DATA STRUCTURES AND ALGORITHMS

CSE2003

PROJECT REPORT

EXPRESSION SIMULATOR

Submitted by:

Rajvardhan Dixit - 16BCB0109

Slot: G1

Faculty: Prof. Gopinath M.P.

VIT® UNIVERSITY

(Estd. u/s 3 of UGC Act 1956)

VELLORE ■ CHENNAI

www.vit.ac.in

INDEX

ABSTRACT

In this project, we aim to implement logical expressions we have learnt, using the concepts of Data structures and algorithms. This course teaches us to integrate both concepts and create real world applications of a program.

In our code, we use the inherent concepts of stacks as linked lists and use this technique for expression evaluation and simulation.

The stacks are dynamically implemented i.e. using linked lists. This implies that memory is dynamically created when the code requires it to be allocated, and there is no wastage of memory space by our program.

The code takes an infix expression, converts it to postfix and displays it. It asks the user what every variable means and then displays a truth table of every possibility the expression can have, running every possible case during simulation.

It then displays a waveform of this output expression in accordance with the truth table in the terminal and using the concepts of file handling in C saves this output in a file for future use.

INTRODUCTION

The code works in two ways.

First, it takes an infix expression and converts it to postfix using stack (made using linked lists), and displays all possible cases that can exist for this expression for different values of input using a truth table. It the displays the output waveform depending on the clock time interval and number of cycles entered by the user. It also shows the success cases from the truth table, where the value of the output case is 1. It finally, saves the waveform in a file for future reference.

Second, an infix expression is input and the user is asked what each variable means. The user is then asked as to what the output expression is supposed to imply. Then again, the expression is converted to postfix, evaluated and the truth table is displayed. The waveform for the output expression and the success cases are displayed. Also, using the definitions provided by the user we show as to when the output definition is correct and will happen. Now this information i.e. the waveform is saved into a file for future reference.

PROBLEM STATEMENT

The program works upon the variable conditions and evaluates them to determine the success cases required to achieve the final output.

All possible permutations of the logical expressions are considered and a truth table is designed.

It then moves on with assessing infix expression, converting it to postfix and then displaying its waveform and success cases.

ALGORITHM

START

1. Include all header files and standard library functions.
2. Define MAX value as 100.
3. Initialize all required variables for the code to be used with appropriate data types.
4. Create a structure node, which acts as a node for the linked list in dynamic implementation of stacks and stores data and has pointer next.
5. Initialize a pointer l1 to NULL.
6. Create function insert to push the element into the stack (linked list) and change the value of the top pointer respectively.
7. Create function delet, to remove the element or pop the element from the stack in the linked list.
8. Create function display, to display all the elements of the linked list with the help of its pointer and display all the elements present in the stack.
9. Create a function prior, which takes in the symbol input and evaluates it and assigns it a priority to be ordered and removed or not from the stack.
10. Create a function cnvrt, to take the input string of the expression and convert it to a form which can be used for evaluation of the given string i.e. postfix expression. This conversion also takes place using the concept of dynamic implementation of stacks using linked list.
11. Create a function count0, to check the uniqueness of a variable in the expression to be evaluated to give a more minimalized output.
12. Create a function no_of_variables, to be able to count the number of unique variable in the expression checked already from the previous function in step 12.
13. Create a function ttable, to initialize a 2 dimensional array to be able to save the input conditions and the output cases in the array to form the truth table.
14. Create a function ttabledisplay, which displays the truth table created in the function in step 14 and displays it with appropriate labeling and bordering and aesthetics.

15. Create a function eval, which takes the previously converted input expression to postfix expression and now evaluates it using the concept of stacks in dynamic memory implementation using linked lists.
16. Create a function graphdisplay, which take the values in the truth table and depending on the frequency input by the users. Now depending on the value 1 or 0 the graph show – or _ respectively an gives and equivalent waveform response for the given expression and hence proves the validity of the given expression.
17. Create a function graphdisplay_out, which uses the concepts of file handling in C to write the output and waveform into a text file to save the result and use it for future implementations.
18. Next create a fn. success_cases, which takes the data of the truth table and checks all the outputs where the output is a success case i.e. 1. It then display all the parameters of such cases i.e. variable input cases.
19. Also another function meaning_of_variable is also created which asks the user what every variable in their input expression means and saaves this meaning of the variables.
20. Later another function final_saying, uses these meanings input by the user to display what the final answer of the input expression would be based on the meaning and the truth cases of evaluation.
21. Finally, the main() function, creates all the additional variable and uses printf() to create the looks of the entire program and using concepts of data abstraction displays only the required information to the user and acts as a driver function to control all the functions created above for their respective job and display all the required details whenever necessary.
22. The main function hence alone acts as the essence of the code and does the complete evaluation of the functions and relevant orderingof function to give a meaningful output.

STOP.

CONCLUSION

Therefore this project tires to encompass and assimilate the integrated concepts of computer science in terms of logical evaluation and C programming. The program can serve as a digital calculator and works on problems entered by users.

