

Project Report on

Evaluation of algebraic expression

Data Structure (CS-1201) Project

Dipjyoti Bisharad : 14-1-4-002

Patel Yashwant Reddy : 14-1-4-006

Mery Hazarika : 14-1-4-102

Dept. : ECE

Sec : B

Sem : 3rd

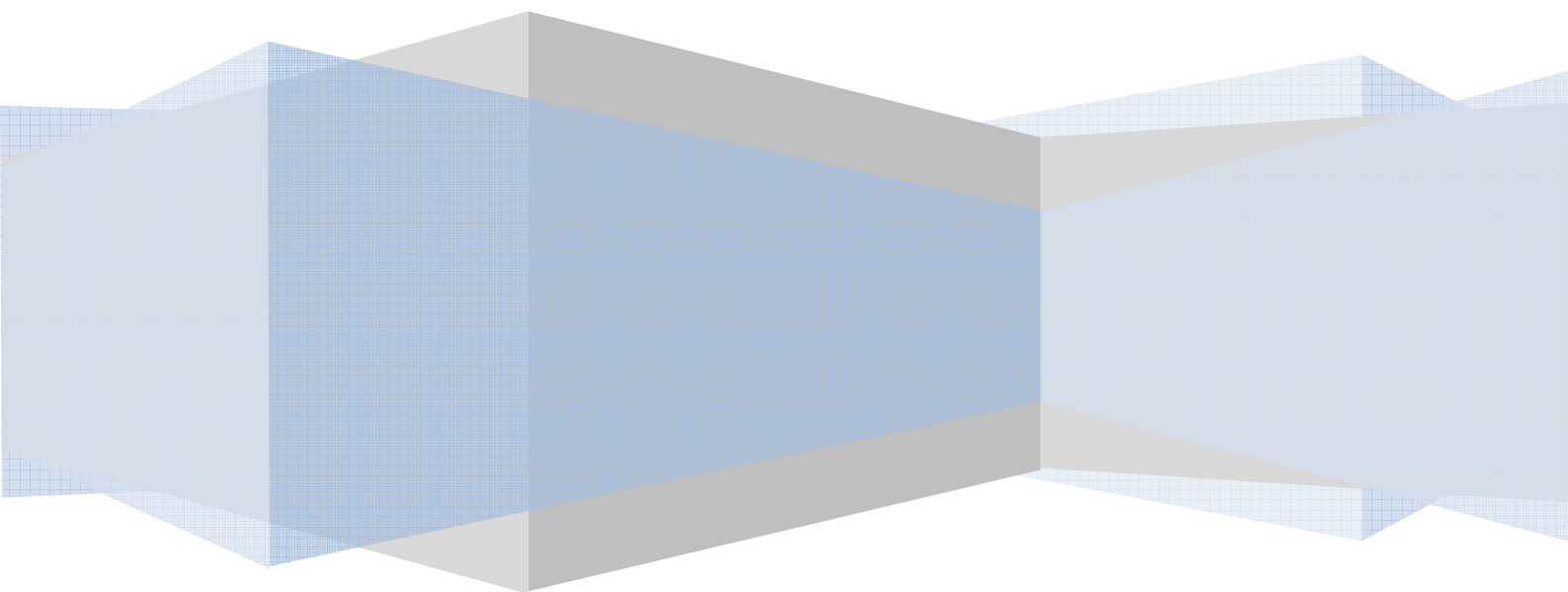


TABLE OF CONTENTS

- **Problem Statement**
- **Theory**
- **Methodology**
- **Platform used**
- **Source code**
- **Discussion on the code**
- **Conclusion**
- **Screenshots of output**

Problem Statement

The user will input a polynomial algebraic expression that may contain one or more variables or none. The program will validate the syntax of the expression. If the expression is valid, the user will then have to enter the values of the variables used, if any. The program will then evaluate the expression and print the result.

Theory & Methodology

The input of the user is accepted in a character array . The expression is subjected to some tests to check its validity. Initially blank spaces, if any, are removed from the expression. Then firstly presence of invalid characters are checked (such as $\&$, $>$, $''$ etc). Then the operator sequencing is checked (e.g. $7y+*$, $z++5$, $x+(/5)$ etc).

