SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chennai-603203.

FACULTY OF ENGINEERING AND TECHNOLOGY

Department of Data Science and Business Systems

Academic Year (2022–2023)

18CSC362J - Compiler Design

SEMESTER–V

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

Kattankulathur, Chennai-603203.

FACULTY OF ENGINEERING AND TECHNOLOGY

18CSC362J-Compiler Design

REG.No:

R	A	2	0	1	1	0	4	2	0	1	0			

BONAFIDECERTIFICATE

Certified that this is the bonafide record of work done by ________of V semester B.Tech COMPUTER SCIENCE AND ENGINEERING during the academic year 2022-2023 in the 18CSC362J -Compiler Design Laboratory.

------------------------------ ----------------------------------

Dr. Chinnasamy A HOD

Staff -Incharge

Submitted for the practical examination held on_____________ at SRM Institute of Science and Technology, Kattankulathur, Chennai-603203.

--------------------------------- ------------------------------

Examiner-1 Examiner-2

Experiment -1 **Implementation of Symbol Table**

Aim: To write a "C" program for the implementation of symbol table with functions to create, insert, modify, search and display.

Algorithm:
1. Declare the variable in the structure as needed.
2. Create a separate function for each operation and use switch case to execute the function.
3. Inside the function insert, search whether the label is inside the symbol table or not
4. If it is already present, ignore insertion else insert in the symbol table
5. Inside the function search, search for the label and if it is found print success else print failure.
6. Inside the function delete, delete the label specified from the symbol table.
7. Inside the function modify, update the table entry with the new values.
8. Inside the function display, display the content of the symbol table.

Program:
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#define null 0
int size=0;
void Insert();
void Display();
void Delete();
int Search(char lab[]);
void Modify();
struct SymbTab
{
 char label[10],symbol[10];
 int addr;
struct SymbTab *next;};
struct SymbTab *first,*last;
void main()
{
 int op,y;
 char la[10];
 do
 {
  printf("\n\tSYMBOL TABLE IMPLEMENTATION\n");
```

```
printf("\n\t1.INSERT\n\t2.DISPLAY\n\t3.DELETE\n\t4.SEARCH\n\t5.MODIFY\n\t
6.END\n");
  printf("\n\tEnter your option : ");
  scanf("%d",&op);
  switch(op)
  {
   case 1:
      Insert();
      break;
   case 2:
      Display();
      break;
   case 3:
      Delete();
      break;
   case 4:
      printf("\n\tEnter the label to be searched : ");
      scanf("%s",la);
      y=Search(la);
      printf("\n\tSearch Result:");
      if(y==1)
    printf("\n\tThe label is present in the symbol table\n");
      else
    printf("\n\tThe label is not present in the symbol table\n");
      break;
   case 5:
      Modify();
      break;
   case 6:
      exit(0);
  }
 }while(op<6);
 getch();
}
void Insert()
{
  int n;
  char l[10];
  printf("\n\tEnter the label : ");
  scanf("%s",l);
  n=Search(l);
  if(n==1)
   printf("\n\tThe label exists already in the symbol table\n\tDuplicate can't be
inserted");
```

```
  else
   {
    struct SymbTab *p;
    p=malloc(sizeof(struct SymbTab));
    strcpy(p->label,l);
    printf("\n\tEnter the symbol : ");
    scanf("%s",p->symbol);
    printf("\n\tEnter the address : ");
    scanf("%d",&p->addr);
    p->next=NULL;
    if(size==0)
     {
      first=p;
      last=p;
     }
    else
     {
      last->next=p;
      last=p;
     }
    size++;
   }
   printf("\n\tLabel inserted\n");
}
void Display()
{
  int i;
  struct SymbTab *p;
  p=first;
  printf("\n\tLABEL\t\tSYMBOL\t\tADDRESS\n");
  for(i=0;i<size;i++)
   {
    printf("\t%s\t\t%s\t\t%d\n",p->label,p->symbol,p->addr);
    p=p->next;
   }
}
int Search(char lab[])
{
 int i,flag=0;
 struct SymbTab *p;
 p=first;
  for(i=0;i<size;i++)
   {
    if(strcmp(p->label,lab)==0)
     flag=1;
```

```
   p=p->next;
  }
 return flag;
}
void Modify()
{
 char l[10],nl[10];
 int add,choice,i,s;
 struct SymbTab *p;
 p=first;
 printf("\n\tWhat do you want to modify?\n");
 printf("\n\t1.Only the label\n\t2.Only the address\n\t3.Both the label and
address\n");
 printf("\tEnter your choice : ");
 scanf("%d",&choice);
 switch(choice)
  {
   case 1:
     printf("\n\tEnter the old label : ");
     scanf("%s",l);
     s=Search(l);
     if(s==0)
   printf("\n\tLabel not found\n");
     else
   {
   printf("\n\tEnter the new label : ");
   scanf("%s",nl);
   for(i=0;i<size;i++)
    {
     if(strcmp(p->label,l)==0)
       strcpy(p->label,nl);
     p=p->next;
    }
   printf("\n\tAfter Modification:\n");
   Display();
   }
  break;
  case 2:
     printf("\n\tEnter the label where the address is to be modified : ");
     scanf("%s",l);
     s=Search(l);
     if(s==0)
   printf("\n\tLabel not found\n");
     else
   {
```

```
     printf("\n\tEnter the new address : ");
     scanf("%d",&add);
     for(i=0;i<size;i++)
      {
       if(strcmp(p->label,l)==0)
        p->addr=add;
       p=p->next;
      }
     printf("\n\tAfter Modification:\n");
     Display();
    }
    break;
    case 3:
       printf("\n\tEnter the old label : ");
       scanf("%s",l);
       s=Search(l);
       if(s==0)
     printf("\n\tLabel not found\n");
       else
    {
     printf("\n\tEnter the new label : ");
     scanf("%s",nl);
     printf("\n\tEnter the new address : ");
     scanf("%d",&add);
     for(i=0;i<size;i++)
      {
       if(strcmp(p->label,l)==0)
        {
         strcpy(p->label,nl);
         p->addr=add;
        }
       p=p->next;
      }
     printf("\n\tAfter Modification:\n");
     Display();
    }
    break;
    }
}
void Delete()
{
  int a;
  char l[10];
  struct SymbTab *p,*q;
  p=first;
```

