

}

printf(" identifier(%d)");  $\rightarrow$  output

for (j=0; j < cc; j++)  $\rightarrow$  i.e. (0 to f-1)

{ printf("%c", identifiers[j]);

}

printf("\n constants: ");

for (j=0; j < cc; j++)  $\rightarrow$  i.e. (0 to f-1)

{ printf("%c", identifiers[j]);

}

(0 to f-1) right side of code

printf("%c", constants[j]);

}

(0 to f-1) right side of code

printf("No operators"); return 0; }  
for (i=0; i<98; i++)  
    & printf(" %c", operator(i));

3

3. Making printf of recognize tokens in not work. or  
output:-

enter string: a=btcte+100

Identifiers : a b c t e

Constants : 100

Operators : = + \* +

but it display

tokens with space after them

obviously both operators &

constants with spaces are

also added with them

-1 Error

eliminate a character ||

eliminate a character &

eliminate =

(a=btcte+100) ans 100

(C:\Users\asus\OneDrive\

\Desktop\2020\

\WPS\10\mos) 31

(V=2019 mos) 31

(S:\mos) 31 100) 31 ans

(V=2019 mos) 31 2020

(S:\mos) 31 2020) 31

(V=2019 mos) 31 100 - 100) 31

(S:\mos) 31 100) 31 ans

(V=2019 mos) 31 100

(S:\mos) 31 100) 31 ans

(V=2019 mos) 31 100

(S:\mos) 31 100) 31 ans

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 [i] == '/')
            printf ("\n it is comment");
        else if (com [i] == '*')
        {
            for (i=2; i<=30; i++)
            {
                if (com [i] == '*' && com [i+1] == '/')
                {
                    print ("\\n is comment");
                    a=1;
                    break;
                }
            }
            if (a==1)
                continue;
        }
    }
}
```

not C++ (class) you can't do this (10, 93)  
can print out whatever you want  
3 print out additional printf fun  
printf C is not comment

so it will print whatever you want  
outputs - print whatever you want

Input : Enter

Comment : // hello

Output : It is a Comment

Comment :

short talk with people

it's like a short talk  
(it's like a short talk)

defined "define" ("#define") - to be absorbed into

"#include" "standard" "math" "math.h"

#include "limits" "stdio" "stdlib" "fopen"

for ("for" "for" "for" "for" "for" "for")

"define" ("define") "define" ("define")

"#include" ("#include") "define" ("define")

"#include" ("#include") "define" ("define")

Exp - 3 :- Design a lexical Analyzer for given language, should ignore the redundant space, tab and new and ignore comment using c.

S: No: 33

Aim:-

To develop Lexical Analyzer for given language  
Should ignore the redundant space, tab & comment 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 <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", "ff",
    "for", "goto", "if", "int", "long", "register", "sh
    "return", "void", "printf", "while"};
    int i, flag = 0;
    for (i=0; i<32; ++i)
    {
```

if (strcmp(keywords[i], buffer) == 0) {  
 flag = 1; /\* flag to indicate if keyword found \*/  
 break; /\* to break the loop if keyword found \*/  
}

return flag;

3. If not keyword [local variable or symbol or  
int main() {  
 FILE \*fp;  
 int i, j = 0;  
 fp = fopen("lex\_input.txt", "r");  
 if (fp == NULL) {  
 printf("Error while opening\n");  
 exit(0);  
 }

3. while ((ch = fgetc(fp)) != EOF) {  
 for (i = 0; i < 6; i++) {  
 if (ch == operator[i])  
 printf("%c operator\n", ch);  
 }  
 if (is\_kw(buffer) == 1)  
 print("key", buffer);

Input: lex\_input.txt int main() {  
 int a, b, c; }  
 print(a, b, c);

Output:

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 recognize operation +, -, \*, /, using regular arithmetic operator 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 s[5];
    printf("Enter any operator");
    gets(s);
    switch(s[0])
    {
        case '+':
            if(s[i] == '=')
                printf("Greater than or equal");
            else
                printf("Greater than");
            break;
        case '<=':
            if(s[i] == '=')
                printf("Less than or equal");
            else
                printf("Less than");
    }
}
```

S.no: 33

