**Question#1,2,3**

**(a)**

**Code:**

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

class base {

protected:

int\* stackArray;

int stackSize;

int top;

public:

base(int size) {

stackArray = new int[size];

stackSize = size;

top = 0;

}

virtual bool isFull() = 0;

virtual bool isEmpty() = 0;

virtual void push(int val) = 0;

virtual void pop(int& val) = 0;

virtual int Top() = 0;

virtual int getMin() = 0;

virtual void reverse() = 0;

};

class IntStack : public base {

public:

IntStack(int size) :base(size){}

~IntStack() {

delete[] stackArray;

}

bool isFull() {

if (stackSize == top) {

return true;

}

return false;

}

bool isEmpty() {

if (top == 0) {

return true;

}

return false;

}

void push(int val) {

stackArray[top] = val;

top++;

}

void pop(int& val) {

if (isEmpty() == true) {

cout << "Error" << endl;

}

else {

top--;

val = stackArray[top];

}

}

int Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return -1;

}

return (stackArray[top]);

}

int getMin() {

int temp = 0;

pop(temp);

int min = temp;

while (!isEmpty()) {

pop(temp);

if (temp < min) {

min = temp;

}

}

return min;

}

void reverse() {

int\* temp = new int[stackSize];

int flag = 0;

while (top != 0) {

pop(temp[flag]);

flag++;

}

top = flag;

stackArray = temp;

}

};

int main() {

IntStack obj(10);

for (int i = 1; i <= 10; i++) {

obj.push(i);

}

cout << "Reverse : " << endl;

obj.reverse();

int temp = 0;

for (int i = 0; i < 10; i++) {

obj.pop(temp);

cout << "[" << temp << "]";

}

cout << endl;

for (int i = 1; i <= 10; i++) {

obj.push(i);

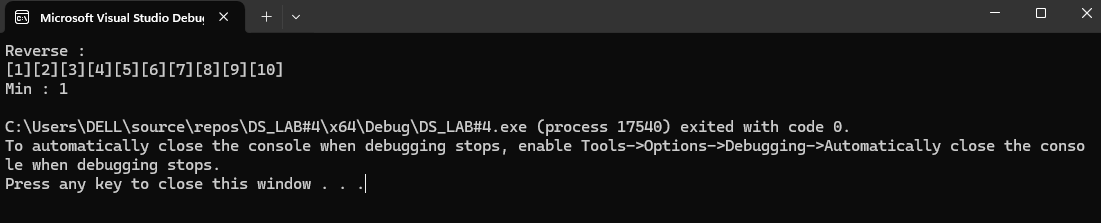
}

cout << "Min : " << obj.getMin() << endl;

return 0;

}

**Output:**

****

**(b)**

**Code:**

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

struct node {

int data;

node\* prev;

node() :data(0), prev(nullptr) {}

};

class base {

protected:

node\* top;

public:

virtual bool isEmpty() = 0;

virtual void push(int val) = 0;

virtual void pop(int& val) = 0;

virtual int Top() = 0;

virtual int getMin() = 0;

};

class IntStack : public base {

public:

IntStack() {

top = nullptr;

}

~IntStack() {

if (top != nullptr) {

node\* temp = top->prev;

while (top->prev != nullptr) {

temp = top->prev;

delete top;

top = temp;

}

}

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

void push(int val) {

node\* temp = new node;

temp->data = val;

temp->prev = top;

top = temp;

}

void pop(int& val) {

if (isEmpty() == true) {

cout << "Empty" << endl;

}

else {

node\* temp = top->prev;

val = top->data;

delete top;

top = temp;

}

}

int Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return -1;

}

return (top->data);

}

int getMin() {

int temp = 0;

pop(temp);

int min = temp;

while (!isEmpty()) {

pop(temp);

if (temp < min) {

min = temp;

}

}

return min;

}

};

int main() {

IntStack obj;

for (int i = 10; i >= 1; i--) {

obj.push(i);

}

int temp = 0;

cout << "Min : " << obj.getMin() << endl;

for (int i = 10; i >= 1; i--) {

obj.push(i);

