**Practical 03**

1. **Stack Demonstration with operation.**

Code:

#include<stdio.h>

int stack[100], c, n, top, x, i;

void push(int);

void pop(void);

void display(void);

int main()

{

top = -1;

printf("\n Enter the size of Stack max is 100 :");

scanf("%d",&n);

printf("\n\t Stack operations using array");

printf("\n\t 1. Push.\n\t 2. Pop.\n\t 3.Display.\n\t 4.Exit");

do

{

printf("\n Enter your choice :\n");

scanf("%d",&c);

switch(c)

{

case 1:

{

printf("\nEnter an element to push\n");

scanf("%d",&x);

push(x);

break;

}

case 2:

{

printf("\nPopping the element\n");

pop();

break;

}

case 3:

{

printf("\nStack Array\n");

display();

break;

}

case 4:

{

printf("\nExit ...\n");

break;

}

default:

{

printf("\nInvalid Option\n");

break;

}

}

}while(c != 4);

return 0;

}

void push(int p)

{

if(top>=n-1)

{

printf("\n Stack Overflow\n");

}

else

{

top++;

stack[top]=p;

}

}

void pop()

{

if(top<0)

{

printf("\n Stack Underflow\n");

}

else

{

printf("\n Popped element in Stack is %d",stack[top]);

top--;

}

}

void display()

{

if(top>=0)

{

printf("\nElements in the Stack\n");

for(i=0; i<=top; i++)

{

printf("%d\t",stack[i]);