```

else
    printf("less than"); // logical operators
break; // logical operators
case '=':
    if (s[i] == '=') {
        printf(" equals");
    }
    else
        printf(" not equal");
    break;
case '<=':
    if (s[i] == '<=') {
        printf(" less than or equal");
    }
    else
        printf(" greater than or equal");
    break;

```

Case '+'

```

printf(" addition");
break;

```

Case '-'

```

printf(" subtraction");
break;

```

case '\*':

```

printf(" multiplication");
break;

```

default:

```

printf(" not a operator");
break;

```

output:

Enter any operator: <=

Less than or equal:

Output: less than or equal

Output: less than or equal

Ex-5: Redesign 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 whitespace 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]; // Input string
    int word = 0, newline = 0, character = 0;
    scanf("%s", 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 == '\n') { // Corner case 2,3  
    word++;  
    newline++;  
}  
printf("Total words: %d\n", words);  
printf("Total lines: %d\n", newline);  
printf("Total character: %d\n", character);  
return 0;
```

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.

Aim:-

S: No. 33

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

Algorithm :- (with time limit 4 min)

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 <ctype.h>
#include <conio.h>

int main()
{
    char a[10];
    int flag, i=1;
    printf("Enter identifier:");
    getch();
    if (isalpha(a[0]))
        flag = 1;
    else
        printf("not a valid identifier");
    while (a[i] != '\0')
    {
        if (!isdigit(a[i]) && !isalpha(a[i]))
        {
```

if (flag == 0) {  
 if (isalpha(c)) {  
 if (c >= 'A' & c <= 'Z') {  
 flag++;  
 } else if (c >= 'a' & c <= 'z') {  
 flag++;  
 } else {  
 break;  
 }  
 } else if (c == '\_') {  
 flag++;  
 } else if (c >= '0' & c <= '9') {  
 flag++;  
 } else {  
 break;  
 }  
}  
if (flag == 1) {  
 printf("valid identifier");  
}

**output:-**

Enter an identifier : abc123

valid identifiers: `fragm bits sbos cells cells`.  
• `std::vector<int> std::vector<double>` - `s`  
• `std::vector<int> apizas` - `s`  
• `sbos` with `new & delete` - `H`

Q.P :- write a c program to find FIRST() -  
predictive parser for given grammar using C

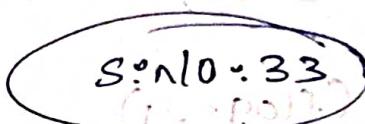
$$S \rightarrow aAb \mid bBa$$

$$A \rightarrow E$$

$$B \rightarrow E$$

S : nlo : 33

(1. 100) 47



### Aim :-

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

### Algorithm :-

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

### Program :-

```
#include <stdio.h>
#include <ctype.h>
void FIRST(char [3], char);
void addToResultSet(char [3], 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);
}
```

S: No: 33

```

for (i=0; i<numOfproductions; i++)
{
    printf("Enter Number of productions >.d : ", i);
    scanf("%d", &i);
}

do
{
    printf("Enter first of %c : ", c);
    scanf("%c", &c);
    FIRST(result, c);
    printf("FIRST(%c) = %s", c, result);
    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';
    for (i=0; i<numOfproductions; i++)
    {
        if (productionSet[i][0] == c)
        {
            if (productionSet[i][2] == '$')
                addToResults(result, '$');
            else
                for (j=0; productionSet[i][j] != '\0'; j++)
                    if (productionSet[i][j] == '#')
                        break;
                    else
                        subResult[k] = productionSet[i][j];
                        k++;
        }
    }
    result[0] = '\0';
    for (i=0; subResult[i] != '\0'; i++)
        addToResults(result, subResult[i]);
}

```

Output :- How many number productions = 4

How many number products = 9

Enter production 1 :  $S = AaAb$

Enter production 2:  $s = BbBg$

Enter production 4 : B = S cancel

Find the first 1000 terms of  $\{a_n\}$ .

$\text{FIRST}(s) = \{\$ab\}$  [;]  $s = ab$  with  $a = \$$

press Y to continue:

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

$$S \rightarrow AaAb \mid EbBa$$

$$A \rightarrow E$$

$$B \rightarrow E$$

$$S : No = 33$$

Aim:-

To find Follow( $S$ ) - predictive parser for given grammar  
using C.

Algorithm:-

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

Program:-

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

int tlimit, x=0, count=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: ");
    scanf("%c", &option);
    for (count=0; option=='a' || option=='b'; count++)
        if (option=='a')
            find-first('a');
        else
            find-follow('b');
}
```

```

{ followbuff mapping a pair of row :- 8 - 9x3
printf ("Number of d.f.s : ", count);
scanf ("%d", production [count]);
y = 0;
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");
    printf ("To continue a);");
    scanf ("%c", &option);
    if (option == 'y' || option == 'Y')
        return 0;
}
while (option == 'y' || option == 'Y');
}

void find_follow (char ch)
{
    int i, j;
    int length = strlen (products [i]);
    if (production [0] [0] == ch)
    {
        arrayManipulation ('$');
        for (i = 0; i < limit; i++)
            for (j = 0; j < length; j++)
                if (production [i] [j] == ch)