}

for (int i = 0; i < 10; i++) {

obj.pop(temp);

cout << "[" << temp << "]";

}

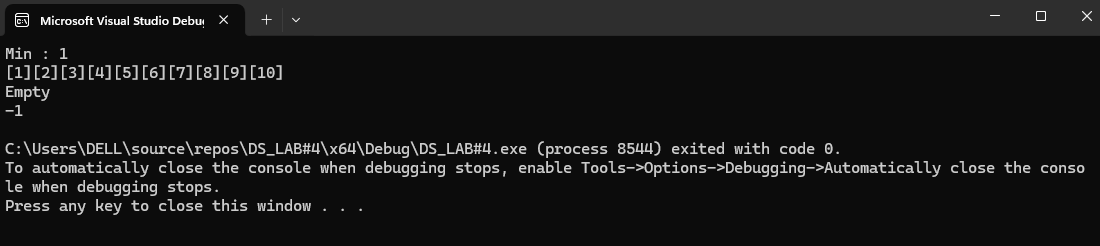
cout << endl;

cout << obj.Top() << endl;

return 0;

}

**Output:**

****

**Questio#4**

**Code:**

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<string>

using namespace std;

struct node {

char data;

node\* prev;

node() :data(' '), prev(nullptr) {}

};

class base {

protected:

node\* top;

public:

virtual bool isEmpty() = 0;

virtual void push(char val) = 0;

virtual void pop(char& val) = 0;

virtual char Top() = 0;

virtual void balanced() = 0;

};

class Stack : public base {

public:

Stack() {

top = nullptr;

}

~Stack() {

if (top != nullptr) {

node\* temp = top->prev;

while (top->prev != nullptr) {

temp = top->prev;

delete top;

top = temp;

}

}

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

void push(char val) {

node\* temp = new node;

temp->data = val;

temp->prev = top;

top = temp;

}

void pop(char& val) {

if (isEmpty() == true) {

cout << "Empty" << endl;

}

else {

node\* temp = top->prev;

val = top->data;

delete top;

top = temp;

}

}

char Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return ' ';

}

return (top->data);

}

void balanced() {

char temp = ' ';

Stack obj;

bool isBalanced = true;

while (top != nullptr) {

if (Top() == ')' || Top() == '}' || Top() == ']') {

pop(temp);

cout << "push : " << temp << endl;

obj.push(temp);

}

else if (Top() == '(' || Top() == '{' || Top() == '[') {

pop(temp);

cout << "matching temp : " << temp << " : " << obj.Top() << endl;

if (obj.Top() == ')' && temp == '(') {

obj.pop(temp);

}

else if (obj.Top() == '}' && temp == '{') {

obj.pop(temp);

}

else if (obj.Top() == ']' && temp == '[') {

obj.pop(temp);

}

else {

isBalanced = false;

}

}

else {

pop(temp);

}

}

if (isBalanced == true) {

cout << "Balanced" << endl;

}

else {

cout << "Not Balanced" << endl;

}

}

};

int main() {

Stack obj;

string str = "{[()()]}";

for (int i = 0; i < str.length(); i++) {

obj.push(str[i]);

}

cout << str << endl;

char temp = ' ';

cout << endl;

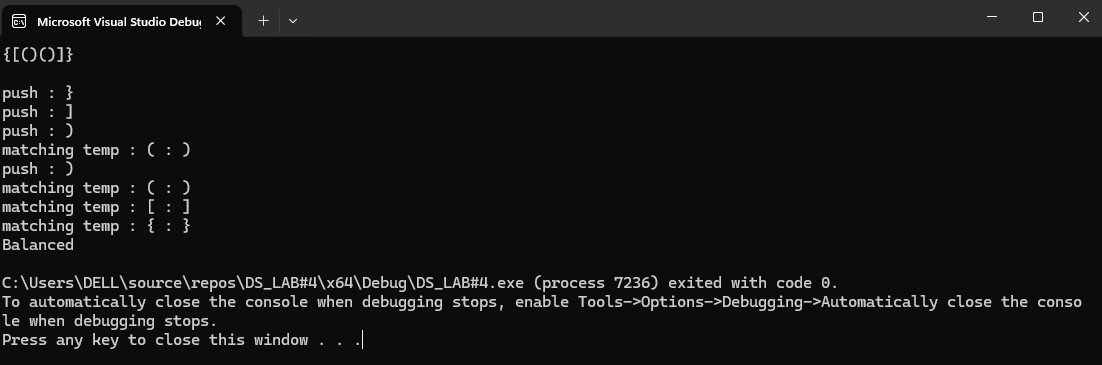
obj.balanced();