Lastly parenthesis balancing is checked using stack. If the expression passes each and every test, the program execution moves further or else an error statement is printed and program stops executing further.

Next, the program asks user to input values for the variables used. The program scans the character array and detects the number of variables used and takes input accordingly.

Now the expression is suitably modified to ease the calculations. For this, each and every character of the expression is loaded in a queue and following operations are performed

- i) Multiplication operators (*) are inserted at necessary places. Expression like $4xy$ is converted to $4*x*y$
- ii) All the constants used in the expression are enclosed in '#'. Expression like $10x+6.56$ is converted to $\#10\#x+\#6.56\#$. This enabled the evaluation of expression for multi digit as well as decimal numbers too.

This step is indeed very crucial as it converts the expression in a form that makes it evaluate able

Finally, to evaluate the polynomial, a two step process is followed. The expression is converted into postfix with the help of a stack. The postfix expression, after replacing its variables with corresponding values, is then evaluated using another stack and we get the final output.

Platform Used

Operating system: Windows 8, 64bit

Programming language used: C++

IDE: Code::Blocks version 13.12 rev 9501

Compiler: GNU GCC Compiler

Output file: .exe

Source code

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<math.h>
struct queue
{
    char c;
    struct queue *next;
};
int *var;
void rmSpace(char[]);    /**Removes blank spaces from expression*/
int fsize(struct queue *);    /**Finds the size of expression after modification*/
int isOperator(char);    /**Checks whether a character is mathematical operator or not*/
int isDot(char[],int);    /**Checks whether a character is dot(.) or not*/
int isBrace(char);    /**Checks whether a character is bracket or not*/
int isValid(char[]);    /**Checks validity of the expression*/
int checkChar(char[]);    /**Checks presence of invalid characters in the expression*/
int checkOperator(char[]);    /**Checks whether the operators are in proper order or not*/
int checkParenthesis(char[]);    /**Checks whether the parenthesis of the expression is balanced or not*/
int checkDivByZero(char[]);    /**Checks whether there is division by zero in the expression*/
int scanExp(char[]);    /**Scans the number of variables present in the expression*/
struct queue* create(char[]);    /**Creates a queue of the characters present in expression*/
void push(char,char[],int *);    /**Pushes/inserts an element into the stack while converting infix to postfix*/
char pop(char[],int *);    /**Pops/deletes an element from the stack while converting infix to postfix*/
void push(double,double[],int *);    /**Pushes/inserts an element into the stack while evaluating postfix*/
double pop(double [],int *);    /**Pops/deletes an element from the stack while evaluating postfix*/
int in_stack_priority(char);    /**Checks the priority of an element present in stack*/
int incoming_priority(char);    /**Checks the priority of current element being processed*/
double evaluate(char[],double[],int);    /**Evaluates the postfix expression*/
void infix_to_postfix(struct queue*,char[]);    /**Converts the infix expression to postfix*/
void EnQueue(struct queue**,struct queue**,char);    /**Inserts element into the queue*/
char DeQueue(struct queue**);    /**Deletes element from the queue*/
```

```

int main()
{
    char exp[100],*stack;
    int nov,i;
    double ans,*val;
    struct queue* front=NULL;
    printf("Enter any algebraic expression:\n\n");
    gets(exp);
    rmSpace(exp);
    if (!isValid(exp))
    {
        printf("\nInvalid expression!!");
        getch();
        return 0;
    }
    for (i=0;i<strlen(exp);i++)
    {
        exp[i]=tolower(exp[i]);
    }
    nov=scanExp(exp);
    if (nov!=0)
    {
        printf("\nEnter the values of variables ");
        for (i=0;i<nov;i++)
        {
            printf("%c ",var[i]);
        }
        printf("\n");
    }
    val=(double *)malloc(nov*sizeof(double));
    for (i=0;i<nov;i++)
    {
        scanf("%lf",&val[i]);
    }
    front=create(exp);
    stack=(char *)malloc(fsize(front)*sizeof(char));
    infix_to_postfix(front,stack);
    ans=evaluate(stack,val,nov);
    printf("\nThe answer is %g",ans);
}

```