This idea can be seen as a prototype that can be developed further into software that can design and analyze circuits and expressions.

REFERNCES

www.stackoverflow.com

www.geeksforgeeks.com

www.sanfoundry.com

CODE:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include <ctype.h>
#include<math.h>
#include<windows.h>

#define MAX 100

int c=0;
char ip[MAX];
char op[MAX];
char var_array[100];
char var[100][100];
char fx[100];
char meaning[10][100];
char meaning_fx[100];
int cycles,time;

//LINKED LIST

//Create a structure
struct node
{
   char data;
   struct node *next;
};

struct node *l1=NULL;
//Insert character into the linked list
void insert(char x)
{
    if(l1==NULL)
    {
        l1=(struct node *)malloc(sizeof(struct node));
        l1->data=x;
        l1->next=NULL;
    }
    else
    {
        struct node *p;
        p=(struct node *)malloc(sizeof(struct node));
        p->data=x;
        p->next=l1;
        l1=p;
    }
}
//Delete character from linked list
char delet()
{
    char c;
    struct node *p;
    if (l1==NULL)
    {
        printf("");
    }
    else
    {
        c=l1->data;
        p=l1->next;
        free (l1);
        l1=p;
    }
    return c;
}
```

```
//Displays Elements of Linked List


void display(struct node *start)
{
    {
        struct node *p;
        p=start;
        if(p==NULL)
            printf("");
        else
        {
            while(p!=NULL)
            {
                printf("%c->",p->data);
                p=p->next;
            }
            printf("");
        }
    }
}
//Checks the priority of input variables
int prior(char s, char c)
{
    if ((c=='^' && s=='+') || s=='-' || s=='@' || s=='*')
        return 1;
    else if(c=='*' || c=='@')
    {
        if(s=='+' || s=='-')
            return 1;
        else
            return 0;
    }
    else if(c=='+' || c=='-')
        return 0;
  return -1;
}
//convert Expression to Calculable form
void cnvrt(char s[], int n)
{
    int i,j;
    for(i=0,j=0;i<n;i++)
    {

        if ((s[i]>='0' && s[i]<='9') || (s[i]>='a' && s[i]<='z') || (s[i]>='A' &&
s[i]<='Z'))
        {
            op[j]=s[i];
            j++;
        }
        else if(s[i]=='(')
        {
            insert(s[i]);
        }
        else if (s[i]=='+' || s[i]=='@' || s[i]=='-' || s[i]=='*' || s[i]=='^')
        {

            if( l1==NULL)
                insert(s[i]);
            else if(l1->data=='(')
                insert(s[i]);
            else if(prior(l1->data, s[i] )!=1)
                insert(s[i]);
            else
            {
                op[j]=delet();
                j++;
            insert(s[i]);
         }
      }
      else if(s[i]==')')
      {
         while(l1!=NULL && l1->data!='(')
         {
            op[j]=delet();
            j++;
         }
         delet();
      }
   }
   while(l1!=NULL)
   {
      op[j]=delet();
      j++;
   }
}
//Checks uniqueness of variables in an expression
int count0( char * ip,int i)
{
        int p;
        for (p=0;p<i;p++)
        {
                if(ip[i]==ip[p])
                return 0;

        }

        return 1;
}
//Number of unique variables
int no_of_variables(char * ip,char * var_array)
```

```
{
        int i,count=0;
        for(i=0;ip[i]!='\0';i++)
        {
                 if((((ip[i]>='A'&& ip[i]<='Z')||(ip[i]>='a'&&
ip[i]<='z'))&&(count0(ip,i))))
                 {
                          var_array[count]=ip[i];
                          count++;
                 }
        }
        return count;
}
//initialise ttable array

int *  ttable(int varc)
{
    int t=pow(2,varc);
    char bit='1';
    int j=0;
    int i=0;
  //for(j=0;j<pow(2,varc);j++)
    for(i=0;i<varc;i++)
    {
           for (j=0;j<t;j++)
           {
              if(j%(int)pow(2,varc-i-1)==0)
              {
                 bit=(bit=='1'?'0':'1');
              }
              var[i][j]=bit;
           }
    }
}
```