```
 printf("\n\tEnter the label to be deleted : ");
 scanf("%s",l);
 a=Search(l);
 if(a==0)
   printf("\n\tLabel not found\n");
 else
  {
   if(strcmp(first->label,l)==0)
   first=first->next;
   else if(strcmp(last->label,l)==0)
    {
     q=p->next;
     while(strcmp(q->label,l)!=0)
      {
       p=p->next;
       q=q->next;
      }
     p->next=NULL;
     last=p;
    }
   else
    {
     q=p->next;
     while(strcmp(q->label,l)!=0)
      {
       p=p->next;
       q=q->next;
      }
     p->next=q->next;
   }
   size--;
   printf("\n\tAfter Deletion:\n");
   Display();
  }
}
```

Result: "C" program for the implementation of symbol table with functions to create, insert, modify, search and display is done successfully.

Experiment -2 **Develop a Lexical Analyser**

Aim: To write a program to implement Lexical Analysis using C.

Algorithm:
1. Start the program.
2. Declare all the variables and file pointers.
3. Display the input program.
4. Separate the keyword in the program and display it.
5. Display the header files of the input program
6. Separate the operators of the input program and display it.
7. Print the punctuation marks.
8. Print the constant that are present in input program.
9. Print the identifiers of the input program.

Program:
```
#include <fstream>
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>


using namespace std;

bool isPunctuator(char ch)                              //check if the given
character is a punctuator or not
{
   if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
      ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
      ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
      ch == '[' || ch == ']' || ch == '{' || ch == '}' ||
      ch == '&' || ch == '|')
      {
         return true;
      }
   return false;
}

bool validIdentifier(char* str)                              //check if the given
identifier is valid or not
{
```

```
    if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
        str[0] == '3' || str[0] == '4' || str[0] == '5' ||
        str[0] == '6' || str[0] == '7' || str[0] == '8' ||
        str[0] == '9' || isPunctuator(str[0]) == true)
        {
            return false;
        }                                                   //if first character of string
is a digit or a special character, identifier is not valid
    int i,len = strlen(str);
    if (len == 1)
    {
        return true;
    }                                                        //if length is one,
validation is already completed, hence return true
    else
    {
    for (i = 1 ; i < len ; i++)                              //identifier cannot
contain special characters
    {
        if (isPunctuator(str[i]) == true)
        {
            return false;
        }
    }
    }
    return true;
}

bool isOperator(char ch)                                     //check if the given
character is an operator or not
{
    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=' || ch == '|' || ch == '&')
    {
       return true;
    }
    return false;
}

bool isKeyword(char *str)                                    //check if the given
substring is a keyword or not
{
    if (!strcmp(str, "if") || !strcmp(str, "else") ||
        !strcmp(str, "while") || !strcmp(str, "do") ||
```

```
        !strcmp(str, "break") ||  !strcmp(str, "continue")
        || !strcmp(str, "int") || !strcmp(str, "double")
        || !strcmp(str, "float") || !strcmp(str, "return")
        || !strcmp(str, "char") || !strcmp(str, "case")
        || !strcmp(str, "long") || !strcmp(str, "short")
        || !strcmp(str, "typedef") || !strcmp(str, "switch")
        || !strcmp(str, "unsigned") || !strcmp(str, "void")
        || !strcmp(str, "static") || !strcmp(str, "struct")
        || !strcmp(str, "sizeof") || !strcmp(str,"long")
        || !strcmp(str, "volatile") || !strcmp(str, "typedef")
        || !strcmp(str, "enum") || !strcmp(str, "const")
        || !strcmp(str, "union") || !strcmp(str, "extern")
        || !strcmp(str,"bool"))
        {
           return true;
        }
    else
    {
      return false;
    }
}

bool isNumber(char* str)                                          //check if the given
substring is a number or not
{
    int i, len = strlen(str),numOfDecimal = 0;
    if (len == 0)
    {
        return false;
    }
    for (i = 0 ; i < len ; i++)
    {
        if (numOfDecimal > 1 && str[i] == '.')
        {
            return false;
        } else if (numOfDecimal <= 1)
        {
            numOfDecimal++;
        }
        if (str[i] != '0' && str[i] != '1' && str[i] != '2'
            && str[i] != '3' && str[i] != '4' && str[i] != '5'
            && str[i] != '6' && str[i] != '7' && str[i] != '8'
            && str[i] != '9' || (str[i] == '-' && i > 0))
            {
                return false;
```

```
        }
    }
    return true;
}

char* subString(char* realStr, int l, int r)                    //extract the required
substring from the main string
{
    int i;

    char* str = (char*) malloc(sizeof(char) * (r - l + 2));

    for (i = l; i <= r; i++)
    {
        str[i - l] = realStr[i];
        str[r - l + 1] = '\0';
    }
    return str;
}


void parse(char* str)                                       //parse the expression
{
    int left = 0, right = 0;
    int len = strlen(str);
    while (right <= len && left <= right) {
        if (isPunctuator(str[right]) == false)              //if character is a digit or an
alphabet
            {
                right++;
            }

        if (isPunctuator(str[right]) == true && left == right)      //if character is a
punctuator
            {
            if (isOperator(str[right]) == true)
            {
                std::cout<< str[right] <<" IS AN OPERATOR\n";
            }
            right++;
            left = right;
            } else if (isPunctuator(str[right]) == true && left != right
                || (right == len && left != right))                 //check if parsed
substring is a keyword or identifier or number
            {
```

```
        char* sub = subString(str, left, right - 1);   //extract substring

        if (isKeyword(sub) == true)
                {
                   cout<< sub <<" IS A KEYWORD\n";
                }
        else if (isNumber(sub) == true)
                {
                   cout<< sub <<" IS A NUMBER\n";
                }
        else if (validIdentifier(sub) == true
              && isPunctuator(str[right - 1]) == false)
              {
                 cout<< sub <<" IS A VALID IDENTIFIER\n";
              }
        else if (validIdentifier(sub) == false
              && isPunctuator(str[right - 1]) == false)
              {
                 cout<< sub <<" IS NOT A VALID IDENTIFIER\n";
              }

        left = right;
        }
   }
   return;
}

int main()
{
   char c[100] = "int a = b * c";
   parse(c);
   return 0;
}
```

Result: The implementation of lexical analyser in C++ was compiled, executed and verified successfully.

Experiment -3 **Conversion from NFA to DFA**

Aim: : To write a program for converting NFA to DFA.

Algorithm:
1. Start
2. Get the input from the user
3. Set the only state in SDFA to “unmarked”.
4. while SDFA contains an unmarked state do:
5. Let T be that unmarked state
6. b. for each a in % do S = e-Closure(MoveNFA(T,a)) c. if S is not in SDFA already then, add S to SDFA (as an “unmarked” state) d. Set MoveDFA(T,a) to S.
7. For each S in SDFA if any s & S is a final state in the NFA then, mark S an a final state in the DFA
8. Print the result.
9. Stop the program.

Program:
```
#include<stdio.h>
#include<string.h>
#include<math.h>