}

}

else

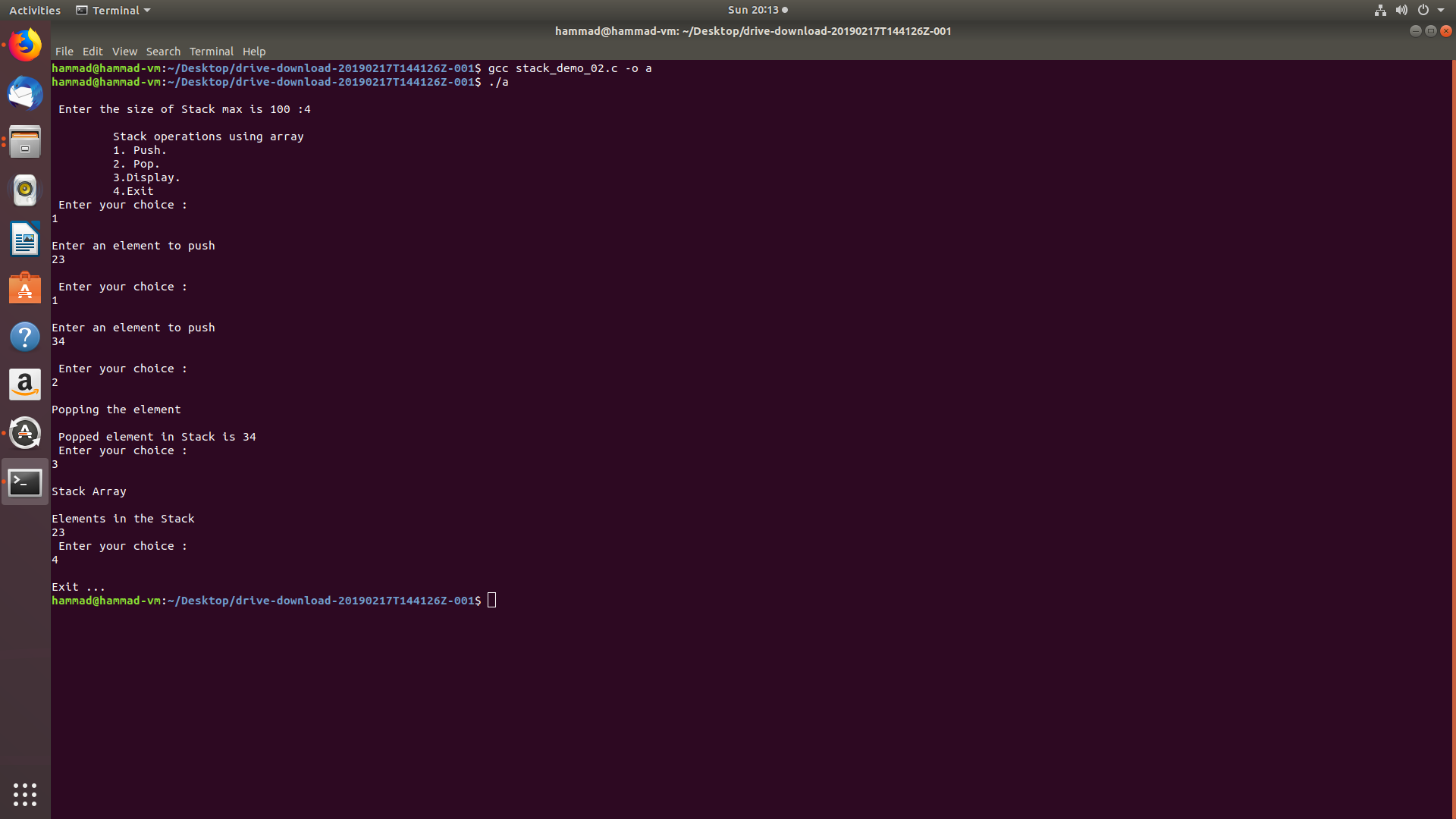
{

printf("\nStack is empty\n");

}

}

Screenshot:



1. **Stack Postfix Evaluation**

Code:

#include<stdio.h>

int stack[100], c, n, top, x, i;

char s;

void push(int);

void op(char);

void display(void);

int main()

{

top = -1;

printf("\n Enter the size of Stack max is 100 :");

scanf("%d",&n);

printf("\n\t Stack operations using array");

printf("\n\t 1. Push an Element.\n\t 2. Operator. \n\t 3. Display.\n\t 4. Exit");

do

{

printf("\n Enter your choice :\n");

scanf("%d",&c);

switch(c)

{

case 1:

{

printf("\n Enter an element to push\n");

scanf("%d",&x);

push(x);

break;

}

case 2:

{

printf("\n Enter Operator\n");

scanf(" %c", &s);

op(s);

break;

}

case 3:

{

printf("\n Stack Array\n");

display();

break;

}

case 4:

{

printf("\n Exit ...\n");

break;

}

default:

{

printf("\n Invalid Option\n");

break;

}

}

}while(c != 4);

return 0;

}

void push(int p)

{

if(top>=n-1)

{

printf("\n Stack Overflow\n");

}

else

{

top++;

stack[top]=p;

}

}

void op(char o)

{

if(top>=0)

{

if(o == '+')

{

printf("\n Addition between top and top-1 element\n");

stack[top-1] = stack[top-1] + stack[top];

top--;

}

else if(o == '-')

{

printf("\n Subtraction between top and top -1\n");

stack[top-1] = stack[top-1] - stack[top];

top--;

}

else if(o == '\*')

{

printf("\n Multipilication between top and top -1\n");

stack[top-1] = stack[top-1] \* stack[top];

top--;

}

else if(o == '/')

{

printf("\n Division between top and top -1 \n");

stack[top-1] = stack[top-1] / stack[top];

top--;

}

}

else

{

printf("Stack is empty choose again!!!");

}

}

void display()

{

if(top>=0)

{

printf("\nElements in the Stack\n");

for(i=0; i<=top; i++)

{

printf("%d\t",stack[i]);