```
//Displays the truthtable
int *  ttabledisplay(int varc)
{
    int t=pow(2,varc);
    int i,j;
  for(j=0;j<pow(2,varc);j++)
    {
       for (i=0;i<varc;i++)
          {
             printf("%c\t",var[i][j]);
          }
   printf("\t%c",fx[j]);
   printf("\n");
    }
   printf("\n\n");
}
//Evaluation of Function
int eval(char * op,int variables,char * var_array)    // Evaluates the expression op
and stores results in Fx array
{
        char ch;
        int c,l,j,a,b,i=0,k=0;
        char temparray[50],op1,op2;
        for(c=0;c<pow(2,variables);c++)
        {
                strcpy(temparray,op);
                for(a=0;a<variables;a++)
        {
                for(b=0;temparray[b]!='\0';b++)
                {
                        if(var_array[a]==temparray[b])
                        {
                                temparray[b]=var[a][c];
                        }
		}
	}

	for(l=1;temparray[l]!='\0';l++)
	{
		if(temparray[l]=='\'')
			if(temparray[l-1]=='0')
				temparray[l-1]='1';
			else
				temparray[l-1]='0';
	}
	i=0;
	while( (ch=temparray[i++]) != '\0')
	{
		if(ch=='0'||ch=='1')
		{	insert(ch);
		}
		else if(ch=='\'')
			continue;
		else
		{
			op2=delet();
			op1=delet();
				switch(ch)
				{
					case '+':
		    if(op1=='1' || op2=='1' ) insert('1');
		    else insert('0');
		    break;
		 case '^':
		    if(op1=='0') insert('1');
		    else if(op1=='1') insert('0');
		    break;
					case '*':
```

```
                        if( op1=='0' || op2=='0' ) insert('0');
                        else insert('1');
                        break;
                     case '@':
                        if(op1=='0' && op2=='0') insert('0');
                        else if(op1=='0'&& op2=='1') insert('1');
                        else if(op1=='1'&& op2=='0') insert('1');
                        else if(op1=='1'&& op2=='1') insert('0');
                        break;
                    case '\':
                            break;

                            }
                    }
            }
            fx[c] = l1->data;
            }
    }
    //Displaying the simulation graph
    void graphdisplay(int varc,char * var_array)
    {
            int i,j,o,n,d,freq;
            printf("Enter Clock Time for Simulation Graph (in multiples of 50 ms) : ");
            scanf("%d",&time);
            printf("Enter the number of cycles : ");
            scanf("%d",&cycles);

            int t=pow(2,varc);
            printf("\n");
            for(i=0;i<varc;i++)
        {
           printf("%c\t",var_array[i]);

           for(t=0;t<cycles;t++)
```

```
          {

             for(j=0;j<pow(2,varc);j++)
             {

                 for(o=0;o<time/50;o++)
                 {
                     if(var[i][j]=='1')
                         printf("-");
                     else
                         printf("_");
                 }
             }
          }
      printf("\n\n");
           }
           printf("Fx\t");
           for(t=0;t<cycles;t++)
           {

          for(j=0;j<pow(2,varc);j++)
          {
             for(n=0;n<time/50;n++)
             {
                          if(fx[j]=='1')
                                 printf("-");
                                 else
                                 printf("_");
             }
          }
           }

           printf("\n\nTime\t");
```

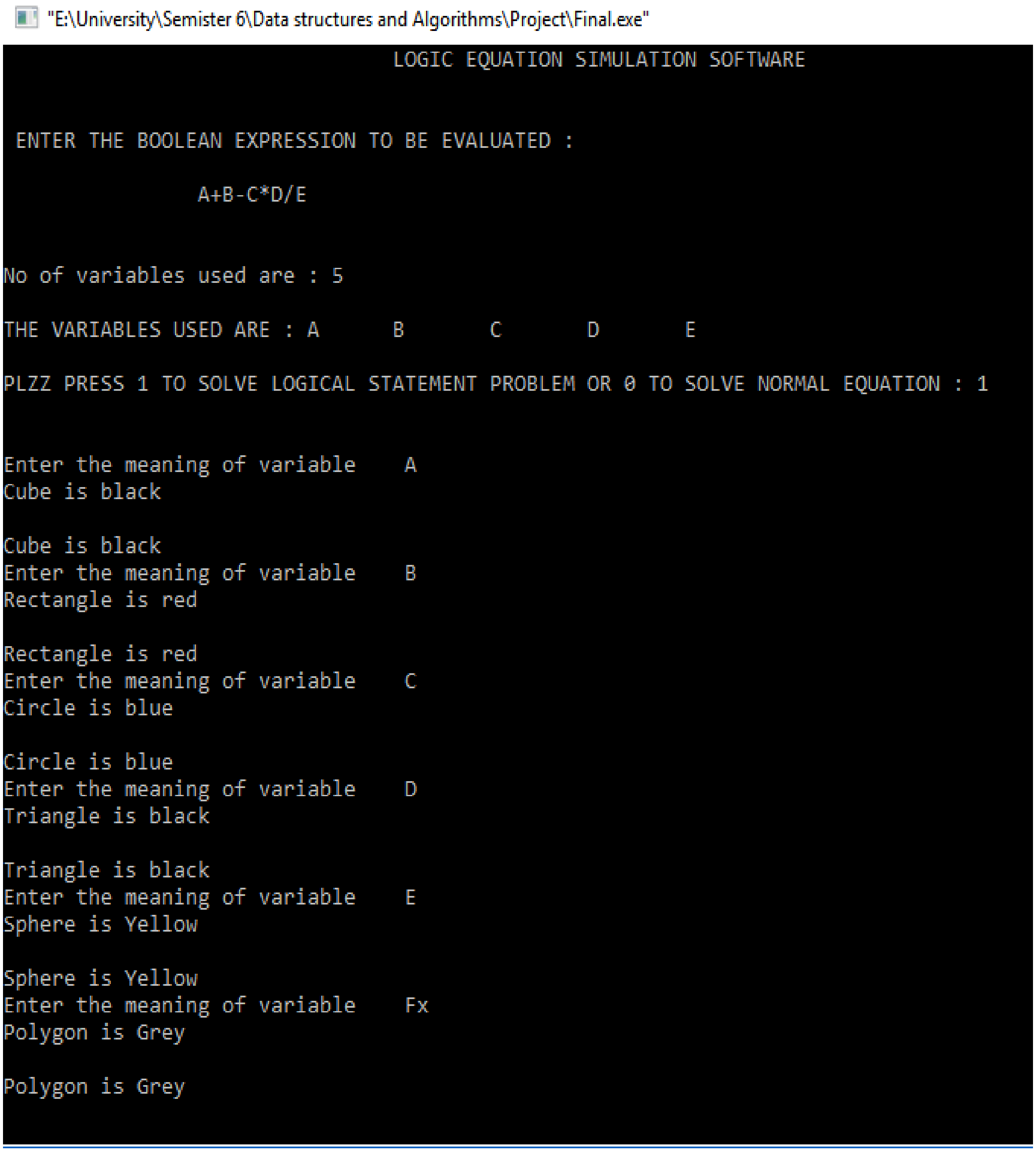
```
        for(d=0;d<time*cycles;d=d+time)
    {
        printf("%d",d);
        for(i=0;i<pow(2,varc);i++)
        {
            for(n=0;n<(time/50);n++)
            {
                printf(" ");
            }
        }