```

Profit (production cost)  $\geq$   $15 \times 10^3$  (target profit)  $\Rightarrow$   $S_{\text{opt}} = 33$

SORF: 33

{ find - follow (production  $i \rightarrow f$ ) ; }.

3

3) Implementations of searching & traversing are  
void find-first (char ch)

```

f int i, k;
if (!isupper(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;
    }
}

```

Output:- [35127 0] (notsuboptimal)

Enter production: 2      [www.xsibsoft.com](http://www.xsibsoft.com)

value of production ( $i$ ) =  $S = AaAb$

value of production:  $GJ = s = BbBq$

Follow values of  $f(x)$  &  $\{f(x)\}$  during

To continue, press  $y_1 = nf$ .) so?

(652 003) abdy (2.1.4) 71802

~~Year C 1945-1946 - 1946-1947~~

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

Aim:-

S.M:33

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

Algorithm:-

1. Start the code and import package.
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 of production");
    scanf ("%d", &num);
    printf ("Enter grammar as E->E-& .");
    for (int i=0; i<num; i++) {
        scanf ("%s", production [i]);
        for (int j=0; j<size; j++) {
            if (production [i][j] == '>') {
                index = j;
            }
        }
    }
}
```

printf ("Grammar : "); production [i];  
non-terminal = production [i] [0]. S:N=33

if (non-terminal == production [i] [index]) {

alpha = products [i] [index+1];

printf ("left recursive"); branching [i];

while (production [i] [index] != & production

[i] [index] != 1) {

index++; if (prod[i][index] == 0) { skip++; } else {

if (production [i] [index] != 0) { skip++; } else {

printf ("non-terminal, alpha, non-terminal");

else

printf ("can't reduced"); more

} else

printf ("is not left recursive");

index++;

if

fibonacci [0] <= n, (0<= i <= n), (0<= j <= n)

output:—

Enter number of production : 2

Enter the grammar as,  $S \rightarrow HGA$ ;

$S \rightarrow L$

$L \rightarrow L, S | S$

(HIGRAMMAR!  $\Rightarrow$  SICL) la is not left

recusive

GRAMMAR!  $\Rightarrow$  L  $\Rightarrow$  S  $\Rightarrow$  S is left recursive.

so,  $\Rightarrow$  L  $\Rightarrow$  S is left recursive.

(HIGRAMMAR!  $\Rightarrow$  SICL) is not left recursive.

exp-10 :- Implement a C program to eliminate left factoring from given CFG.

~~S: NO: 33~~

Aim:- To understand the concept of reflection.

To implement c program to eliminate left factors from given CFG.

## Algorithm :-

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

Program :- (a) *calculator* (b) *string*

