**Data Structure**

**(UCS301)**

**Assignment**

**4**



Submitted By-

Name: Ananya Agarwal

Roll No: 102083036

Batch: 2CO14

1. **Develop a menu driven program demonstrating the following operations on a Stack: push(), pop(), isEmpty(), isFull(), display(), and peek().**

Sol 1-

#include<iostream>

using namespace std;

struct node

{

int data;

node \*next;

};

class stack

{

node \*top;

public:

stack()

{

top=NULL;

}

bool isEmpty();

bool isFull();

void push(int a);

void pop();

int peek();

void display();

};

bool stack::isEmpty()

{

return (top==NULL);

}

bool stack::isFull()

{

node \*temp=new node;

if(temp==NULL)

{

delete temp;

return true;

}

else

{

delete temp;

return false;

}

}

void stack::push(int a)

{

if(isFull())

cout<<"Stack is full.....Overflow condition!!!\n";

else

{

node \*temp=new node;

temp->data=a;

temp->next=top;

top=temp;

}

}

void stack::pop()

{

if(isEmpty())

cout<<"Underflow! \n";

else

{

node \*temp=top;

top=top->next;

cout<<temp->data<<" deleted! \n";

delete temp;

}

}

int stack::peek()

{

return top->data;

}

void stack::display()

{

if(isEmpty())

cout<<"Underflow! \n";

else

{

cout<<"Stack elements are as follows : \n";

node \*temp=top;

while(temp!=NULL)

{

cout<<temp->data<<endl;

temp=temp->next;

}

cout<<endl;

}

}

int main()

{

stack s;

int ch;

char c;

do

{

cout<<"Menu of choices : \n";

cout<<"1) Push \n";

cout<<"2) Pop \n";

cout<<"3) Is Empty \n";

cout<<"4) Is Full \n";

cout<<"5) Display \n";

cout<<"6) Peek \n";

cout<<"7) Exit \n";

cout<<"\nEnter your choice : ";

cin>>ch;

cout<<endl;

switch(ch)

{

case 1:

int a;

cout<<"Enter data : ";

cin>>a;

s.push(a);

break;

case 2:

s.pop();

break;

case 3:

if(s.isEmpty())

cout<<"Yes! Stack is empty!!! \n";

else

cout<<"No! Stack is not empty!!! \n";

break;

case 4:

if(s.isFull())

cout<<"Yes! Stack is full!!! \n";

else

cout<<"No! Stack is not full!!! \n";

break;

case 5:

s.display();

break;

case 6:

cout<<"Data at top node : "<<s.peek()<<endl;

break;

case 7:

exit(0);

default:

cout<<"Invalid choice! \n";

}

cout<<"\nDo you want to continue? (y/n) : ";

cin>>c;

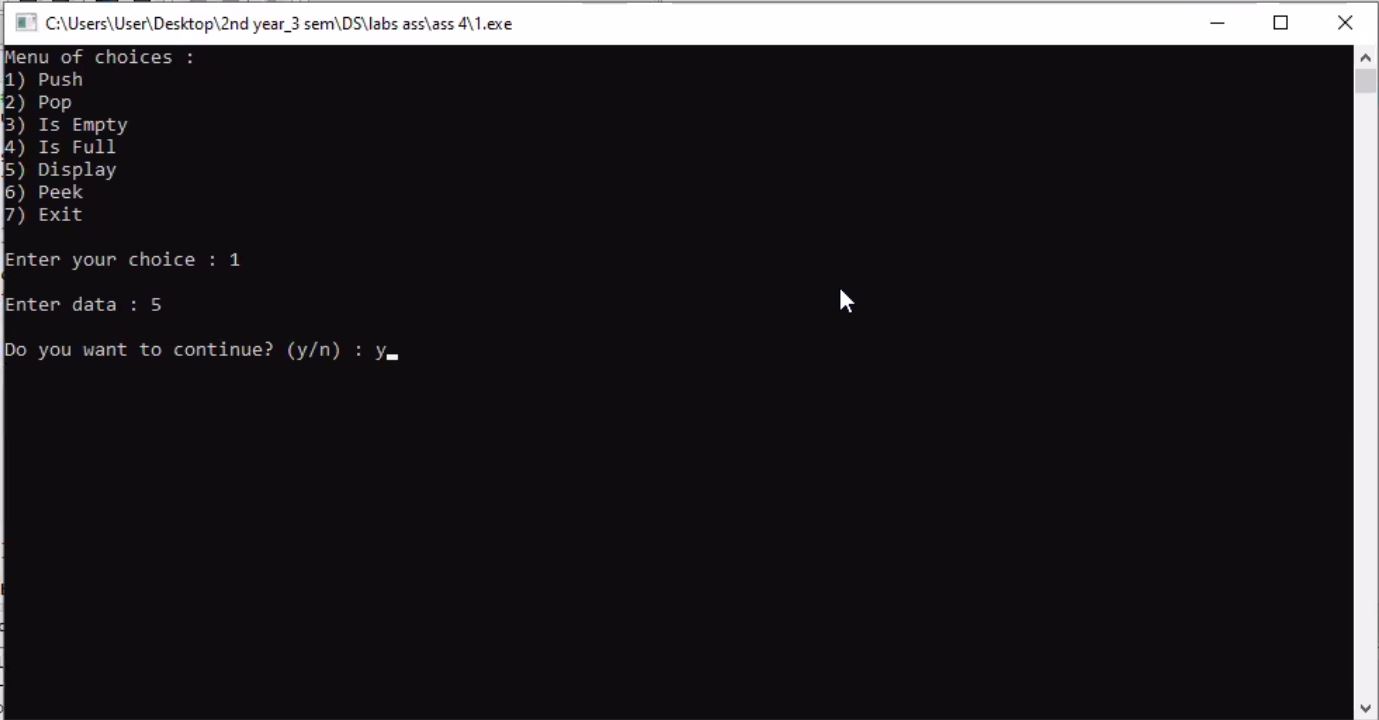
system("cls");