//cout << obj.Top() << endl;

return 0;

}

**Output:**

****

**Question#5**

**Code:**

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<string>

using namespace std;

struct node {

string data;

node\* prev;

node() :data(" "), prev(nullptr) {}

};

class Stack {

protected:

node\* top;

public:

Stack() {

top = nullptr;

}

~Stack() {

if (top != nullptr) {

node\* temp = top->prev;

while (top->prev != nullptr) {

temp = top->prev;

delete top;

top = temp;

}

}

}

bool isEmpty() {

if (top == nullptr) {

return true;

}

return false;

}

void push(string val) {

node\* temp = new node;

temp->data = val;

temp->prev = top;

top = temp;

}

void pop(string& val) {

if (isEmpty() == true) {

cout << "Empty" << endl;

}

else {

node\* temp = top->prev;

val = top->data;

delete top;

top = temp;

}

}

string Top() {

if (isEmpty() == true) {

cout << "Empty" << endl;

return " ";

}

return (top->data);

}

void evaluatePostfix() {

string temp = " ";

Stack operators;

Stack obj;

int ans = 0;

int digit1 = 0;

int digit2 = 0;

while (top != nullptr) {

pop(temp);

cout << "Temp : " << temp << endl;

if (temp == "+" || temp == "-" || temp == "\*" || temp == "/" || temp == "%") {

operators.push(temp);

}

else if (temp >= "0" && temp <= "9") {

while (top != nullptr) {

digit2 = stoi(temp);

cout << "Digit#2 : " << digit2 << endl;

pop(temp);

digit1 = stoi(temp);

cout << "Digit#1 : " << digit1 << endl;

operators.pop(temp);

cout << "Operator : " << temp << endl;

if (temp == "+") {

ans = digit1 + digit2;

cout << "Ans : " << ans << endl;

push(to\_string(ans));

}

else if (temp == "-") {

ans = digit1 - digit2;

cout << "Ans : " << ans << endl;

push(to\_string(ans));

}

else if (temp == "\*") {

ans = digit1 \* digit2;

cout << "Ans : " << ans << endl;

push(to\_string(ans));

}

else if (temp == "/") {

ans = digit1 / digit2;

cout << "Ans : " << ans << endl;

push(to\_string(ans));

}

else if (temp == "%") {

ans = digit1 % digit2;

cout << "Ans : " << ans << endl;

push(to\_string(ans));

}

}

}

else {

cout << "Invalid Expression" << endl;

break;

}

}

pop(temp);

cout << "Ans : " << temp << endl;

}

};

int main() {

Stack obj;

string str = "5 6 2 + \* 12 4 / -";

string temp = " ";

cout << "Length : " << str.length() << endl;

for (int i = 0; i < str.length(); i++) {

temp = str[i];

int count = 1;

bool flag = false;

if (temp != " ") {

while (str[i + count] != ' ' && i < str.length() - 1) {

temp = temp + str[i + count];

count++;

flag = true;

}

if (flag == true) {

i++;

}

obj.push(temp);

cout << temp << " ";

}

}

cout << endl;

cout << str << endl;

//cout << "Min : " << obj.getMin() << endl;

/\*for (int i = 0; i < 10; i++) {

obj.pop(temp);

cout << "[" << temp << "]";

}\*/

cout << endl;

obj.evaluatePostfix();

//obj.balanced();

//cout << obj.Top() << endl;

return 0;

}