Ex. No. 1 **Date:**

SYMBOL TABLE

AIM

To write a c program for implementing the symbol table.

ALGORITHM

Step1. Start the program

Step2. Create a structure “syt” containing fields for the c datatypes and its associated memory bytes.

Step3. Create modules for performing the symbol table maintainence operation.

Step4. Get the input file “out.c” from the user.

Step5. Compare the values from the input file with the structure variables.

- **a.** If a match occurs between the C Keywords like “int”,”float”,”double”,”long”,”short” then they will be displayed under Datatype columns of the symtab then the corresponding bytes occupied will be displayed under the bytes column of the symbol table.
- **b.** If no match occurs, display it under the variables column of the symtab.

Step6. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

struct syt

{

        char v[10],t[10];

        int s;

} st[50];

char *tt[] = {"int","float","long","double","short"};

int vt[5]= {2,4,6,8,1};

FILE *fp;

void main()
```

```
{
        char *fn,u[20],t[20],s[100],*p;
        int i=0,j,k,l,y,sp=0;
        clrscr();
        printf("\n Enter file name: ");
        scanf("%s",fn);
        printf("\n The given file name is %s",fn);
        fp=fopen(fn,"r");
        printf("\nSymbol table maintenance");
        printf("\n\tVariable\tType\t\tsize\n");
        while(!feof(fp))
        {

                fscanf(fp,"%s",s);
                for(i=0;i<5;i++)
                if(strcmp(s,tt[i])==0)
                {
                        fscanf(fp,"%s",s);
                        p=strchr(s,',');
                        if(p)
                        {
                                j=0;
                                while(s[j]!=',')
                                        j++;
                                for(k=0;k<j;k++)
                                        t[k]=s[k];
```

```
                t[k]='\0';
                printf("\n%10s\t%10s\t%10d",t,tt[i],vt[i]);
                strcpy(st[sp].v,t);
                strcpy(st[sp].t,tt[i]);
                st[sp].s=vt[i];
                sp++;
                kjk:
                y=j;
                j++;
                while(s[j]!='\0'&&s[j]!=',')
                        j++;
                for(l=y;l<j;l++)
                        t[l-2]='\0';
                printf("\n%10s\t%10s\t%10d",t,tt[i],vt[i]);
                strcpy(st[sp].t,tt[i]);
                st[sp].s=vt[i];
                sp++;
                if(s[j]==',')
                        goto kjk;
        }
        else
        {
                printf("\n%10s\t%10s\t%10d",s,tt[i],vt[i]);
                strcpy(st[sp].v,t);
                strcpy(st[sp].t,tt[i]);
                st[sp].s=vt[i];
```

```
                }
        }
}
fclose(fp);
for(i=0;i<sp;i++)
        strcpy(t,st[i].v);
k=0;
for(j=0;j<strlen(t);j++)
{
        if(t[i]!=',')
        {
                u[k]=u[j];
                k++;
        }
        u[k]='\0';
        strcpy(st[i].v,u);
}
for(i=0;i<sp-2;i++)
{
        for(j=i+1;j<sp;j++)
        {
                if(strcmp(st[i].v,st[j].v)==0)
                        printf("\n\nMultiple Declaration for %s",st[i].v);
        }
}
getch();
```

```
}
```

INPUT FILE: out.c

```
main()
{
    int a
    float b
    double c
    long d
}
```

OUTPUT

```
 Enter file name: out.c

 The given file name is out.c
Symbol table maintenance
       Variable        Type            size

        a              int              2
        b            float              4
        c           double              8
        d             long              6_
```

RESULT

Thus, the C Program For implementing Symbol Table has been executed and verified successfully.

Ex. No. 2 **Date:**

ASSEMBLER

AIM

To write a c program for implementing the assembler.

ALGORITHM

Step1. Start the program

Step2. Get the input file "res.cpp" containing the assembly codes from user

Step3. if opcode is "MVI" read the operands from the input file and store it in the registers 'AREG' and 'BREG'

Step4. Create Modules For Performing the 'ADD','SUB','Mul' operations.

Step5. compare the opcode from the input file with the keywords Step

- **a.** If the opcode is ADD perform addition operation on the operands. Step
- **b.** If opcode is SUB perform subtraction operation on the operands Step
- **c.** If opcode is MUL perform Multiplication operation on the operands. Step 6: If opcode is "STOP" then exit.

Step6. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

#include<stdlib.h>

void main()

{

        FILE *f;

        char x[40],data[40],reg[40];

        int a=0,b=0,i,j,result=0,temp,c,d;

        clrscr();
```

```
printf("\t\tOUTPUT\n");
f=fopen("res.cpp","r");
while(!feof(f))
{

        fscanf(f,"%s",x);
        if(strcmp(x,"STOP")==0)
                exit(0);
        else if(strcmp(x,"MVI")==0)
        {
                fscanf(f,"%s%s",reg,data);
                if(strcmp(reg,"AREG,")==0)
                        a=atoi(data);
                else if(strcmp(reg,"BREG,")==0)
                        b=atoi(data);
        }
        else
        {
                if(strcmp(x,"ADD")==0)
                {
                        result=a+b;
                        printf("\n\tI:%d",a);
                        printf("\n\tJ:%d",b);
                        printf("\nRESULT(I+J):%d",result);
                }
                else if(strcmp(x,"SUB")==0)
```

```
                {
                        result=a-b;
                        printf("\n\tI:%d",a);
                        printf("\n\tJ:%d",b);
                        printf("\nResult(I-J):%d",result);
                }
                else if(strcmp(x,"MUL")==0)
                {
                        result=a*b;
                        printf("\nI:%d",a);
                        printf("\nJ:%d",b);
                        printf("\nresult(I*J):%d",result);
                }
                getch();
        }
    }
}
```

INPUT FILE: res.cpp

```
MVI AREG, 10
MVI BREG, 15
ADD AREG, BREG
MOVE RES, AREG
PRINT RES
Res DS 1
STOP
```

OUTPUT

RESULT

Thus, the C Program For implementing assembler has been executed and verified successfully.

Ex. No. 3 **Date:**

ABSOLUTE LOADER

AIM

Implementation of Absolute loader using c program.

ALGORITHM

Step1. Start the program

Step2. Assign the required variable

Step3. Open the files

- **a.** fp1=fopen("input.dat","r");
- **b.** fp2=fopen("output.dat","w");

Step4. Read the content

Step5. Using while loop perform the loop until character is not equal to E

Step6. **while(strcmp(input,"E")!=0)**

Step7. Then compare the character is equal to H

Step8. **If H then**

- **a.** fscanf(fp1,"%d",&start);

Step9. Like that get length, and input

Step10. Else if the character is T Then

Step11. Then perform the frprintf in fp1 for input file for ,

- **a.** input[0],inuput[1] for address
- **b.** input[2],inuput[3] for address+1
- **c.** input[4],inuput[5] for address+2
- **d.** Else if it is not H or T

Step12. Then perform the frprintf in fp2 for output file for ,

- **a.** input[0],inuput[1] for address
- **b.** input[2],inuput[3] for address+1
- **c.** input[4],inuput[5] for address+2
- **d.** fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
- **e.** fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
- **f.** fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
- **g.** address+=3;
- **h.** fscanf(fp1,"%s",input);

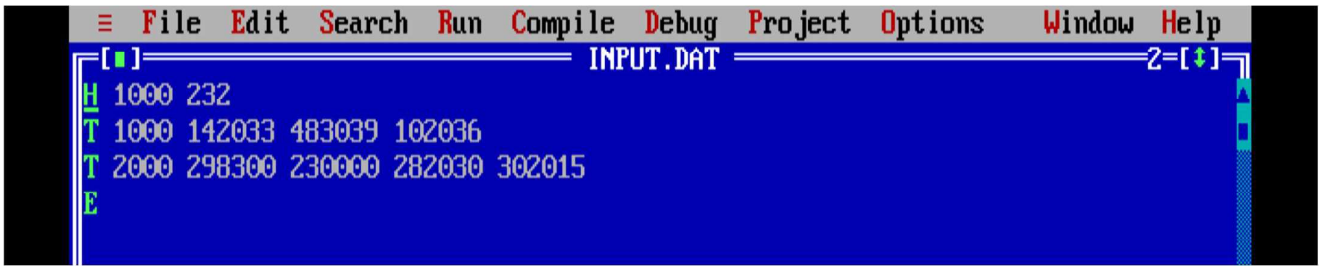
Step13. Finally terminate the program Stop the Program.