    }
        /*for(d=0;d<=time*cycles*pow(2,varc);d=d+time*cycles)
        {
                printf("%d\t",d);
        }*/

}
//Makes a Txt file of the simulation graph
void graphdisplay_out(int varc,char * var_array)
{


        int i,j,o,n,d;
        int t=pow(2,varc);
        FILE *fp;
        fp=fopen("Graph.txt","w");                              //File handeling
Functions//


        fprintf(fp,"\n");
        fprintf(fp,"\nGRAPH FOR THE LOGIC EXPRESSION : ");
        for(i=0;i<strlen(ip);i++)
    {
        fprintf(fp,"%c ",ip[i]);
    }
```

```
fprintf(fp,"\n\n");
    for(i=0;i<varc;i++)
{
   fprintf(fp,"%c\t",var_array[i]);

   for(t=0;t<cycles;t++)
   {

      for(j=0;j<pow(2,varc);j++)
      {

         for(o=0;o<time/50;o++)
         {
            if(var[i][j]=='1')
               fprintf(fp,"-");
            else
               fprintf(fp,"_");
         }
      }
   }
fprintf(fp,"\n\n");
    }
    fprintf(fp,"Fx\t");
    for(t=0;t<cycles;t++)
    {

   for(j=0;j<pow(2,varc);j++)
   {
      for(n=0;n<time/50;n++)
      {
                    if(fx[j]=='1')
                            fprintf(fp,"-");
                            else
                            fprintf(fp,"_");
            }
        }
         }

         fprintf(fp,"\n\nTime\t");

         for(d=0;d<time*cycles;d=d+time)
    {
        fprintf(fp,"%d",d);
        for(i=0;i<pow(2,varc);i++)
        {
            for(n=0;n<(time/50);n++)
            {
                fprintf(fp," ");
            }
        }
    }
         /*for(d=0;d<=time*cycles*pow(2,varc);d=d+time*cycles)
         {
                  fprintf(fp,"%d\t",d);
         }*/
         fclose(fp);
         printf("\n\nTHE GRAPH FILE HAS BEEN SAVED TO YOUR LOCAL
DIRECTORY...PLZZ SAVE IT FOR USING NEXT TIME \n\n");

}
//Prints the success cases of the given equation
void success_cases(int varc)
{
    printf("SUCCESS CASES \n\n");
    int t=pow(2,varc);
    int i,j;
    for(i=0;i<varc;i++)
    {
        printf("%c\t",var_array[i]);
         }
         printf("\n");
    for(j=0;j<pow(2,varc);j++)
     {
         if(fx[j]=='0')
         {
             continue;
         }
         else
         {
             for (i=0;i<varc;i++)
            {
                printf("%c\t",var[i][j]);
            }
         }
    printf("\n");
     }
    printf("\n\n");
}
//Takes input of the meaning of variables and fx
void meaning_of_variables(char *var_array,int variables)
{
    int i=0;
    char mean[100];
    fgets(mean, 100, stdin);