int ninputs;
int dfa[100][2][100] = {0};
int state[10000] = {0};
char ch[10], str[1000];
int go[10000][2] = {0};
int arr[10000] = {0};

int main()
{
    int st, fin, in;
    int f[10];
    int i,j=3,s=0,final=0,flag=0,curr1,curr2,k,l;
    int c;

    printf("\nFollow the one based indexing\n");

    printf("\nEnter the number of states::");
    scanf("%d",&st);

    printf("\nGive state numbers from 0 to %d",st-1);

    for(i=0;i<st;i++)
```

```
	state[(int)(pow(2,i))] = 1;

printf("\nEnter number of final states\t");
scanf("%d",&fin);

printf("\nEnter final states::");
for(i=0;i<fin;i++)
{
   scanf("%d",&f[i]);
}

int p,q,r,rel;

printf("\nEnter the number of rules according to NFA::");
scanf("%d",&rel);

printf("\n\nDefine transition rule as \"initial state input symbol final state\"\n");



for(i=0; i<rel; i++)
{
   scanf("%d%d%d",&p,&q,&r);
   if (q==0)
     dfa[p][0][r] = 1;
   else
     dfa[p][1][r] = 1;
}

printf("\nEnter initial state::");
scanf("%d",&in);

in = pow(2,in);

i=0;

printf("\nSolving according to DFA");

int x=0;
for(i=0;i<st;i++)
{
	for(j=0;j<2;j++)
	{
		int stf=0;
		for(k=0;k<st;k++)
```

```c
			{
				if(dfa[i][j][k]==1)
				   stf = stf + pow(2,k);
			}


			go[(int)(pow(2,i))][j] = stf;
			printf("%d-%d-->%d\n",(int)(pow(2,i)),j,stf);
			if(state[stf]==0)
			   arr[x++] = stf;
			state[stf] = 1;
		}

	}


	//for new states
	for(i=0;i<x;i++)
	{
		printf("for %d ---- ",arr[x]);
		for(j=0;j<2;j++)
		{
			int new=0;
			for(k=0;k<st;k++)
			{
				if(arr[i] & (1<<k))
				{
					int h = pow(2,k);

					if(new==0)
					   new = go[h][j];
					new = new | (go[h][j]);


				}
			}

			if(state[new]==0)
			{
			   arr[x++] = new;
			   state[new] = 1;
			}
		}
	}
```

```
printf("\nThe total number of distinct states are::\n");

printf("STATE     0   1\n");

for(i=0;i<10000;i++)
{
      if(state[i]==1)
      {
            //printf("%d**",i);
            int y=0;
            if(i==0)
               printf("q0 ");

            else
            for(j=0;j<st;j++)
            {
                  int x = 1<<j;
                  if(x&i)
                  {
                     printf("q%d ",j);
                     y = y+pow(2,j);
                     //printf("y=%d  ",y);
                  }
            }
            //printf("%d",y);
            printf("      %d   %d",go[y][0],go[y][1]);
            printf("\n");
      }
}



j=3;
while(j--)
{
      printf("\nEnter string");
      scanf("%s",str);
      l = strlen(str);
      curr1 = in;
      flag = 0;
      printf("\nString takes the following path-->\n");
      printf("%d-",curr1);

      for(i=0;i<l;i++)
      {
```

```
            curr1 = go[curr1][str[i]-'0'];
            printf("%d-",curr1);
        }

        printf("\nFinal state - %d\n",curr1);

        for(i=0;i<fin;i++)
        {
            if(curr1 & (1<<f[i]))
            {
                flag = 1;
                break;
            }
        }

        if(flag)
            printf("\nString Accepted");
        else
            printf("\nString Rejected");

    }


   return 0;
}
```

Input/Output-

Follow the one based indexing
Enter the number of states::3
Give state numbers from 0 to 2
Enter number of final states 1
Enter final states::4
Enter the number of rules according to NFA::4
Define transition rule as "initial state input symbol final state"
1 0 1
1 1 1
1 0 2
2 0 4
Enter initial state::1
Solving according to DFA1-0-->0
1-1-->0
2-0-->6
2-1-->2
4-0-->0

4-1-->0
for 0 ---- for 0 ----
The total number of distinct states are::
STATE 0 1
q0 0 0
q0 0 0
q1 6 2
q2 0 0
q1 q2 0 0

Result: The implementation of converting NFA to DFA in C was compiled, executed and verified successfully.

Experiment -4 **Conversion from Regular Expression to NFA**

Aim: To write a program for converting Regular Expression to NFA.

Algorithm:

1. Start
2. Get the input from the user
3. Initialize separate variables and functions for Postfix , Display and NFA
4. Create separate methods for different operators like +,*, .
5. By using Switch case Initialize different cases for the input
6. For ' . ' operator Initialize a separate method by using various stack functions do the same for the other operators like ' * ' and ' + '.
7. Regular expression is in the form like a.b (or) a+b
8. Display the output
9. Stop

Program:

```
#include<stdio.h>
#include<string.h>
int main()
{
        char reg[20]; int q[20][3],i=0,j=1,len,a,b;
        for(a=0;a<20;a++) for(b=0;b<3;b++) q[a][b]=0;
        scanf("%s",reg);
        printf("Given regular expression: %s\n",reg);
        len=strlen(reg);
        while(i<len)
        {
                if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*') { q[j][0]=j+1; j++; }
                if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][1]=j+1; j++;    }
                if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*') {     q[j][2]=j+1; j++;    }
                if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
                {
                  q[j][2]=((j+1)*10)+(j+3); j++;
                  q[j][0]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+1; j++;
                        i=i+2;
                }
                if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
                {
                        q[j][2]=((j+1)*10)+(j+3); j++;
                        q[j][1]=j+1; j++;
                        q[j][2]=j+3; j++;
                        q[j][0]=j+1; j++;
```

