GD GOENKA UNIVERSITY

Sohna road, Gurugram, Haryana

COMPILER DESIGN

CSE4001L

SCHOOL OF ENGINEERING AND SCIENCES

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

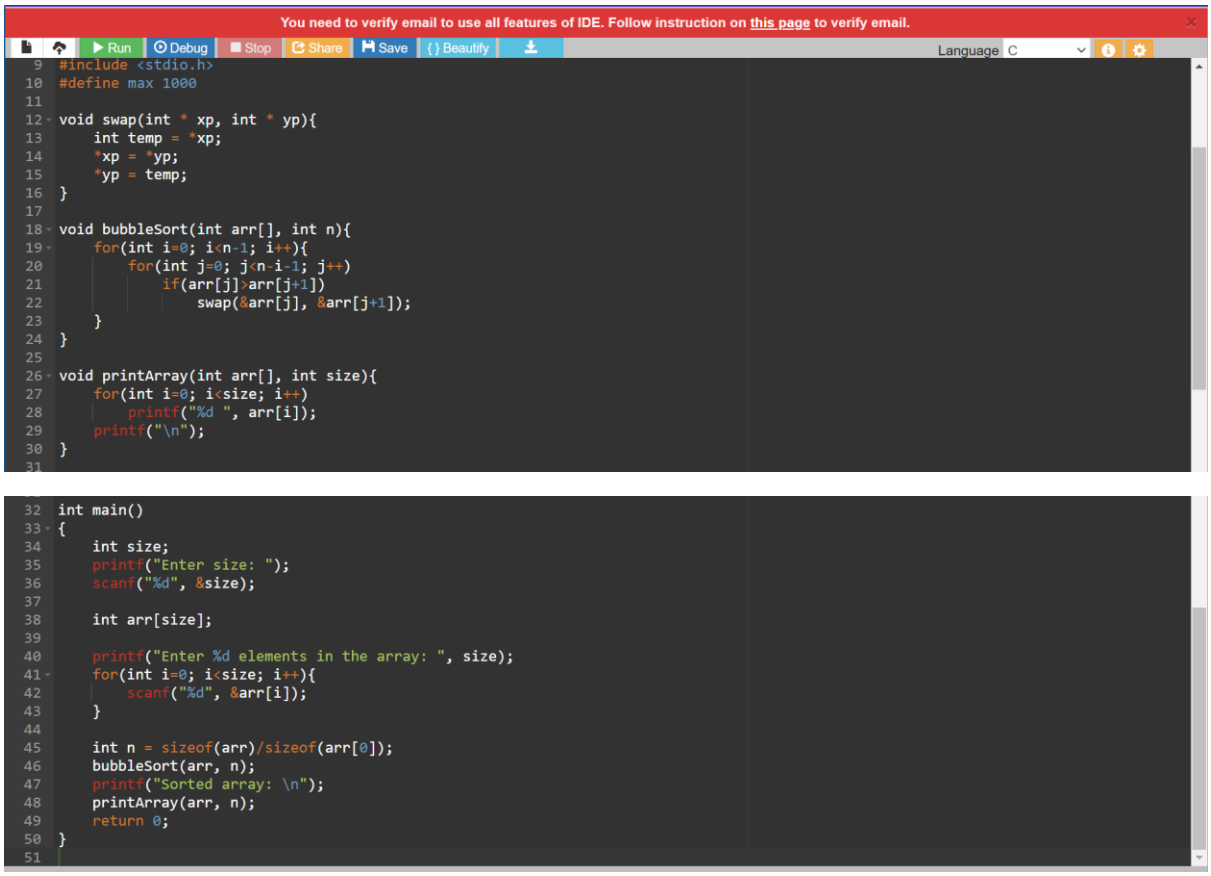
Submitted By	
Student Name	Ayan Chowdhury
Enrolment No.	190020203008
Section/Group	T1
Department	Computer Science and Engineering
Session/Semester	2021-22/ Odd Semester
Submitted To	
Faculty Name	Alina Banerjee

TABLE OF CONTENT

Experiment:1

Aim: Write a program in C for Bubble Sort.