PROGRAM

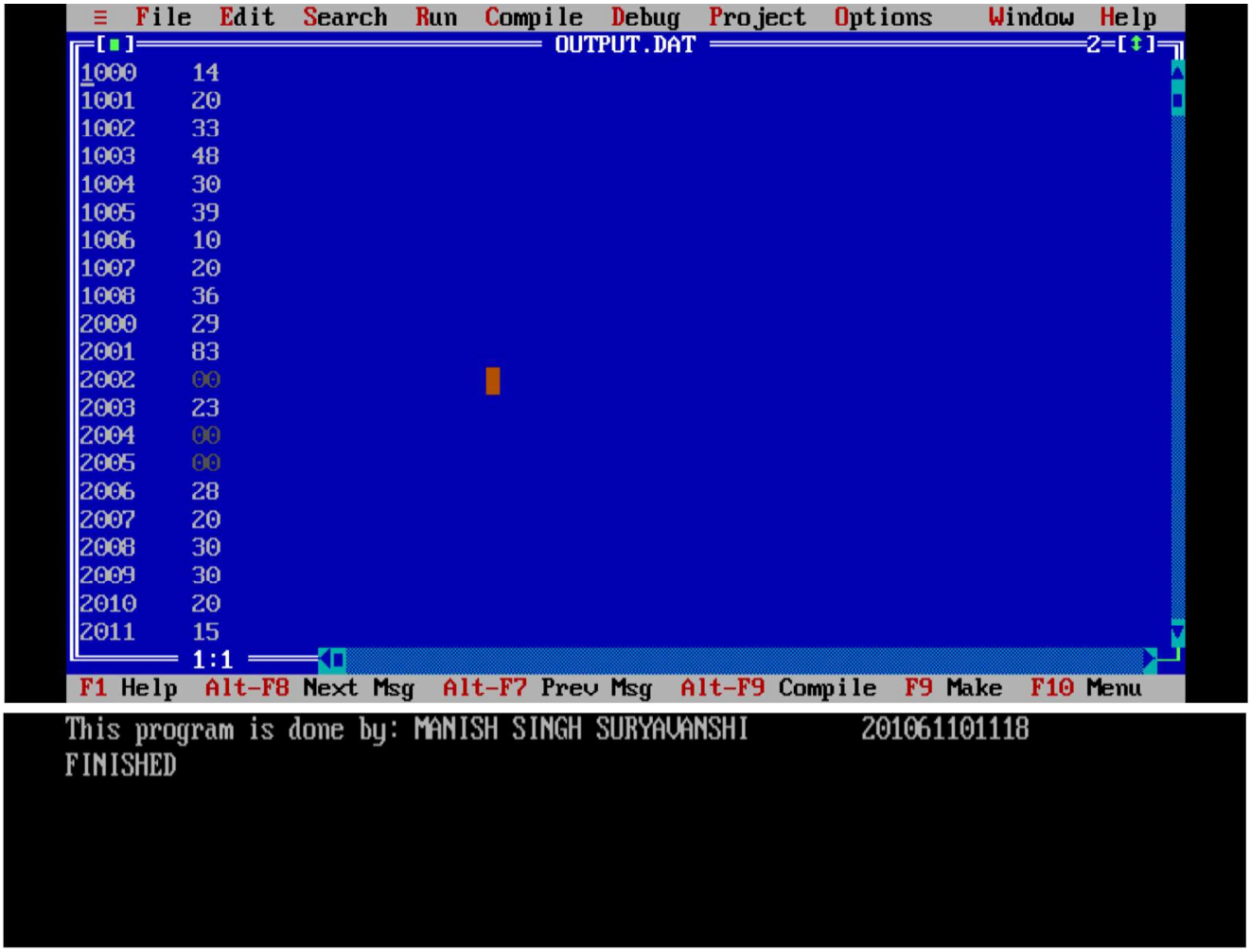
```
# include <stdio.h>
# include <conio.h>
# include <string.h>
void main()
{
        char input[10];
        int start,length,address;
        FILE *fp1,*fp2;
        clrscr();
         fp1=fopen("input.dat","r");
        fp2=fopen("output.dat","w");
        fscanf(fp1,"%s",input);
        while(strcmp(input,"E")!=0)
        {

                if(strcmp(input,"H")==0)
                {
                        fscanf(fp1,"%d",&start);
                        fscanf(fp1,"%d",&length);
                        fscanf(fp1,"%s",input);
                }
                else if(strcmp(input,"T")==0)
                {
                        fscanf(fp1,"%d",&address);
                        fscanf(fp1,"%s",input);
```

```
                fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
                fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
                fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
                address+=3;
                fscanf(fp1,"%s",input);
            }
            else
            {
                fprintf(fp2,"%d\t%c%c\n",address,input[0],input[1]);
                fprintf(fp2,"%d\t%c%c\n",(address+1),input[2],input[3]);
                fprintf(fp2,"%d\t%c%c\n",(address+2),input[4],input[5]);
                address+=3;
                fscanf(fp1,"%s",input);
            }
        }
        fclose(fp1);
        fclose(fp2);
        printf("FINISHED");
        getch();
    }
```

INPUT FILE: input.dat



OUTPUT: output.dat



RESULT

Thus, the C Program For implementing Absolute Loader has been executed and verified successfully.

Ex. No. 4 **Date:**

LEXICAL ANALYZER

AIM

To write a c program to generate Lexical Analyzer.

ALGORITHM

Step1. Start the program

Step2. Reads an input stream of characters.

Step3. Copies the input stream to an output stream.

Step4. Breaks the input stream into smaller strings that match the extended regular expressions in the lex specification file.

Step5. Executes an action for each extended regular expression that it recognizes.

Step6. These actions are C language program fragments in the lex specification file.

Step7. Each action fragment can call actions or subroutines outside of itself.

Step8. Stop the Program.

PROGRAM

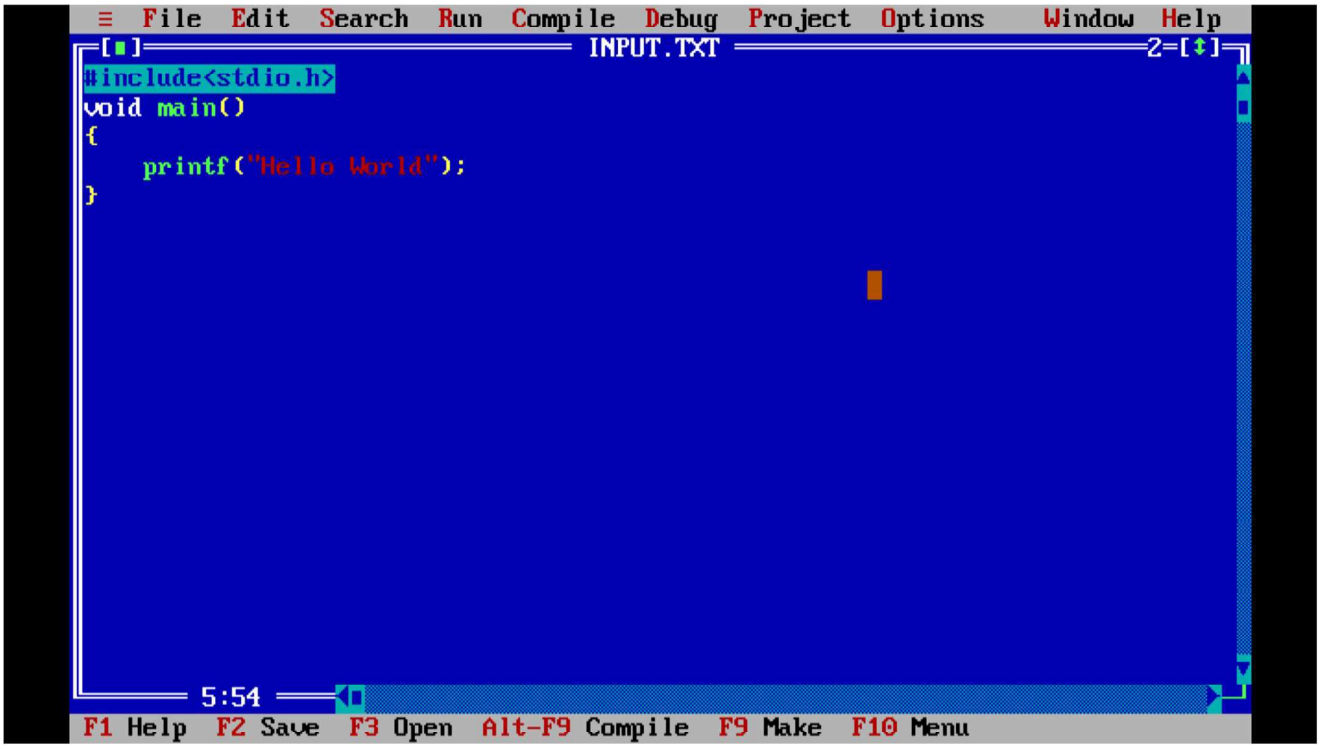
```
#include<string.h>
#include<ctype.h>
#include<stdio.h>
int main()
{
   FILE *input, *output;
   int l=1, t=0, j=0, i, flag;
   char ch, str[20];
   char keyword[30][30]={"int", "main", "if", "else", "do", "while"};
   clrscr();
        input=fopen("input.txt", "r");
   output=fopen("output.txt", "w");
```

```
fprintf(output, "Line no. \tToken no. \t Token \t\t Lexeme\n\n");
while(!feof(input))
{
        i=0;
        flag=0;
        ch=fgetc(input);
        if(ch=='+'||ch=='-'||ch=='*'||ch=='/')
        {
                fprintf(output, "%7d\t\t%7d\t\tOperator\t\t%7c\n", l, t, ch);
                t++;
        }
        else if(ch==';'||ch=='{'||ch=='}'||ch=='('||ch==')'||ch=='?'||ch=='@'||ch=='!'||ch=='%')
        {
                fprintf(output, "%7d\t\t%7d\t\tSpecial Symbol\t%7c\n", l, t, ch);
                t++;
        }
        else if(isdigit(ch))
        {
                fprintf(output, "%7d\t\t%7d\t\tDigit\t\t%7c\n", l, t, ch);
                t++;
        }
        else if(isalpha(ch))
        {
                str[i]=ch;
                i++;
                ch=fgetc(input);
```