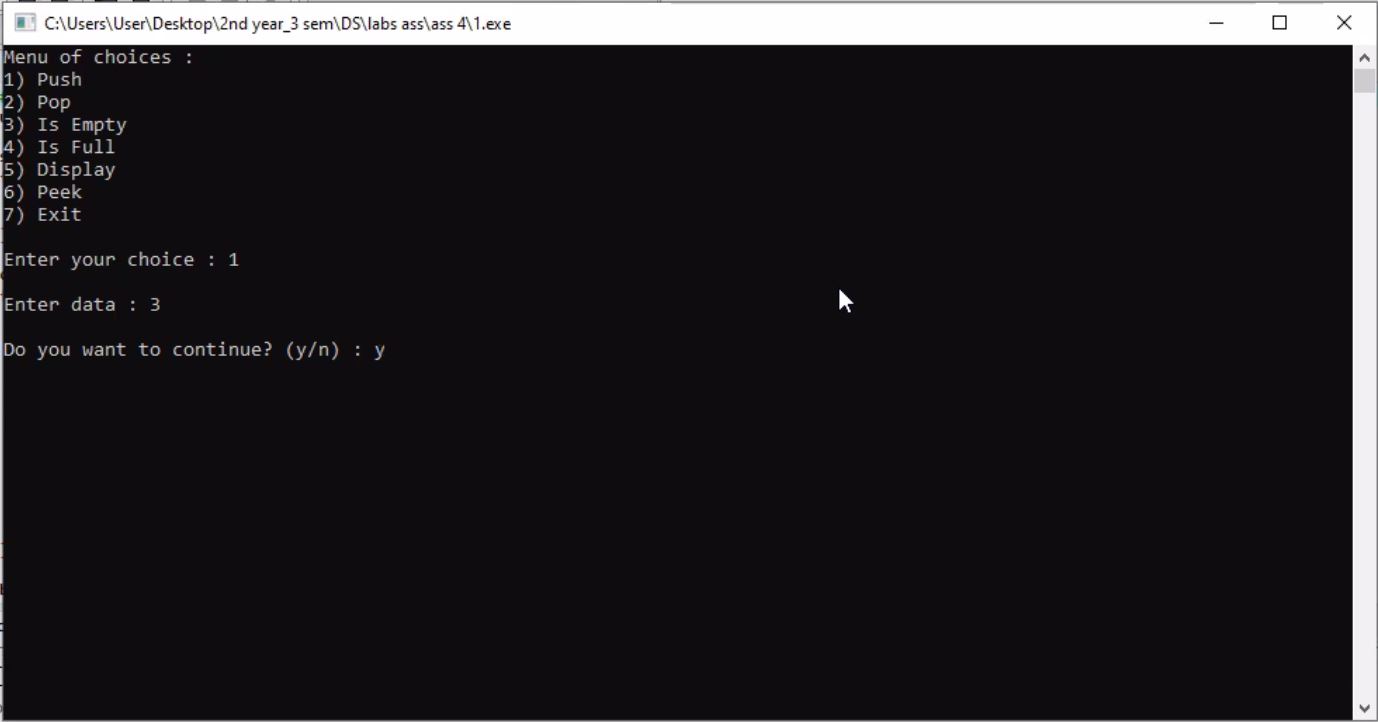
}while(c=='y'||c=='Y');

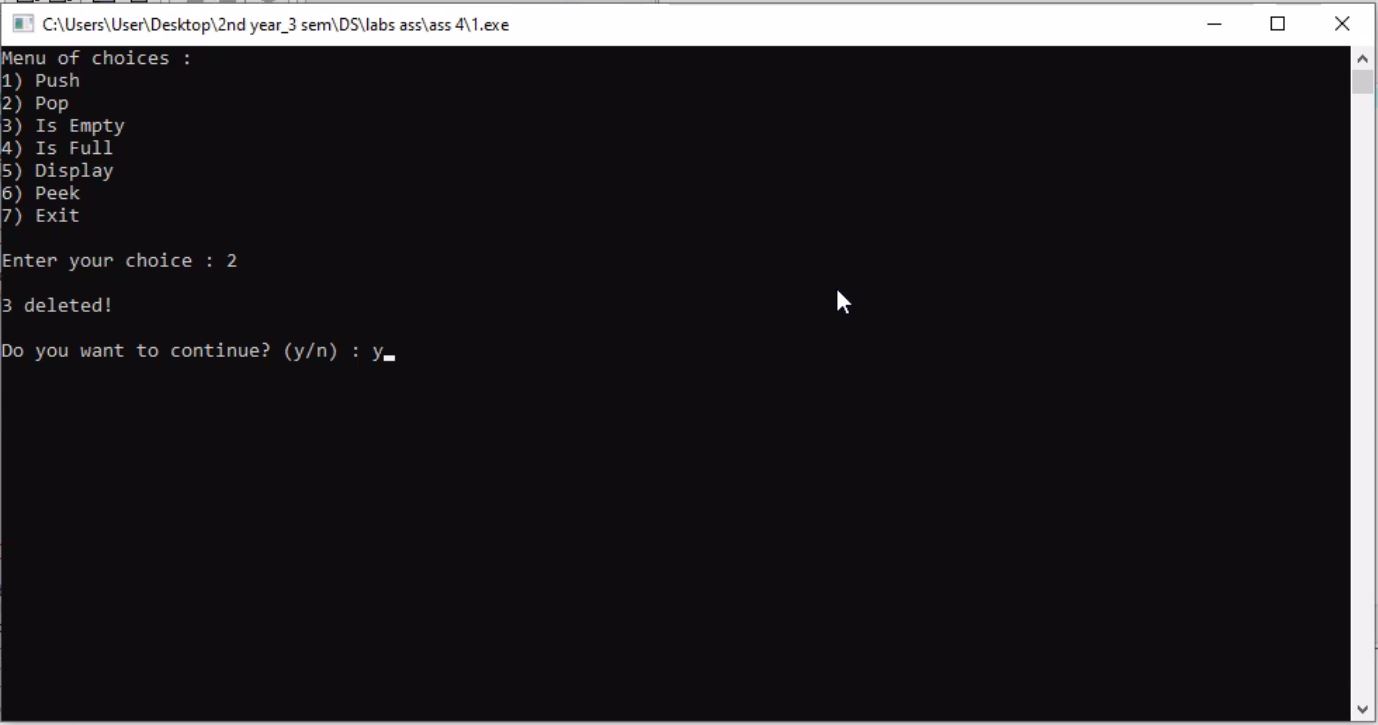
return 0;

