DS Lab

Infix, Prefix and Postfix evaluation

**To Design and implement an algorithm to evaluate an arithmetic expressions which may be any form (postfix, prefix, infix), and demonstrate its working with suitable examples.**

Theory

Postfix and Prefix expressions are evaluated faster as compared to infix expressions and parentheses is not required in these expressions.

1. Infix Evaluation (simple calculator)

Pseudocode

Scan infix expression – num1 op num2

Switch(op)

Case ‘+’:

Print num1+num2

Case ‘-’:

Print num1-num2

Case ‘\*’:

Print num1\*num2

Case ‘/’:

Print num1/num2

Case ‘$’:

Print pow(num1, num2)

**Code**

#include<stdio.h>  //include standard input/output header file

int main()

{

        int num1, num2;  //declaration of variables to store operands

        char op;  //variable to store operator

        printf("Enter expression: ");

        scanf("%d%c%d", &num1, &op, &num2);

        int res=num1;  //declaration of res. if operator is '$' res defined by num1 will be used

        switch(op)  //selecting operation to be performed

        {

                case '+':   //add the operands

                        printf("Output: %d\n", num1+num2);

                        break;

                case '-':   //subtract the operands

                        printf("Output: %d\n", num1-num2);

                        break;

                case '\*':  //multiply

                        printf("Output: %d\n", num1\*num2);

                        break;

                case '/':  //divide

                        printf("Output: %d\n", num1/num2);

                        break;

                case '$':  //num1 power num2

                        for(int i=1; i<num2; i++)

                        {

                                res\*=num1;

                        }

                        printf("Output: %d\n", res);

                        break;

                default:

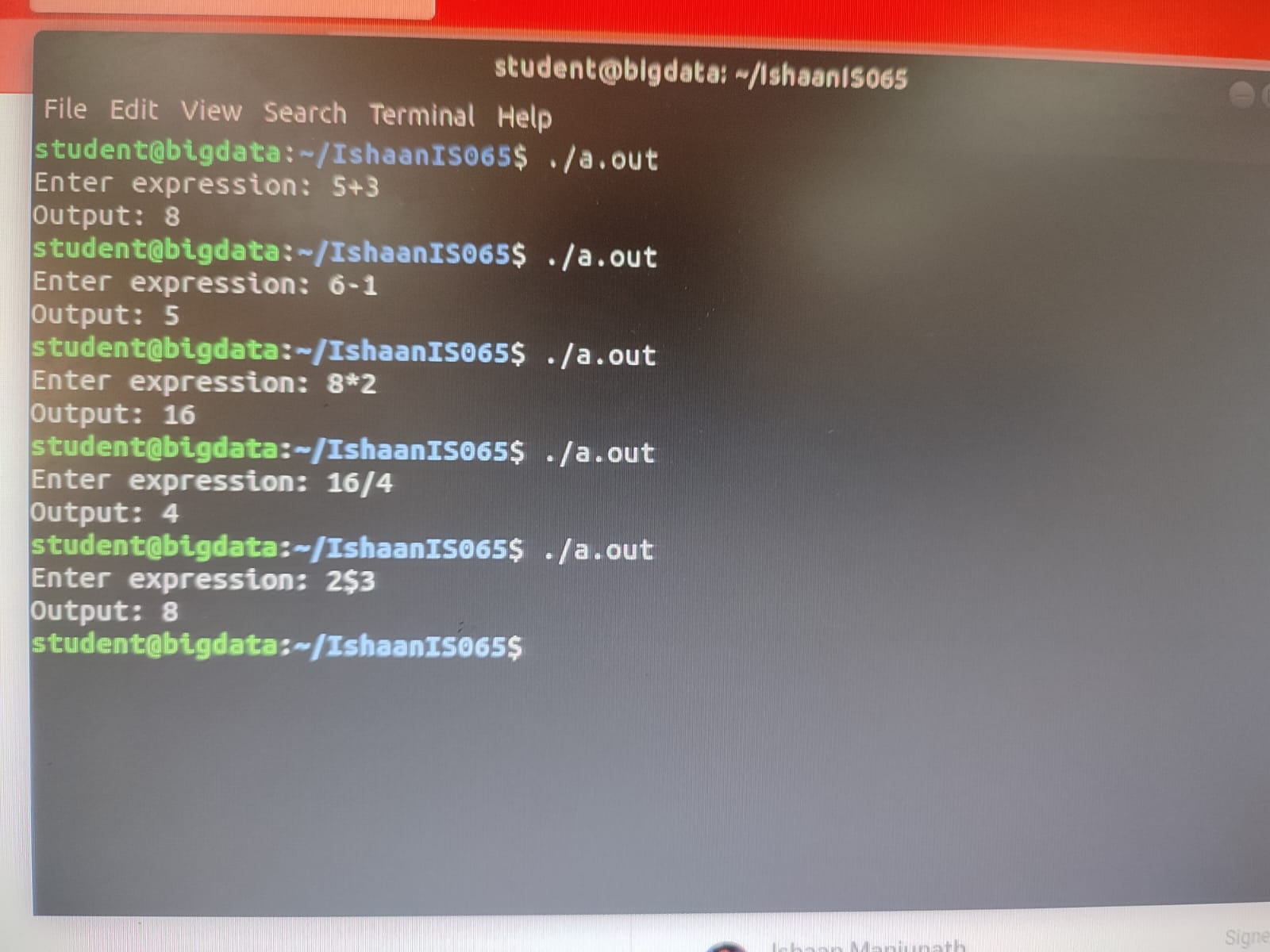
                        printf("Enter valid expression\n");