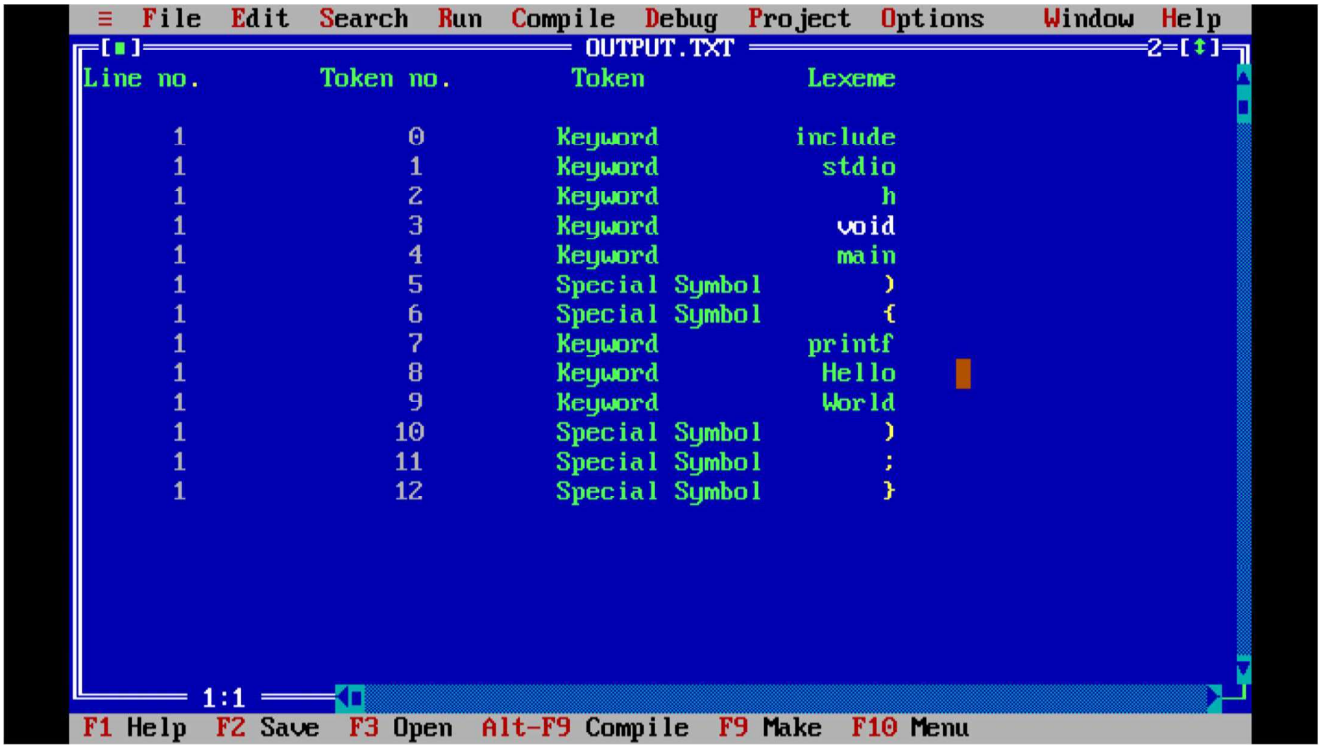
```
        while(isalnum(ch)&&ch!=' ')
        {
        str[i]=ch;
        i++;
        ch=fgetc(input);
        }
        str[i]='\0';
        for(j=0; j<=30; j++)
        {
        if(strcmp(str, keyword[j])==0)
        {
                flag=1;
                break;
        }
        }
        if(flag==1)
        {
         fprintf(output, "%7d\t\t%7d\t\tKeyword\t\t%7s\n", l, t, str);
         t++;
        }
        else
        {
         fprintf(output, "%7d\t\t%7d\t\tIdentifiers\t\t%7s\n", l, t, str);
         t++;
        }
}
```

```
            else if(ch=='\n')
            {
                i++;
            }
        }
        fclose(input);
        fclose(output);
        getch();
        return 0;
    }
```

INPUT FILE: input.txt



OUTPUT



RESULT

Thus, the C Program for implementing Lexical Analyser has been executed and verified successfully.

Ex. No. 5 **Date:**

CONSTRUCTING NFA FROM A REGULAR EXPRESSION

AIM

To write a C program to convert the regular expression to NFA.

ALGORITHM

Step1. Start the program

Step2. Declare all necessary header files.

Step3. Define the main function.

Step4. Declare the variables and initialize variables r & c to '0'.

Step5. Use a for loop within another for loop to initialize the matrix for NFA states.

Step6. Get a regular expression from the user & store it in 'm'.

Step7. Obtain the length of the expression using strlen() function and store it in 'n'.

Step8. Use for loop upto the string length and follow steps 8 to 12.

Step9. Use switch case to check each character of the expression.

Step10. If case is '/', set the links as 'E' or suitable inputs as per rules.

Step11. If case is '*', set the links as 'E' or suitable inputs as per rules.

Step12. If case is '+', set the links as 'E' or suitable inputs as per rules.

Step13. Check the default case, i.e.,for single alphabet or 2 consecutive alphabets and set the links to respective alphabet.

Step14. End the switch case.

Step15. Use for loop to print the states along the matrix.

Step16. Use a for loop within another for lop and print the value of respective links.

Step17. Print the states start state as '0' and final state.

Step18. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

void main()

{

   char m[20], t[10][10];

   int n, i, j, r=0, c=0;
```

```
clrscr();
        printf("\n\t\t\tSIMULATION OF NFA");
printf("\n\t\t\t*****************");
for(i=0; i<10; i++)
{

        for(j=0; j<10; j++)
          t[i][j]=' ';
}
printf("\n\nEnter a regular expression: ");
scanf("%s", m);
n=strlen(m);
for(i=0; i<n; i++)
{
        switch(m[i])
        {
           case '|': {  t[r][r+1]='E';
                        t[r+1][r+2]=m[i-1];
                        t[r+2][r+5]='E';
                        t[r][r+3]='E';
                        t[r+4][r+5]='E';
                        t[r+3][r+4]=m[i+1];
                        r=r+5;
                        break;
                     }
           case '*': {  t[r-1][r]='E';
```

```
                t[r][r+1]='E';
                t[r][r+3]='E';
                t[r+1][r+2]=m[i-1];
                t[r+2][r+1]='E';
                t[r+2][r+3]='E';
                r=r+3;
                break;
            }
case '+': {  t[r][r+1]=m[i-1];
                t[r+1][r]='E';
                r=r+1;
                break;
            }
default:
{
      if(c==0)
      {
         if((isalpha(m[i]))&&(isalpha(m[i+1])))
         {
               t[r][r+1]=m[i];
               t[r+1][r+2]=m[i+1];
               r=r+2;
               c=1;
         }
         c=1;
      }
```

```
                else if(c==1)
                {
                   if(isalpha(m[i+1]))
                   {
                        t[r][r+1]=m[i+1];
                        r=r+1;
                        c=2;
                   }
                }
                else
                {
                   if(isalpha(m[i+1]))
                   {
                        t[r][r+1]=m[i+1];
                        r=r+1;
                        c=3;
                   }
                }
           }
           break;
       }
}
printf("\n");
for(j=0; j<=r; j++)
       printf(" %d", j);
printf("\n_____");
```

```
        printf("\n");
        for(i=0; i<=r; i++)
        {
              for(j=0; j<=r; j++)
                 printf(" %c ", t[i][j]);
              printf("|%d", i);
              printf("\n");
        }
        printf("\nStart state: 0\nFinal state: %d", i-1);
        getch();
    }
```

OUTPUT

```
                        SIMULATION OF NFA
                        ******************

Enter a regular expression: b*c

 0 1 2 3
_____________________________
    E     E |0
      b     |1
    E     E |2
            |3

Start state: 0
Final state: 3
```

RESULT

Thus, the C Program to convert the regular expression to NFA has been executed and verified successfully.

Ex. No. 6 **Date:**

CONSTRUCTING DFA FROM A REGULAR EXPRESSION

AIM

To write a C program to convert the regular expression to DFA.

ALGORITHM

Step1. Start the program

Step2. A structure is created and the variables are declared.

Step3. Define the main function.

Step4. Declare the array for states and inpstr.

Step5. Define the readtt function to initialize the number of states, number of inputs and transition from one state to other.

Step6. Get the no. of states and no. of inputs from the user.

Step7. Start the for loop and continue till it satisfies the condition.

Step8. Print the states according to the condition.

Step9. Generate the initial and final states of the string.

Step10. Define the dfasimul function to simulate it step 10-16.

Step11. Initialize the variable start to '0' and begin for loop.

Step12. Check whether start is equal to '-1', then print "DFA Halted".

Step13. Get the string from the user and save in 'inpstr'.

Step14. Read the string and until the end-of-string reached do step 15.

Step15. Go from one state to other based on the transition entered and update start by the final state.

Step16. Print "string is accepted" if a state is equal to nstates-1.

Step17. Else print "String is not accepted".

Step18. Display the output.

Step19. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

typedef struct

{

        int input[2];
```

```
}state;
void readtt(state *states,int nstates,int ninps)
{
        int state,inp;
        printf("Enter the state transition:\n");
        for(state=0;state<nstates;state++)
        {
                for(inp=0;inp<ninps;inp++)
                {
                printf("(state:%d,%d):",state,inp);
                scanf("%d",&states[state].input[inp]);
                }
        }
}
void dfasimul(state *states,int nstates,char *inpstr)
{
        int i,start;
        start=0;
        for(i=0;inpstr[i]!='\0';i++)
                start=states[start].input[inpstr[i]-'0'];
        if(start==-1)
        {
                printf("DFA halted");
                return;
        }
        else if(start==nstates-1)
```

```
                printf("String is accepted");
        else
                printf("String is not accepted");
}
void main()
{
        state states[100];
        int nstates,ninps;
        char inpstr[20];
        clrscr();
        printf("\n\t\t\t SIMULATION OF DFA");
        printf("\n\t\t\t *****************");
        printf("\nEnter the no. of states: ");
        scanf("%d",&nstates);
        printf("Enter the no.of inputs: ");
        scanf("%d",&ninps);
        readtt(states,nstates,ninps);
        printf("\nInitial state: 0\nFinal state: %d",nstates-1);
        printf("\n\nEnter the string: ");
        scanf("%s",inpstr);
        dfasimul(states,nstates,inpstr);
        getch();

}
```

OUTPUT

```
                        SIMULATION OF DFA
                        *****************
Enter the no. of states: 2
Enter the no.of inputs: 2
Enter the state transition:
(state:0,0):0
(state:0,1):1
(state:1,0):0
(state:1,1):1

Initial state: 0
Final state: 1

Enter the string: 0110
String is not accepted
```

```
                        SIMULATION OF DFA
                        *****************
Enter the no. of states: 2
Enter the no.of inputs: 3
Enter the state transition:
(state:0,0):1
(state:0,1):0
(state:0,2):0
(state:1,0):1
(state:1,1):0
(state:1,2):1

Initial state: 0
Final state: 1

Enter the string: 01110
String is accepted
```

RESULT

Thus, the C Program to convert the regular expression to DFA has been executed and verified successfully.

