**Write a C++ program to demonstrate working of Stack using Array.**

**Code:**

**OperationsOnStack.cpp**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

#define STACKSIZE 5

using namespace std;

class stack{

int arr[STACKSIZE];

int stack\_top;

public:

stack(){

stack\_top = -1;

}

void push(int val){

stack\_top = stack\_top+1;

arr[stack\_top] = val;

}

int pop(){

int val;

val = arr[stack\_top];

stack\_top--;

return val;

}

bool isEmpty(){

if(stack\_top == -1){

return true;

}else{

return false;

}

}

bool isFull(){

if(stack\_top == STACKSIZE-1){

return true;

}else{

return false;

}

}

int size(){

return stack\_top+1;

}

void display(){

if(stack\_top == -1){

cout << "Not any Element present in the Stack";

}

else{

cout << "Elements in the Stack is: ";

for(int i = 0 ; i <= stack\_top; i++){

cout << arr[i] << " ";

}

cout << endl;

}

}

};

int main(){

stack myStack;

int val;

int choice;

while(1){

cout <<"\n Press:\n 1.Push\n 2.Pop\n 3.Display\n 4.Exit";

cout << "\nEnter the Choice: ";

cin >> choice;

switch(choice){

case 1:

if(myStack.isFull() == false){

cout << "\nEnter Value:";

cin >> val;

myStack.push(val);

}else{

cout << "\nStack is Full.";

}

break;

case 2:

if(myStack.isEmpty() == false){

val = myStack.pop();

cout << "\nValue is " << val;

}

else{

cout << "\nStack is Empty.";

}

break;

case 3:

myStack.display();

break;

case 4:

exit(1);

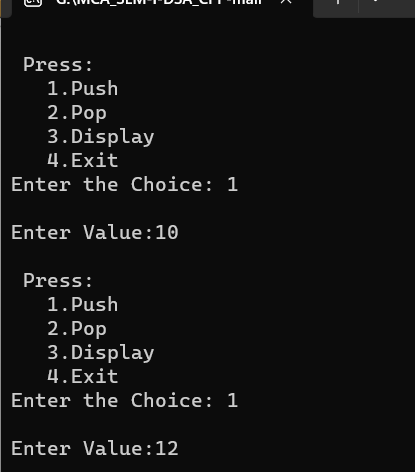
}

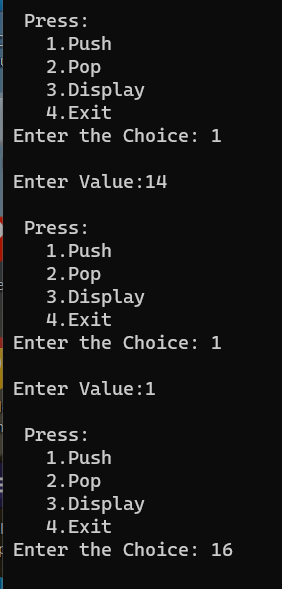
}

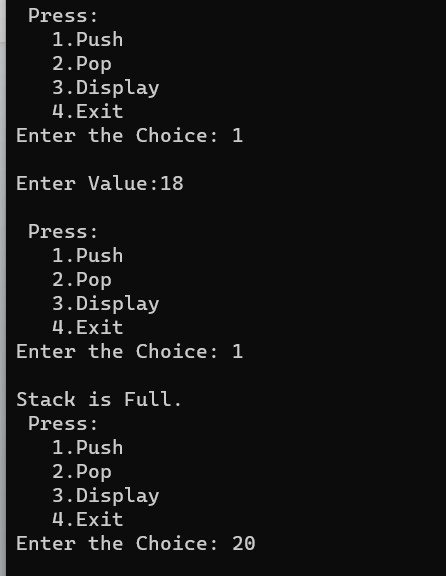
return 0;

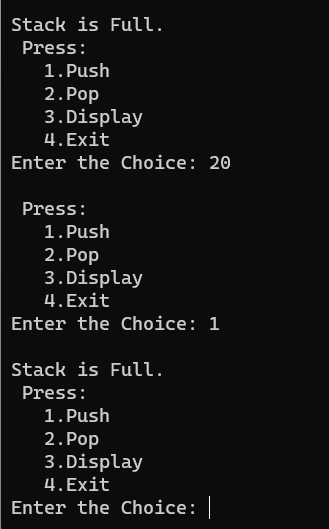
}

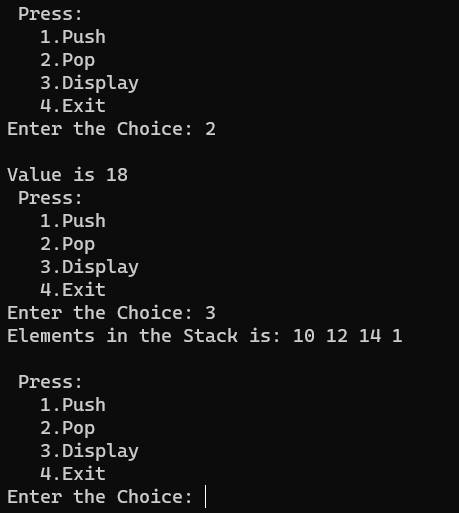
**Output:**











**Write a C++ program to demonstrate working of Stack using Linked List.**

**Code:**

#include <bits/stdc++.h>

#include<iostream>

#include<conio.h>

#include<stdlib.h>

using namespace std;

class Node {

public:

int data;