```
                q[j][2]=j+1; j++;
                i=i+2;
            }
            if(reg[i]=='a'&&reg[i+1]=='*')
            {
                q[j][2]=((j+1)*10)+(j+3); j++;
                q[j][0]=j+1; j++;
                q[j][2]=((j+1)*10)+(j-1); j++;
            }
            if(reg[i]=='b'&&reg[i+1]=='*')
            {
                q[j][2]=((j+1)*10)+(j+3); j++;
                q[j][1]=j+1; j++;
                q[j][2]=((j+1)*10)+(j-1); j++;
            }
            if(reg[i]==')'&&reg[i+1]=='*')
            {
                q[0][2]=((j+1)*10)+1;
                q[j][2]=((j+1)*10)+1;
                j++;
            }
            i++;
        }
        printf("\n\tTransition Table \n");
        printf("_____________________________________________\n");
        printf("Current State |\tInput |\tNext State");
        printf("\n_____________________________________________\n");
        for(i=0;i<=j;i++)
        {
            if(q[i][0]!=0) printf("\n  q[%d]\t      |   a   |  q[%d]",i,q[i][0]);
            if(q[i][1]!=0) printf("\n  q[%d]\t      |   b   |  q[%d]",i,q[i][1]);
            if(q[i][2]!=0)
            {
                if(q[i][2]<10) printf("\n  q[%d]\t      |   e   |  q[%d]",i,q[i][2]);
                else printf("\n  q[%d]\t      |   e   |  q[%d] , q[%d]",i,q[i][2]/10,q[i]
[2]%10);
            }
        }
        printf("\n_____________________________________________\n");
        return 0;
}
```

Input: (a|b)*a
Output:
Given regular expression: (a|b)*a

Transition Table

Current State	Input	Next State
q[0]	e	q[7] , q[1]
q[1]	e	q[2] , q[4]
q[2]	a	q[3]
q[3]	e	q[6]
q[4]	b	q[5]
q[5]	e	q[6]
q[6]	e	q[7] , q[1]
q[7]	a	q[8]

Result: The implementation of converting Regular Expression to NFA in C was compiled, executed and verified successfully.

Experiment -5 **First and Follow**

Aim: To write a program to implement Lexical Analysis using C.

Algorithm:

First:
To find the first() of the grammar symbol, then we have to apply the following set of rules to the given grammar:-

- If X is a terminal, then First(X) is {X}.
- If X is a non-terminal and X tends to aα is production, then add 'a' to the first of X. if X->ε, then add null to the First(X).
- If X_>YZ then if First(Y)=ε, then First(X) = { First(Y)-ε} U First(Z).
- If X->YZ, then if First(X)=Y, then First(Y)=teminal but null then First(X)=First(Y)=terminals.

Follow:
To find the follow(A) of the grammar symbol, then we have to apply the following set of rules to the given grammar:-

- $ is a follow of 'S'(start symbol).
- If A->αBβ,β!=ε, then first(β) is in follow(B).
- If A->αB or A->αBβ where First(β)=ε, then everything in Follow(A) is a Follow(B).

Program:

```
// C program to calculate the First and
// Follow sets of a given grammar
#include<stdio.h>
#include<ctype.h>
#include<string.h>

// Functions to calculate Follow
void followfirst(char, int, int);
void follow(char c);

// Function to calculate First
void findfirst(char, int, int);
```

```
int count, n = 0;

// Stores the final result
// of the First Sets
char calc_first[10][100];

// Stores the final result
// of the Follow Sets
char calc_follow[10][100];
int m = 0;

// Stores the production rules
char production[10][10];
char f[10], first[10];
int k;
char ck;
int e;

int main(int argc, char **argv)
{
	int jm = 0;
	int km = 0;
	int i, choice;
	char c, ch;
	count = 8;

	// The Input grammar
	strcpy(production[0], "E=TR");
	strcpy(production[1], "R=+TR");
	strcpy(production[2], "R=#");
	strcpy(production[3], "T=FY");
	strcpy(production[4], "Y=*FY");
	strcpy(production[5], "Y=#");
	strcpy(production[6], "F=(E)");
	strcpy(production[7], "F=i");

	int kay;
	char done[count];
	int ptr = -1;

	// Initializing the calc_first array
	for(k = 0; k < count; k++) {
		for(kay = 0; kay < 100; kay++) {
			calc_first[k][kay] = '!';
```

```
		}
	}
	int point1 = 0, point2, xxx;

	for(k = 0; k < count; k++)
	{
		c = production[k][0];
		point2 = 0;
		xxx = 0;

		// Checking if First of c has
		// already been calculated
		for(kay = 0; kay <= ptr; kay++)
			if(c == done[kay])
				xxx = 1;

		if (xxx == 1)
			continue;

		// Function call
		findfirst(c, 0, 0);
		ptr += 1;

		// Adding c to the calculated list
		done[ptr] = c;
		printf("\n First(%c) = { ", c);
		calc_first[point1][point2++] = c;

		// Printing the First Sets of the grammar
		for(i = 0 + jm; i < n; i++) {
			int lark = 0, chk = 0;

			for(lark = 0; lark < point2; lark++) {

				if (first[i] == calc_first[point1][lark])
				{
					chk = 1;
					break;
				}
			}
			if(chk == 0)
			{
				printf("%c, ", first[i]);
				calc_first[point1][point2++] = first[i];
			}
```

```
        }
        printf("}\n");
        jm = n;
        point1++;
    }
    printf("\n");
    printf("-----------------------------------------------\n\n");
    char donee[count];
    ptr = -1;

    // Initializing the calc_follow array
    for(k = 0; k < count; k++) {
        for(kay = 0; kay < 100; kay++) {
            calc_follow[k][kay] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for(e = 0; e < count; e++)
    {
        ck = production[e][0];
        point2 = 0;
        xxx = 0;

        // Checking if Follow of ck
        // has alredy been calculated
        for(kay = 0; kay <= ptr; kay++)
            if(ck == donee[kay])
                xxx = 1;

        if (xxx == 1)
            continue;
        land += 1;

        // Function call
        follow(ck);
        ptr += 1;

        // Adding ck to the calculated list
        donee[ptr] = ck;
        printf(" Follow(%c) = { ", ck);
        calc_follow[point1][point2++] = ck;

        // Printing the Follow Sets of the grammar
        for(i = 0 + km; i < m; i++) {
```

```
                    int lark = 0, chk = 0;
                    for(lark = 0; lark < point2; lark++)
                    {
                        if (f[i] == calc_follow[point1][lark])
                        {
                            chk = 1;
                            break;
                        }
                    }
                    if(chk == 0)
                    {
                        printf("%c, ", f[i]);
                        calc_follow[point1][point2++] = f[i];
                    }
                }
                printf(" }\n\n");
                km = m;
                point1++;
            }
}

void follow(char c)
{
    int i, j;