    for(i=0;i<variables;i++)
    {
        printf("Enter the meaning of variable \t %c \n",var_array[i]);
        gets(mean);
        strcpy(meaning[i],mean);
        printf("\n%s\n",meaning[i]);
    }
    printf("Enter the meaning of variable \t Fx \n");
    gets(mean);
    strcpy(meaning_fx,mean);
    printf("\n%s\n",meaning_fx);
}
//Prints out the final saying of the function as per meaning
void final_saying(char * ip,int n,int variables)
{
    int i,j;
    printf("\t\tTHE FINAL SAYING IS : \n\n");
    printf("%s if ",meaning_fx);
    for(i=0;i<n;i++)
    {
        if(ip[i]=='+')
        {
            printf("Or ");
            continue;
        }
        else if(ip[i]=='*')
        {
            printf("And ");
            continue;
        }
        else
        {
            for(j=0;j<variables;j++)
            {
                if(ip[i]==var_array[j])
                {
                    printf("%s ",meaning[j]);
                }
            }
        }
    }
    printf("\n");
}

void fordelay(int j)
{   int i,k;
    for(i=0;i<j;i++)
        k=i;
}
//MAIN function
int main()
{
    int i,n,meaning_c;
    while(1)
    {
        printf("\t\t\t\tLOGIC EQUATION SIMULATION SOFTWARE\n\n\n");
        printf(" ENTER THE BOOLEAN EXPRESSION TO BE EVALUATED :
\n\n\t\t");
    scanf("%s",ip);
    n=strlen(ip);

    cnvrt(ip,n);

    int variables=no_of_variables(ip,var_array);

    printf("\n\nNo of variables used are : %d\n",variables);
    printf("\nTHE VARIABLES USED ARE : ");
    for(i=0;i<variables;i++)
    {
        printf("%c\t",var_array[i]);
    }
    printf("\n\nPLZZ PRESS 1 TO SOLVE LOGICAL STATEMENT
PROBLEM OR 0 TO SOLVE NORMAL EQUATION : ");
    scanf("%d",&meaning_c);
    if(meaning_c==1)
```

```
   {
      printf("\n\n");
      meaning_of_variables(var_array,variables);
   }

   ttable(variables);

   printf("\n\nTHE FUNCTION USED BY COMPUTER FOR EVALUATION
IS : ");
   for(i=0;i<n;i++)
   {
      printf("%c",op[i]);
   }
   printf("\n\n");

   eval(op,variables,var_array);   // evaluates the logic expression
   printf("\n\nTRUTH TABLE\n\n");
       for(i=0;i<variables;i++)
       {
      printf("%c\t",var_array[i]);
       }
       printf(":\tFx\n");

       ttabledisplay(variables);

   printf("\n\n");

   graphdisplay(variables,var_array);

   graphdisplay_out(variables,var_array);

   printf("\n\n");