Ex. No. 7 **Date:**

ELIMINATING LEFT FACTORING

AIM

To write a C program to eliminate left factoring.

ALGORITHM

Step1. Start the program

Step2. Following the first rule, **S->cAd** to parse S

Step3. The next non=term in line A is parsed using first rule, **A -> ab** , but turns out INCORRECT, parser backtracks

Step4. Next rule to parse A is taken **A->a**, turns out CORRECT. String parsed completely, parser stops.

Step5. Stop the Program.

PROGRAM

```
#include<iostream.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
struct production
{
        char lf;
        char rt[10];
        int prod_rear;
        int fl;
};
struct production prodn[20],prodn_new[20]; //Creation of object
int b=-1,d,f,q,n,m=0,c=0;
char terminal[20],nonterm[20],alpha[10],extra[10];
```

```
char epsilon='^';
void main()
{
	clrscr();

	cout<<"\nEnter the number of Special characters(except non-terminals): ";
	cin>>q;
	cout<<"Enter the special characters for your production: ";
	for(int cnt=0;cnt<q;cnt++)
		cin>>alpha[cnt];
	cout<<"\nEnter the number of productions: ";
	cin>>n;
	for(cnt=0;cnt<=n-1;cnt++)
	{

		cout<<"Enter the "<< cnt+1<<" production: ";
		cin>>prodn[cnt].lf;
		cout<<"->";
		cin>>prodn[cnt].rt;
		prodn[cnt].prod_rear=strlen(prodn[cnt].rt);
		prodn[cnt].fl=0;
	}
	for(int cnt1=0;cnt1<n;cnt1++)
	{
		for(int cnt2=cnt1+1;cnt2<n;cnt2++)
		{
			if(prodn[cnt1].lf==prodn[cnt2].lf)
```

```
{
        cnt=0;
        int p=-1;
        while((prodn[cnt1].rt[cnt]!='\0')&&(prodn[cnt2].rt[cnt]!='\0'))
        {
            if(prodn[cnt1].rt[cnt]==prodn[cnt2].rt[cnt])
            {
                extra[++p]=prodn[cnt1].rt[cnt];
                prodn[cnt1].fl=1;
                prodn[cnt2].fl=1;
            }
            else
            {
                if(p==-1)
                    break;
                else
                {
                    int h=0,u=0;
                    prodn_new[++b].lf=prodn[cnt1].lf;
                    strcpy(prodn_new[b].rt,extra);
                    prodn_new[b].rt[p+1]=alpha[c];
                    prodn_new[++b].lf=alpha[c];
                    for(int g=cnt;g<prodn[cnt2].prod_rear;g++)
                        prodn_new[b].rt[h++]=prodn[cnt2].rt[g];
                    prodn_new[++b].lf=alpha[c];
                    for(g=cnt;g<=prodn[cnt1].prod_rear;g++)
```

```
                                        prodn_new[b].rt[u++]=prodn[cnt1].rt[g];
                                        m=1;
                                        break;
                                }
                        }
                        cnt++;
                }
                if((prodn[cnt1].rt[cnt]==0)&&(m==0))
                {
                        int h=0;
                        prodn_new[++b].lf=prodn[cnt1].lf;
                        strcpy(prodn_new[b].rt,extra);
                        prodn_new[b].rt[p+1]=alpha[c];
                        prodn_new[++b].lf=alpha[c];
                        prodn_new[b].rt[0]=epsilon;
                        prodn_new[++b].lf=alpha[c];
                        for(int g=cnt;g<prodn[cnt2].prod_rear;g++)
                        prodn_new[b].rt[h++]=prodn[cnt2].rt[g];
                }
                if((prodn[cnt2].rt[cnt]==0)&&(m==0))
                {
                        int h=0;
                        prodn_new[++b].lf=prodn[cnt1].lf;
                        strcpy(prodn_new[b].rt,extra);
                        prodn_new[b].rt[p+1]=alpha[c];
                        prodn_new[++b].lf=alpha[c];
```

```
                                prodn_new[b].rt[0]=epsilon;
                                prodn_new[++b].lf=alpha[c];
                                for(int g=cnt;g<prodn[cnt1].prod_rear;g++)
                                prodn_new[b].rt[h++]=prodn[cnt1].rt[g];
                        }
                        c++;
                        m=0;
                }
        }
}
cout<<"\n\n*********************************";
cout<<"\n AFTER LEFT FACTORING ";
cout<<"\n*********************************";
cout<<endl;
for(int cnt3=0;cnt3<=b;cnt3++)
{
        cout<<"Production "<<cnt3+1<<" is: ";
        cout<<prodn_new[cnt3].lf;
        cout<<"->";
        cout<<prodn_new[cnt3].rt;
        cout<<endl<<endl;
}
for(int cnt4=0;cnt4<n;cnt4++)
{
        if(prodn[cnt4].fl==0)
        {
```

```
                cout<<"Production "<<cnt3++<<" is: ";
                cout<<prodn[cnt4].lf;
                cout<<"->";
                cout<<prodn[cnt4].rt;
                cout<<endl<<endl;
            }
        }
        getche();
    }
```

OUTPUT

```
Enter the number of Special characters(except non-terminals): 1
Enter the special characters for your production: R

Enter the number of productions: 4
Enter the 1 production: S
->iCtS
Enter the 2 production: S
->iCtSeS
Enter the 3 production: S
->a
Enter the 4 production: C
->b

**********************************
 AFTER LEFT FACTORING
**********************************
Production 1 is: S->iCtSR

Production 2 is: R->^

Production 3 is: R->eS

Production 3 is: S->a

Production 4 is: C->b
```

RESULT

Thus, the program to eliminate left factoring has been executed and verified successfully.

Ex. No. 8 **Date:**

TOP-DOWN PARSING

AIM

To write a C program to implement top-down using arrays.

ALGORITHM

Step1. Start the program

Step2. A structure is created and the variables are declared.

Step3. Define the main function.

Step4. Declare the array for states and inpstr.

Step5. Define the readtt function to initialize the number of states ,number of inputs,and transition from one state to other.

Step6. Get the no. of states and no. of inputs from the user.

Step7. Start the for loop and continue till it satisfies the condition.

Step8. Print the states according to the condition.

Step9. Generate the initial and final states of the string.

Step10. Define the dfasimul function to simulate it step 10-16.

Step11. Initialize the variable start to '0' and begin for loop.

Step12. Check whether start is equal to '-1', then print "DFA Halted".

Step13. Get the string from the user and save in 'inpstr'.

Step14. Read the string and until the end-of-string reached do step 15.

Step15. Go from one state to other based on the transition entered and update start by the final state.

Step16. Print "string is accepted" if a state is equal to nstates-1.

Step17. Else print "String is not accepted".

Step18. Display the output.

Step19. Stop the Program.

PROGRAM

```
#include<iostream.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>

void error()
```

```
{
        cout<<"Not accepted";
        getch();
        exit(0);
}
int main()
{
        clrscr();
         cout<<"Enter the string in single letter such as a+a*a \n";
        char string[10],stack[10];
        int i=0,k,top=0;
        cout<<"Enter the Expression :";
        cin>>string;
        int j=strlen(string);
        string[j]='$';
        string[++j]='\0';
        cout<<string<<endl;
        stack[top]='E';
        cout<<"$"<<stack<<"\t";
        for(k=i;k<j;k++)
                cout<<string[k];
        cout<<endl;
        top++;
        while(top!=0)
        {
```

```
if(stack[top-1]==string[i])
{
        i++;
        top--;
        stack[top]='\0';
}
else if(stack[top-1]=='E')
{
        if(string[i]=='a')
        {
                top--;
                stack[top]='D';
                stack[++top]='T';
                top++;
                stack[top]='\0';
        }
        else if(string[i]=='(')
        {
                top--;
                stack[top]='D';
                stack[++top]='T';
                top++;
                stack[top]='\0';
        }
        else error();
}
```

```
else if(stack[top-1]=='D')
{
        if(string[i]=='+')
        {
                top--;
                stack[top]='D';
                stack[++top]='T';
                stack[++top]='+';
                top++;
                stack[top]='\0';
        }
        else if(string[i]==')')
        {
                top--;
                stack[top]='\0';
        }
        else if(string[i]=='$')
        {
                top--;
                stack[top]='\0';
        }
        else error();
}
else if(stack[top-1]=='T')
{
        if(string[i]=='a')
```

```
            {
                top--;
                stack[top]='s';
                stack[++top]='F';
                top++;
                stack[top]='\0';
            }
            else if(string[i]=='(')
            {
                top--;
                stack[top]='s';
                stack[++top]='F';
                top++;
                stack[top]='\0';
            }
            else error();
        }
        else if(stack[top-1]=='s')
        {
            if(string[i]=='+')
            {
                top--;
                stack[top]='\0';
            }
            else if(string[i]=='*')
            {
```

```
                top--;
                stack[top]='s';
                stack[++top]='F';
                stack[++top]='*';
                top++;
                stack[top]='\0';
        }
        else if(string[i]==')')
        {
                top--;
                stack[top]='\0';
        }
        else if(string[i]=='$')
        {
                top--;
                stack[top]='\0';
        }
        else error;
}
else if(stack[top-1]=='F')
{
        if(string[i]=='a')
        {
                top--;
                stack[top]='a';
                top++;
```

```
                    stack[top]='\0';
                }
                else if(string[i]=='(')
                {
                    top--;
                    stack[top]=')';
                    stack[++top]='E';
                    stack[++top]='(';
                    top++;
                    stack[top]='\0';
                }
                else error();
            }
            else error();
            cout<<"$"<<stack<<"\t";
            for(k=i;k<j;k++)
                cout<<string[k];
            cout<<endl;
        }
        cout<<"accepted";
        getch();
        return 0;
}
```

OUTPUT

RESULT

Thus, the C program to implement top down parsing using arrays has been executed and verified successfully.