```

#include <stdio.h> 3213
#include <iostream.h> Aman
#include <string.h> (Soham)
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] != '\n'; i++)
    {
        if (gram[i] == ' ')
            continue;
        else
        {
            part1[j] = gram[i];
            j++;
            if (j == 10)
                break;
        }
    }
    part1[j] = '\0';
    for (j = ++i; i < 20; j++)
        gram[i] = ' ';
    for (j = 0; gram[j] != '\0'; j++)
        cout << gram[j];
}

```

part2[ij] = gram[ij];

S: No: 33

part2[ij] = 'o';

for (i=0; i<strlen(part1) || i<strlen(part2);

{ if (part1[ij] == part2[ij])

{ modified Gram[k] = part1[ij];  
k++;

pos = i+1; // move after first +

for (i=pos, j=0; part1[ij] != 'o'; i++, j++) {

new Gram[ij] = part1[ij];

modified Gram[k] = 'x';

modified Gram[k+1] = 'o';

new Gram[ij] = 'o';

printf("\n s->xs", modified Gram);

printf("\n x->.s\n", new Gram);

}

Output :-

Enter production = s->iets|iesla

s->ietsx

x->iesla

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

Aim :- To implement C program to perform symbol table operations.

To implement C program to perform symbol table operations.

algorithm :-

1. Start the code and import package.

2. Assign the variable.

3. Declare the variable

4. start the Run the code.

program :-

```
#include <stdio.h> // main function
```

```
#include <string.h> // main function
```

```
int cnt=0;
```

```
struct symtab
```

```
{char val[20]; char lab[10]; int add;
```

```
char label [20];
```

```
int addr;
```

```
sy [10];
```

```
void insert(); // insertion fun
```

```
int search (char t); // search fun
```

```
void display(); // display fun
```

```
void modify();
```

```
{
```

```
int ch, val;
```

```
char lab [10];
```

```
do
```

```
{
```

1. Insert 2. display 3. Search 4. modify  
5. exit (0);

Scanf ("Enter", "abcd"); if (a == 0) exit (0);

switch (ch)

{

Case 1:

insert();

break;

case 2 :

display();

break;

case 3 :

printf ("Enter label"); if (c == 3) exit (0);

scanf ("%s", lab); if (lab[0] == 'a') val = 1;

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 [5];

int symbol;

printf ("Enter label");

scanf ("%s", lab);

```

int Search (char ts)
{
    int flag=0; i; for (i=0; i<cnt; i++)
    {
        if (strcmp (sy[i].label, s) == 0)
        {
            flag=1;
            break;
        }
    }
    return flag;
}

void display ()
{
    int i;
    for (i=0; i<cnt; i++)
        printf ("%s %s %s\n", sy[i].label, sy[i].addr);
}

```

Output:-

1. insert

2. display (3, 60; 100) H100

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.

Aim :-

To write C program to Construct the recursive descent for 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 <string.h>

char input[100];
int i, l;
void main()
{
    printf("Recursive parsing : ");
    printf(" E → TE' | E' → f+e @ n → f+e");
    printf(" enter string : ");
    gets(input);
    if (input[i] == 'o')
    {
        if (input[i+1] == 'l')
            printf("String accepted");
        else
            printf("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: nlo = 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 package
2. Declare the variables
3. Assign the variable
4. Start & Run the code .

Program :-

```
#include <stdio.h>
#include <string.h>
void main()
{
    char string[10];
    int flag, count = 0;
    printf("the grammar is → S→as,S→gb,
           S→ab");
    printf("Enter string:");
    gets(string);
    if (string[0] == 'a') {
        flag = 0;
        for (count = 1; string[count] != 'b'; count++)
            if (string[count] == 'g')
```

if (string [count] == 'b') {

    upravida flag = 1;      s.no: 33  
    upravida string[0] to suppl. b

    for i = 0 to m-1 do {      fo cito  
        if (string[i] == 'a') {      fo  
            if (flag == 1) {      fo  
                if (string[i] == 'b') {      fo  
                    flag = 0;      fo  
                }      fo  
            }      fo  
        }      fo  
    }      fo

    Continue;

    {      fo  
        if (flag == 1) {      fo  
            if (string[i] == 'a') {      fo  
                if (string[i+1] == 'b') {      fo  
                    flag = 0;      fo  
                }      fo  
            }      fo  
        }      fo  
    }      fo

    else if (flag == 1) && (string [count] == 'a') {      fo  
        if (string [count] == 'a') {      fo  
            if (string [count+1] == 'b') {      fo  
                flag = 0;      fo  
            }      fo  
        }      fo  
    }      fo

    }      fo  
    printf ("the string is ");      fo  
    break;      fo  
}      fo

else {      fo  
    printf ("string accepted.\n");      fo  
}      fo

}      fo  
}      fo

Output:-

the grammar is :  $S \rightarrow aS, S \rightarrow bS, S \rightarrow ab$

enter the string : abb

String accepted.

Exp-IV :- Implement the concept of shift-reduce parsing in C programming.

SIN 01 23

Aim:-

To implement the shift-reduce parsing using C programming.

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

char fp_sym[15], stack[15]; int ip_ptr=0, st_ptr;
char temp[2], temp2[2];
char act[15];
void check(); int main()
{
    printf("Shift Reduce parser");
    printf("Grammar:");
    printf("E->E+E");
    printf("E->e");
    printf("enter the input symbol");
    get(fp_sym);
    printf("stack table");
    printf("stack");
}
```