    // Adding "$" to the follow
    // set of the start symbol
    if(production[0][0] == c) {
        f[m++] = '$';
    }
    for(i = 0; i < 10; i++)
    {
        for(j = 2;j < 10; j++)
        {
            if(production[i][j] == c)
            {
                if(production[i][j+1] != '\0')
                {
                    // Calculate the first of the next
                    // Non-Terminal in the production
                    followfirst(production[i][j+1], i, (j+2));
                }

                if(production[i][j+1]=='\0' && c!=production[i][0])
                {
```

```
					// Calculate the follow of the Non-Terminal
					// in the L.H.S. of the production
					follow(production[i][0]);
				}
			}
		}
	}
}

void findfirst(char c, int q1, int q2)
{
	int j;

	// The case where we
	// encounter a Terminal
	if(!(isupper(c))) {
		first[n++] = c;
	}
	for(j = 0; j < count; j++)
	{
		if(production[j][0] == c)
		{
			if(production[j][2] == '#')
			{
				if(production[q1][q2] == '\0')
					first[n++] = '#';
				else if(production[q1][q2] != '\0'
						&& (q1 != 0 || q2 != 0))
				{
					// Recursion to calculate First of New
					// Non-Terminal we encounter after epsilon
					findfirst(production[q1][q2], q1, (q2+1));
				}
				else
					first[n++] = '#';
			}
			else if(!isupper(production[j][2]))
			{
				first[n++] = production[j][2];
			}
			else
			{
				// Recursion to calculate First of
				// New Non-Terminal we encounter
				// at the beginning
```

```
                    findfirst(production[j][2], j, 3);
                }
            }
        }
}

void followfirst(char c, int c1, int c2)
{
        int k;

        // The case where we encounter
        // a Terminal
        if(!(isupper(c)))
                f[m++] = c;
        else
        {
                int i = 0, j = 1;
                for(i = 0; i < count; i++)
                {
                        if(calc_first[i][0] == c)
                                break;
                }

                //Including the First set of the
                // Non-Terminal in the Follow of
                // the original query
                while(calc_first[i][j] != '!')
                {
                        if(calc_first[i][j] != '#')
                        {
                                f[m++] = calc_first[i][j];
                        }
                        else
                        {
                                if(production[c1][c2] == '\0')
                                {
                                        // Case where we reach the
                                        // end of a production
                                        follow(production[c1][0]);
                                }
                                else
                                {
                                        // Recursion to the next symbol
                                        // in case we encounter a "#"
                                        followfirst(production[c1][c2], c1, c2+1);
```

```
                    }
                }
                j++;
            }
        }
}
```

Result: The FIRST and FOLLOW sets of the non-terminals of a grammar were found successfully using python language.

Experiment -6 **Predictive Parsing Table**

Aim: To write a a program for Predictive Parsing table.

Algorithm:

For the production A → α of Grammar G.

- For each terminal, a in FIRST (α) add A → α to M [A, a].
- If ε is in FIRST (α), and b is in FOLLOW (A), then add A → α to M[A, b].
- If ε is in FIRST (α), and $ is in FOLLOW (A), then add A → α to M[A, $].
- All remaining entries in Table M are errors.

Program:

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

char prol[7][10] = { "S", "A", "A", "B", "B", "C", "C" };
char pror[7][10] = { "A", "Bb", "Cd", "aB", "@", "Cc", "@" };
char prod[7][10] = { "S->A", "A->Bb", "A->Cd", "B->aB", "B->@", "C->Cc", "C-
>@" };
char first[7][10] = { "abcd", "ab", "cd", "a@", "@", "c@", "@" };
char follow[7][10] = { "$", "$", "$", "a$", "b$", "c$", "d$" };
char table[5][6][10];

int numr(char c)
{
  switch (c)
  {
    case 'S':
      return 0;

    case 'A':
      return 1;

    case 'B':
      return 2;

    case 'C':
      return 3;

    case 'a':
```

```
      return 0;

    case 'b':
      return 1;

    case 'c':
      return 2;

    case 'd':
      return 3;

    case '$':
      return 4;
  }

  return (2);
}

int main()
{
  int i, j, k;

  for (i = 0; i < 5; i++)
    for (j = 0; j < 6; j++)
      strcpy(table[i][j], " ");

  printf("The following grammar is used for Parsing Table:\n");

  for (i = 0; i < 7; i++)
    printf("%s\n", prod[i]);

  printf("\nPredictive parsing table:\n");

  fflush(stdin);

  for (i = 0; i < 7; i++)
  {
    k = strlen(first[i]);
    for (j = 0; j < 10; j++)
      if (first[i][j] != '@')
        strcpy(table[numr(prol[i][0]) + 1][numr(first[i][j]) + 1], prod[i]);
  }

  for (i = 0; i < 7; i++)
  {
```

```
        if (strlen(pror[i]) == 1)
        {
          if (pror[i][0] == '@')
          {
            k = strlen(follow[i]);
            for (j = 0; j < k; j++)
              strcpy(table[numr(prol[i][0]) + 1][numr(follow[i][j]) + 1], prod[i]);
          }
        }
      }

      strcpy(table[0][0], " ");

      strcpy(table[0][1], "a");

      strcpy(table[0][2], "b");

      strcpy(table[0][3], "c");

      strcpy(table[0][4], "d");

      strcpy(table[0][5], "$");

      strcpy(table[1][0], "S");

      strcpy(table[2][0], "A");

      strcpy(table[3][0], "B");

      strcpy(table[4][0], "C");

      printf("\n--------------------------------------------------------\n");

      for (i = 0; i < 5; i++)
        for (j = 0; j < 6; j++)
        {
          printf("%-10s", table[i][j]);
          if (j == 5)
            printf("\n--------------------------------------------------------\n");
        }
    }
```

Result: The implementation and creation of predictive parse table using c was executed successfully.

Experiment -7 **Shift Reduce Parsing**

Aim: To write a program to implement Lexical Analysis using C.

Algorithm:

- Shift reduce parsing is a process of reducing a string to the start symbol of a grammar.
- Shift reduce parsing uses a stack to hold the grammar and an input tape to hold the string.
-
- Sift reduce parsing performs the two actions: shift and reduce. That's why it is known as shift reduces parsing.
- At the shift action, the current symbol in the input string is pushed to a stack.
- At each reduction, the symbols will replaced by the non-terminals. The symbol is the right side of the production and non-terminal is the left side of the production.