                        break;

        }

}

**Output**

****

1. Postfix evaluation using stack

Pseudocode

Scan postfix expression from left to right

If char is operand

Push (operand-‘0’) //integer value of character

Else if char is operator

num2=pop()

num1=pop()

Switch(operator)

Case ‘+’:

Temp= num1+num2

Case ‘-’:

Temp= num1-num2

Case ‘\*’:

Temp=num1\*num2

Case ‘/’:

Temp= num1/num2

Case ‘$’:

Temp= pow(num1, num2)

Push(Temp)

Print(Stack[top])

**Code**

#include<stdio.h>  //include header files required

#include<ctype.h>

#include<string.h>

int intStack[25]; //integer stack

int intTop=-1;

void pushInt(int item) //push function

{

        intStack[++intTop]=item;

}

int popInt()  //pop function

{

        return intStack[intTop--];

}

int evaluate(char \*postfix)  //function to evaluate postfix and return value

{

        char ch;

        int i=0, operand1, operand2;

        while((ch=postfix[i++])!='\0')  //scans postfix expression from left to right

        {

                if(isdigit(ch))  //if the character is a digit

                {

                        pushInt(ch-'0');  //pushes integer value of character value of digit from postfix

                }

                else

                {

                        operand2 = popInt();  //stack follows FILO. so first operand is popped after second operand

                        operand1 = popInt();

                        switch(ch)

                        {

                                case '+':  //addition

                                        pushInt(operand1+operand2);

                                        break;

                                case '-':  //subtraction

                                        pushInt(operand1-operand2);

                                        break;

                                case '\*':  //multiplication

                                        pushInt(operand1\*operand2);

                                        break;

                                case '/':  //division

                                        pushInt(operand1/operand2);

                                        break;

                                case '^':  //power

                                        pushInt(operand1^operand2);

                                        break;