Code:



```c
#include <stdio.h>
#define max 1000

void swap(int * xp, int * yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void bubbleSort(int arr[], int n){
    for(int i=0; i<n-1; i++){
        for(int j=0; j<n-i-1; j++)
            if(arr[j]>arr[j+1])
                swap(&arr[j], &arr[j+1]);
    }
}

void printArray(int arr[], int size){
    for(int i=0; i<size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main()
{
    int size;
    printf("Enter size: ");
    scanf("%d", &size);

    int arr[size];

    printf("Enter %d elements in the array: ", size);
    for(int i=0; i<size; i++){
        scanf("%d", &arr[i]);
    }

    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Enter size: 5
Enter 5 elements in the array:
22
53
11
87
2
Sorted array:
2 11 22 53 87


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment:2

Aim : Write a program in C for Matrix Multiplication.

Code:

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

int main(){
    int a[10][10], b[10][10], pro[10][10], r, c;
    printf("Enter number of rows: ");
    scanf("%d", &r);
    printf("Enter number of columns: ");
    scanf("%d", &c);
    printf("Enter elements of first matrix: \n");
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            scanf("%d", &a[i][j]);
        }
    }

    printf("Enter elements of second matrix: \n");
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            scanf("%d", &b[i][j]);
        }
    }

    printf("Product of the matrices: \n");
    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            pro[i][j]=0;
            for(int k=0; k<c; k++){
                pro[i][j]+=a[i][k]*b[k][j];
            }
        }
    }

    for(int i=0; i<r; i++){
        for(int j=0; j<c; j++){
            printf("%d\t", pro[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

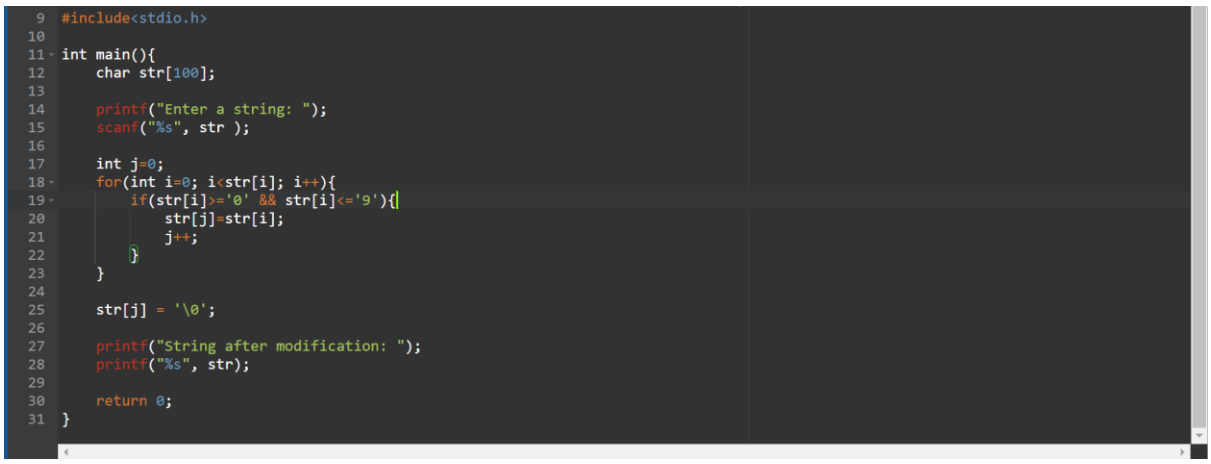
Output:

```
Enter number of rows: 3
Enter number of columns: 1
Enter elements of first matrix:
2
67
21
Enter elements of second matrix:
9
4
20
Product of the matrices:
18
603
189


...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment: 3

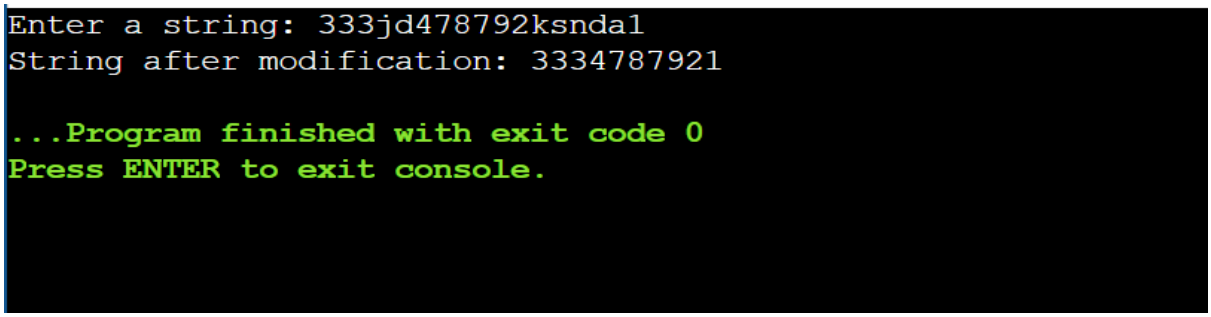
Aim: Write a C program to remove all characters from string keeping all numbers.