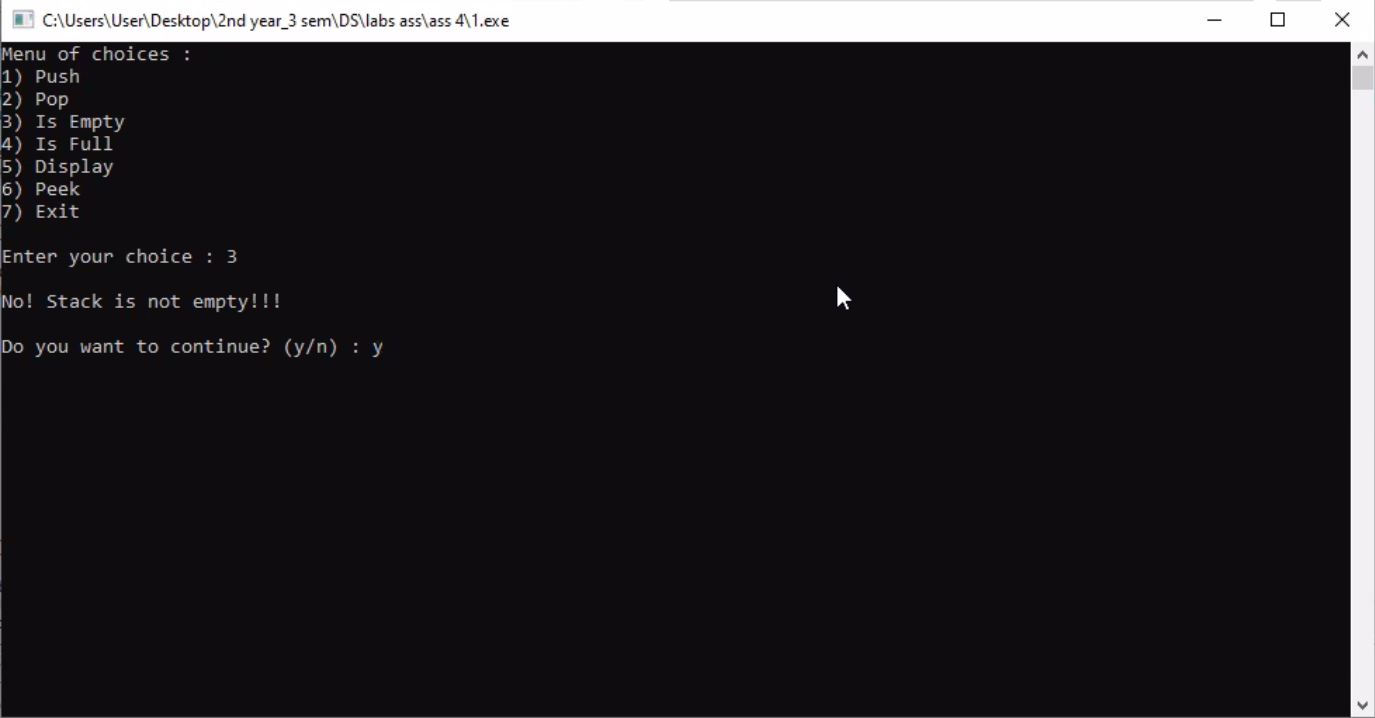
}

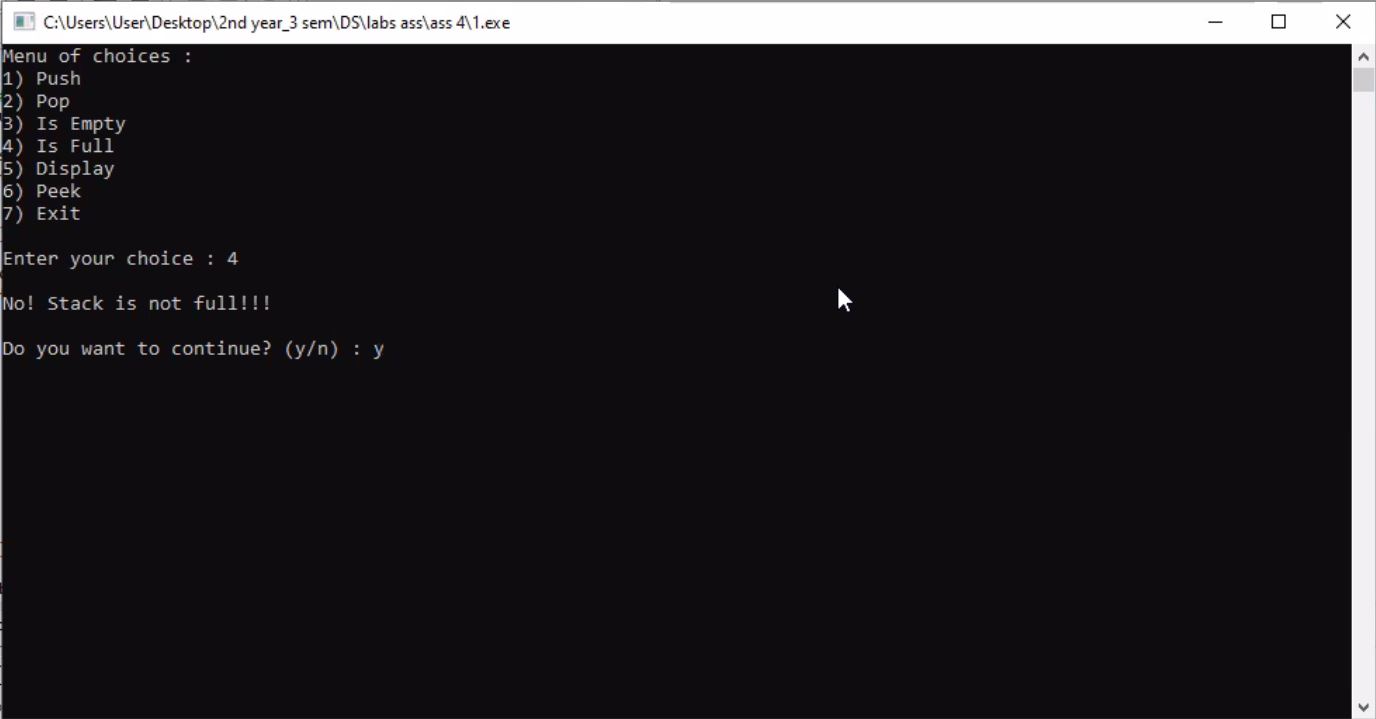
Output

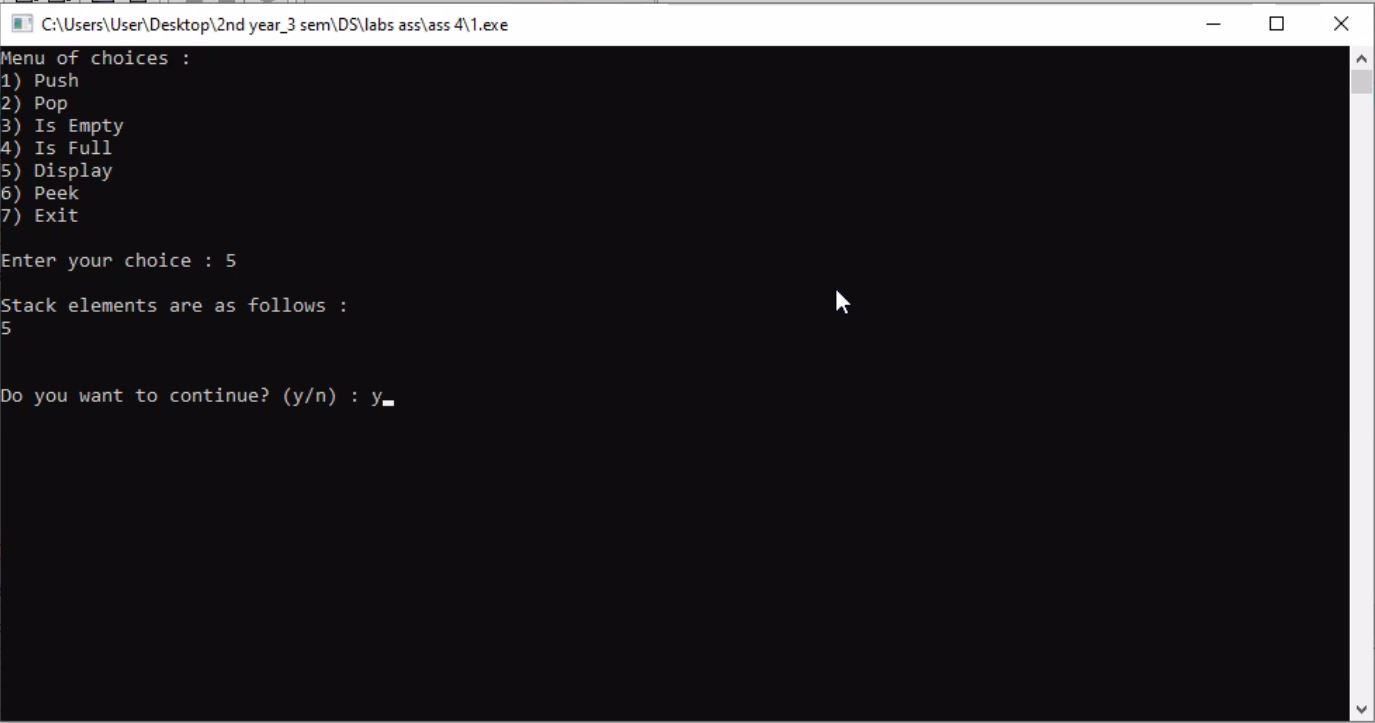


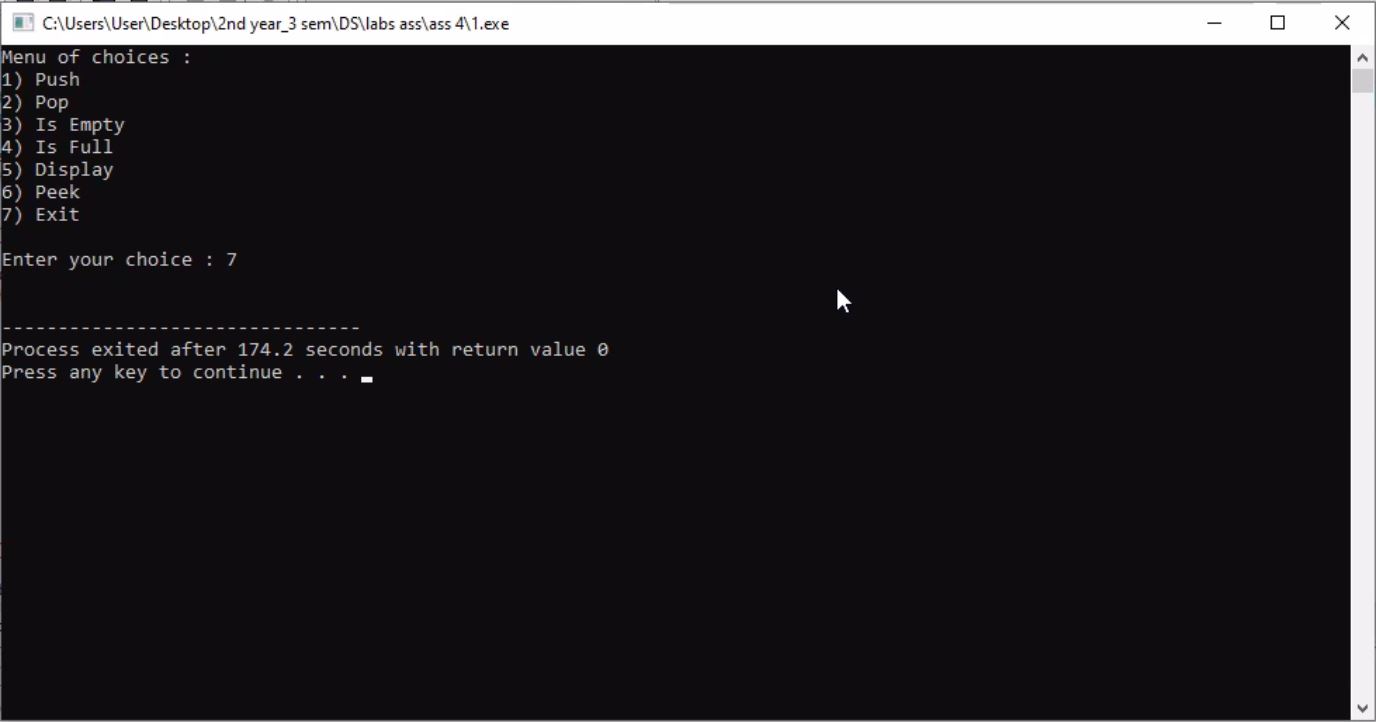
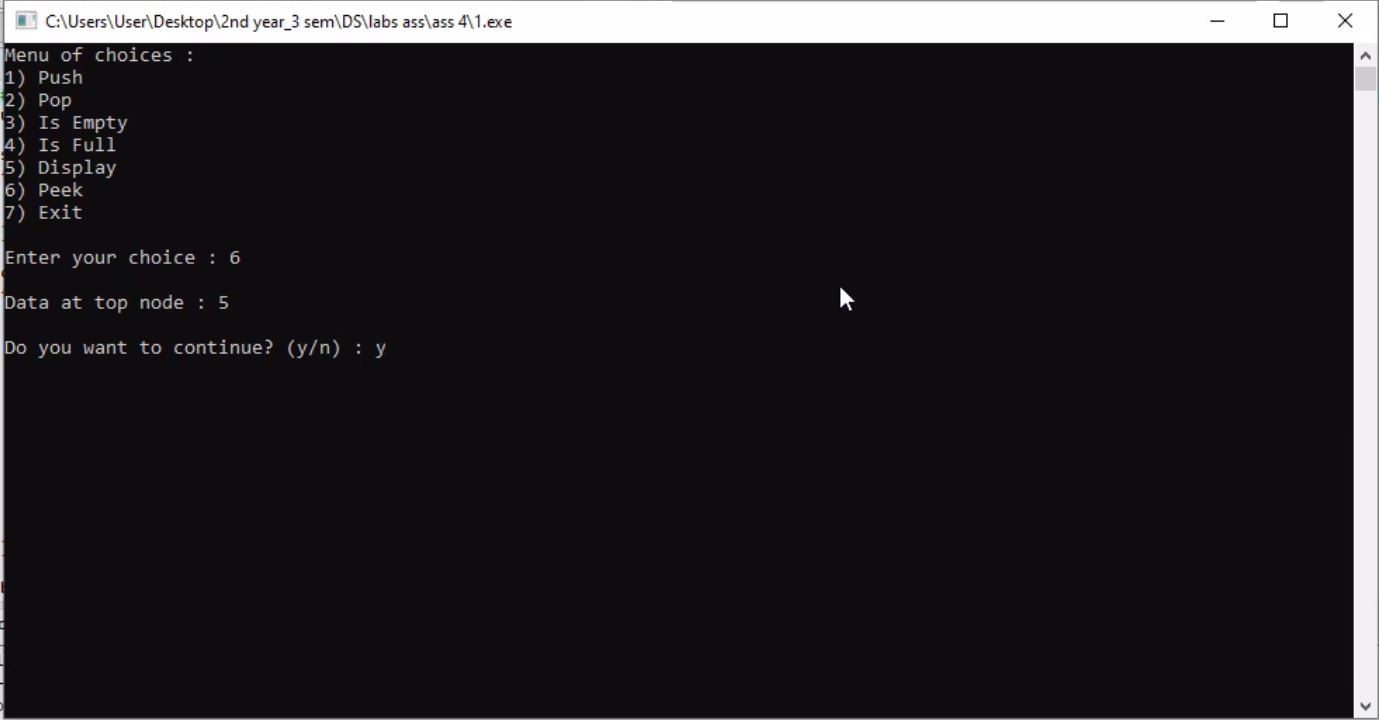












1. **Given a String, Reverse it using STACK. For example “data structure” should be output as “erutcurtsatad.”**

Sol 2

#include<iostream>

using namespace std;

struct node

{

char data;

node \*next;

};

class stack

{

node \*top;

public:

stack()

{

top=NULL;

}

bool isEmpty();

bool isFull();

void push(char a);

void pop();

char peek();

void display();

};

bool stack::isEmpty()

{

return (top==NULL);

}

bool stack::isFull()

{

node \*temp=new node;

if(temp==NULL)

{

delete temp;

return true;

}

else

{

delete temp;

return false;

}

}

void stack::push(char a)

{

if(isFull())

cout<<"Overflow! \n";

else

{

node \*temp=new node;

temp->data=a;

temp->next=top;

top=temp;

}

}

void stack::pop()

{

if(isEmpty())

cout<<"Underflow! \n";

else

{

node \*temp=top;

top=top->next;

cout<<temp->data<<" deleted! \n";

delete temp;

}

}

char stack::peek()

{

return top->data;

}

void stack::display()

{

if(isEmpty())

cout<<"Underflow! \n";

else

{

node \*temp=top;

while(temp!=NULL)

{

cout<<temp->data;

temp=temp->next;

}

cout<<endl;

}

}

int main()

{

stack s;

int i;

char str[20];

cout<<"Enter a string : ";

cin.getline(str,20);

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

s.push(str[i]);