temp[0] = ip\_sym(ip\_ptr); temp[0] = '0'

strcat(act, temp); S: No 33

{ stack[st\_ptr] = ip\_sym(ip\_ptr),

stack[st\_ptr] = '0'; ip\_sym(ip\_ptr) =>

printf("%d,%s,%c",

temp[0], stack[st\_ptr], ip\_ptr);

st\_ptr++;

}

st\_ptr++; check();

}

void check()

int flag = 0; temp2[0] = stack[st\_ptr];

if (!strcmp(temp2, "a")) || (!strcmp(temp2, "A"))

{

Stack[st\_ptr] = '1'; if (!strcmp(temp, "a"))

printf("a is all %c", stack, ip\_sym)

flag = 1;

}

if (C != strcmp(stack, "E")) || (!strcmp

(stack, E))

{

strcpy(stack, "E"); flag = 1;

printf("E is all %c", stack, ip\_sym)

else

printf("E is not all %c", stack, ip\_sym)

if (Flag == 0) /\* flag == 0 means no match \*/ S: No 2-33

{ printf("rejecting because no match with  
exit(0);  
}

return;

}

Output:-

shift Reduce parse stack to DFA.

GRAMMER

$E \rightarrow E + E$

$E \rightarrow E / E$

$E \rightarrow E * E$

$E \rightarrow a \cup b$

input: a+b

stack

\$

#a

input

a+b\$

#b\$

left side of stack

left part of action stack

(#a) # b \$

shift

(#a) # b \$ : E (0) a+b (\$ 1) b \$ : b b a # → a word

(#a) # b \$ : E (0) a+b (\$ 1) b \$ : shift

\$ E + b

\$

shift

\$ E + E

\$ [P] (S)

E → b

\$ E

\$

E → E + E

\$ E

\$

ACCEPT

(0) Action

Exp-15:- write a c program to implement the operator precedence parsing.

$$3 \times 10 = 33$$

Aim:-

To implement the operator precedence parsing using c program.

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 <iostream.h>
char *input;
int i=0;
char lastHandle[6], stack[50]; handle()
=>{if(c=='+', precedence=1); if(c=='-',
int top=0,i;
char prec[a][9]={}
int getIndex(char c)
{
switch(c)
{
case '+':return 0;
case '-':return 1;
case '*':return 2;
case '/':return 3;
case '^':return 4;
case ')':return 5;
case '(':return 6;
case '=':return 7;
case ',':return 8;
}
```

case 'A': return 2;

case 'B': return 3;

case 'C': return 4;

case 'D': return 5;

case 'E': return 6;

case 'F': return 7;

case 'G': return 8;

case 'H': return 9;

3 int shift (char c)

{ stack [top] = \* (input + it);

stack [top+1] = '0';

3 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;

3 if (found == 1)

{ stack [top+1] = 'E';

return 1;

```

    top = top - 1;
    stack[i] = '$';
    return 1;
}
else {
    printf("Accepted");
}
}