Program:

```
#include<stdio.h>
#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[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(a);
      c=strlen(a);
      strcpy(act,"SHIFT->");
      puts("stack \t input \t action");
      for(k=0,i=0; j<c; k++,i++,j++)
       {
        if(a[j]=='i' && a[j+1]=='d')
          {
            stk[i]=a[j];
            stk[i+1]=a[j+1];
            stk[i+2]='\0';
            a[j]=' ';
            a[j+1]=' ';
```

```
            printf("\n$%s\t%s$\t%sid",stk,a,act);
            check();
          }
        else
          {
            stk[i]=a[j];
            stk[i+1]='\0';
            a[j]=' ';
            printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
            check();
          }
      }

   }
void check()
   {
     strcpy(ac,"REDUCE TO E");
     for(z=0; z<c; z++)
       if(stk[z]=='i' && stk[z+1]=='d')
         {
          stk[z]='E';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          j++;
         }
     for(z=0; z<c; z++)
       if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
         {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+2]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          i=i-2;
         }
     for(z=0; z<c; z++)
       if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
         {
          stk[z]='E';
          stk[z+1]='\0';
          stk[z+1]='\0';
          printf("\n$%s\t%s$\t%s",stk,a,ac);
          i=i-2;
         }
     for(z=0; z<c; z++)
       if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
```

```
        {
         stk[z]='E';
         stk[z+1]='\0';
         stk[z+1]='\0';
         printf("\n$%s\t%s$\t%s",stk,a,ac);
         i=i-2;
        }
  }
```

Result: The implementation of shift reduce parsing was executed and verified successfully.

Experiment -8 **Computation of Lead and Trail**

Aim: To write a program to compute of Lead and Trail.

Algorithm:
1. For Leading, check for the first non-terminal.
2. If found, print it.
3. Look for next production for the same non-terminal.
4. If not found, recursively call the procedure for the single non-terminal present before the comma or End Of Production String.
5. Include it's results in the result of this non-terminal.
6. For trailing, we compute same as leading but we start from the end of the production to the beginning.
7. Stop

Program:
```
#include<iostream>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
using namespace std;

int vars,terms,i,j,k,m,rep,count,temp=-1;
char var[10],term[10],lead[10][10],trail[10][10];
struct grammar
{
	int prodno;
	char lhs,rhs[20][20];
}gram[50];
void get()
{
	cout<<"\nLEADING AND TRAILING\n";
	cout<<"\nEnter the no. of variables : ";
	cin>>vars;
	cout<<"\nEnter the variables : \n";
	for(i=0;i<vars;i++)
	{
		cin>>gram[i].lhs;
		var[i]=gram[i].lhs;
	}
	cout<<"\nEnter the no. of terminals : ";
```

```
	cin>>terms;
	cout<<"\nEnter the terminals : ";
	for(j=0;j<terms;j++)
		cin>>term[j];
	cout<<"\nPRODUCTION DETAILS\n";
	for(i=0;i<vars;i++)
	{
		cout<<"\nEnter the no. of production of "<<gram[i].lhs<<":";
		cin>>gram[i].prodno;
		for(j=0;j<gram[i].prodno;j++)
		{
			cout<<gram[i].lhs<<"->";
			cin>>gram[i].rhs[j];
		}
	}
}
void leading()
{
	for(i=0;i<vars;i++)
	{
		for(j=0;j<gram[i].prodno;j++)
		{
			for(k=0;k<terms;k++)
			{
				if(gram[i].rhs[j][0]==term[k])
					lead[i][k]=1;
				else
				{
					if(gram[i].rhs[j][1]==term[k])
						lead[i][k]=1;
				}
			}
		}
	}
	for(rep=0;rep<vars;rep++)
	{
		for(i=0;i<vars;i++)
		{
			for(j=0;j<gram[i].prodno;j++)
			{
				for(m=1;m<vars;m++)
				{
					if(gram[i].rhs[j][0]==var[m])
					{
						temp=m;
```

```
                                        goto out;
                                }
                        }
                        out:
                        for(k=0;k<terms;k++)
                        {
                                if(lead[temp][k]==1)
                                        lead[i][k]=1;
                        }
                }
            }
        }
}
void trailing()
{
        for(i=0;i<vars;i++)
        {
                for(j=0;j<gram[i].prodno;j++)
                {
                        count=0;
                        while(gram[i].rhs[j][count]!='\x0')
                                count++;
                        for(k=0;k<terms;k++)
                        {
                                if(gram[i].rhs[j][count-1]==term[k])
                                        trail[i][k]=1;
                                else
                                {
                                        if(gram[i].rhs[j][count-2]==term[k])
                                                trail[i][k]=1;
                                }
                        }
                }
        }
        for(rep=0;rep<vars;rep++)
        {
                for(i=0;i<vars;i++)
                {
                        for(j=0;j<gram[i].prodno;j++)
                        {
                                count=0;
                                while(gram[i].rhs[j][count]!='\x0')
                                        count++;
                                for(m=1;m<vars;m++)
                                {
```

```
                    if(gram[i].rhs[j][count-1]==var[m])
                        temp=m;
                }
                for(k=0;k<terms;k++)
                {
                    if(trail[temp][k]==1)
                        trail[i][k]=1;
                }
            }
        }
    }
}
void display()
{
    for(i=0;i<vars;i++)
    {
        cout<<"\nLEADING("<<gram[i].lhs<<") = ";
        for(j=0;j<terms;j++)
        {
            if(lead[i][j]==1)
                cout<<term[j]<<",";
        }
    }
    cout<<endl;
    for(i=0;i<vars;i++)
    {
        cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
        for(j=0;j<terms;j++)
        {
            if(trail[i][j]==1)
                cout<<term[j]<<",";
        }
    }
}
int main()
{

    get();
    leading();
    trailing();
    display();

}
```

Input: Enter the no. of variables : 3

Enter the variables :
E
T
F

Enter the no. of terminals : 5

Enter the terminals : (
)
+
*
id

PRODUCTION DETAILS

Enter the no. of production of E:2
E->E+T
E->T

Enter the no. of production of T:2
T->T*F
T->F

Enter the no. of production of F:2
F->(E)
F->id

Result: The program to find lead and trail was successfully compiled and run.

Experiment -9 **Computation of LR[0]**

Aim: To write a program to implement LR(0) items.