                        }

                }

        }

        return intStack[intTop];   //final value is stored in stack

}

void main()  //main function

{

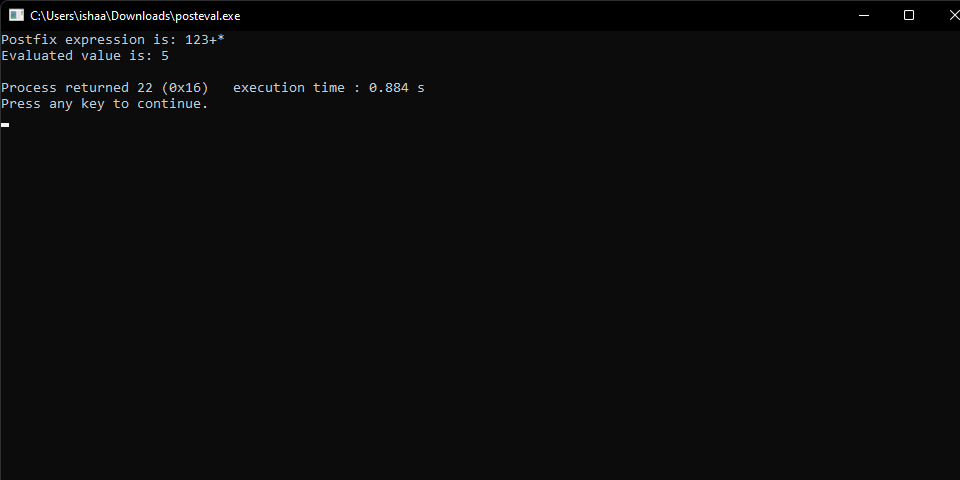
        char postfix[]="123+\*";

        printf("Postfix expression is: %s\n", postfix);

        printf("Evaluated value is: %d\n", evaluate(postfix));

}

**Output**



|  |  |
| --- | --- |
| Expression | Stack |
| 1 | 1 |
| 2 | 12 |
| 3 | 123 |
| + | 1 5 (2+3=5) |
| \* | 5(final value) |

1. Prefix Evaluation using stack

Pseudocode

Scan prefix from right to left

If char is operand

Push (operand-‘0’) //integer value of character

Else if char is operator

num1=pop()

num2=pop()

Switch(operator)

Case ‘+’:

Temp= num1+num2

Case ‘-’:

Temp= num1-num2

Case ‘\*’:

Temp=num1\*num2

Case ‘/’:

Temp= num1/num2

Case ‘$’:

Temp= pow(num1, num2)

Push(Temp)

Print(Stack[top])

**Code**

#include<stdio.h>   //include necessary header files

#include<ctype.h>

#include<string.h>

int intStack[25];  //integer stack

int intTop=-1;

void pushInt(int item)  //push function for integer stack

{

        intStack[++intTop]=item;

}

int popInt()  //pop function for integer stack

{

        return intStack[intTop--];

}

int evaluate(char \*prefix)  //prefix evaluation function

{

        char ch;   //variable initialization

        int i, operand1, operand2;

        for(i=strlen(prefix)-1; i>-1; i--)   //scans prefix from right to left

        {

                ch=prefix[i];

                if(isdigit(ch))   //if character is digit push integer value of character to stack

                {

                        pushInt(ch-'0');

                }

                else

                {

                        operand1 = popInt();   //scanning from right to left, first operand gets pushed last and hence gets popped out first

                        operand2 = popInt();

                        switch(ch)

                        {

                                case '+':  //addition

                                        pushInt(operand1+operand2);

                                        break;

                                case '-':  //subtraction

                                        pushInt(operand1-operand2);

                                        break;

                                case '\*':  //multiplication

                                        pushInt(operand1\*operand2);

                                        break;

                                case '/':  //division

                                        pushInt(operand1/operand2);

                                        break;

                                case '^':  //power

                                        pushInt(operand1^operand2);

                                        break;

                        }

                }

        }

        return intStack[intTop];  //returns top value of stack which hold evaluated value