Code:



```c
#include<stdio.h>

int main(){
    char str[100];

    printf("Enter a string: ");
    scanf("%s", str );

    int j=0;
    for(int i=0; i<str[i]; i++){
        if(str[i]>='0' && str[i]<='9'){
            str[j]=str[i];
            j++;
        }
    }

    str[j] = '\0';

    printf("String after modification: ");
    printf("%s", str);

    return 0;
}
```

Output:



```
Enter a string: 333jd478792ksnda1
String after modification: 3334787921

...Program finished with exit code 0
Press ENTER to exit console.
```

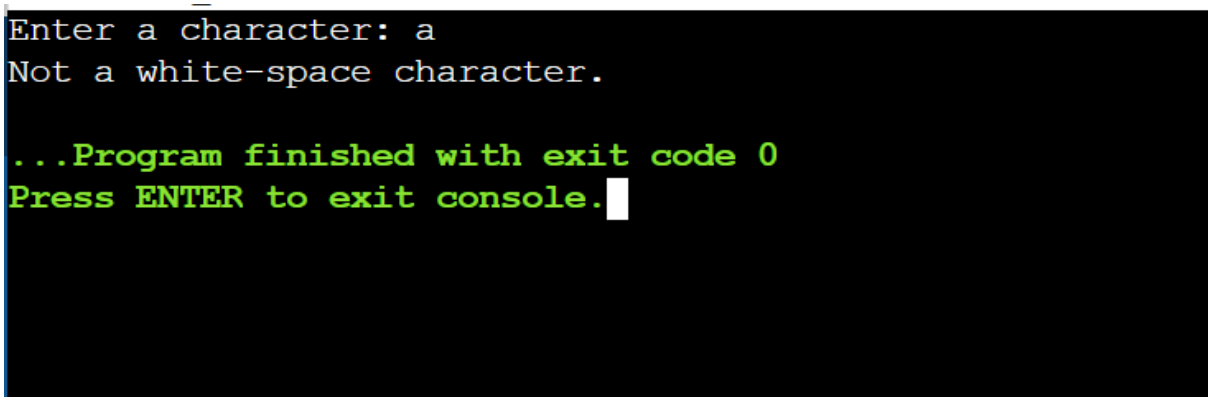
Experiment: 4

Aim: Write a C program to check if a character is white-space or not.

Code:

```c
#include <stdio.h>
#include <ctype.h>
int main()
{
    char c;
    int result;

    printf("Enter a character: ");
    scanf("%c", &c);

    result = isspace(c);

    if (result == 0)
    {
        printf("Not a white-space character.");
    }
    else
    {
        printf("White-space character.");
    }

    return 0;
}
```

Output:



```
Enter a character: a
Not a white-space character.

...Program finished with exit code 0
Press ENTER to exit console.
```

Experiment: 5

a) **Aim: Write a Python program to test whether a given identifier is valid or not in a Python program.**

Code:

```
# Function that returns true if str1
# is a valid identifier
def isValid(str1, n):
# If first character is invalid
    if (((ord(str1[0]) >= ord('a') and
        ord(str1[0]) <= ord('z')) or
        (ord(str1[0]) >= ord('A') and
        ord(str1[0]) <= ord('Z')) or
        ord(str1[0]) == ord('_')) == False):
        return False
# Traverse the for the rest of the characters
    for i in range(1, len(str1)):
        if (((ord(str1[i]) >= ord('a') and
            ord(str1[i]) <= ord('z')) or
            (ord(str1[i]) >= ord('A') and
            ord(str1[i]) <= ord('Z')) or
            (ord(str1[i]) >= ord('0') and
            ord(str1[i]) <= ord('9')) or
            ord(str1[i]) == ord('_')) == False):
            return False
# is a valid identifier
    return True
# Driver code
str1 = "_Hello123"
n = len(str1)
if (isValid(str1, n)):
    print("Valid")
else:
    print("Invalid")

```

Output:

```
Valid
>
```

b) Aim: Write a C program to test whether a given keyword is valid or not in a C program.

Code:

```c
#include <stdio.h>
#include <string.h>
int main() {
    char keyword[32][10]={
        "auto","double","int","struct","break","else","long",
        "switch","case","enum","register","typedef","char",
        "extern","return","union","const","float","short",
        "unsigned","continue","for","signed","void","default",
        "goto","sizeof","voltile","do","if","static","while"
    } ;
    char str[]="which";
    int flag=0,i;
        for(i = 0; i < 32; i++) {
            if(strcmp(str,keyword[i])==0) {
            flag=1;
        }
    }
    if(flag==1)
        printf("%s is a keyword",str);
    else
        printf("%s is not a keyword",str);
}

```

Output:

```
which is not a keyword
```

Experiment: 6

a) **Aim: Construct a DFA which accepts set of all strings over ∑ = {a, b} of length 2.**

Code:

```
def checkStateA(n):
    if(len(n)==1):
        print('String not accepted')
    else:
        if(n[0]=='a' or n[0]=='b'):
            stateB(n[1:]);

def stateB(n):
    if(len(n)!=1):
        print('String not accepted')
    else:
        stateC(n[1:])

def stateC(n):
    if(len(n)==0):
        print('String accepted')
    else:
        print('String not accepted')

n=input()
checkStateA(n);
```

Output:

```
bb
String accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

b) Aim: Construct a DFA to accept a string containing a zero followed by a one.

Code:

```
wrd = "011"

if "01" in wrd:
    print("Accepted")
else:
    print("Not Accepted")
```

Output:

```
input
Accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

c) Aim: Construct a DFA to accept a string containing two consecutive zero's followed by two consecutive ones.