   success_cases(variables);
```

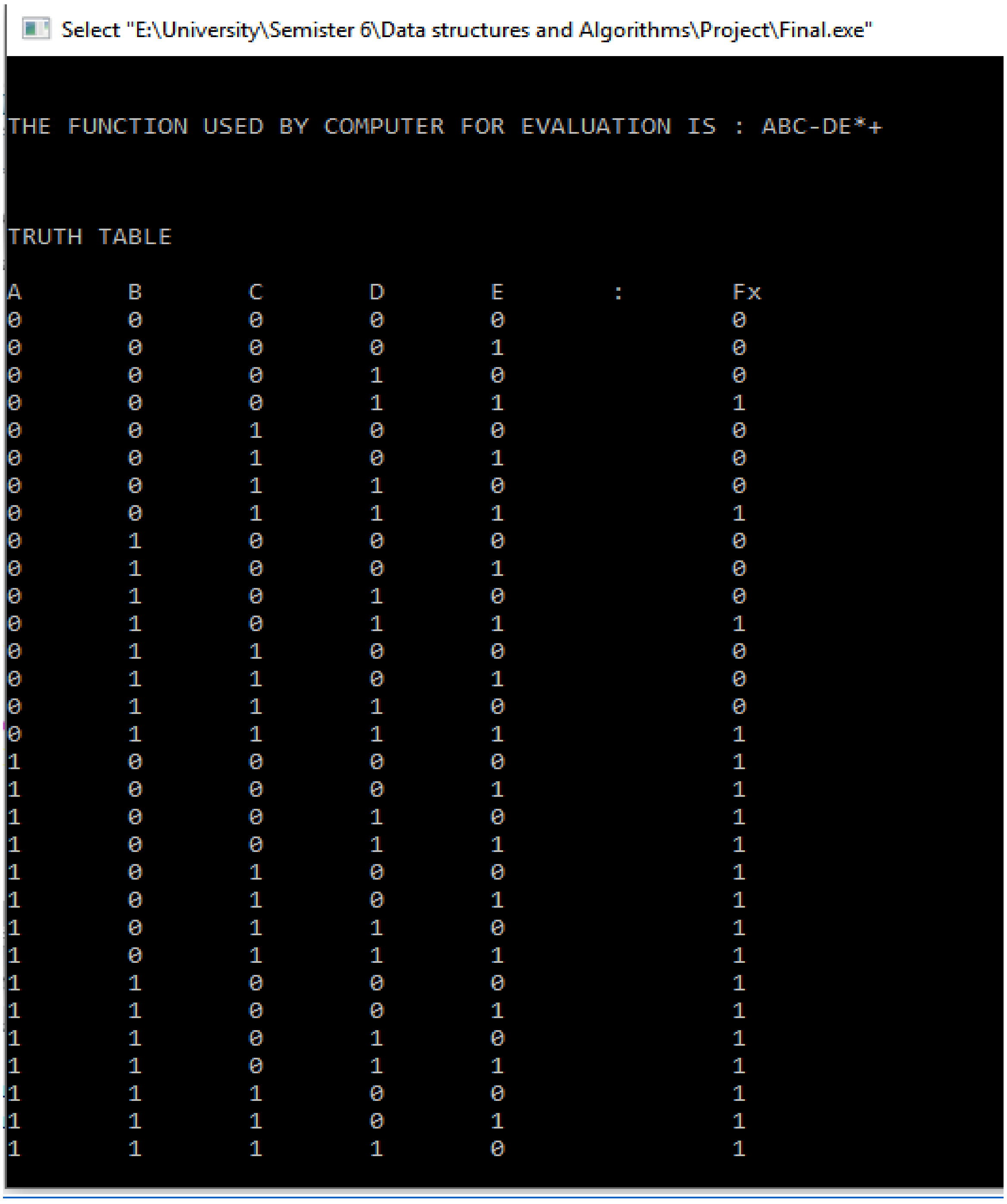
```
    if(meaning_c==1)
    final_saying(ip,n,variables);
    printf("\n\nTO EXIT PRESS 0 OR 1 TO CONTINUE :\n");
    int exit_status;
    scanf("%d",&exit_status);
    if(exit_status==0)
        exit(0);
    printf("PLEASE SAVE YOUR DATA BEFORE PROCEEDING
        AHEAD\n\n");
    l1=NULL;
    system("cls");
    }
    return 0;
}
```

FINAL EXECUTION



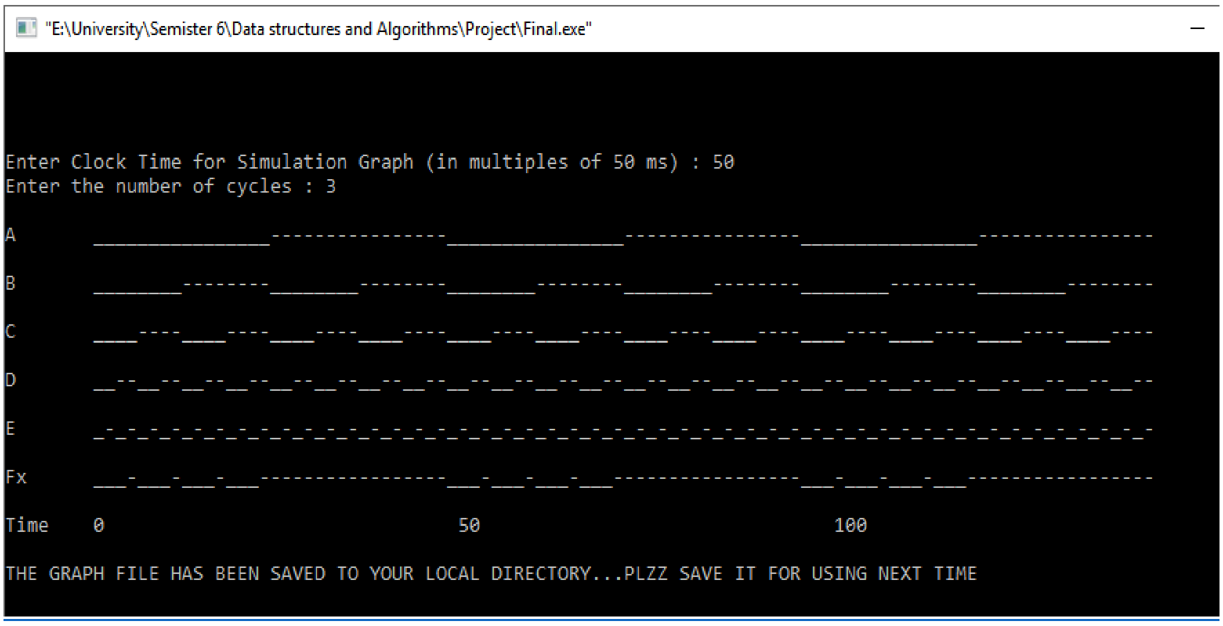
"E:\University\Semister 6\Data structures and Algorithms\Project\Final.exe"

```
                                LOGIC EQUATION SIMULATION SOFTWARE


 ENTER THE BOOLEAN EXPRESSION TO BE EVALUATED :