```

    getch();
    free(val);
    free(var);
    free(stack);
    return 0;
}
void rmSpace(char s[])
{
    int i,l=strlen(s),j,k=0;
    for (i=0;i<l;i++)
    {
        if (s[i]==' ')
        {
            for (j=i;j<l-1;j++)
            {
                s[j]=s[j+1];
            }
            s[l-(++k)]='\0';
            i--;
        }
    }
}
int isValid(char s[])
{
    if (!checkChar(s))
        return 0;
    if (!checkOperator(s))
        return 0;
    if (!checkParenthesis(s))
        return 0;
    if (!checkDivByZero(s))
        return 0;
    return 1;
}

```



```

        if (c==0&&j!=l)
            return 0;
        return 1;
    }
}
return 0;
}
int checkOperator(char s[])
{
    int i,l=strlen(s);
    for (i=0;i<l;i++)
    {
        if(isOperator(s[i]))
        {
            if (i==l-1 || i==0)
                return 0;
            if (!(isalpha(s[i-1]) || isdigit(s[i-1]) || s[i-1]=='.' || s[i-1]=='-' || s[i-1]=='/'))
                return 0;
            if (!(isalpha(s[i+1]) || isdigit(s[i+1]) || s[i+1]=='.' || s[i+1]=='-' || s[i+1]=='/'))
                return 0;
        }
    }
}
int checkParenthesis(char s[])
{
    char *c,ch;
    int count=0,l,i,pos=0;
    l=strlen(s);
    for (i=0;i<l;i++)
    {
        if (s[i]=='(' || s[i]=='{' || s[i]=='[')
            count++;
    }
    c=(char *)malloc(count*sizeof(char));
    for (i=0;i<l;i++)
    {
        if (s[i]=='(' || s[i]=='{' || s[i]=='[')
            push(s[i],c,&pos);
    }
}

```

```

    if (s[i]==' ' || s[i]=='}' || s[i]==']')
    {
        if (pos<0)
            return 0;
        ch=pop(c,&pos);
        if (ch=='(')
        {
            if (s[i]!='')
                return 0;
        }
        if (ch=='{')
        {
            if (s[i]!='}')
                return 0;
        }
        if (ch=='[')
        {
            if (s[i]!='']')
                return 0;
        }
    }
}
free(c);
if (pos!=0)
    return 0;
return 1;
}
int checkDivByZero(char s[])
{
    int i,l;
    l=strlen(s);
    for (i=0;i<l-1;i++)
    {
        if (s[i]=='/'&& s[i+1]=='0')
            return 0;
    }
    return 1;
}

```

```

int scanExp(char s[])
{
    int c=0,i,l,k=0;
    l=strlen(s);
    char *t;
    t=(char *)calloc(26,sizeof(char));
    for (i=0;i<l;i++)
    {
        if (isalpha(s[i]))
            t[(int)(s[i])-97]++;
    }
    for (i=0;i<26;i++)
    {
        if (t[i]>0)
            c++;
    }
    var=(int *)malloc(c*sizeof(int));
    for (i=0;i<26;i++)
    {
        if (t[i]>0)
            var[k++]=i+97;
    }
    free(t);
    return c;
}

struct queue* create(char s[])
{
    struct queue *front=NULL,*rear=NULL;
    int i,l,f=0,k=0;
    l=strlen(s);
    for (i=0;i<l;i++)
    {
        if (s[i]=='{' || s[i]=='[')
            s[i]='(';
        else if (s[i]=='}' || s[i]==']')
            s[i]=')';
    }
    if (isdigit(s[0]))
        EnQueue(&front,&rear, '#');
}

```

```

if (s[0]=='.')
{
    EnQueue(&front,&rear, '#');
    EnQueue(&front,&rear, '0');
}
for (i=0;i<l;i++)
{
    if(i>0)
    {
        if (isdigit(s[i])&&(!(isdigit(s[i-1])))&&s[i-1]!='.')
```

$$\text{EnQueue}(\&\text{front},\&\text{rear}, \text{'\#'});$$