Node\* link;

Node(int n)

{

this->data = n;

this->link = NULL;

}

};

class Stack {

Node\* top;

public:

Stack() { top = NULL; }

void push(int data)

{

Node\* temp = new Node(data);

if (!temp) {

cout << "\nStack Overflow";

exit(1);

}

temp->data = data;

temp->link = top;

top = temp;

}

bool isEmpty()

{

return top == NULL;

}

int peek()

{

if (!isEmpty())

return top->data;

else

exit(1);

}

void pop()

{

Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow" << endl;

exit(1);

}

else {

temp = top;

top = top->link;

free(temp);

}

}

void display()

{

Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow";

exit(1);

}

else {

temp = top;

while (temp != NULL) {

cout << temp->data;

temp = temp->link;

if (temp != NULL)

cout << " -> ";

}

}

}

};

int main()

{

Stack s;

s.push(11);

s.push(22);

s.push(33);

s.push(44);

s.display();

cout << "\nTop element is " << s.peek() << endl;

s.pop();

s.pop();

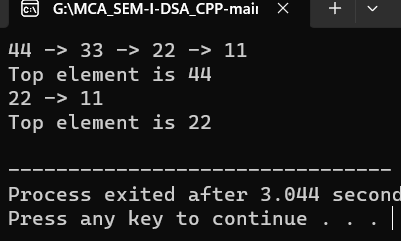
s.display();

cout << "\nTop element is " << s.peek() << endl;

return 0;

}

**Output:**



**Demonstrate application of stack “evaluation of postfix expression”.**

**Code:**

#include<iostream>

#include<stack>

using namespace std;

bool isOperator(char x){

switch(x){

case '+':

case '-':

case '\*':

case '/':

return true;

}

return false;

}

string prefixConvertToPostfix(string prefix){

stack <string> expression;

int length = prefix.size();

for(int i = length-1; i >= 0; i--){

if(isOperator(prefix[i])){

string op1 = expression.top();

expression.pop();

string op2 = expression.top();

expression.pop();

string temp = op1 + op2 + prefix[i];

expression.push(temp);

}

else{

expression.push(string(1,prefix[i]));

}

}

return expression.top();

}

int main(){

string prefix = "+/ab/cd";

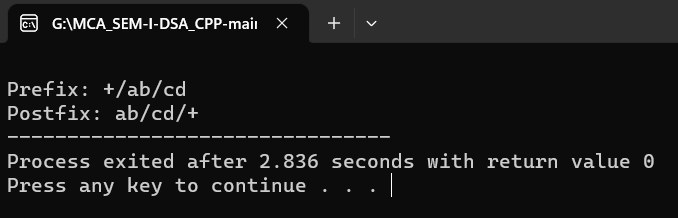
cout << "\nPrefix: " << prefix;

cout << "\nPostfix: " << prefixConvertToPostfix(prefix);

return 0;

}

**Output:**



**Demonstrate application of stack “balancing of parenthesis”.**

**Code:**

#include<iostream>

#include<conio.h>

#include<stdlib.h>

#define STACKSIZE 5

using namespace std;

class stack{

int arr[STACKSIZE];

int stack\_top;

public:

stack(){

stack\_top = -1;

}

void push(int val){

stack\_top = stack\_top+1;

arr[stack\_top] = val;

}

int pop(){

int val;

val = arr[stack\_top];

stack\_top--;

return val;

}

bool isEmpty(){

if(stack\_top == -1){

return true;

}else{

return false;

}

}

bool isFull(){

if(stack\_top == STACKSIZE-1){

return true;

}else{

return false;

}

}

int size(){

return stack\_top+1;

}

void display(){

if(stack\_top == -1){

cout << "Not any Element present in the Stack";

}

else{

cout << "Elements in the Stack is: ";

for(int i = 0 ; i <= stack\_top; i++){

cout << arr[i] << " ";

}

cout << endl;

}

}

};

int main(){

stack myStack;

char exp[] = {'<','{','[',']','(',')',']','}','>','\0'};

int size = 0;

int op1, op2, res, i;

int isValid = 1;

char test;

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

test = exp[i];

char comp;

if(test == '{' || test == '<' || test == '[' || test == '(' ){

myStack.push(test);

}

else{

if(myStack.isEmpty() == true){

isValid = 0;

break;

}

else{

comp = myStack.pop();

if(test == '}' && comp != '{'){

isValid = 0;

break;

}

if(test == '>' && comp != '<'){

isValid = 0;

break;

}

if(test == ']' && comp != '['){

isValid = 0;

break;

}

if(test == ')' && comp != '('){

isValid = 0;

break;

}

}

}

}

if(isValid == 1 && myStack.isEmpty() == true){

cout << "\nValid Parenthesis!!!";