Ex. No. 9 **Date:**

SHIFT REDUCE PARSING

AIM

To write a C program to perform the shift reduce parsing.

ALGORITHM

Step1. Start the program

Step2. Define the main function.

Step3. Declare array for string and stack and other necessary variables.

Step4. Get the expression from the user and store it as string.

Step5. Append $ to the end of the string.

Step6. Store $ into the stack.

Step7. Print three columns as 'Stack', 'String' and 'Action' for the respective actions.

Step8. Use for loop from i as 0 till string length and check the string.

Step9. If string has some operator or id, push it to the stack.

Step10. Mark this action as 'Shift'.

Step11. Print the stack, string and action values.

Step12. If stack contains some production on shifting, reduce it.

Step13. Mark this action as 'Reduce'.

Step14. Print the stack, string and action values.

Step15. Repeat steps 9 to 14 again and again till the for loop is valid.

Step16. Now check the string and the stack.

Step17. If the string contains only $ and the stack has only $E within it, then print that the 'given string is valid'.

Step18. Else print that the 'given string is invalid'.

Step19. Stop the Program.

PROGRAM

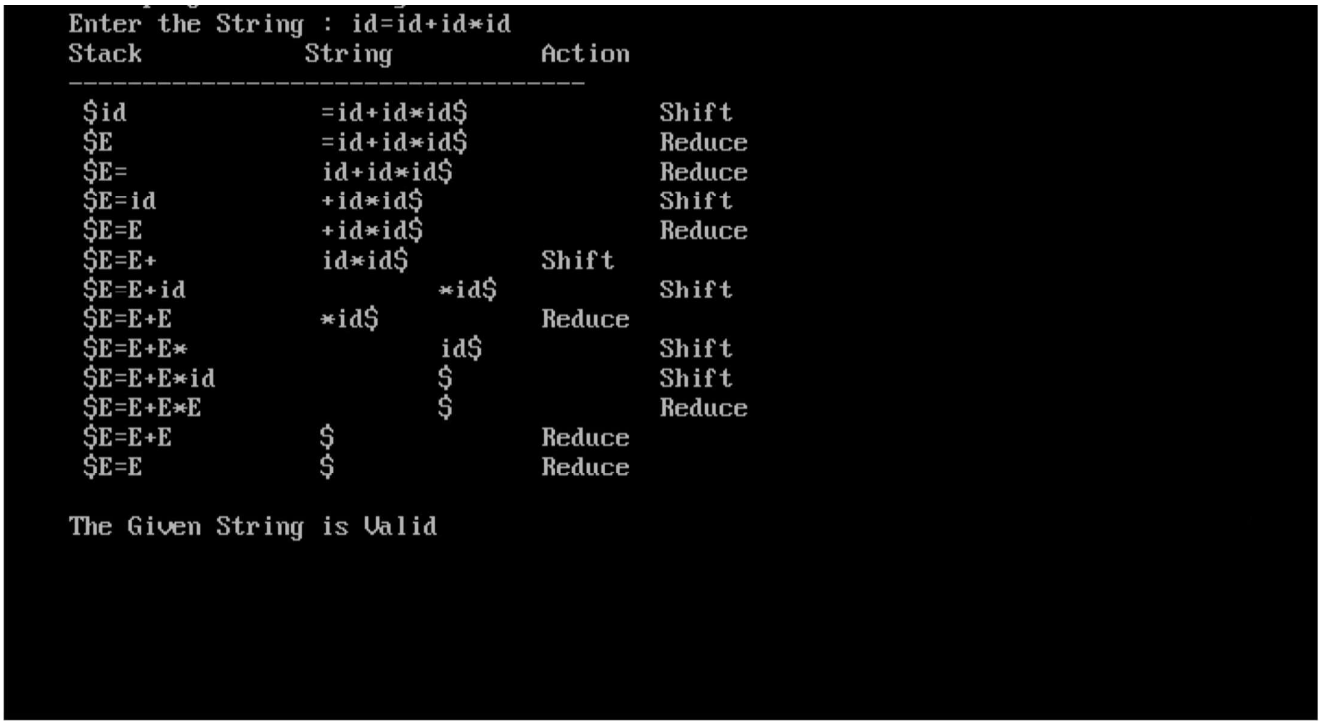
```
#include<stdio.h>

#include<string.h>

#include<conio.h>

void main()

{
```

```
char str[25],stk[25];
int i,j,t=0,l,r;
clrscr();
printf("Enter the String : ");
scanf("%s",&str);
l=strlen(str);
str[l]='$';
stk[t]='$';
printf("Stack\t\tString\t\tAction\n---------------------------------\n ");
for(i=0;i<l;i++)
{

        if(str[i]=='i')
        {
                t++;
                stk[t]=str[i];
                stk[t+1]=str[i+1];
                for(j=0;j<=t+1;j++)
                        printf("%c",stk[j]);
                printf("\t\t ");
                for(j=i+2;j<=l;j++)
                        printf("%c",str[j]);
                printf("\t\tShift");
                printf("\n ");
                stk[t]='E';
                i++;
```

```
        }
        else
        {
                t++;
                stk[t]=str[i];
        }
        for(j=0;j<=t;j++)
                printf("%c",stk[j]);
        printf("\t\t ");
        for(j=i+1;j<=l;j++)
                printf("%c",str[j]);
        if(stk[t]=='+' || stk[t]=='*')
                printf("\t\tShift");
        else
                printf("\t\tReduce");
        printf("\n ");
}
while(t>1)
{
        if(stk[t]=='E' && (stk[t-1]=='+' || stk[t-1]=='*') && stk[t-2]=='E')
        {
                t-=2;
                for(j=0;j<=t;j++)
                        printf("%c",stk[j]);
                printf("\t\t");
                printf(" %c",str[l]);
```

```
                printf("\t\tReduce\n ");
            }
            else
                t-=2;
        }
        if(t==1 && stk[t]!='+' && stk[t]!='*')
            printf("\nThe Given String is Valid\n\n");
        else
            printf("\nThe Given String is Invalid\n\n");
        getch();
    }
```

OUTPUT



RESULT

Thus, the C Program to perform the shift reduce parsing has been executed and verified successfully.

Ex. No. 10 **Date:**

OPERATOR PRECEDENCE PARSING

AIM

To write a C program for implementing operator precedence parsing for given grammar.

ALGORITHM

Step1. Start the program

Step2. Declare all necessary header files and variables

Step3. Assign all the precedence relation for given grammar in a 2D array

Step4. Get input from user in an array

Step5. Append $ symbol at the end of the input

Step6. Create stack using array and put $ as first element

Step7. Display the contents of stack and input array

Step8. If last element of stack and input is $ then the parsing is completed

Step9. Compare top of stack and top of input use find function to get element

Step10. If the symbol is space print parser error

Step11. If the symbol is < then shift top element of input to stack.

Step12. Display the contents of stack and input array

Step13. If the symbol is > then remove all elements from stack

Step14. Display the contents of stack and input array

Step15. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

int find(char a)

{

        switch(a)

        {

                case 'a':

                        return 0;

                case '+':
```

```
                return 1;
        case '*':
                return 2;
        case '$':
                return 3;
    }
}
void display(char a[],int top1,char b[],int top2)
{
    int i;
    for(i=0;i<=top1;i++)
        printf("%c",a[i]);
    printf("\t");
    for(i=top2;i<strlen(b);i++)
        printf("%c",b[i]);
    printf("\n");
}
void main()
{
    char table[][4]= {' ','>','>','>','<','<','<','>','<','>','<','>','<','<','<',' '};
    char input[10];
    char stack[10]={'$'};
    int top_stack=0,top_input=0,i,j;
    clrscr();
    printf("operator precedence parsing for E->E+E/E*E/a\n");
```

```
printf("enter the string\n");
scanf("%s",input);
strcat(input,"$");
printf("stack\tinput\n");
display(stack,top_stack,input,top_input);
while(1)
{
	if((stack[top_stack]=='$')&&(input[top_input]=='$'))
	{
		printf("string accepted");
		break;
	}
	if(table[find(stack[top_stack])][find(input[top_input])]==' ')
	{
		printf("parse error");
		getch();
		exit(0);
	}
	if(table[find(stack[top_stack])][find(input[top_input])]=='<')
	{
		stack[++top_stack]='<';
		stack[++top_stack]=input[top_input];
		top_input++;
		display(stack,top_stack,input,top_input);
		continue;
	}
```

```
                if(table[find(stack[top_stack])][find(input[top_input])]=='>')
                {
                        stack[++top_stack]='>';
                        display(stack,top_stack,input,top_input);
                        top_stack-=3;
                        display(stack,top_stack,input,top_input);
                }
        }
        getch();
}
```

OUTPUT