```

    }
    if (i==l-1)
        break;
    if ((isalpha(s[i])&&isdigit(s[i+1])) || (isdigit(s[i])&&isalpha(s[i+1])))
    {
        EnQueue(&front,&rear, s[i]);
        k=1;
    }
    else if ((isalpha(s[i])&&s[i+1]=='') || (isdigit(s[i])&&s[i+1]==''))
    {
        EnQueue(&front,&rear, s[i]);
        k=1;
    }
    else if ((isalpha(s[i+1])&&s[i]=='') || (isdigit(s[i+1])&&s[i]==''))
    {
        EnQueue(&front,&rear, s[i]);
        k=1;
    }
    else if (isalpha(s[i])&&isalpha(s[i+1]))
    {
        EnQueue(&front,&rear, s[i]);
        k=1;
    }
    else
        EnQueue(&front,&rear,s[i]);
    if (isdigit(s[i])&&(!(isdigit(s[i+1])))&&s[i+1]!='.')
```

$$\text{EnQueue}(\&\text{front},\&\text{rear}, \text{'\#'});$$

```

        if (k)
            EnQueue(&front,&rear,'*');
        k=0;
    }
    EnQueue(&front,&rear,s[i]);
    if (isdigit(s[i]))
        EnQueue(&front,&rear, '#');
    return front;
}

void EnQueue(struct queue **front,struct queue **rear,char s)
{
    struct queue* temp=(struct queue*)malloc(sizeof(struct queue));
    temp->c=s;
    temp->next=NULL;
    if (*front==NULL)
    {
        *front=temp;
        *rear=temp;
        return;
    }
    (*rear)->next=temp;
    *rear=temp;
}

char DeQueue(struct queue **front)
{
    struct queue* temp=*front;
    (*front)=(*front)->next;
    char c=temp->c;
    free(temp);
    return c;
}

int fsize(struct queue* front)
{
    struct queue* temp=front;
    int c=0;

```

```

while(temp!=NULL)
{
    temp=temp->next;
    c++;
}
return c;
}
void infix_to_postfix(struct queue* front,char pf[])
{
    int i,TOP=0,m=0;
    char n,k,*s;
    s=(char *)malloc(fsize(front)*sizeof(char));
    while (front!=NULL)
    {
        k=DeQueue(&front);
        switch(k)
        {
            case '(':
                push(k,s,&TOP);
                break;
            case ')':
                while((n=pop(s,&TOP))!='(')
                    pf[m++]=n;
                break;
            case '+':
            case '-':
            case '*':
            case '/':
            case '%':
            case '^':
                while(TOP>=-1&&in_stack_priority(s[TOP])>=incoming_priority(k))
                    pf[m++]=pop(s,&TOP);
                push(k,s,&TOP);
                break;
            default:
                pf[m++]=k;
        }
    }
}

```

```

while(TOP!=-1)
    pf[m++]=pop(s,&TOP);
pf[m-1]='\0';
free(s);
}
void push(char c,char s[],int *TOP)
{
    s[++(*TOP)]=c;
}
char pop(char s[],int *TOP)
{
    return s[(--*TOP)];
}
int in_stack_priority(char c)
{
    switch(c)
    {
        case '(':
            return 0;
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
        case '%':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}

```

```

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

        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
        case '%':
            return 2;
        case '^':
            return 4;
        default:
            return 0;
    }
}

double evaluate(char s[],double val[],int nov)
{
    double ans,a,b,temp,*c,m=0;
    int i,l,TOP=-1,flag=0,j=0,f;
    l=strlen(s);
    c=(double *)malloc(l*sizeof(double));
    for(i=0;i<l;i++)
    {
        if (!(isOperator(s[i])))
        {
            if (isalpha(s[i]))
            {
                for (f=0;f<nov;f++)
                {
                    if (var[f]==(int)s[i])
                        break;
                }
                push(val[f],c,&TOP);
            }
        }
    }
}

```