}

}

else

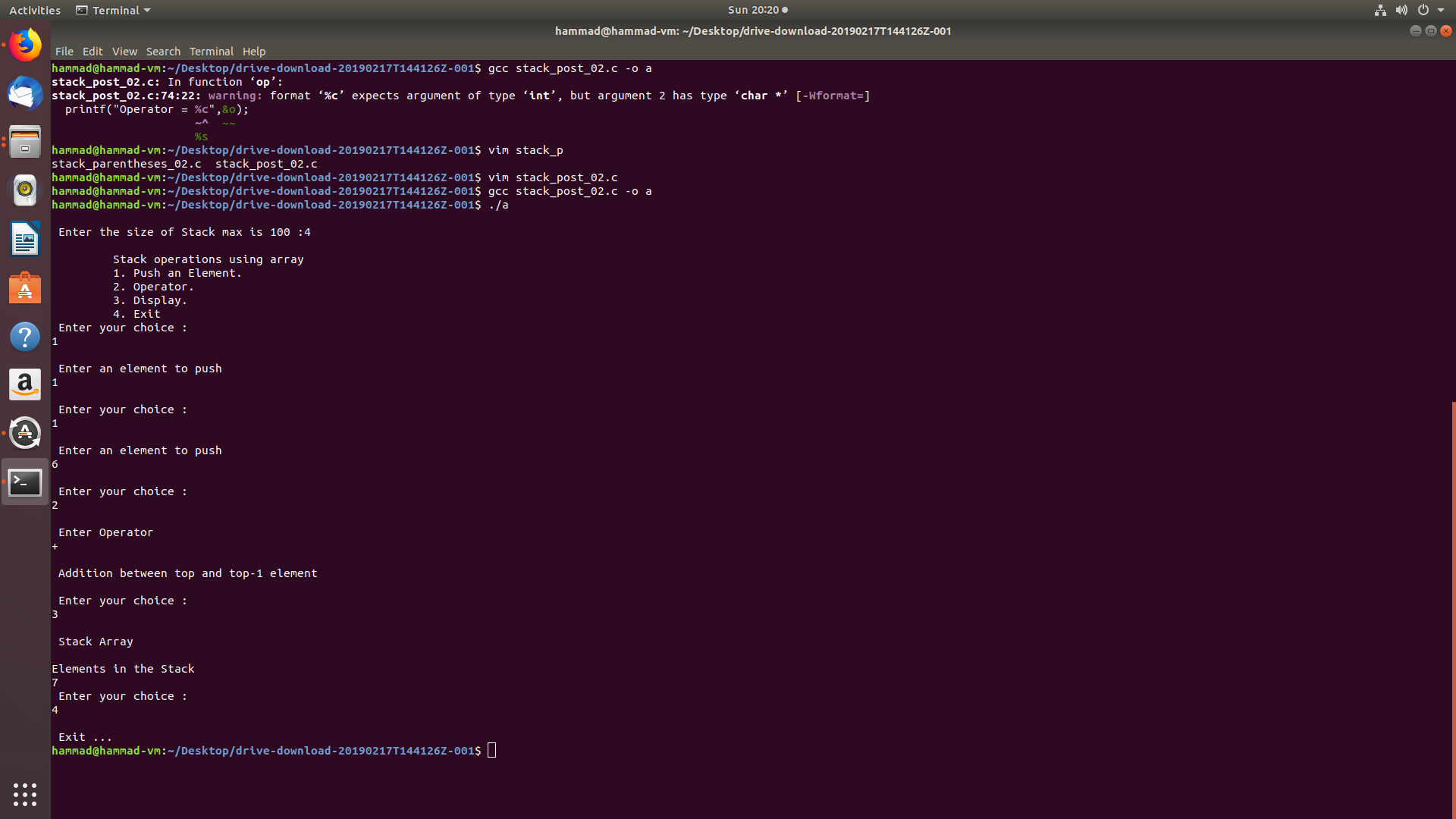
{

printf("\nStack is empty\n");

}

}

Screenshot:



1. **Stack Infix to Postfix Conversion**

Code:

#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>

#define max 100

int top = -1;

char stack[max];

void push(char);

char pop();

void find\_top();

int precedence(char);

int isOperator(char);

void InfixToPostfix(char[] , char[]);

int main()

{

int i;

char infix\_temp[max], postfix\_temp[max];

printf("\nEnter Infix expression : \n");

gets(infix\_temp);

InfixToPostfix(infix\_temp, postfix\_temp);

printf("\nPostfix Expression : \n");

puts(postfix\_temp);

return 0;

}

void push(char item)

{

if(top >= max-1)

{

printf("\nStack Overflow.\n");

}

else

{

top = top+1;

stack[top] = item;

}

}

char pop()

{

char item ;

if(top <0)

{

printf("\nStack Underflow.\n");

exit(1);

}

else

{

item = stack[top];

top = top-1;

return(item);