```
operator precedence parsing for E->E+E/E*E/a
enter the string
a+a*a
stack   input
$       a+a*a$
$<a     +a*a$
$<a>    +a*a$
$       +a*a$
$<+     a*a$
$<+<a   *a$
$<+<a>  *a$
$<+     *a$
$<+<*   a$
$<+<*<a $
$<+<*<a>        $
$<+<*   $
$<+<*>  $
$<+     $
$<+>    $
$       $
string accepted
```

RESULT

Thus, the C program for implementing operator precedence parsing for given grammar has been executed and verified successfully.

Ex. No. 11 **Date:**

SLR PARSING TABLE CONSTRUCTION

AIM

To write a C++ program to generate a parsing table for SLR Parsing.

ALGORITHM

Step1. Start the program

Step2. Declare all necessary header files and variables.

Step3. construct SLR parsing table for all the states and fill the Action and Goto first.

Step4. Insert the initial state as 0 in the stack.

Step5. Compare with the first input symbol.

Step6. Action is taken according to general parsing table.

Step7. If action=Shift, goto step 8 else step 9

Step8. push the symbol in stack.

Step9. If action= Reduce, pop the symbol from the stack, replace by the non terminal symbol, else goto step 10.

Step10. if action=goto, push the state number in the stack.

Step11. If action= accept, it implies the parsing is successful.

Step12. Stop the Program.

PROGRAM

```
#include<bits/stdc++.h>

#define error(x) cerr<<#x<<" = "<<x<<'\n'

using namespace std;



set<char> ss;

map<char,vector<vector<char>>> mp;

bool dfs(char i, char org, char last, map<char,vector<vector<char>>> &mp){

    bool rtake = false;

    for(auto r : mp[i]){

        bool take = true;
```

```
        for(auto s : r){
            if(s == i) break;
            if(!take) break;
            if(!(s>='A'&&s<='Z')&&s!='e'){
                ss.insert(s);
                break;
            }
            else if(s == 'e'){
                if(org == i||i == last)
                ss.insert(s);
                rtake = true;
                break;
            }
            else{
                take = dfs(s,org,r[r.size()-1],mp);
                rtake |= take;
            }
        }
    }
    return rtake;
}

map<int,map<char,set<pair<deque<char>,deque<char>>>>> f;
map<int,vector<pair<int,char>>> g;

int num = -1;
```

```
void dfs2(char c, char way, int last, pair<deque<char>,deque<char>> curr){
    map<char,set<pair<deque<char>,deque<char>>>> mp2;
    int rep = -2;
    if(last != -1){
        for(auto q : g[last]){
            if(q.second == way){
                rep = q.first;
                mp2 = f[q.first];
            }
        }
    }
    mp2[c].insert(curr);
    int count = 10;
    while(count--){
        for(auto q : mp2){
            for(auto r : q.second){
                if(!r.second.empty()){
                    if(r.second.front()>='A'&&r.second.front()<='Z'){
                        for(auto s : mp[r.second.front()]){
                            deque<char> st,emp;
                            for(auto t : s) st.push_back(t);
                            mp2[r.second.front()].insert({emp,st});
                        }
                    }
                }
            }
```

```
    }
}
for(auto q : f){
    if(q.second == mp2){
        g[last].push_back({q.first,way});
        return;
    }
}
if(rep == -2){
    f[++num] = mp2;
    if(last != -1)
    g[last].push_back({num,way});
}
else{
    f[rep] = mp2;
}
int cc = num;
for(auto q : mp2){
    for(auto r : q.second){
        if(!r.second.empty()){
            r.first.push_back(r.second.front());
            r.second.pop_front();
            dfs2(q.first,r.first.back(),cc,r);
        }
    }
}
```

```
}



int main(){
    int i,j;
    ifstream fin("inputslr.txt");
    string num;
    vector<int> fs;
    vector<vector<int>> a;
    char start;
    bool flag = 0;
        cout<<"Grammar: "<<'\n';
    while(getline(fin,num)){

        if(flag == 0) start = num[0],flag = 1;
        cout<<num<<'\n';
        vector<char> temp;
        char s = num[0];
        for(i=3;i<num.size();i++){
            if(num[i] == '|'){
                mp[s].push_back(temp);
                temp.clear();
            }
            else temp.push_back(num[i]);
        }
        mp[s].push_back(temp);
```

```
}
map<char,set<char>> fmp;
for(auto q : mp){
   ss.clear();
   dfs(q.first,q.first,q.first,mp);
   for(auto g : ss) fmp[q.first].insert(g);
}

cout<<'\n';
cout<<"FIRST: "<<'\n';
for(auto q : fmp){
   string ans = "";
   ans += q.first;
   ans += " = {";
   for(char r : q.second){
      ans += r;
      ans += ',';
   }
   ans.pop_back();
   ans+="}";
   cout<<ans<<'\n';
}

map<char,set<char>> gmp;
gmp[start].insert('$');
int count = 10;
```

```
while(count--){
    for(auto q : mp){
        for(auto r : q.second){
            for(i=0;i<r.size()-1;i++){
                if(r[i]>='A'&&r[i]<='Z'){
                    if(!(r[i+1]>='A'&&r[i+1]<='Z')) gmp[r[i]].insert(r[i+1]);
                    else {
                        char temp = r[i+1];
                        int j = i+1;
                        while(temp>='A'&&temp<='Z'){
                            if(*fmp[temp].begin()=='e'){
                                for(auto g : fmp[temp]){
                                    if(g=='e') continue;
                                    gmp[r[i]].insert(g);
                                }
                                j++;
                                if(j<r.size()){
                                    temp = r[j];
                                    if(!(temp>='A'&&temp<='Z')){
                                        gmp[r[i]].insert(temp);
                                        break;
                                    }
                                }
                                else{
                                    for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
                                    break;
```

```
                        }
                    }
                    else{
                        for(auto g : fmp[temp]){
                            gmp[r[i]].insert(g);
                        }
                        break;
                    }
                }
            }
        }
        if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
            for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
        }
    }
}

cout<<'\n';
cout<<"FOLLOW: "<<'\n';
for(auto q : gmp){
    string ans = "";
    ans += q.first;
    ans += " = {";
    for(char r : q.second){
```

```
            ans += r;
            ans += ',';
        }
        ans.pop_back();
        ans+="}";
        cout<<ans<<'\n';
    }
    string temp = "";
    temp+='.';
    temp+=start;

    deque<char> emp;
    deque<char> st;
    st.push_back(start);
    dfs2('!','k',-1,{emp,st});

    cout<<"\nProductions: "<<'\n';
    int cc = 1;
    set<char> action,go;
    map<pair<char,deque<char>>,int> pos;
    for(auto q : mp){
        go.insert(q.first);
        for(auto r : q.second){
            cout<<"r"<<cc<<": ";
            string ans = "";
            ans += q.first;
```

```
            ans+="->";
            deque<char> temp;
            for(auto s : r) ans += s,temp.push_back(s);
            pos[{q.first,temp}] = cc;
            for(auto s : r){
                if(s>='A'&&s<='Z') go.insert(s);
                else action.insert(s);
            }
            cout<<ans<<'\n';
            cc++;
        }
    }

    cout<<"\nGraph: "<<'\n';
    for(auto mp2 : f){
        cout<<'\n';
        cout<<"I";
        cout<<mp2.first<<": \n";
        for(auto q : mp2.second){
            string ans = "";
            ans += q.first;
            ans += "->";
            for(auto r : q.second){
                for(auto t : r.first) ans+=t;
                ans+='.';
                for(auto t : r.second) ans+=t;
```

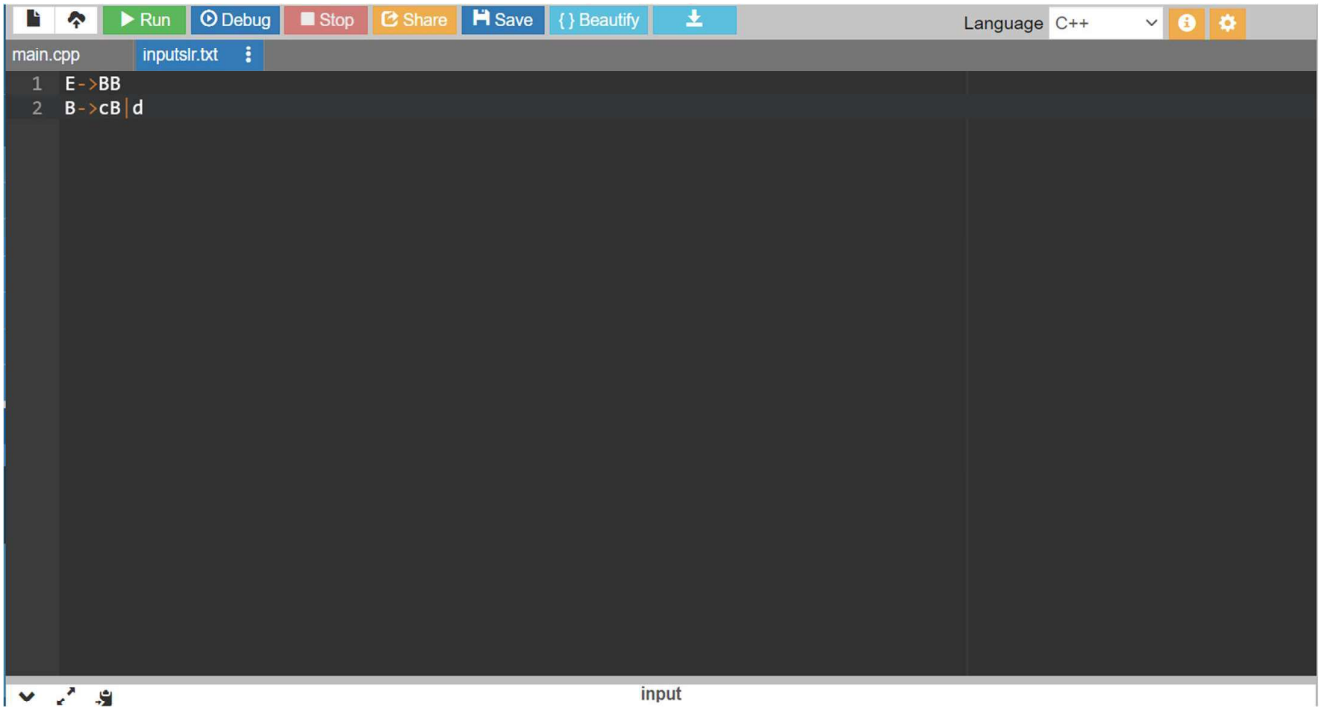
```
            ans+='|';
        }
        ans.pop_back();
        for(auto tt : ans){
            if(tt == '!') cout<<start<<'\'';
            else cout<<tt;
        }
        cout<<'\n';
    }
}
cout<<'\n';
cout<<"Edges: "<<'\n';
for(auto q : g){
    for(auto r : q.second){
        cout<<"I"<<q.first<<" -> "<<r.second<<" -> "<<"I"<<r.first<<"\n";
    }
}
action.insert('$');
cout<<"\nParsing Table:"<<'\n';
cout<<"St.\t\tAction & Goto"<<'\n';
int tot = f.size();
cout<<"  \t";
for(auto q : action) cout<<q<<'\t';
for(auto q : go) cout<<q<<'\t';
cout<<'\n';
for(i=0;i<tot;i++){
```