Code:

main.py

```
wrd = "111111010"

if "0011" in wrd:
    print("Accepted")
else:
    print("Not Accepted")
```

Output:

```
input
Not Accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

d) Aim: Construct a DFA to accept a string containing even number of zeros and any number of ones.

Code:

main.py

```
count = 0

wrd = "001011111"

for letter in wrd:
    if letter == '0':
        count += 1

if count % 2 == 0:
    print("Accepted")
else:
    print("Not Accepted")
```

Output:

```
input
Not Accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

e) Aim: Construct a DFA to accept all strings which do not contain three consecutive zeros.

Code:

main.py

```
import re


wrd = "10111000"
matches = re.findall("0+", wrd)

zeroes = map(lambda x: len(x), matches)

if max(zeroes) == 3:
    print("Not Accepted")
else:
    print("Accepted")
```

Output:

input

```
Not Accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

f) Aim: Construct a DFA to accept all strings containing even number of zeros and even number of ones.

Code:

main.py

```
wrd = "00110011"

count_zero = 0
count_one = 0

for letter in wrd:
    if letter == '0':
        count_zero += 1
    if letter == '1':
        count_one += 1

if count_one % 2 == 0 and count_zero % 2 == 0:
    print("Accepted")
else:
    print("Not Accepted")
```

Output:

g) Aim: Construct the smallest DFA which will accept all strings over ∑ = {a, b}.

Code:

main.py

```
1  myList = ['a', 'b']
2
3  wrd = 'ababb'
4  count = 0
5
6  for letter in wrd:
7      if myList.__contains__(letter):
8          count += 1
9
10
11 if count == len(wrd):
12     print("Accepted")
13 else:
14     print("Not Accepted")
```

Output:

h) Aim: Construct a DFA with ∑ = {0, 1} which accepts the strings which start with 1 and ends with 0.

Code:

```
main.py
1  wrd = "10010010"
2
3  if wrd[0] == '1' and wrd[len(wrd) - 1] == '0':
4      print("Accepted")
5  else:
6      print("Not Accepted")
```

Output:

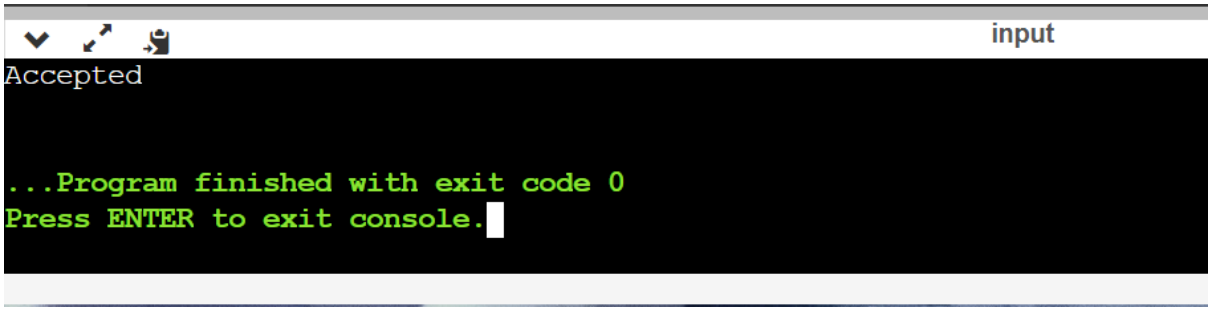
```
input
Accepted


...Program finished with exit code 0
Press ENTER to exit console.
```

i) Aim: Construct a DFA with ∑ = {0, 1} which accepts only input 101.

Code:

```
main.py
1  wrd = "101"
2
3  if wrd == "101":
4      print("Accepted")
5  else:
6      print("Not Accepted")
```

Output:



j) Aim: Construct a DFA with ∑ = {a, b} which accepts all strings of length greater than equal to 2 and less than equal to 2.

Code:

Output:

Experiment: 7

a) Aim: Write a C Program to implement a SR parser.

Code:

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

//Global Variables
int z = 0, i = 0, j = 0, c = 0;


// Modify array size to increase
// length of string to be parsed
char a[16], ac[20], stk[15], act[10];


// This Function will check whether
// the stack contain a production rule
// which is to be Reduce.
// Rules can be E->2E2 , E->3E3 , E->4
void check()
{
    // Copying string to be printed as action
    strcpy(ac,"REDUCE TO E -> ");

    // c=length of input string
    for(z = 0; z < c; z++)
    {
    //checking for producing rule E->4
        if(stk[z] == '4')
        {
            printf("%s4", ac);
            stk[z] = 'E';
            stk[z + 1] = '\0';
            //printing action
            printf("\n$%s\t%s$\t", stk, a);
        }
```

```
35      }
36
37
38      for(z = 0; z < c - 2; z++)
39      {
40          //checking for another production
41          if(stk[z] == '2' && stk[z + 1] == 'E' && stk[z + 2]
                == '2')
42          {
43              printf("%s2E2", ac);
44              stk[z] = 'E';
45              stk[z + 1] = '\0';
46              stk[z + 2] = '\0'; printf("\n$%s\t%s$\t", stk, a
                    ); i = i - 2;
47          }
48
49
```

```
50      }
51
52
53      for(z=0; z<c-2; z++)
54      {
55          //checking for E->3E3
56          if(stk[z] == '3' && stk[z + 1] == 'E' && stk[z + 2]
                == '3')
57          {
58              printf("%s3E3", ac);
59              stk[z]='E';
60              stk[z + 1]='\0';
61              stk[z + 1]='\0';
62              printf("\n$%s\t%s$\t", stk, a); i = i - 2;
63          }
64      }
65      return ; //return to main
```

```
66  }
67
68  //Driver Function
69  int main()
70  {
71      printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
72
73
74      // a is input string
75      strcpy(a,"32423");
76
77      // strlen(a) will return the length of a to c
78      c=strlen(a);
79
80
81      // "SHIFT" is copied to act to be printed
82      strcpy(act,"SHIFT");
83
84      // This will print Labels (column name)
85      printf("\nstack \t input \t action");
86
87      // This will print the initial
88      // values of stack and input
89      printf("\n$\t%s$\t", a);
90
91      // This will Run upto length of input string
92      for(i = 0; j < c; i++, j++)
93      {
94          // Printing action
95          printf("%s", act);
96
97          // Pushing into stack
98          stk[i] = a[j];
99          stk[i + 1] = '\0';
```

```
100         // Moving the pointer
101         a[j]=' ';
102
103         // Printing action
104         printf("\n$%s\t%s$\t", stk, a);
105
106         // Call check function ..which will
107         // check the stack whether its contain
108         // any production or not check();
109     }
110
111
112     // Rechecking last time if contain
113     // any valid production then it will
114     // replace otherwise invalid check();
115
116     // if top of the stack is E(starting symbol)
117     // then it will accept the input
118     if(stk[0] == 'E' && stk[1] == '\0')
119         printf("Accept\n");
120     else //else reject
121         printf("Reject\n");
122 }
123
```

Output:

```
▲ ~ # gcc sr_parser.c
▲ ~ # ./a.out
GRAMMAR is -
E->2E2
E->3E3
E->4

stack    input    action
$       32423$  SHIFT
$3       2423$  SHIFT
$32       423$  SHIFT
$324       23$  REDUCE TO E -> 4
$32E       23$  SHIFT
$32E2       3$  REDUCE TO E -> 2E2
$3E         3$  SHIFT
$3E3         $  REDUCE TO E -> 3E3
$E           $  Accept
▲ ~ #
```

b) Aim: Write a C program to implement Operator precedence parser.

Code:

```c
#include<stdio.h>
#include<string.h>
#include<cstdlib>

char *input;
int i=0;
char lasthandle[6],stack[50],handles[][5]={")E(","E*E","E+E"
    ,"i","E^E"};
//(E) becomes )E( when pushed to stack

int top=0,l;
char prec[9][9]={

                    // /input/

        /*stack    +    -    *    /    ^    i    (    )    $  */

        /*  + */  '>', '>','<','<','<','<','<','>','>',

        /*  - */  '>', '>','<','<','<','<','<','>','>',

        /*  * */  '>', '>','>','>','<','<','<','>','>',

        /*  / */  '>', '>','>','>','<','<','<','>','>',

        /*  ^ */  '>', '>','>','>','<','<','<','>','>',

        /*  i */  '>', '>','>','>','>','e','e','>','>',

        /*  ( */  '<', '<','<','<','<','<','<','>','e',

        /*  ) */  '>', '>','>','>','>','e','e','>','>',

        /*  $ */  '<', '<','<','<','<','<','<','<','>',

            };