                A+B-C*D/E


No of variables used are : 5

THE VARIABLES USED ARE : A      B       C       D       E

PLZZ PRESS 1 TO SOLVE LOGICAL STATEMENT PROBLEM OR 0 TO SOLVE NORMAL EQUATION : 1


Enter the meaning of variable    A
Cube is black

Cube is black
Enter the meaning of variable    B
Rectangle is red

Rectangle is red
Enter the meaning of variable    C
Circle is blue

Circle is blue
Enter the meaning of variable    D
Triangle is black

Triangle is black
Enter the meaning of variable    E
Sphere is Yellow

Sphere is Yellow
Enter the meaning of variable    Fx
Polygon is Grey

Polygon is Grey
```

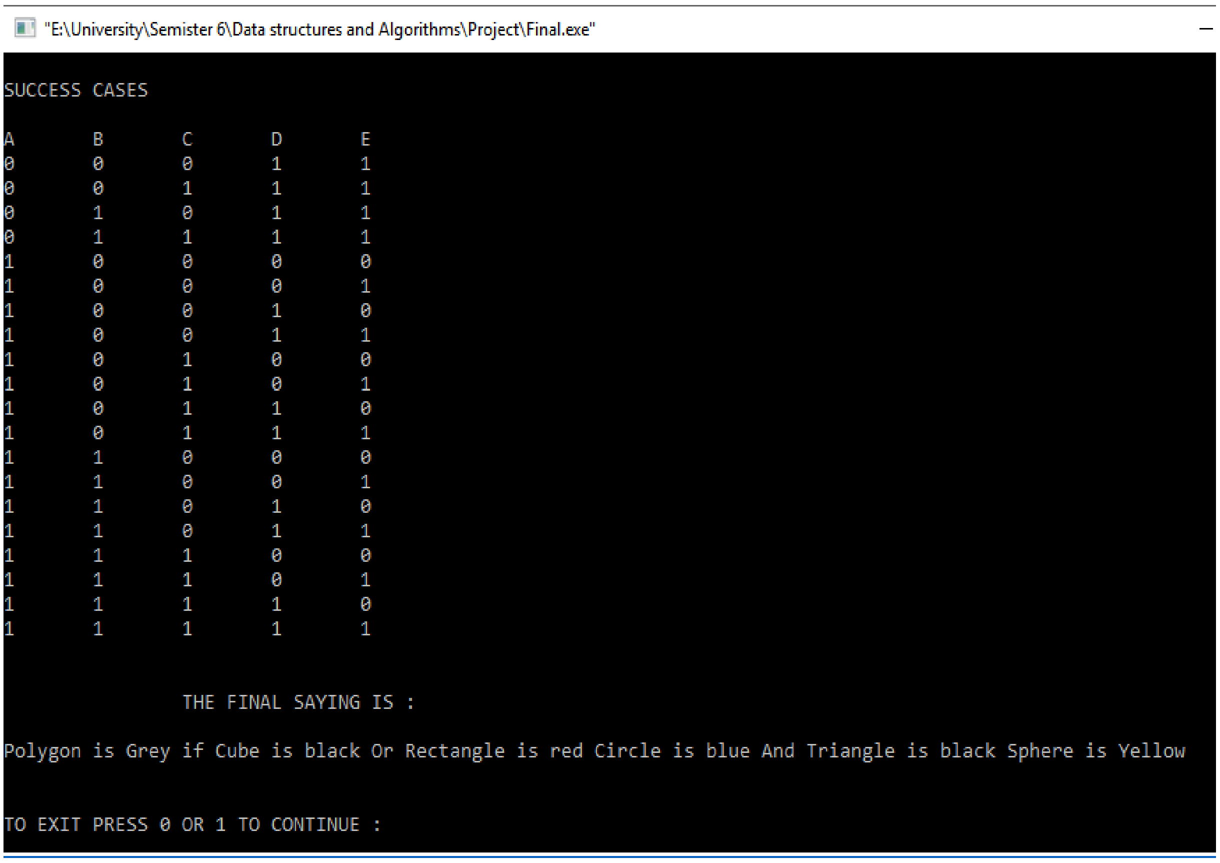


Select "E:\University\Semister 6\Data structures and Algorithms\Project\Final.exe"