Algorithm:
1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End.

Program:
```
#include<iostream>
#include<conio.h>
#include<string.h>

using namespace std;

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";
int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];
int noitem=0;

struct Grammar
{
        char lhs;
        char rhs[8];
}g[20],item[20],clos[20][10];
```

```
int isvariable(char variable)
{
	for(int i=0;i<novar;i++)
		if(g[i].lhs==variable)
			return i+1;
	return 0;
}
void findclosure(int z, char a)
{
	int n=0,i=0,j=0,k=0,l=0;
	for(i=0;i<arr[z];i++)
	{
		for(j=0;j<strlen(clos[z][i].rhs);j++)
		{
			if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)
			{
				clos[noitem][n].lhs=clos[z][i].lhs;
				strcpy(clos[noitem][n].rhs,clos[z][i].rhs);
				char temp=clos[noitem][n].rhs[j];
				clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];
				clos[noitem][n].rhs[j+1]=temp;
				n=n+1;
			}
		}
	}
	for(i=0;i<n;i++)
	{
		for(j=0;j<strlen(clos[noitem][i].rhs);j++)
		{
			if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem]
[i].rhs[j+1])>0)
			{
				for(k=0;k<novar;k++)
				{
					if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)
					{
						for(l=0;l<n;l++)
							if(clos[noitem][l].lhs==clos[0][k].lhs
&& strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)
								break;
						if(l==n)
						{
							clos[noitem][n].lhs=clos[0][k].lhs;
						strcpy(clos[noitem][n].rhs,clos[0][k].rhs);
							n=n+1;
```

```
                                        }
                                }
                        }
                }
        }
    }
    arr[noitem]=n;
    int flag=0;
    for(i=0;i<noitem;i++)
    {
        if(arr[i]==n)
        {
            for(j=0;j<arr[i];j++)
            {
                int c=0;
                for(k=0;k<arr[i];k++)
                    if(clos[noitem][k].lhs==clos[i][k].lhs &&
strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)
                        c=c+1;
                if(c==arr[i])
                {
                    flag=1;
                    goto exit;
                }
            }
        }
    }
    exit:;
    if(flag==0)
        arr[noitem++]=n;
}

int main()
{
    cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :
\n";
    do
    {
        cin>>prod[i++];
    }while(strcmp(prod[i-1],"0")!=0);
    for(n=0;n<i-1;n++)
    {
        m=0;
        j=novar;
        g[novar++].lhs=prod[n][0];
```

```
		for(k=3;k<strlen(prod[n]);k++)
		{
			if(prod[n][k] != '|')
			g[j].rhs[m++]=prod[n][k];
			if(prod[n][k]=='|')
			{
				g[j].rhs[m]='\0';
				m=0;
				j=novar;
				g[novar++].lhs=prod[n][0];
			}
		}
	}
	for(i=0;i<26;i++)
		if(!isvariable(listofvar[i]))
			break;
	g[0].lhs=listofvar[i];
	char temp[2]={g[1].lhs,'\0'};
	strcat(g[0].rhs,temp);
	cout<<"\n\n augumented grammar \n";
	for(i=0;i<novar;i++)
		cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

	for(i=0;i<novar;i++)
	{
		clos[noitem][i].lhs=g[i].lhs;
		strcpy(clos[noitem][i].rhs,g[i].rhs);
		if(strcmp(clos[noitem][i].rhs,"ε")==0)
			strcpy(clos[noitem][i].rhs,".");
		else
		{
			for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)
				clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];
			clos[noitem][i].rhs[0]='.';
		}
	}
	arr[noitem++]=novar;
	for(int z=0;z<noitem;z++)
	{
		char list[10];
		int l=0;
		for(j=0;j<arr[z];j++)
		{
			for(k=0;k<strlen(clos[z][j].rhs)-1;k++)
			{
```

```
                    if(clos[z][j].rhs[k]=='.')
                    {
                        for(m=0;m<l;m++)
                            if(list[m]==clos[z][j].rhs[k+1])
                                break;
                        if(m==l)
                            list[l++]=clos[z][j].rhs[k+1];
                    }
                }
            }
            for(int x=0;x<l;x++)
                findclosure(z,list[x]);
        }
        cout<<"\n THE SET OF ITEMS ARE \n\n";
        for(int z=0; z<noitem; z++)
        {
            cout<<"\n I"<<z<<"\n\n";
            for(j=0;j<arr[z];j++)
                cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

        }

}
```

Input:
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
0

Result: The program for computation of LR[0] was successfully compiled and run.

Experiment -10 **Construction of DAG**

Aim: To write a program to construct a direct acyclic graph.

Algorithm:
1. Start the program
2. Include all the header files
3. Check for postfix expression and construct the in order DAG representation
4. Print the output
5. Stop the program

Program:
```
#include<stdio.h>
#include<string.h>
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
 int pos;
 char op;
}k[15];
void main()
{

 printf("\t\tINTERMEDIATE CODE GENERATION OF DAG\n\n");

 scanf("%s",str);
 printf("The intermediate code:\t\tExpression\n");
 findopr();
 explore();

}
void findopr()
{
 for(i=0;str[i]!='\0';i++)
  if(str[i]==':')
   {
  k[j].pos=i;
  k[j++].op=':';
```

```
 }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='/')
  {
  k[j].pos=i;
  k[j++].op='/';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='*')
  {
  k[j].pos=i;
  k[j++].op='*';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='+')
  {
  k[j].pos=i;
  k[j++].op='+';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='-')
  {
  k[j].pos=i;
  k[j++].op='-';
  }
}
void explore()
{
 i=1;
 while(k[i].op!='\0')
 {
  fleft(k[i].pos);
  fright(k[i].pos);
  str[k[i].pos]=tmpch--;
  printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
  for(j=0;j <strlen(str);j++)
   if(str[j]!='$')
    printf("%c",str[j]);
  printf("\n");
  i++;
 }
 fright(-1);
 if(no==0)
 {
  fleft(strlen(str));
```

```
  printf("\t%s := %s",right,left);
 }
 printf("\t%s :=  %c",right,str[k[--i].pos]);

}
void fleft(int x)
{
 int w=0,flag=0;
 x--;
 while(x!= -1 &&str[x]!= '+' &&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!
='-'&&str[x]!='/'&&str[x]!=':')
 {
  if(str[x]!='$'&& flag==0)
   {
  left[w++]=str[x];
  left[w]='\0';
  str[x]='$';
  flag=1;
   }
  x--;
 }
}
void fright(int x)
{
 int w=0,flag=0;
 x++;
 while(x!= -1 && str[x]!= '+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!
=':'&&str[x]!='-'&&str[x]!='/')
 {
  if(str[x]!='$'&& flag==0)
   {
  right[w++]=str[x];
  right[w]='\0';
  str[x]='$';
  flag=1;
   }
  x++;
 }
}
```

Input:
a=b*-c+b*-c

Result: The program for computation of direct acyclic graph was successfully compiled and run.