```
cout<<"I"<<i<<'\t';
for(auto q : action){
   if(g.count(i)){
      int flag = 0;
      for(auto r : g[i]){
         if(r.second == q){
            flag = 1;
            cout<<"S"<<r.first<<"\t";
            break;
         }
      }
      if(!flag) cout<<"-"<<'\t';
   }
   else{
      int flag = 0;
      for(auto r : f[i]){
         if(r.first == '!'){
            if(q == '$'){
            cout<<"AC\t";
            flag = 1;
            }
            else cout<<"-\t";
         }

      }
      if(!flag){
```

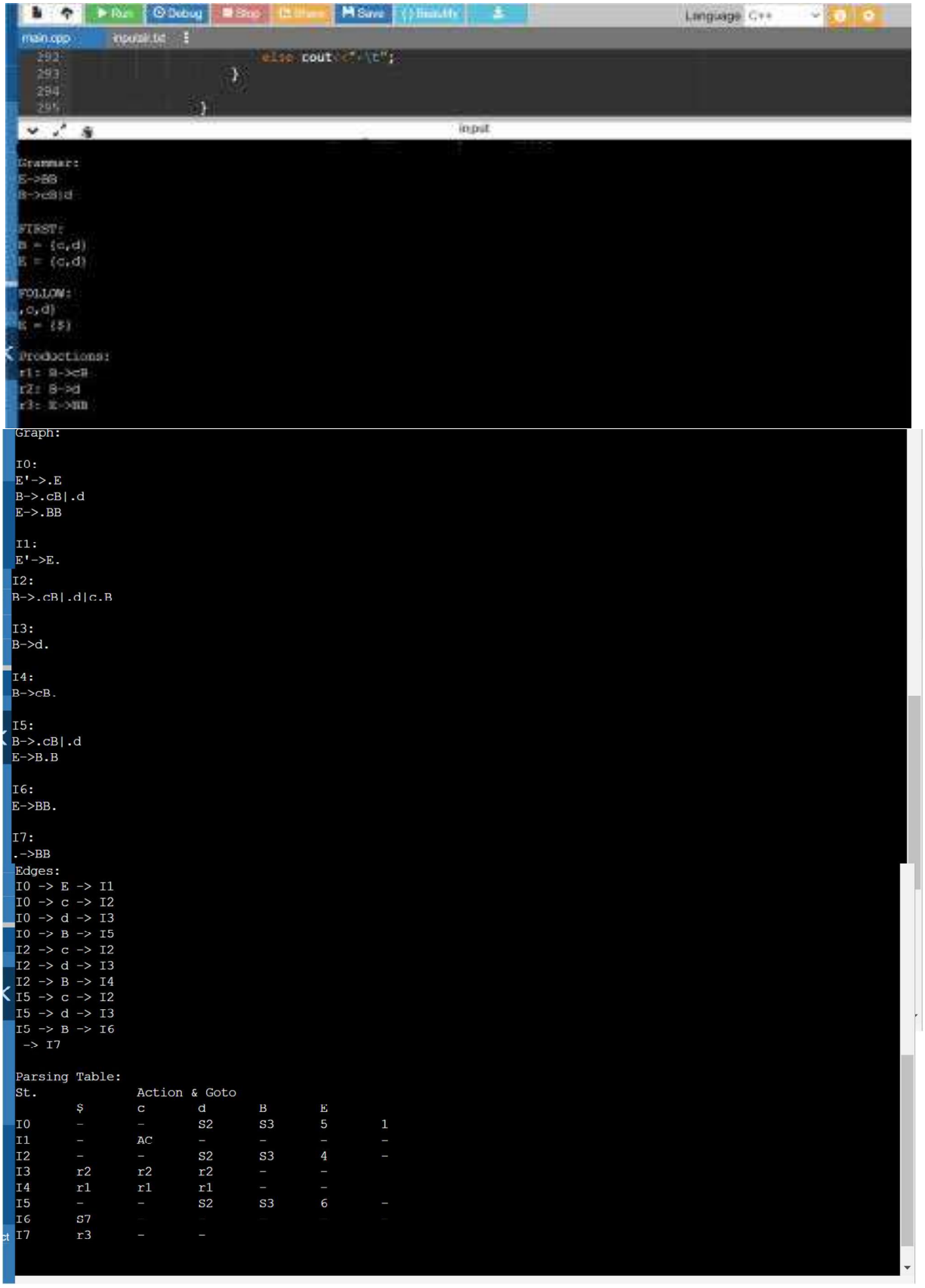
```
            for(auto r : f[i]){
                char ccc = r.first;
                deque<char> chk = (*r.second.begin()).first;
                int cou = 1;
                for(auto r : gmp[ccc]){
                    if(q == r){
                        cout<<"r"<<pos[{ccc,chk}]<<"\t";
                    }
                    cou++;
                }
            }
        }
    }
}
for(auto q : go){
    if(g.count(i)){
        int flag = 0;
        for(auto r : g[i]){
            if(r.second == q){
                flag = 1;
                cout<<r.first<<"\t";
                break;
            }
        }
        if(!flag) cout<<"-"<<'\t';
    }
```

```
            else{
                cout<<"-"<<'\t';
            }
        }
        cout<<'\n';
    }

    return 0;
}
```

INPUT: inputslr.txt



OUTPUT



```
Grammar:
E->BB
B->cB|d

FIRST:
B = {c,d}
E = {c,d}

FOLLOW:
,c,d}
E = {$}

Productions:
r1: B->cB
r2: B->d
r3: E->BB

Graph:

I0:
E'->.E
B->.cB|.d
E->.BB

I1:
E'->E.

I2:
B->.cB|.d|c.B

I3:
B->d.

I4:
B->cB.

I5:
B->.cB|.d
E->B.B

I6:
E->BB.

I7:
.->BB
Edges:
I0 -> E -> I1
I0 -> c -> I2
I0 -> d -> I3
I0 -> B -> I5
I2 -> c -> I2
I2 -> d -> I3
I2 -> B -> I4
I5 -> c -> I2
I5 -> d -> I3
I5 -> B -> I6
 -> I7

Parsing Table:
St.             Action & Goto
        $       c       d       B       E
I0      -       -       S2      S3      5       1
I1      -       AC      -       -       -       -
I2      -       -       S2      S3      4       -
I3      r2      r2      r2      -       -
I4      r1      r1      r1      -       -
I5      -       -       S2      S3      6       -
I6      S7
I7      r3      -       -
```

RESULT

Thus, the C++ program generate a parsing table for SLR Parsing has been executed and verified.

Ex. No. 12 **Date:**

INTERMEDIATE CODE GENERATOR

AIM

To write a C++ program to generate the intermediate code.

ALGORITHM

Step1. Start the program

Step2. Declare the array and other necessary variables.

Step3. Define the main function.

Step4. Get the expression from the user.

Step5. Scan the expression and put it in array 't'.

Step6. Check the position of 2, 0 & 4 characters is alphabets.

Step7. Print the expression and MOV R t the character in 2nd position.

Step8. Else print "Enter the correct expression".

Step9. Start switch case for 3rd symbol and follow the steps 10 to

Step10. If '*' then print MUL character at 4th position with R.

Step11. Print MOV R to character at position 0.

Step12. If '+' then print ADD character at 4 th position with R.

Step13. Print MOV R to character at position 0.

Step14. If '-' then print SUB character at 4th position with R.

Step15. Print MOV R to character at position 0.

Step16. If '/' then print DIV character at 4th position with R.

Step17. Print MOV R to character at position 0.

Step18. For default case, print "Invalid operator"

Step19. Stop the Program.

PROGRAM

```
#include<stdio.h>

#include<conio.h>

#include<ctype.h>

char s[20],t[20];

void main()

{
```

```
clrscr();
printf("Enter the expression: ");
scanf("%s",&t);
if(isalpha(t[2])&&isalpha(t[0])&&isalpha(t[4]))

        printf("Mov%c,R \n",t[2]);
else
        printf("Enter correct exp: ");
switch(t[3])
{
        case '*':
                        printf("MUL%c,R \n",t[4]);
                        printf("MOVR,%c \n",t[0]);
                        break;
        case '+':
                        printf("ADD%c,R \n",t[4]);
                        printf("MOVR,%c \n",t[0]);
                        break;
        case '-':
                        printf("SUB%c,R \n",t[4]);
                        printf("MOVR,%c \n",t[0]);
                        break;
        case '/':
                        printf("DIV%c,R \n",t[4]);
                        printf("MOVR,%c \n",t[0]);
                        break;
```