}

}

void find\_top()

{

if(top>0)

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

else

printf("\nValid Expression.\n");

}

int precedence(char s)

{

if(s == '^')

{

return(3);

}

else if(s == '\*' || s == '/')

{

return(2);

}

else if(s == '+' || s == '-')

{

return(1);

}

else

{

return(0);

}

}

int is\_operator(char c)

{

if(c == '^' || c == '\*' || c == '/' || c == '+' || c =='-')

{

return 1;

}

else

{

return 0;

}

}

void InfixToPostfix(char infix[], char postfix[])

{

int i, j;

char item;

char x;

push('(');

strcat(infix,")");

i=0;

j=0;

item=infix[i];

while(item != '\0')

{

if(item == '(')

{

push(item);

}

else if( isdigit(item) || isalpha(item))

{

postfix[j] = item;

j++;

}

else if(is\_operator(item) == 1)

{

x=pop();

while(is\_operator(x) == 1 && precedence(x)>= precedence(item))

{

postfix[j] = x;

j++;

x = pop();

}

push(x);

push(item);

}

else if(item == ')')

{

x = pop();

while(x != '(')

{

postfix[j] = x;

j++;

x = pop();

}

}

else

{

printf("\nInvalid infix Expression.\n");

getchar();

exit(1);

}

i++;

item = infix[i];