```

else if (s[i]=='#')
{
    while(s[++i]!='#')
    {
        if (s[i]=='.')
        {
            flag=1;
            continue;
        }
        if (!flag)
            m=m*10+(int)(s[i]-'0');
        else
            m=m+((int)(s[i]-'0'))/pow(10,++j);
    }
    push(m,c,&TOP);
    flag=0;
    j=0;
    m=0;
}
}
else
{
    a=pop(c,&TOP);
    b=pop(c,&TOP);
    switch(s[i])
    {
        case '+':
            temp=b+a;
            break;
        case '-':
            temp=b-a;
            break;
        case '*':
            temp=b*a;
            break;
    }
}

```

```

        case '/':
            if (a==0)
            {
                printf("\nCannot divide by zero!!!");
                getch();
                exit(0);
            }
            temp=b/a;
            break;
        case '%':
            temp=(int)b%(int)a;
            break;
        case '^':
            temp=pow(b,a);
            break;
    }
    push(temp,c,&TOP);
}
}
ans=pop(c,&TOP);
free(c);
return ans;
}
void push(double c,double s[],int *TOP)
{
    s[++(*TOP)]=c;
}
double pop(double s[],int *TOP)
{
    return s[(*TOP)--];
}

```

Discussions on code implemented

In this code all the linear data structure namely array (static and dynamic), linked list, stack and queue have been used. While stacks have been implemented using dynamic array, the queue is implemented using linked list.

The whole program has been divided into 21 distinct functions each performing a particular task. The description of the task of each function is commented in their declaration statement. While operating using stacks and queue, standard insert and delete functions i.e. PUSH(),POP() in case of stack and EnQueue() and DeQueue() have been used. It is to be noted that two instances of function PUSH() and POP() have been used in this program using the concept of function overloading. One set of PUSH(), POP() operation is used while converting infix to postfix while another set is used in evaluating the postfix.

Some notably important functions used in the program are described in detail below:

- i) **checkParenthesis()** – This function checks whether all the parenthesis used in the expression are balanced or not. To do so, concept of stack is used. All the opening brackets were pushed into the stack and on encountering closing brackets, a element in stack was popped and its corresponding closing bracket is matched with the closing bracket encountered. If they match execution moves further or else false flag is returned. Unequal numbers of opening and closing brackets were also checked accordingly.
- ii) **create()** – This function is the most important one in the entire program. It is this function that converts the expression in suitable form that could be evaluated. In this function each character in the expression is loaded in a queue after inserting '#' to enclose constant numbers used in the expression.
- iii) **infix_to_postfix()** – As name suggests, it converts a infix statement to its corresponding postfix. The infix expression is the queue generated by the create() function. The postfix expression is evaluated using stack and stored in a dynamic character array. The priority and associativity of various operators are taken care by two functions namely in_stack_priority() and incoming_priority().

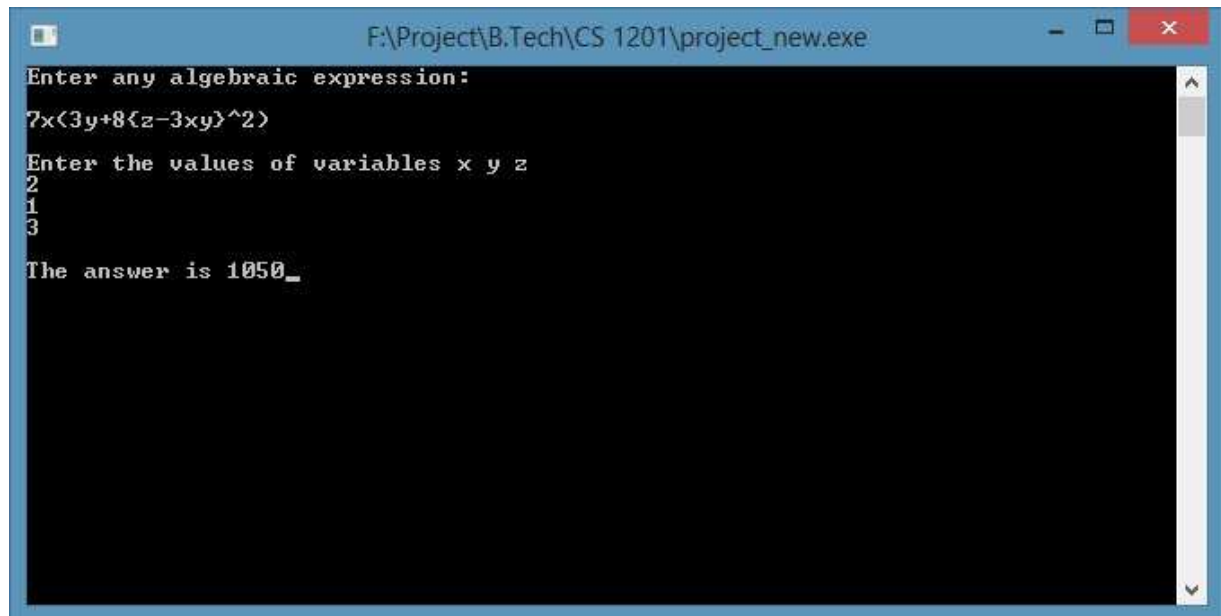
iv) evaluate() – This is the last major function of the program that finally evaluates the polynomial expression from its postfix form. In this stage the variables are replaced by their corresponding values taken from the user. The constants used in the program are reconstructed from the postfix expression in decimal format and pushed into the stack created in the function. The function returns answer upto 6 decimal places.

Conclusion

The program works extensively well for various forms of input. A number of checks of syntactical errors performed on the expression ensures that it does not end up in runtime error. Both integer and decimal constants work fine with the program. The program responds well to the errors such as divide by zero. All the standard requirements to evaluate any expression is strictly followed in the program. This program can be improved to work with mathematical functions such as sin, cos, square root, log etc and may be used in the firmware of calculators

Output

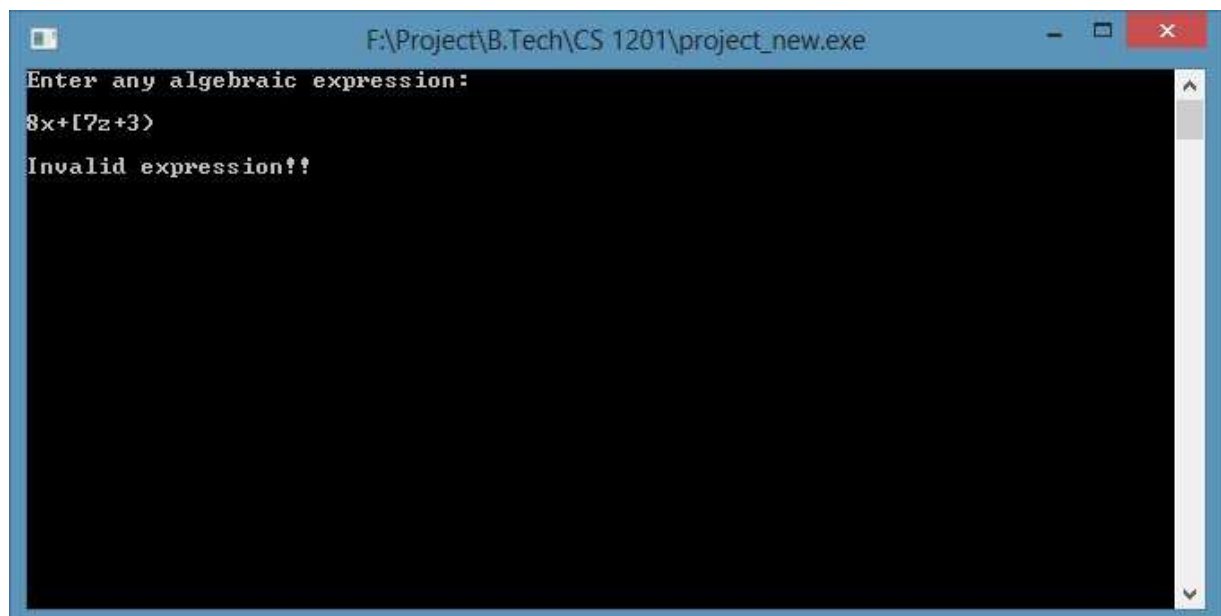
1)