}else{

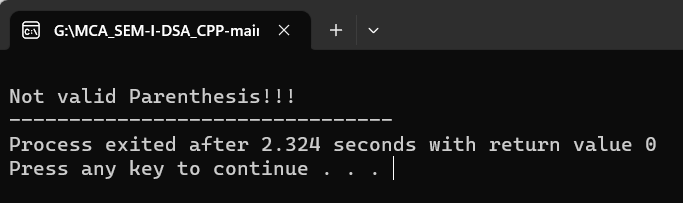
cout << "\nNot valid Parenthesis!!!";

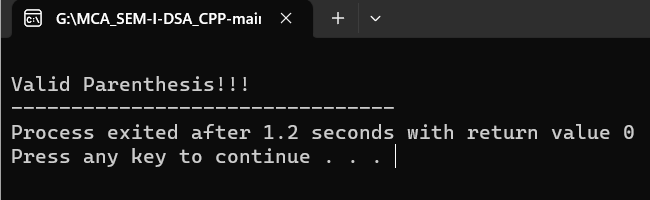
}

return 0;

}

**Output:**





**Write Linked List code with functions perform on it.**

**Code:**

#include<iostream>

#include<bits/stdc++.h>

#include<stdlib.h>

using namespace std;

struct node

{

int data;

struct node\* next;

};

struct node\* head=NULL;

struct node\* current=NULL;

void printList(){

struct node\* p=head;

cout<<"\n [";

while(p!=NULL){

cout<<" "<<p->data<<" ";

p=p->next;

}

cout<<"]";

}

//count

int countNode(){

struct node\* temp = head;

int i = 0;

while(temp != NULL){

i++;

temp = temp -> next;

}

return i;

}

//insert

void insertatBegin(int data){

struct node\* lk=(struct node\*)malloc(sizeof(struct node));

lk->data=data;

lk->next=head;

head=lk;

}

void insertLast(int data){

struct node\* temp = new node;

temp -> data = data;

temp -> next = NULL;

struct node\* trav = head;

if(trav != NULL){

while(trav -> next != NULL){

trav = trav -> next;

}

trav -> next = temp;

}

else{

head = temp;

}

}

void insertAt(int pos, int data){

struct node\* temp = new node;

struct node\* trav = head;

int k = 1;

int cnt = countNode();

if(head == NULL){

insertatBegin(data);

}

else{

if(pos > cnt){

cout << "Wrong Position!";

}

else{

while(k== pos-1){

trav = trav -> next;

k++;

}

temp -> data = data;

temp -> next = trav->next;

trav->next = temp;

}

}

}

//remove

void removeFirst(){

head=head->next;

}

void removeAt(int pos){

int node\_cnt = countNode();

if(node\_cnt < pos){

cout << "Wrong Position!";

}

else{

struct node\* trav = head;

int k = 1;

while(k < pos-1){

trav = trav -> next;

k++;

}

struct node\* temp = trav -> next;

trav->next = temp->next;

}

}

void removeLast(){

struct node\* trav = head;

if(trav->next == NULL){

head = NULL;

}

else{

while(trav->next->next != NULL){

trav = trav -> next;

}

struct node\* temp = trav->next;

trav -> next = NULL;

temp = NULL;

}

}

//reverse

void reverseList(struct node\*\* head){

struct node\* prev=NULL,\*cur=\*head,\*tmp;

while(cur!=NULL){

tmp=cur->next;

cur->next=prev;

prev=cur;

cur=tmp;

}

\*head=prev;

}

//sort

void sort(){

struct node\*sort\_head = head;

struct node\* trav = head;

while(trav !=NULL){

if(trav->data < sort\_head->data){

int temp = trav->data;

trav->data= sort\_head->data;

sort\_head->data =temp;

}

trav = trav->next;

}

sort\_head = sort\_head->next;

}

//searching

void search(int key){

struct node\* trav = head;

bool flag = true;

while(trav != NULL && flag == true){

if(trav -> data == key){

flag = false;

break;

}

else{

trav = trav -> next;

}

}

if(flag == false){

cout << "Element Found!";

}

else{

cout << "Element Not Found!";

}

}

int main(){

insertatBegin(12);

insertatBegin(22);

insertatBegin(30);

insertatBegin(44);

insertatBegin(50);

cout<<"\n Linked list: ";

printList();

insertLast(10);

cout<<"\nSearching Element in the Linked list: ";

int n;

cin >> n;

search(n);

cout<<"\n Insert At Last Linked list: ";

printList();

insertAt(2,90);

cout<<"\n Insert At Linked list: ";

printList();

removeFirst();

cout<<"\n After Remove first Node Linked list: ";

printList();

removeLast();

cout<<"\n After Remove Last Node Linked list: ";

printList();

removeAt(3);

cout<<"\n After Remove At position Node Linked list: ";

printList();

reverseList(&head);

insertAt(1,10);

cout<<"\n Insert At Linked list: ";

printList();

cout<<"\n Reversed Linked list: ";

printList();

sort();

cout<<"\n Sorted Link List";

printList();

cout << "\n Count of the Node is " << countNode();

/\*cout<<"\nSearching Element in the Linked list: ";

int n;

cin >> n;

search(n);

\*/

}

**Output:**