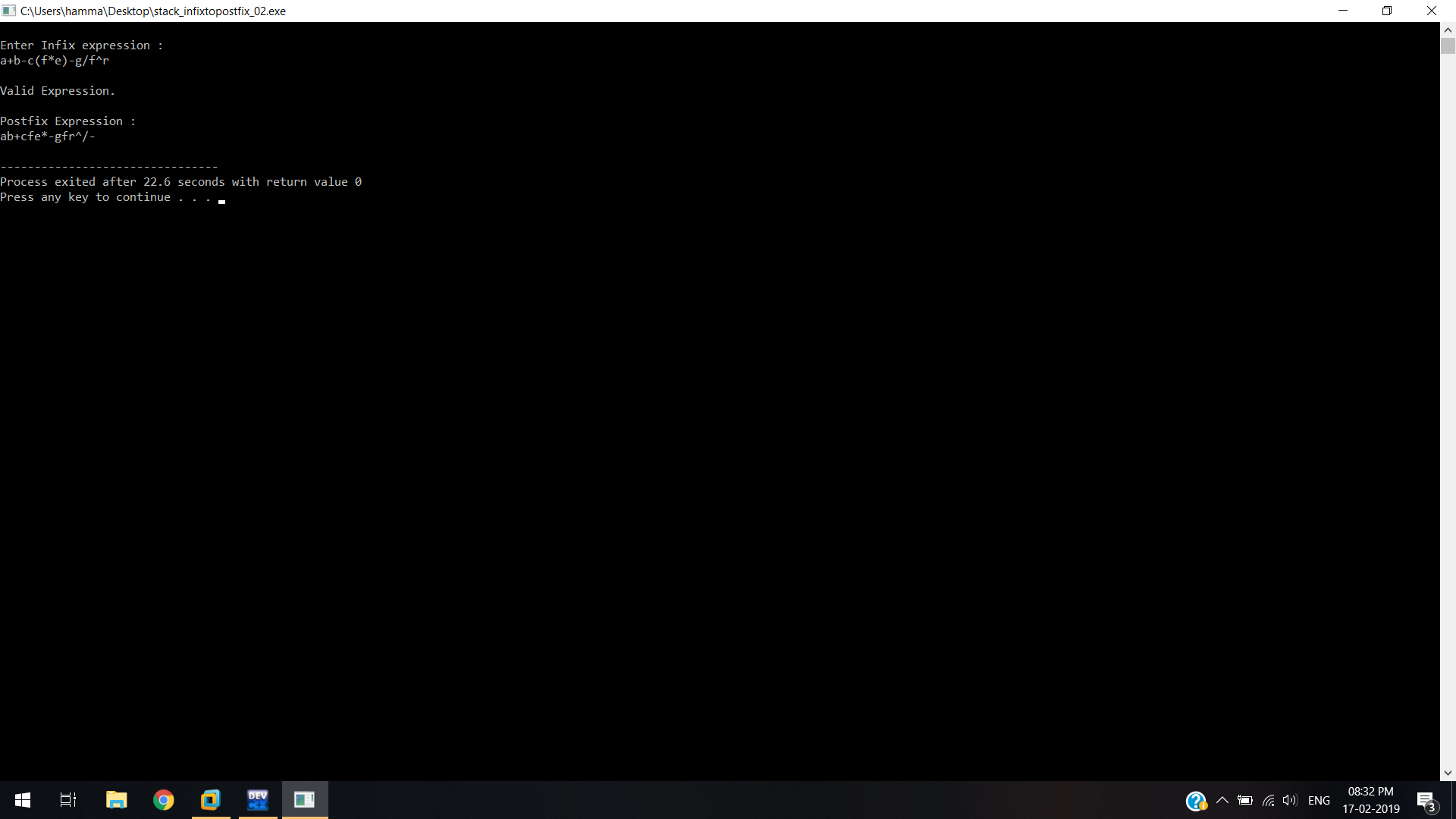
}

find\_top();

postfix[j] = '\0';

}

Screenshot:



1. **Stack Parentheses Balancing**

Code:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define max 100

int top = -1;

char stack[max];

void push(char);

void pop();

void find\_top();

int main()

{

int i;

char temp[max];

printf("\nEnter expression\n");

scanf("%s", &temp);

for (i = 0; temp[i] != '\0';i++)

{

if (temp[i] == '(')

{

push(temp[i]);

}

else if (temp[i] == ')')

{

pop();

}

else if (temp[i] == '[')

{

push(temp[i]);

}

else if (temp[i] == ']')

{

pop();

}

else if (temp[i] == '{')

{

push(temp[i]);

}

else if (temp[i] == '}')

{

pop();

}

}

find\_top();

return 0;

}

void push(char p)

{

stack[top] = p;

top++;

}

void pop()

{

if (top == -1)

{

printf("\nStack Underflow\n");

exit(0);

}

else

{

top--;

}

}

void find\_top()

{

if (top == -1)

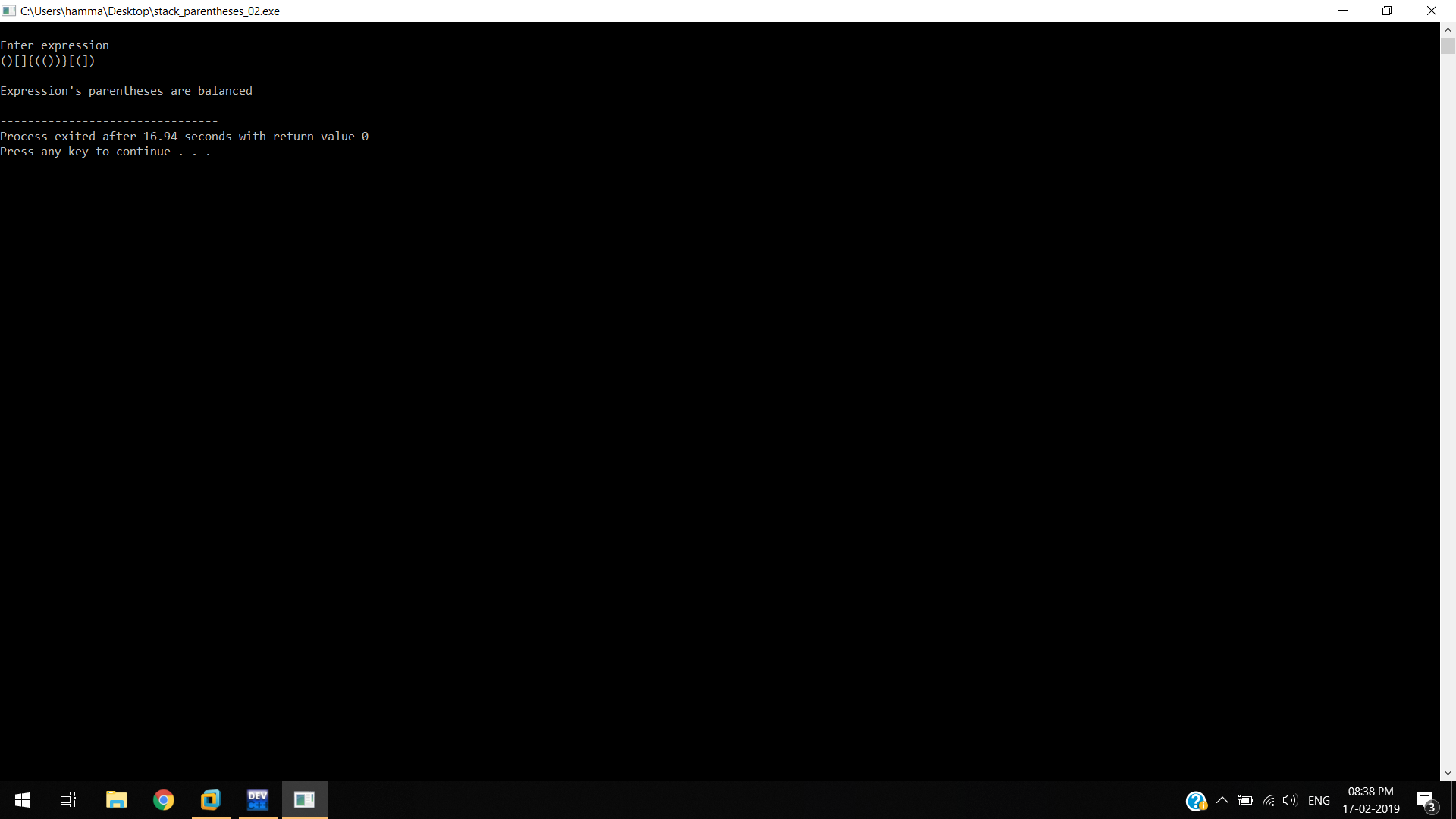
printf("\nExpression's parentheses are balanced\n");