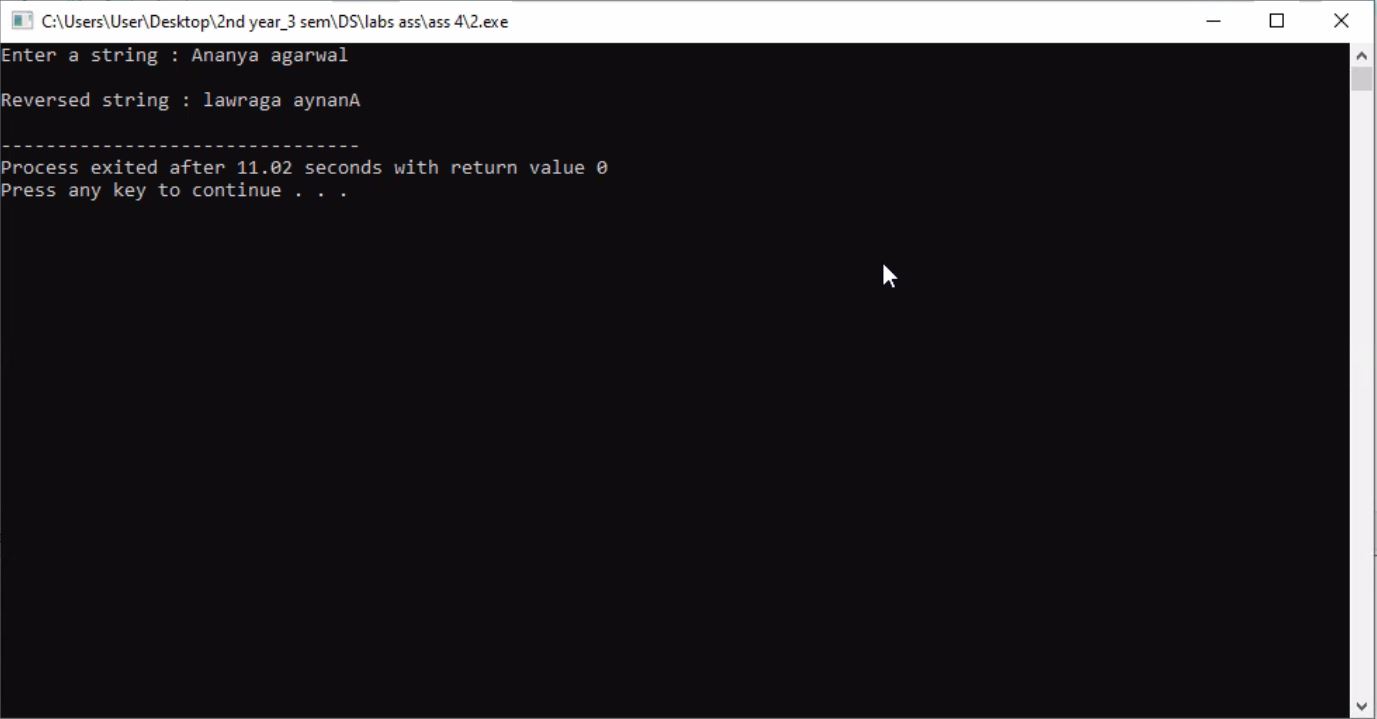
cout<<"\nReversed string : ";

s.display();

return 0;

}

Output



1. **Write a program that checks if an expression has balanced parentheses.**

Sol 3

#include<iostream>

using namespace std;

struct node

{

char data;

node \*next;

} ;

class stack

{

node \*top;

public:

stack()

{

top = NULL;

}

void push(char ch);

void pop();

char peek();

bool isEmpty();

};

void stack::push(char ch)

{

node \*temp = new node;

temp->data = ch;

temp->next = top;

top = temp;

}

void stack::pop()

{

if(!isEmpty())

{

node \*temp = top;

top = top->next;

delete temp;

}

}

char stack::peek()

{

if(!isEmpty())

return top->data;

}

bool stack::isEmpty()

{

if(top == NULL)

return true;

return false;

}

int main()

{

int i;

stack s;

string expr;

cout<<"Enter the expression : ";

getline(cin,expr);

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

{

if(expr[i] == ' ')

continue;

else if(s.isEmpty() == true)

s.push(expr[i]);

else if((s.peek() == '(' && expr[i] == ')') || (s.peek() == '[' && expr[i] == ']') || (s.peek() == '{' && expr[i] == '}'))

s.pop();

else

s.push(expr[i]);

}

if(s.isEmpty() == true)

cout<<"\nThe given expression is balanced!!!!\n";