void main()
{
    int f;
    char *input = (char *) malloc(50 * size(char));
    printf("Enter string");
    scanf("%s", input);
    input = strcat(input, '$');
    if (strlen(input) == 1)
        printf("Accepted");
    else
        printf("Not Accepted");
}

```

Output:-

Stack	Input	Action
\$	i	+ (i+1) \$ shift
\$ \$	i	+ (i+1) \$ shift
\$ \$ +	i	+ (i+1) \$ shift
\$ \$ + (i+1)	i	R E\$ i
\$ \$ + (i+1) R		\$ shift
\$ \$		R E\$ i
\$		shift

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

Aim :-

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

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 <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);
        s[len].temp = d1;
        if(s[len].data[0] == '0')
            s[len].temp[0] = '1';
        else
            s[len].temp[0] = '0';
        fprintf(f2, "%s", s[len].temp);
        j++;
    }
}
```

```

if (!strcmp(s[3].data, "4"))
{
    printf("not correct answer\n");
    s[3].data = "33";
}

printf("%s = %s + %s", s[j].temp,
       s[j+2].data);
j++;
}

else if (!strcmp(s[3].data, "2"))
{
    printf("%s = %s - %s", s[j].temp,
           s[j+1].data);
    for (i=4; i<len; i+=2)
    {
        itoa(c[i], d1, 7);
        strcat(d2, d1);
        strcat(s[0].temp, d2);
        if (!strcmp(s[i+1].data))
            printf("%s = %s + %s", s[j].temp);
        else
            strcpy(d1, s[i+1].data);
        strcpy(d2, s[i+1].data);
        getch();
    }
}

```

Output :-

Input : int t1, t2, in2, in3

$$t1 = int1 + in2$$

$$t2 = t1 - in3$$

$$out = t2 / 11.$$

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 package.
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;
    Scan f [e.g. [1~]"], &str);
    for (int i=0; str[i] != '\0'; i++)
    {
        if (str[i] == ' ')
        {
            words++;
        }
        else if (str[i] == '\n')
        {
            newline++;
        }
    }
}
```

if (ch >= 'A' & ch <= 'Z') or (ch >= 'a' & ch <= 'z') {  
 else if (str[i] != ' ' & str[i] != '\n') {

character++; total++; s: No = 33

else if (str[i] == ' ') {

if (character > 0) total++;

character = 0; word++;

if (word > 1) total++;

if (str[i] == '\n') character = 0;

newline++;

printf("Total words = %d\n", words);

printf("Total lines = %d\n", lines);

printf("Total character = %d\n", character);

return 0;

Output :-

```
void main()
{
    int a, b, c;
    a = 100;
    b = 200;
    c = a + b;
    printf("Value of a = %d\n", a);
    printf("Value of b = %d\n", b);
    printf("Value of c = %d\n", c);
}
```

Total number of words = 18

(Total lines = 5 Total words = 18)

(Total characters = 100 + 200 + 18 = 318)

(Total characters = 100 + 200 + 18 = 318)

(Total characters = 100 + 200 + 18 = 318)

(Total characters = 100 + 200 + 18 = 318)

(Total characters = 100 + 200 + 18 = 318)

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 char address code-r,d,%h1");
        for (j=0; j<6; j++)
        {
            scanf ("%c", &a[i][j]);
        }
    }
    printf ("The generated is:");
}
```

```

3/37-133-134997 at (array) 2 of 18: No:33
for (i=0; i<n; i++)
{
    printf (" mov r.c, R+I.d", a[i][s], i);
    if (a[i][u]=='-')
        printf (" sub r.c, R+I.d", a[i][s], i);
    if (a[i][u]=='+')
        printf (" add r.c, R+I.d", a[i][s], i);
    if (a[i][u]=='*')
        printf (" multi r.c, R+I.d", a[i][s], i);
    printf (" move r.d, -r.c, i, a[i][t]);
    printf ("\n");
}
return 0;

```

Output :-

Enter no: intermediate code : 2

Enter 3 address : 1: a+b+c;

(Enterprise's address 1 : d=notd, Enterprise's address 2 : d=notd, ) 71089

generated code:

Mov b, R0 Chiamata (call) rot.

Bridg C, RD

mov - PaB

Mov , 20,9

MOV R1,R1

MOV D, R

MOV R1,d

EXP-19: Write a program to compute LEADING operator precedence parser for given grammar.

## Aim :-

To write c program to Computer Combinatorics  
operator precedence parser for grammar :-

## Algorithm:-

1. start the code and import package.
  2. declare the variable.
  3. assign the variable.
  4. start & run the code.