```
THE FUNCTION USED BY COMPUTER FOR EVALUATION IS : ABC-DE*+


TRUTH TABLE

A        B        C        D        E        :        Fx
0        0        0        0        0                 0
0        0        0        0        1                 0
0        0        0        1        0                 0
0        0        0        1        1                 1
0        0        1        0        0                 0
0        0        1        0        1                 0
0        0        1        1        0                 0
0        0        1        1        1                 1
0        1        0        0        0                 0
0        1        0        0        1                 0
0        1        0        1        0                 0
0        1        0        1        1                 1
0        1        1        0        0                 0
0        1        1        0        1                 0
0        1        1        1        0                 0
0        1        1        1        1                 1
1        0        0        0        0                 1
1        0        0        0        1                 1
1        0        0        1        0                 1
1        0        0        1        1                 1
1        0        1        0        0                 1
1        0        1        0        1                 1
1        0        1        1        0                 1
1        0        1        1        1                 1
1        1        0        0        0                 1
1        1        0        0        1                 1
1        1        0        1        0                 1
1        1        0        1        1                 1
1        1        1        0        0                 1
1        1        1        0        1                 1
1        1        1        1        0                 1
```

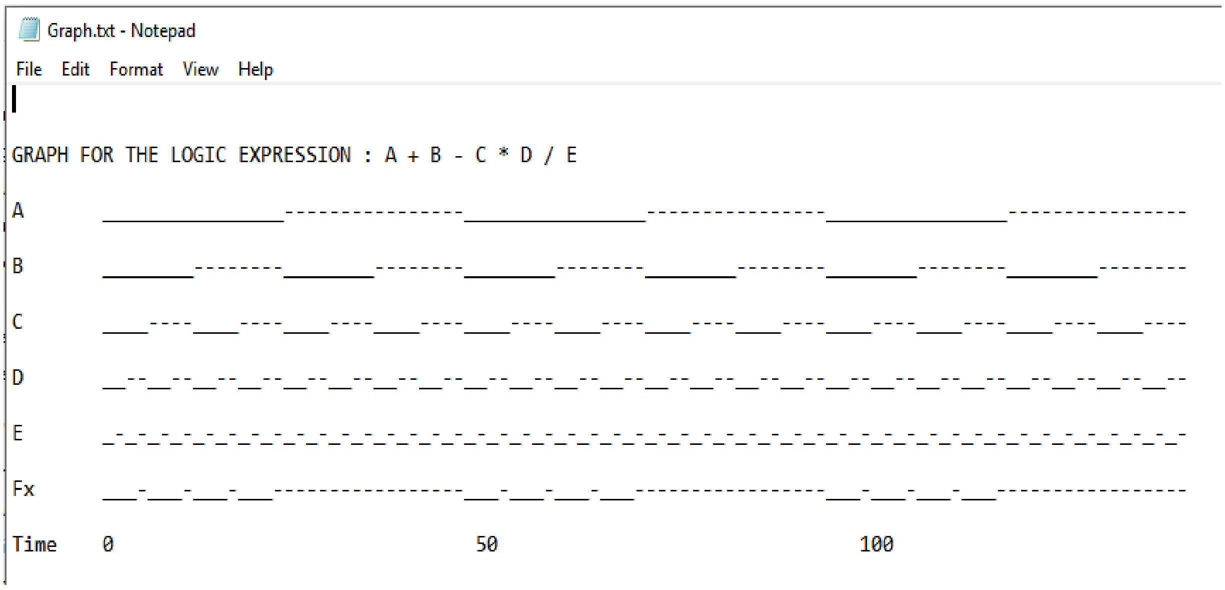

"E:\University\Semister 6\Data structures and Algorithms\Project\Final.exe"

```
Enter Clock Time for Simulation Graph (in multiples of 50 ms) : 50
Enter the number of cycles : 3

A       ________________----------------________________----------------________________----------------

B       ________--------________--------________--------________--------________--------________--------

C       ____----____----____----____----____----____----____----____----____----____----____----____----

D       __--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--

E       _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-

Fx      ___-___-___-___-----------------___-___-___-___-----------------___-___-___-___-----------------

Time    0                                 50                                100

THE GRAPH FILE HAS BEEN SAVED TO YOUR LOCAL DIRECTORY...PLZZ SAVE IT FOR USING NEXT TIME
```


"E:\University\Semister 6\Data structures and Algorithms\Project\Final.exe"

```
SUCCESS CASES

A       B       C       D       E
0       0       0       1       1
0       0       1       1       1
0       1       0       1       1
0       1       1       1       1
1       0       0       0       0
1       0       0       0       1
1       0       0       1       0
1       0       0       1       1
1       0       1       0       0
1       0       1       0       1
1       0       1       1       0
1       0       1       1       1
1       1       0       0       0
1       1       0       0       1
1       1       0       1       0
1       1       0       1       1
1       1       1       0       0
1       1       1       0       1
1       1       1       1       0
1       1       1       1       1


                THE FINAL SAYING IS :

Polygon is Grey if Cube is black Or Rectangle is red Circle is blue And Triangle is black Sphere is Yellow


TO EXIT PRESS 0 OR 1 TO CONTINUE :
```

Graph saved in .txt file through file handling operation



```
GRAPH FOR THE LOGIC EXPRESSION : A + B - C * D / E

A       ________________----------------________________----------------________________----------------

B       ________--------________--------________--------________--------________--------________--------

C       ____----____----____----____----____----____----____----____----____----____----____----____----

D       __--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--__--

E       _-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-

Fx      ___-___-___-___-----------------___-___-___-___-----------------___-___-___-___-----------------

Time    0                                50                              100
```