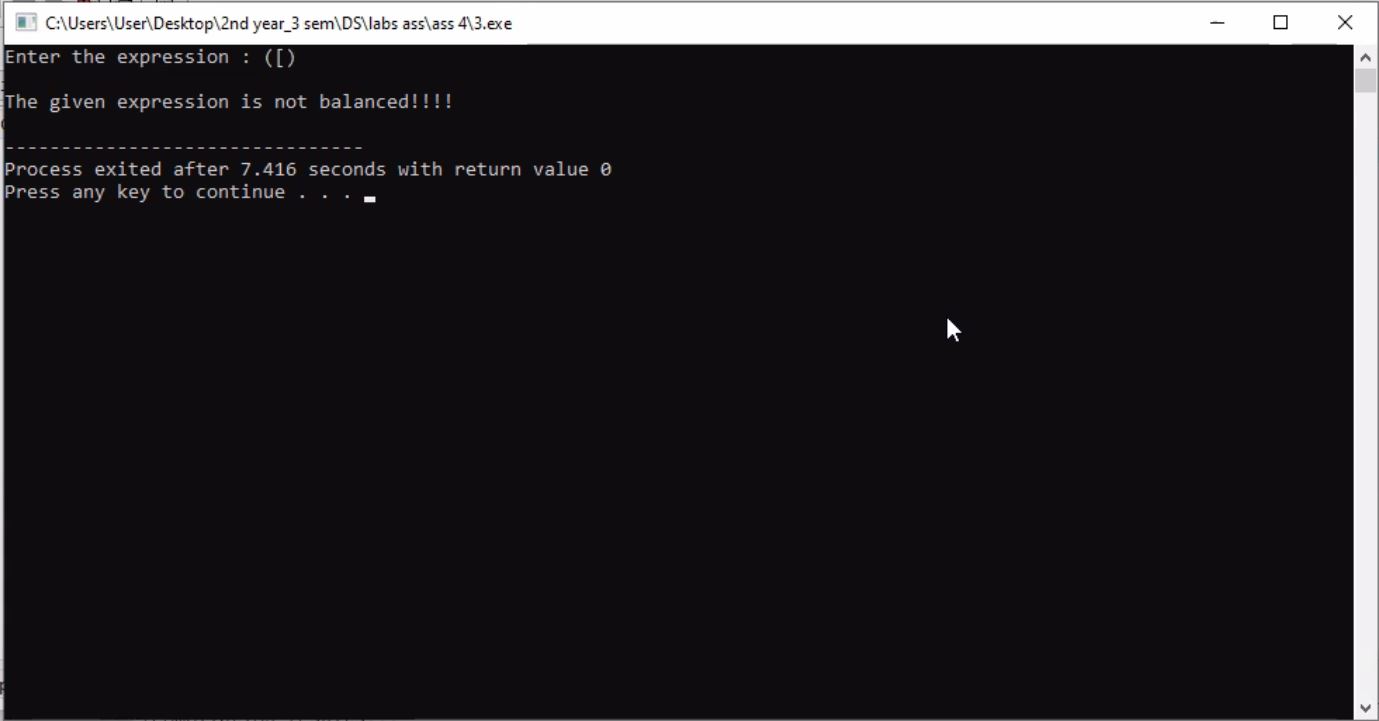
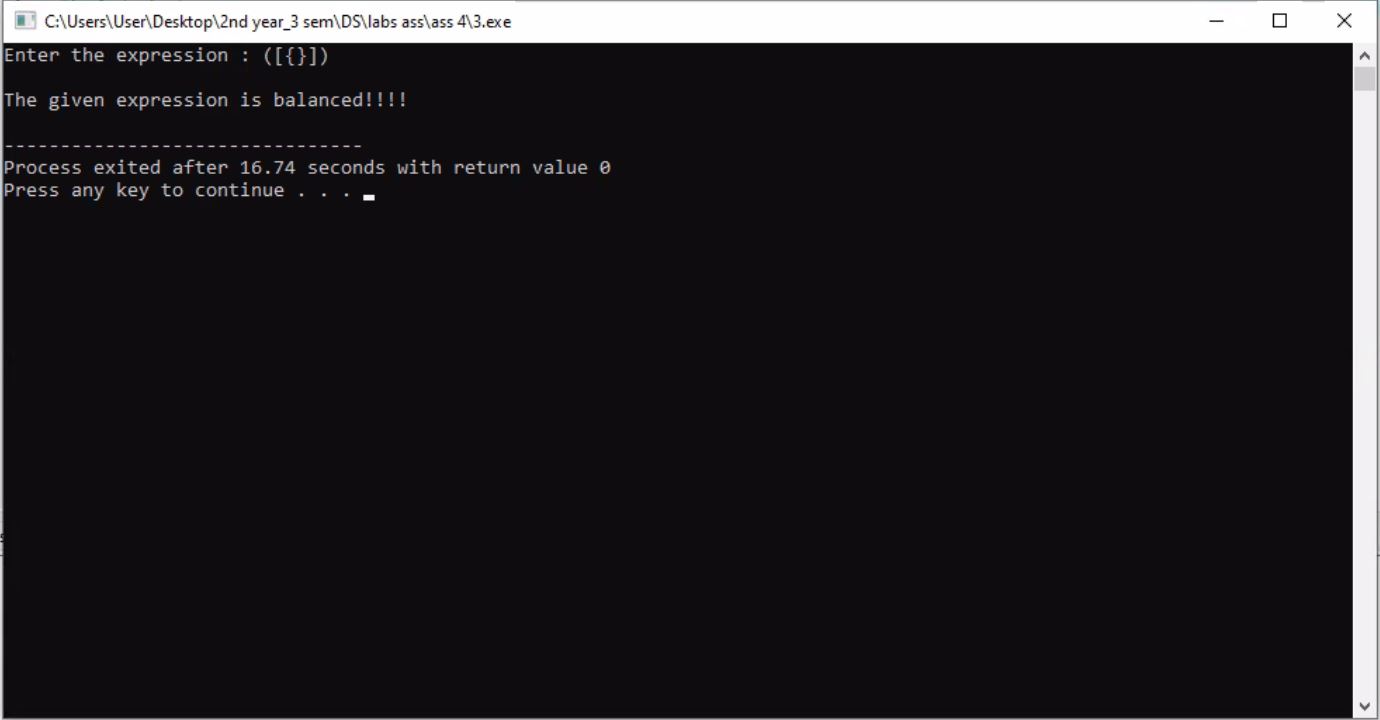
else

cout<<"\nThe given expression is not balanced!!!! \n";

return 0;

}

Output



1. **Write a program to convert an Infix expression into a Postfix expression.**

Sol 4

#include<iostream>

using namespace std;

struct node

{

char data;

node \*next;

};

class stack

{

node \*top;

public:

stack()

{

top = NULL;

}

void push(char d);

void pop();

char peek();

bool isEmpty();

};

void stack::push(char d)

{

node \*temp = new node;

temp->data = d;

temp->next = top;

top = temp;

}

void stack::pop()

{

if(top!=NULL)

{

node \*temp = top;

top = top->next;

delete temp;

}

}

char stack::peek()

{

if(top!=NULL)

{

return top->data;

}

}

bool stack::isEmpty()

{

if(top == NULL)

return true;

return false;

}

int priority(char a)

{

if(a == '^')

return 3;

else if((a == '\*') || (a == '/') || (a == '%'))

return 2;

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

return 1;

else

return 0;

}

int main()

{

int i;

stack s;

string expr;

string postfix;

cout<<"Enter the infix expression : ";

getline(cin,expr);

s.push('(');

expr.push\_back(')');

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

{

if(expr[i] == ' ')

continue;

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

s.push(expr[i]);

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

{

while(s.peek()!='(')

{

postfix.push\_back(s.peek());

postfix.push\_back(' ');

s.pop();

}

s.pop();

}

else if((expr[i] == '+') || (expr[i] == '-') || (expr[i] == '\*') || (expr[i] == '/') || (expr[i] == '%') || (expr[i] == '^'))