```

```
int getindex(char c)
{
switch(c)
    {
    case '+':return 0;
    case '-':return 1;
    case '*':return 2;
    case '/':return 3;
    case '^':return 4;
    case 'i':return 5;
    case '(':return 6;
    case ')':return 7;
    case '$':return 8;
    }
    return 0;
}


int shift()
{
stack[++top]=*(input+i++);
stack[top+1]='\0';
return 0;
}


int reduce()
{
int i,len,found,t;
for(i=0;i<5;i++)//selecting handles
    {
    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-t+1]='E';
                top=top-t+1;
                strcpy(lasthandle,handles[i]);
                stack[top+1]='\0';
                return 1;//successful reduction
                }
            }
    }
return 0;
}


void dispstack()
{
int j;
for(j=0;j<=top;j++)
    printf("%c",stack[j]);
}


void dispinput()
{
int j;
for(j=i;j<l;j++)
    printf("%c",*(input+j));
}


int main()
{
int j;

```

```
117  input=(char*)malloc(50*sizeof(char));
118  printf("\nEnter the string\n");
119  scanf("%s",input);
120  input=strcat(input,"$");
121  l=strlen(input);
122  strcpy(stack,"$");
123  printf("\nSTACK\tINPUT\tACTION");
124  while(i<=l)
125      {
126      shift();
127      printf("\n");
128      dispstack();
129      printf("\t");
130      dispinput();
131      printf("\tShift");
132      if(prec[getindex(stack[top])][getindex(input[i])]=='>')
133          {
134          while(reduce())
135              {
136              printf("\n");
137              dispstack();
138              printf("\t");
139              dispinput();
140              printf("\tReduced: E->%s",lasthandle);
141              }
142          }
143      }
144
145  if(strcmp(stack,"$E$")==0)
146      printf("\nAccepted;");
147  else
148      printf("\nNot Accepted;");
149
150  }
```

Output:

```
Enter the string
i*<i+i>*i
STACK   INPUT   ACTION
$i  *<i+i>*i$   Shift
$E  *<i+i>*i$   Reduced: E->i
$E* <i+i>*i$    Shift
$E*<    i+i>*i$ Shift
$E*<i   +i>*i$  Shift
$E*<E   +i>*i$  Reduced: E->i
$E*<E+  i>*i$   Shift
$E*<E+i >*i$    Shift
$E*<E+E >*i$    Reduced: E->i
$E*<E   >*i$    Reduced: E->E+E
$E*<E>  *i$ Shift
$E*<E>* i$  Shift
$E*<E>*i    $   Shift
$E*<E>*E    $   Reduced: E->i
$E*<E>*E$       Shift
$E*<E>*E$   Shift
Not Accepted;
```

Experiment:8

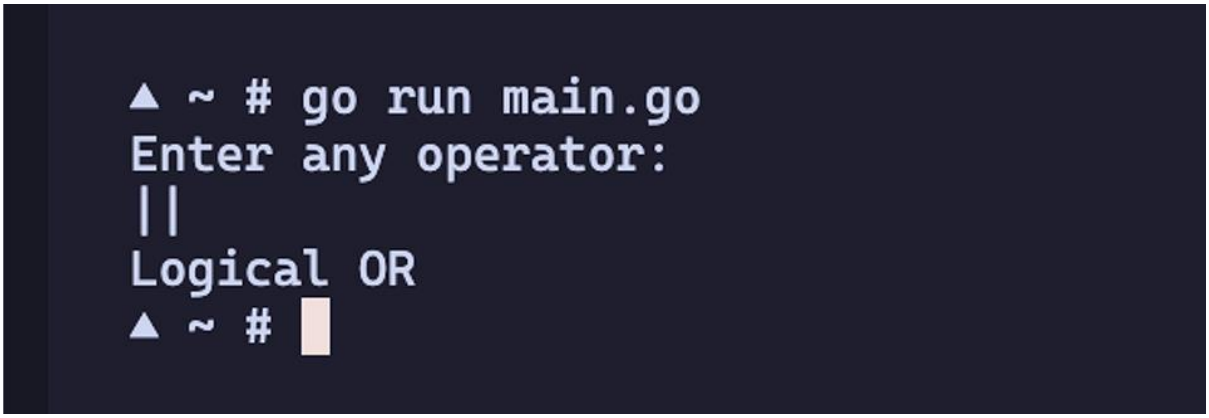
Aim: Write a program to simulate lexical analyser for validating operators.

Code:

```
package main

import (
    "fmt"
)

func main() {
    var operator string
    fmt.Printf("Enter any operator: \n")
    fmt.Scanln(&operator)

    switch {
    case operator == ">=":
        fmt.Printf("Greater than equal")

    case operator == "<=":
        fmt.Printf("Less than equal")

    case operator == ">":
        fmt.Printf("Greater")

    case operator == "<":
        fmt.Printf("Less \n")

    case operator == "=":
        fmt.Printf("Equal \n")

    case operator == "!=":
        fmt.Printf("Not Equal \n")
```

```
    case operator == "&&":
        fmt.Printf("Logical AND \n")

    case operator == "||":
        fmt.Printf("Logical OR \n")

    case operator == "&":
        fmt.Printf("Bitwise AND \n")

    case operator == "|":
        fmt.Printf("Bitwise OR \n")

    case operator == "+":
        fmt.Printf("Addition \n")

    case operator == "-":
        fmt.Printf("Subtraction \n")

    case operator == "/":
        fmt.Printf("Division \n")

    case operator == "*":
        fmt.Printf("Multiplication \n")

    case operator == "%":
        fmt.Printf("Modulus \n")

    default:
        fmt.Printf("Not an operator! \n")
    }

}
```

Output:



Experiment:9

Aim: Write a C program to implement Follow of function.