else

printf("\nExpression's parentheses not balanced\n");

}

Screenshot:



5. **Stack Using Linked List**

**Code:**

#include<stdio.h>

#include<stdlib.h>

void push();

void display();

void pop();

int counter=0;

// A linked list node

struct Node

{

int data;

struct Node \*next;

}\*r, \*temp, \*top = NULL, \*q, \*p;

//menu driven program for insertion deletion and to display linked list

int main()

{

int ch;

do

{

printf("\n Enter Choice :-\n");

printf("\n 1. Push an element.\n 2. Pop an element.\n 3. Display Stack.\n 4. Exit.\n");

scanf("%d",&ch);

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

printf("\n Bye!!!\n");

exit(0);

break;

default:

printf("\n Invalid Choice\n");

break;

}

}while(ch!=4);

}

void push()

{

int data1;

if(top==NULL)

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1; //setting very first element->next to null eg list->data = data1 and list->next = NULL.

p->next = NULL;

top = p;

}

else

{

p = (struct Node \*)malloc(sizeof(struct Node));//To assign memory location.

printf("\n Enter an element to insert.\n");//input.

scanf("%d",&data1);

p->data = data1;

p->next = NULL; //NULL because it'll be the last element of the linked list.

q = top;

while(q->next != NULL)

{

q = q->next; //To check pointer q has reached last location of the linked list.

}

q->next = p; //setting last element to p which will be like adding a new element at the last location.

}

}

void pop()

{

if(top == NULL)

{

printf("\n Stack is empty.\n"); //to check whether it is empty or not

}

else

{

q = top;

if(q->next == NULL)

{

top = NULL;

}

while(q->next!=NULL) //while to point q pointer at the very last location of linked list, we're running it till end.