```
F:\Project\B.Tech\CS 1201\project_new.exe
Enter any algebraic expression:
7x(3y+8(z-3xy)^2)
Enter the values of variables x y z
2
1
3
The answer is 1050_
```

A screenshot of a Windows command prompt window titled "F:\Project\B.Tech\CS 1201\project_new.exe". The window has a blue title bar with standard Windows window controls. The command prompt shows the following text: "Enter any algebraic expression:", "7x(3y+8(z-3xy)^2)", "Enter the values of variables x y z", "2", "1", "3", and "The answer is 1050_". The text is displayed in a monospaced font on a black background.

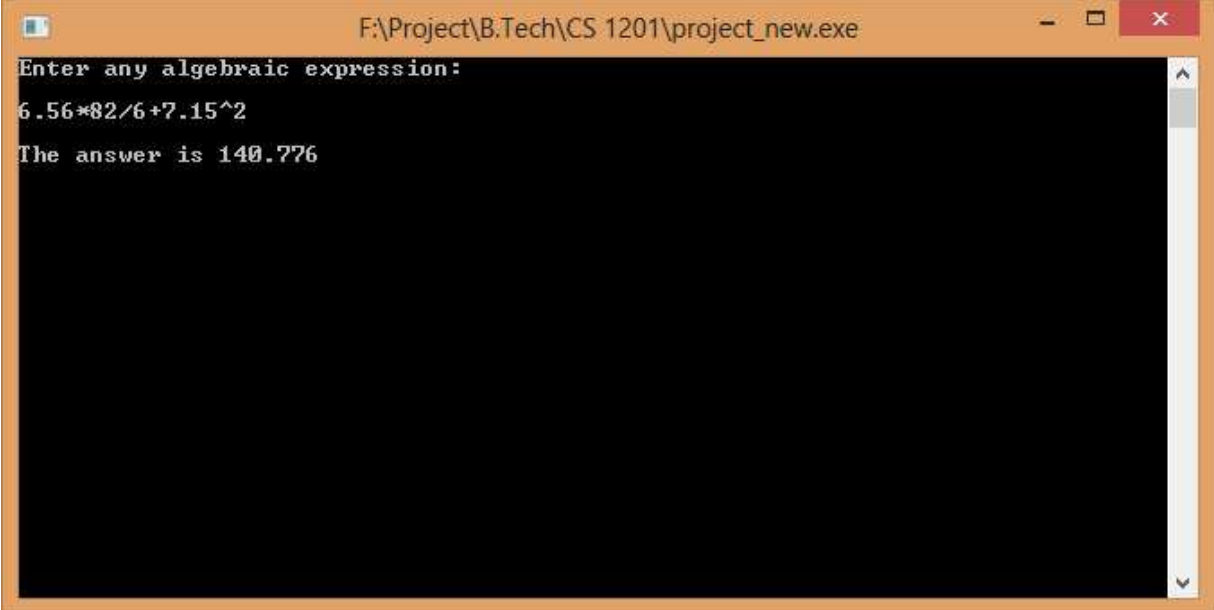
2)



```
F:\Project\B.Tech\CS 1201\project_new.exe
Enter any algebraic expression:
8x+[7z+3)
Invalid expression!!
```

A screenshot of a Windows command prompt window titled "F:\Project\B.Tech\CS 1201\project_new.exe". The window has a blue title bar with standard Windows window controls. The command prompt shows the following text: "Enter any algebraic expression:", "8x+[7z+3)", and "Invalid expression!!". The text is displayed in a monospaced font on a black background.

3)



```
F:\Project\B.Tech\CS 1201\project_new.exe
Enter any algebraic expression:
6.56*82/6+7.15^2
The answer is 140.776
```

4)



```
F:\Project\B.Tech\CS 1201\project_new.exe
Enter any algebraic expression:
836/x
Enter the values of variables x
0
Cannot divide by zero!!!_
```