### program : -

int r; s: nlo = 33  
 for (i=0; i < 18; i++) {  
 if (arr[i][0] == pre && arr[i][1] == re) {  
 arr[i][2] = 'T'; // mark it as visited  
 break; // no need to search remaining nodes  
 ++top; // push to stack  
 stack[top][0] = proj; // save old cell from  
 stack[top][1] = res; // save new cell  
 stack[top][2] = 'L'; // destroy old cell  
 } // end of if block  
 int main () {  
 int i=0, j; // init all cells to blank  
 char pre, re, pri='L'; // initial projectile  
 for (i=0; i < 6; i++) {  
 for (j=0; j < 3 && res[i][j] != 'L'; j++) {  
 if (res[i][j] == 'L' || res[i][j] == '\*') {  
 res[i][j] = 'C'; // mark as visited  
 install (prod[i], res, i, j); // install projectile  
 break; // no need to search further  
 } // end of inner loop  
 } // end of outer loop  
 for (i=0; i < 8; i++) {  
 printf ("%c", arr[i][0]); // print row  
 for (j=0; j < 3; j++) {  
 printf ("-%c", arr[i][j]); // print cell  
 } // end of inner loop  
 getch(); // wait for key press

printf("%d\n");

S: N0 = 33

for(i=0; i<18; i++)

{ if (pr != arr[i][0])

{ pri = arr[i][0];

printf("%c", pri);

}

getch();

Output:-

output:-

E + T

E \* T

E C T

E > F

E i T

{ E \$ + F } (i) = E [ ] (i) is valid

{ E C F } + E F (i) is valid

E i F + E F (i) is valid

{ E + F } (i) is valid

{ E \* F } (i) is valid

T \* T

T C T

{ T C T } (i) is valid

{ T \$ F } (i) is valid

E → + \* (i)

F → (i)

T → + C (i)

EXP - 20 :- write c program to computer

TRAILING() - operator parser to given s

Aim :-

To write c program to computer TRAILING

- Operator parser for grammar & tokens

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 <conio.h>
char arr[18][3] = {{'t', '+', 'F'}, { // T + F
    {'t', '*', 'F'}, { // T * F
    {'t', '^', 'F'}, { // T ^ F
    {'t', '(', 'F'}, { // T ( F
    {'t', ')', 'F'}, { // T ) F
    {'t', '+', 'T'}, { // T + T
    {'t', '*', 'T'}, { // T * T
    {'t', '^', 'T'}, { // T ^ T
    {'t', '(', 'T'}, { // T ( T
    {'t', ')', 'T'}, { // T ) T
    {'t', '+', 'F'}, { // F + F
    {'t', '*', 'F'}, { // F * F
    {'t', '^', 'F'}, { // F ^ F
    {'t', '(', 'F'}, { // F ( F
    {'t', ')', 'F'} // F ) F
}}}}
```

Char prod [6] = "FETTFFF"

```
char res [6][5] = {{'t', '+', 'F', 'F', 'F'}, // T + F F F
    {'t', '*', 'F', 'F', 'F'}, // T * F F F
    {'t', '^', 'F', 'F', 'F'}, // T ^ F F F
    {'t', '(', 'F', 'F', 'F'}, // T ( F F F
    {'t', ')', 'F', 'F', 'F'}, // T ) F F F
    {'t', '+', 'T', 'F', 'F'}} // T + T F F
```

char stack[5][2]; S.N: 33

int top = -1; || top = -1 [L1C1] & ||

void install (char pro, char re) {

int i;

for (i=0; i<18; i++)

{ if arr[i][0] == pro && arr[i][1] == re)

{

([0][1]) res = i + 1;

}

([0][1]) res = i + 1;

top++;

arr[2] = (res, i + 1); // res = 1, i + 1 = 2

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') then res[i][j] = 'F'

If res[i][j] == '+' || res[i][j] == '-'

break; // or next operation

sinh=33

for (i=0; i<18; i++)  
{ arr[i][0] = arr[i][18]; arr[i][18] = arr[i][0]; }

if (pri != arr[i][0])

pri = arr[i][0];

printf("\n't c->', pri);

if (arr[i][2] == '+')

printf(" -r.c ", arr[i][1]);

Output :-

E + F

E + F

E < F

E > F

E i F

F \$ F

if F(i>F) & i>F == F(F) & i>F

F & F

F & F

F > F

F & F

+ & F

+ & F