Code:

```c
#include<stdio.h>
#include<string.h>
#include <cctype>
#include <iostream>
#include <cstring>
int n,m=0,p,i=0,j=0;
char a[10][10],followResult[10];
void follow(char c);
void first(char c);
void addToResult(char);
int main()
{
    int i;
    int choice;
    char c,ch;
    printf("Enter the no.of productions: ");
    scanf("%d", &n);
    printf(" Enter %d productions\nProduction with multiple
        terms should be give as separate productions \n", n);
    for(i=0;i<n;i++)
        scanf("%s%c",a[i],&ch);
        // gets(a[i]);
    do
    {
        m=0;
        printf("Find FOLLOW of -->");
        scanf(" %c",&c);
        follow(c);
        printf("FOLLOW(%c) = { ",c);
        for(i=0;i<m;i++)
            printf("%c ",followResult[i]);
        printf(" }\n");
        printf("Do you want to continue(Press 1 to continue.
            )?");
```

```
        scanf("%d%c",&choice,&ch);
    }
    while(choice==1);
}
void follow(char c)
{
    if(a[0][0]==c)addToResult('$'); for(i=0;i<n;i++)
    {
        for(j=2;j<strlen(a[i]);j++)
        {
            if(a[i][j]==c)
            {
                if(a[i][j+1]!='\0') first(a[i][j+1]);
                if(a[i][j+1]=='\0'&&c!=a[i][0])
                    follow(a[i][0]);
            }
        }
    }
}
void first(char c)
{

    int k;

    if(!(isupper(c)))
        //f[m++]=c;
        addToResult(c);

    for(k=0;k<n;k++)
    {
        if(a[k][0]==c)
        {
            if(a[k][2]=='$') follow(a[i][0]);
            else if(islower(a[k][2]))
```

```
                //f[m++]=a[k][2];
                addToResult(a[k][2]);
            else first(a[k][2]);
        }
    }
}
void addToResult(char c)
{
    int i;
    for( i=0;i<=m;i++)
    if(followResult[i]==c) return;
    followResult[m++]=c;
}
```

Output:

```
Enter the no.of productions: 6
Enter 6 productions
Production with multiple terms should be give as separate
    productions
S=aBDh
b=cC
C=bC
D=EF
E=g
F=f
Find FOLLOW of -->D
FOLLOW(D) = { h  }
Do you want to continue(Press 1 to continue.    )?1
Find FOLLOW of -->C
FOLLOW(C) = { b  }
Do you want to continue(Press 1 to continue.    )?1
Find FOLLOW of -->S
FOLLOW(S) = { $  }
```

Experiment:10

Aim: Write a program to implement LR(0) parser

Code:

```
import os
from collections import Counter
import pyfiglet
import termtables as tt


def append_dot(a):
    jj = a.replace("->", "->.")
    return jj


def compress_name(name: str):
    res = Counter(name)
    comp = ''
    for r in res:
        comp += r + str(res[r])

    return comp


def save_file(final_string, grammar, name):
    directory = os.path.dirname("parsable_strings/" + str(grammar) + "/")
    if not os.path.exists(directory):
        os.makedirs(directory)

    with open("parsable_strings/{0}/{1}.txt".format(grammar, name), 'w') as f:
        f.write(final_string)
```