{

while((priority(expr[i]) <= priority(s.peek())) && (s.peek()!='('))

{

postfix.push\_back(s.peek());

postfix.push\_back(' ');

s.pop();

}

s.push(expr[i]);

}

else

{

postfix.push\_back(expr[i]);

postfix.push\_back(' ');

}

}

if(s.isEmpty())

cout<<"\nPostfix Expression : "<<postfix<<endl;

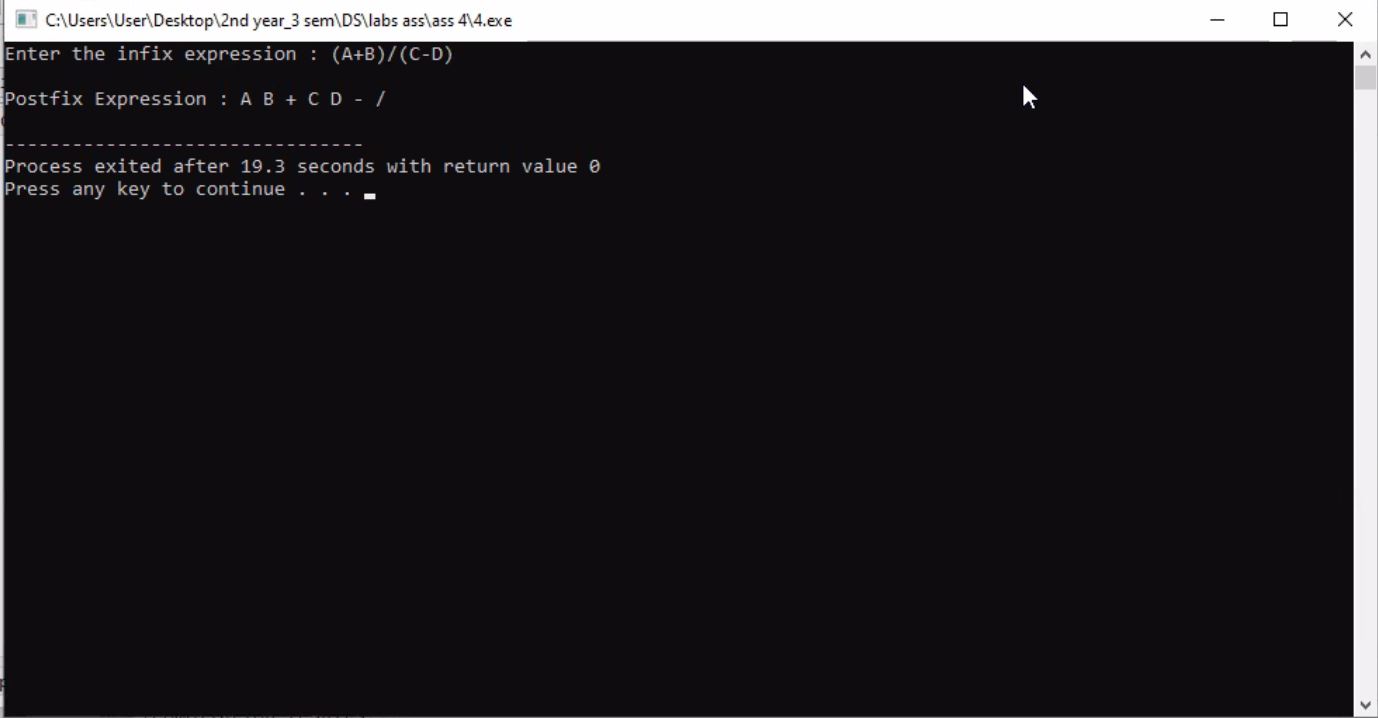
else

cout<<"\nInvalid Expression! \n";

return 0;

}

Output



1. **Write a program for the evaluation of a Postfix expression.**

Sol 5

#include<iostream>

#include<math.h>

using namespace std;

struct node

{

int data;

node \*next;

};

class stack

{

node \*top;

public:

stack()

{

top = NULL;

}

void push(char d);

void pop();

char peek();

bool isEmpty();

};

void stack::push(char d)

{

node \*temp = new node;

temp->data = d;

temp->next = top;

top = temp;

}

void stack::pop()

{

if(top!=NULL)

{

node \*temp = top;

top = top->next;

delete temp;

}

}

char stack::peek()

{

if(top!=NULL)

{

return top->data;

}

}

bool stack::isEmpty()

{

if(top == NULL)

return true;

return false;

}

int EvaluatePostfix(string expr);

int PerformOperation(char operation, int operand1, int operand2);

bool IsOperator(char C);

bool IsNumericDigit(char C);

int main()

{

string expr;

cout<<"Enter Postfix Expression \n";

getline(cin,expr);

int result = EvaluatePostfix(expr);

cout<<"Output = "<<result<<"\n";

}

int EvaluatePostfix(string expr)

{

int i;

stack s;

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

{

if(expr[i] == ' ' || expr[i] == ',')

continue;

else if(IsOperator(expr[i]))

{

// Pop two operands

int operand2 = s.peek(); s.pop();

int operand1 = s.peek(); s.pop();

// Perform operation

int result = PerformOperation(expr[i], operand1, operand2);

s.push(result);

}

else if(IsNumericDigit(expr[i]))

{

int operand = 0;

while(expr[i]!='\0' && IsNumericDigit(expr[i]))

{

operand = (operand\*10) + (expr[i] - '0');

i++;

}

i--;

s.push(operand);

}

}

return s.peek();

}

bool IsNumericDigit(char C)

{

if(C >= '0' && C <= '9')

return true;

return false;

}

bool IsOperator(char C)

{

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

return true;

return false;

}

int PerformOperation(char operation, int operand1, int operand2)

{

if(operation == '+') return operand1 +operand2;

else if(operation == '-') return operand1 - operand2;

else if(operation == '\*') return operand1 \* operand2;

else if(operation == '/') return operand1 / operand2;

else if(operation == '^') return pow(operand1,operand2);

else cout<<"Unexpected Error \n";

return -1;

}

Output