```
            default:
                        printf("INVALID OPERATOR");
                        break;
        }
        getch();
    }
```

OUTPUT

RESULT

Thus, the C Program For implementing Symbol Table has been executed and verified successfully.

Ex. No. 13 **Date:**

INTERMEDIATE CODE GENERATOR

AIM

To write a C++ program to generate the machine code.

ALGORITHM

Step1. Start the program

Step2. Loads each operand only once.

Step3. Performs each operation only once.

Step4. Does no stores.

Step5. Minimal number of registers having the above three properties.

- **a.** Show need L registers to produce a result with label L
- **b.** Must compute one side and no use the register containing its answer before finishing the other side.
- **c.** Apply this argument recursively

Step6. Load: LD R, x

- **a.** Desc(R) = x (removing everything else from Desc(R))
- **b.** Add R to Desc(x) (leaving alone everything else in Desc(x))
- **c.** Remove R from Desc(w) for all w ≠ x (not in 2e please check)

Step7. Store: ST x, R

- **a.** Add the memory location of x to Desc(x)

Step8. Operation: OP Rx , Ry , Rz implementing the quad OP x, y, z

- **a.** Desc(Rx) = x
- **b.** Desc(x) = Rx
- **c.** Remove Rx from Desc(w) for all w ≠ x

Step9. Copy: For x = y after processing the load (if needed)

- **a.** Add x to Desc(Ry) (note y not x)
- **b.** Desc(x) = R

Step10. Stop the Program.

PROGRAM

```
#include<stdlib.h>

#include<string.h>

#include<fstream.h>

#include<stdio.h>
```

```
#include<conio.h>
int label[20];
int no = 0;
int check_label(int k)
{
        int i;
        for (i = 0; i < no; i++)
        {
                if (k == label[i])
                        return 1;
        }
        return 0;
}
int main()
{
        clrscr();
        FILE *fp1, *fp2;
        char fname[10], op[10], ch;
        char operand1[8], operand2[8], result[8];
        int i = 0, j = 0;
        printf("\n Enter filename of the intermediate code: ");
        scanf("%s", &fname);
        fp1 = fopen(fname, "r");
        fp2 = fopen("target.txt", "w");
        if (fp1 == NULL || fp2 == NULL)
```

```
{
        printf("\n Error opening the file");
        exit(0);
}
while (!feof(fp1))
{
        fprintf(fp2, "\n");
        fscanf(fp1, "%s", op);
        i++;
        if (check_label(i))
                fprintf(fp2, "\nlabel#%d", i);
        if (strcmp(op, "print") == 0)
        {
                fscanf(fp1, "%s", result);
                fprintf(fp2, "\n\t OUT %s", result);
        }
        if (strcmp(op, "goto") == 0)
        {
                fscanf(fp1, "%s %s", operand1, operand2);
                fprintf(fp2, "\n\t JMP %s,label#%s", operand1, operand2);
                label[no++] = atoi(operand2);
        }
        if (strcmp(op, "[]=") == 0)
        {
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n\t STORE %s[%s],%s", operand1, operand2, result);
```

```
}
if (strcmp(op, "uminus") == 0)
{
        fscanf(fp1, "%s %s", operand1, result);
        fprintf(fp2, "\n\t LOAD -%s,R1", operand1);
        fprintf(fp2, "\n\t STORE R1,%s", result);
}
switch (op[0])
{
        case '*':
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD", operand1);
                fprintf(fp2, "\n \t LOAD %s,R1", operand2);
                fprintf(fp2, "\n \t MUL R1,R0");
                fprintf(fp2, "\n \t STORE R0,%s", result);
                break;
        case '+':
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
                fprintf(fp2, "\n \t LOAD %s,R1", operand2);
                fprintf(fp2, "\n \t ADD R1,R0");
                fprintf(fp2, "\n \t STORE R0,%s", result);
                break;
        case '-':
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
```

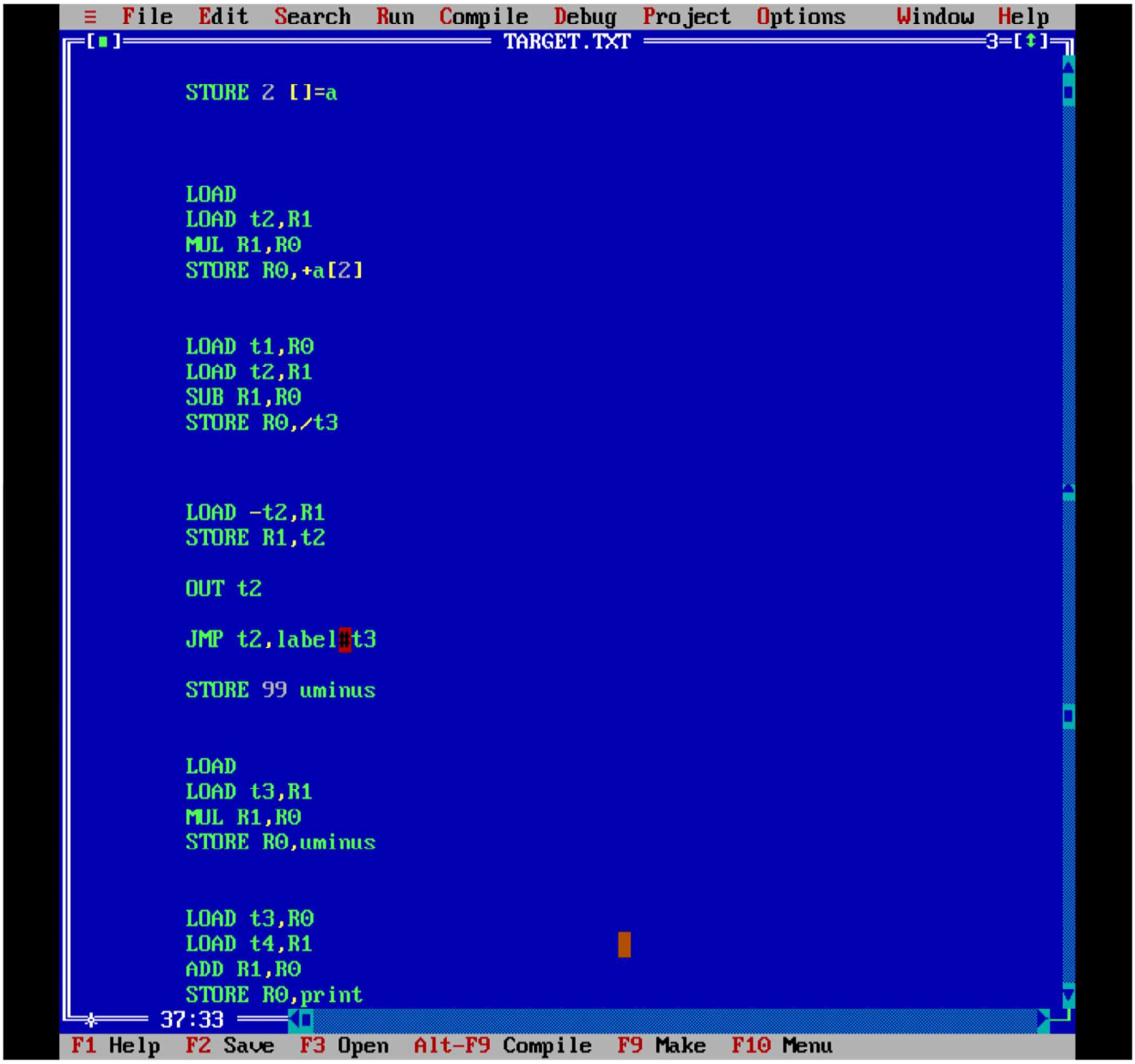
```
                fprintf(fp2, "\n \t LOAD %s,R1", operand2);
                fprintf(fp2, "\n \t SUB R1,R0");
                fprintf(fp2, "\n \t STORE R0,%s", result);
                break;
        case '/':
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
                fprintf(fp2, "\n \t LOAD %s,R1", operand2);
                fprintf(fp2, "\n \t DIV R1,R0");
                fprintf(fp2, "\n \t STORE R0,%s", result);
                break;
        case '%':
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
                fprintf(fp2, "\n \t LOAD %s,R1", operand2);
                fprintf(fp2, "\n \t DIV R1,R0");
                fprintf(fp2, "\n \t STORE R0,%s", result);
                break;
        case '=':
                fscanf(fp1, "%s %s", operand1, result);
                fprintf(fp2, "\n\t STORE %s %s", operand1, result);
                break;
        case '>':
                j++;
                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
```

```
                                fprintf(fp2, "\n\t JGT %s,label#%s", operand2, result);
                                label[no++] = atoi(result);
                                break;
                        case '<':
                                fscanf(fp1, "%s %s %s", operand1, operand2, result);
                                fprintf(fp2, "\n \t LOAD %s,R0", operand1);
                                fprintf(fp2, "\n\t JLT %s,label#%d", operand2, result);
                                label[no++] = atoi(result);
                                break;
                }
        }
        fclose(fp2);
        fclose(fp1);
        fp2 = fopen("target.txt", "r");
        if (fp2 == NULL)
        {
                printf("Error opening the file\n");
                exit(0);
        }
        do
        {
                ch = fgetc(fp2);
                printf("%c", ch);
        }while (ch != EOF);
        fclose(fp1);
        return 0;
```

```
}
```

INPUT: Input13.txt

OUTPUT: target.txt



RESULT

Thus, the C++ Program to generate the machine code has been executed and verified successfully.