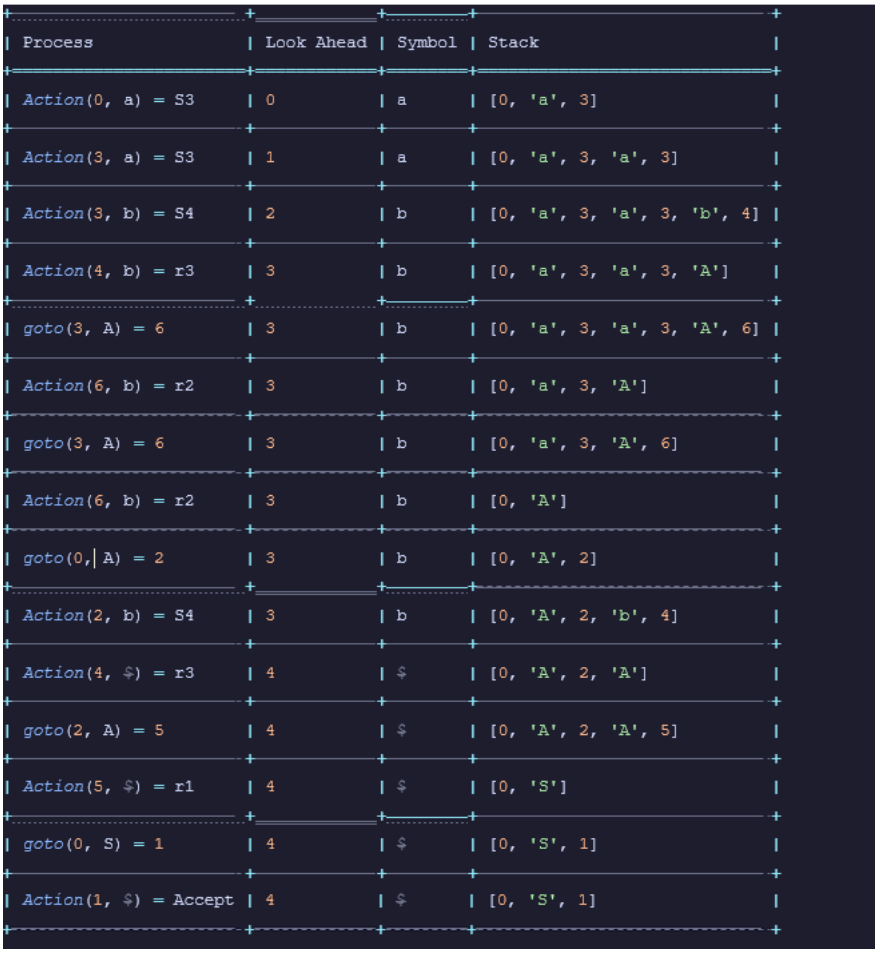
```
def closure(a):
    temp = [a]
    for it in temp:
```

```
        jj = it[it.index(".") + 1]
        if jj != len(it) - 1:
            for k in prod:
                if k[0][0] == jj and (append_dot(k)) not in temp:
                    temp.append(append_dot(k))
        else:
            for k in prod:
                if k[0][0] == jj and it not in temp:
                    temp.append(it)

    return temp


def swap(new, pos):
    new = list(new)
    temp = new[pos]
    if pos != len(new):
        new[pos] = new[pos + 1]
        new[pos + 1] = temp
        new1 = "".join(new)
        return new1
    else:
        return "".join(new)


def goto1(x1):
    hh = []
    pos = x1.index(".")
    if pos != len(x1) - 1:
        jj = list(x1)
        kk = swap(jj, pos)
        if kk.index(".") != len(kk) - 1:
            jjj = closure(kk)
            return jjj
        else:
            hh.append(kk)
            return hh
    else:
        return x1


def get_terminals(gram):
```

```
    terms = set()
    for p in gram:
        x1 = p.split('->')
        for t in x1[1].strip():
            if not t.isupper() and t != '.' and t != '':
                terms.add(t)

    terms.add('$')

    return terms


def get_non_terminals(gram):
    terms = set()
    for p in gram:
        x1 = p.split('->')
        for t in x1[1].strip():
            if t.isupper():
                terms.add(t)

    return terms


def get_list(graph, state):
    final = []
    for g in graph:
        if int(g.split()[0]) == state:
            final.append(g)
    return final
```

The full code can be found at https://github.com/ayanchowdhury2810/LR-0--Parsing

Output:



Process	Look Ahead	Symbol	Stack
Action(0, a) = S3	0	a	[0, 'a', 3]
Action(3, a) = S3	1	a	[0, 'a', 3, 'a', 3]
Action(3, b) = S4	2	b	[0, 'a', 3, 'a', 3, 'b', 4]
Action(4, b) = r3	3	b	[0, 'a', 3, 'a', 3, 'A']
goto(3, A) = 6	3	b	[0, 'a', 3, 'a', 3, 'A', 6]
Action(6, b) = r2	3	b	[0, 'a', 3, 'A']
goto(3, A) = 6	3	b	[0, 'a', 3, 'A', 6]
Action(6, b) = r2	3	b	[0, 'A']
goto(0, A) = 2	3	b	[0, 'A', 2]
Action(2, b) = S4	3	b	[0, 'A', 2, 'b', 4]
Action(4, $) = r3	4	$	[0, 'A', 2, 'A']
goto(2, A) = 5	4	$	[0, 'A', 2, 'A', 5]
Action(5, $) = r1	4	$	[0, 'S']
goto(0, S) = 1	4	$	[0, 'S', 1]
Action(1, $) = Accept	4	$	[0, 'S', 1]