}

void main()  //main function

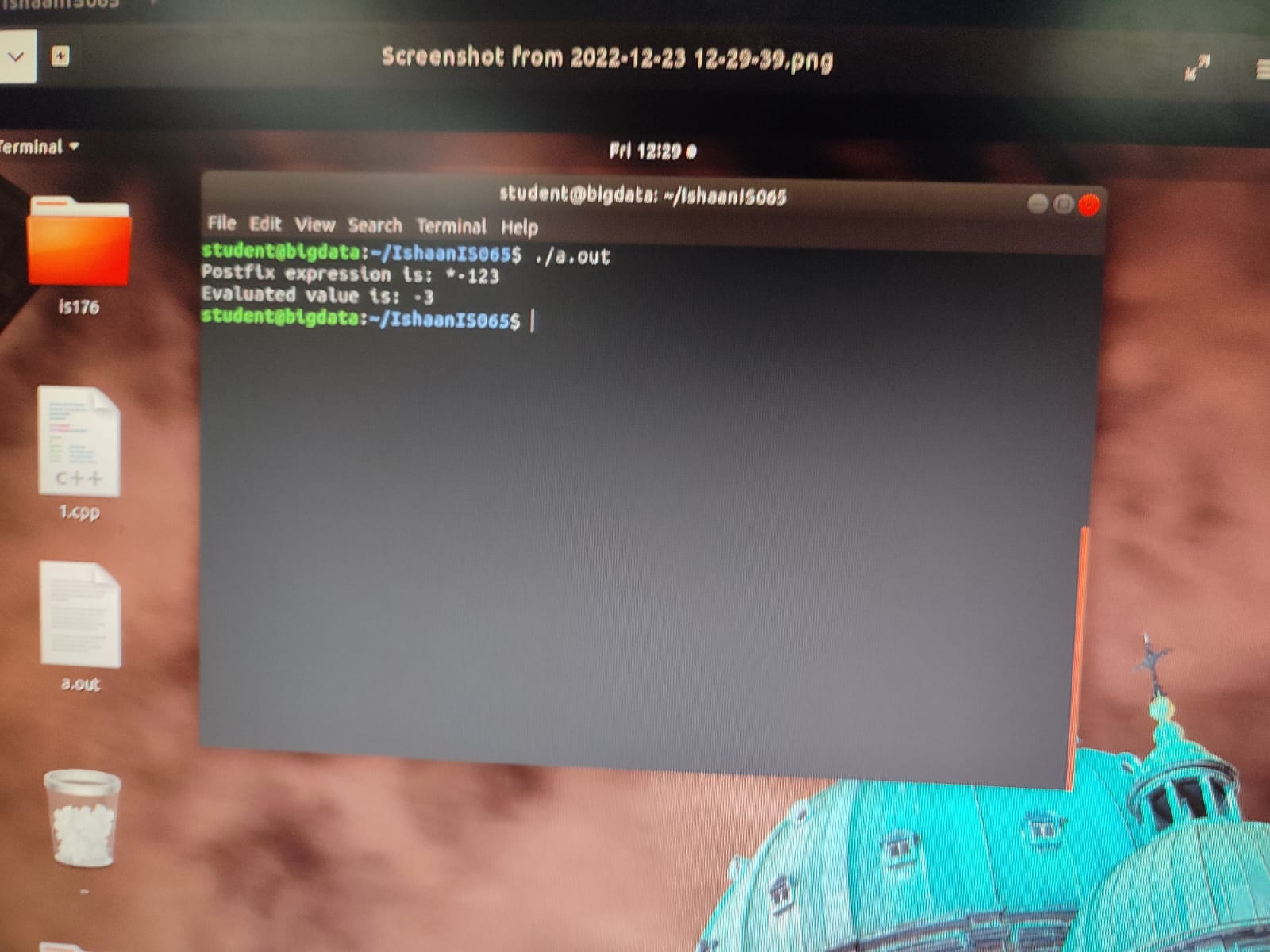
{

        char prefix[]="\*-123";

        printf("Postfix expression is: %s\n", prefix);

        printf("Evaluated value is: %d\n", evaluate(prefix));

}



Using reversed expression – 321-\*

|  |  |
| --- | --- |
| Expression | Stack |
| 3 | 3 |
| 2 | 32 |
| 1 | 321 |
| - | 3 -1(1-2=-1) |
| \* | -3 (final value) |