Experiment -11 **Construction of DAG**

Aim: To write a program to implement type checking.

Algorithm:
- Track the global scope type information (e.g. classes and their members)
- Determine the type of expressions recursively, i.e. bottom-up, passing the resulting types upwards.
- If type found correct, do the operation
- Type mismatches, semantic error will be notified

Program:
```
#include<stdio.h>
#include<stdlib.h>
int main()
{
int n,i,k,flag=0;
char vari[15],typ[15],b[15],c;
printf("Enter the number of variables:");
scanf(" %d",&n);
for(i=0;i<n;i++)
{
printf("Enter the variable[%d]:",i);
scanf(" %c",&vari[i]);
printf("Enter the variable-type[%d](float-f,int-i):",i);
scanf(" %c",&typ[i]);
if(typ[i]=='f')
flag=1;
}
printf("Enter the Expression(end with $):");
i=0;
getchar();
while((c=getchar())!='$')
{
b[i]=c;
i++;  }
k=i;
for(i=0;i<k;i++)
{
if(b[i]=='/')
{
flag=1;
break;  }  }
```

```
for(i=0;i<n;i++)
{
if(b[0]==vari[i])
{
if(flag==1)
{
if(typ[i]=='f')
{  printf("\nthe datatype is correctly defined..!\n");
break;  }
else
{  printf("Identifier %c must be a float type..!\n",vari[i]);
break;  }  }
else
{  printf("\nthe datatype is correctly defined..!\n");
break;  }  }
}
return 0;
}
```

Input:

```
Enter the number of variables:4
Enter the variable[0]:A
Enter the variable-type[0](float-f,int-i):i
Enter the variable[1]:B
Enter the variable-type[1](float-f,int-i):i
Enter the variable[2]:C
Enter the variable-type[2](float-f,int-i):f
Enter the variable[3]:D
Enter the variable-type[3](float-f,int-i):i
Enter the Expression(end with $):A=B*C/D
$
Identifier A must be a float type..!
```

Result: The program for implementation of Type Checking was successfully compiled and run.

Experiment -12 **Dataflow Analysis**

Aim: To write a program to implement type checking.

Algorithm:
- Start the Program Execution.
- Read the total Numbers of Expression
- Read the Left and Right side of Each Expressions
- Display the Expressions with Line No
- Display the Data flow movement with Particular Expressions
- Stop the Program Execution.

Program:
```
#include <stdio.h>
#include <string.h>
struct op
{
char l[20];
char r[20];
}
op[10], pr[10];
void main()
{
int a, i, k, j, n, z = 0, m, q,lineno=1;
char * p, * l;
char temp, t;
char * tem;char *match;
printf("enter no of values");
scanf("%d", & n);
for (i = 0; i < n; i++)
{
printf("\tleft\t");
scanf("%s",op[i].l);
printf("\tright:\t");
scanf("%s", op[i].r);
}
printf("intermediate Code\n");
for (i = 0; i < n; i++)
{   printf("Line No=%d\n",lineno);
printf("\t\t\t%s=", op[i].l);
printf("%s\n", op[i].r);lineno++;
```

```
}
printf("***Data Flow Analysis for the Above Code ***\n");
for(i=0;i<n;i++)
{
for(j=0;j<n;j++)
{
match=strstr(op[j].r,op[i].l);
if(match)
{
printf("\n %s is live at  %s \n ", op[i].l,op[j].r);
}
}
}}
```

Input:

```
enter no of values4
        left    a
        right:  a+b
        left    b
        right:  a+c
        left    c
        right:  a+b
        left    d
        right:  b+c+d
```

Result: The program for dataflow analysis was successfully compiled and run.

Experiment -11 **Storage Allocation**

Aim: Program to implement any one storage allocation strategies using stack.

Algorithm:

- Initially check whether the stack is empty
- Insert an element into the stack using push operation
- Insert more elements onto the stack until stack becomes full
- Delete an element from the stack using pop operation
- Display the elements in the stack
- Stop the program by exit

Program:

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 5
struct stack
{
    int stk[MAXSIZE];
    int top;
};
typedef struct stack ST;
ST s;
/*Function to add an element to stack */
void push ()
{
    int num;
    if (s.top == (MAXSIZE - 1))
    {
        printf ("Stack is Full\n");
        return;
    }
    else
    {
        printf ("\nEnter element to be pushed : ");
        scanf ("%d", &num);
        s.top = s.top + 1;
        s.stk[s.top] = num;
    }
```

```
    return;
}
/*Function to delete an element from stack */
int pop ()
{
    int num;
    if (s.top == - 1)
    {
        printf ("Stack is Empty\n");
        return (s.top);
    }
    else
    {
        num = s.stk[s.top];
        printf ("poped element is = %d\n", s.stk[s.top]);
        s.top = s.top - 1;
    }
    return(num);
}
/*Function to display the status of stack */
void display ()
{
    int i;
    if (s.top == -1)
    {
        printf ("Stack is empty\n");
        return;
    }
    else
    {
        printf ("\nStatus of elements in stack : \n");
        for (i = s.top; i >= 0; i--)
        {
            printf ("%d\n", s.stk[i]);
        }
    }
}
int main ()
{
    int ch;
    s.top = -1;

    printf ("\tSTACK OPERATIONS\n");
    printf("---------------------------\n");
    printf("      1. PUSH\n");
```

```
    printf("     2. POP\n");
    printf("     3. DISPLAY\n");
    printf("     4. EXIT\n");
    //printf("---------------------------\n");
    while(1)
    {
        printf("\nChoose operation : ");
        scanf("%d", &ch);
        switch (ch)
        {
            case 1:
                push();
            break;
            case 2:
                pop();
            break;
            case 3:
                display();
            break;
            case 4:
                exit(0);
            default:
                printf("Invalid operation \n");
        }
    }
    return 0;
}
```

Result: The program for implement of any one storage allocation strategies STACK was successfully compiled and run.