{

r = q; //r will store previous pointed out location of linked list

q=q->next; //to increment q

}

r->next=NULL; //setting previous location's next pointer to NULL since it's very last element so it should point to NULL.

free(q); // delocating memory of q

printf("\n Popped an element from the stack.\n");

}

}

void display()

{

if(top==NULL)

{

printf("\n Stack is empty.\n");

}

else

{

printf(" Elements :-\t");

q=top;

counter = 0;

while(q != NULL)

{

printf("%d\t",q->data);

q = q->next;

counter++; // to count total number of elements in the linked list

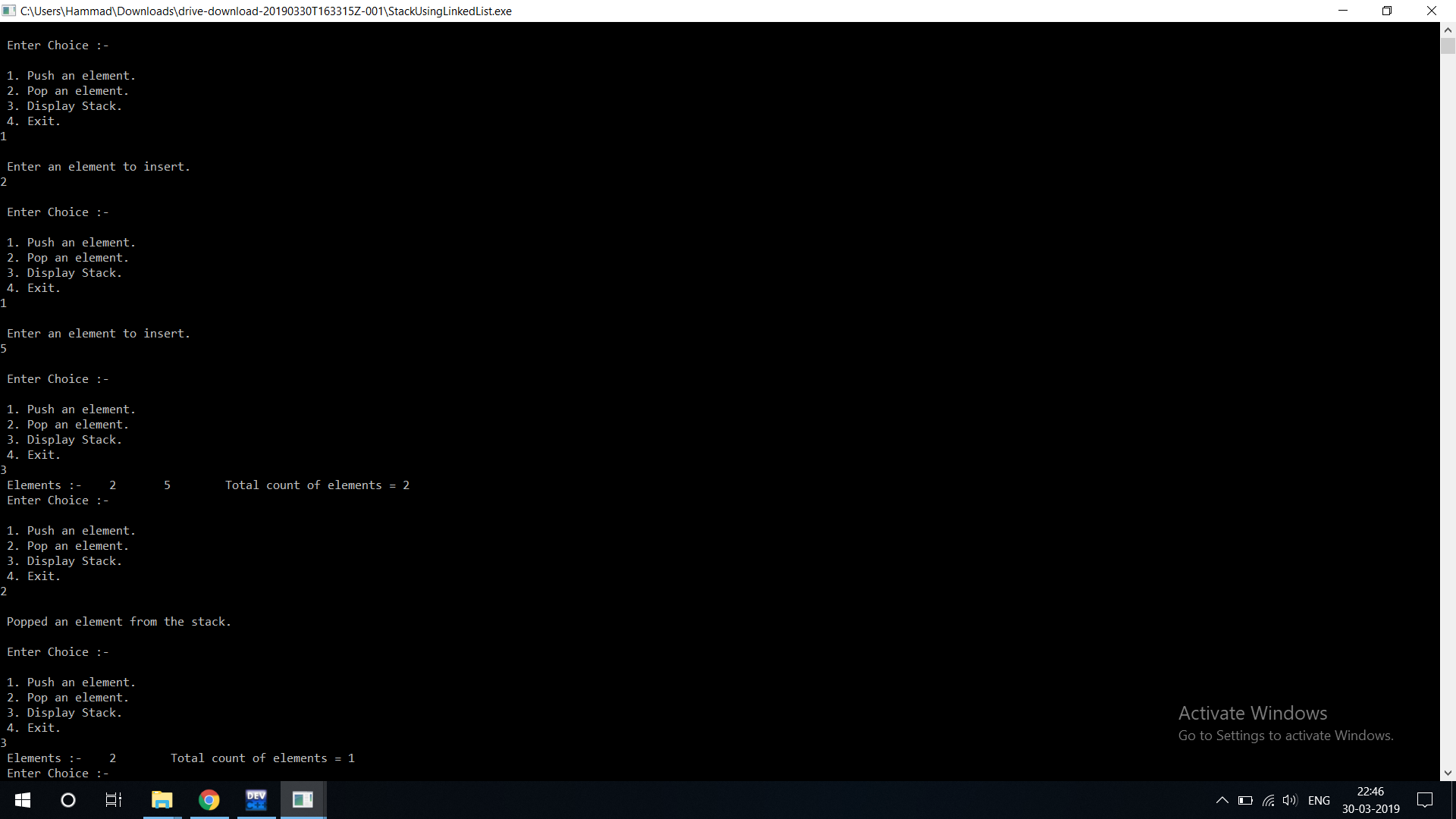
}

printf(" Total count of elements = %d",counter);

}

}

**Screenshot :**

****