**Name: Talha Abdullah Bangyal**

**Roll number: 22F-3194**

**Section: BS(AI)-3A**

**Lab: 4**

**--------------------------------------------------------------------**

**Task 1 (a)**

#include<iostream>

using namespace std;

class IntStack {

private:

int\* StackArray;

int StackSize;

int Top;

public:

IntStack(int);

~IntStack();

bool push(int);

bool pop(int&);

bool isFull();

bool isempty();

int top();

void printStack();

};

IntStack::IntStack(int size) {

StackArray = new int[size];

StackSize = size;

Top = -1;

}

IntStack::~IntStack() {

delete[] StackArray;

}

bool IntStack::push(int val) {

if (isFull()) {

cout << "The stack is full." << endl;

return false;

}

Top++;

StackArray[Top] = val;

return true;

}

bool IntStack::pop(int& val) {

if (isempty())

{

cout << "The stack is empty.\n";

return false;

}

val = StackArray[Top];

Top--;

return true;

}

bool IntStack::isFull() {

if (Top == StackSize - 1)

return true;

else

return false;

}

bool IntStack::isempty() {

return (Top == -1);

}

int IntStack::top() {

if (isempty())

{

cout << "The stack is empty.\n";

return -1;

}

return StackArray[Top];

}

void IntStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

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

cout << StackArray[i] << " ";

}

cout << endl;

}

}

int main() {

int val, choice;

IntStack stack(5);

while (true) {

cout << "Menu:" << endl;

cout << "1. Push an element onto the stack" << endl;

cout << "2. Pop an element from the stack" << endl;

cout << "3. Check if the stack is empty" << endl;

cout << "4. Check if the stack is full" << endl;

cout << "5. Get the top element of the stack" << endl;

cout << "6. Print the stack" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter an integer to push onto the stack: ";

cin >> val;

if (stack.push(val)) {

cout << "Element pushed successfully." << endl;

}

break;

}

case 2: {

if (stack.pop(val)) {

cout << "Popped element: " << val << endl;

}

break;

}

case 3: {

if (stack.isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "The stack is not empty." << endl;

}

break;

}

case 4: {

if (stack.isFull()) {

cout << "The stack is full." << endl;

}

else {

cout << "The stack is not full." << endl;

}

break;

}

case 5: {

val = stack.top();

if (val != -1) {

cout << "Top element: " << val << endl;

}

break;

}

case 6: {

stack.printStack();

break;

}

case 0: {

cout << "Exiting..." << endl;

return 0;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

}

return 0;

}

Task 1 (b)

#include<iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class IntStack {

private:

Node\* Top;

public:

IntStack();

~IntStack();

bool push(int);

bool pop(int&);

bool isempty();

int top();

void printStack();

};

IntStack::IntStack() {

Top = nullptr;

}

IntStack::~IntStack() {

while (Top != nullptr) {

Node\* temp = Top;

Top = Top->next;

delete temp;

}

}

bool IntStack::push(int val) {

Node\* newNode = new Node;

newNode->data = val;

newNode->next = Top;

Top = newNode;

return true;

}

bool IntStack::pop(int& val) {

if (isempty()) {

cout << "The stack is empty.\n";

return false;

}

val = Top->data;

Node\* temp = Top;

Top = Top->next;

delete temp;

return true;

}

bool IntStack::isempty() {

return (Top == nullptr);

}

int IntStack::top() {

if (isempty()) {

throw std::runtime\_error("Stack is empty");

}

return Top->data;

}

void IntStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

Node\* temp = Top;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

int main() {

int val, choice;

IntStack stack;

while (true) {

cout << "Menu:" << endl;

cout << "1. Push an element onto the stack" << endl;

cout << "2. Pop an element from the stack" << endl;

cout << "3. Check if the stack is empty" << endl;

cout << "4. Check if the stack is full" << endl;

cout << "5. Get the top element of the stack" << endl;

cout << "6. Print the stack" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter an integer to push onto the stack: ";

cin >> val;

if (stack.push(val)) {

cout << "Element pushed successfully." << endl;

}

break;

}

case 2: {

if (stack.pop(val)) {

cout << "Popped element: " << val << endl;

}

break;

}

case 3: {

if (stack.isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "The stack is not empty." << endl;

}

break;

}

case 4: {

cout << "The stack is not full." << endl;

break;

}

case 5: {

try {

val = stack.top();

cout << "Top element: " << val << endl;

}

catch (const std::exception& e) {

cout << e.what() << endl;

}

break;

}

case 6: {

stack.printStack();

break;

}

case 0: {

cout << "Exiting..." << endl;

return 0;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

}

return 0;

}

**Task 2**

#include<iostream>

using namespace std;

class IntStack {

private:

int\* StackArray;

int StackSize;

int Top;

int Min;

public:

IntStack(int);

~IntStack();

bool push(int);

bool pop(int&);

bool isFull();

bool isempty();

int top();

int getMin();

void printStack();

};

IntStack::IntStack(int size) {

StackArray = new int[size];

StackSize = size;

Top = -1;

Min = INT\_MAX;

}

IntStack::~IntStack() {

delete[] StackArray;

}

bool IntStack::push(int val) {

if (isFull()) {

cout << "The stack is full." << endl;

return false;

}

Top++;

StackArray[Top] = val;

if (val < Min) {

Min = val;

}

return true;

}

bool IntStack::pop(int& val) {

if (isempty())

{

cout << "The stack is empty.\n";

return false;

}

val = StackArray[Top];

if (val == Min) {

Min = INT\_MAX;

for (int i = 0; i <= Top - 1; i++) {

if (StackArray[i] < Min) {

Min = StackArray[i];

}

}

}

Top--;

return true;

}

bool IntStack::isFull() {

if (Top == StackSize - 1)

return true;

else

return false;

}

bool IntStack::isempty() {

return (Top == -1);

}

int IntStack::top() {

if (isempty())

{

cout << "The stack is empty.\n";

return -1;

}

return StackArray[Top];

}

int IntStack::getMin() {

if (isempty())

{

cout << "The stack is empty.\n";

return -1;

}

return Min;

}

void IntStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

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

cout << StackArray[i] << " ";

}

cout << endl;

}

}

int main() {

int val, choice;

IntStack stack(5);

while (true) {

cout << "Menu:" << endl;

cout << "1. Push an element onto the stack" << endl;

cout << "2. Pop an element from the stack" << endl;

cout << "3. Check if the stack is empty" << endl;

cout << "4. Check if the stack is full" << endl;

cout << "5. Get the top element of the stack" << endl;

cout << "6. Get the minimum element of the stack" << endl;

cout << "7. Print the stack" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter an integer to push onto the stack: ";

cin >> val;

if (stack.push(val)) {

cout << "Element pushed successfully." << endl;

}

break;

}

case 2: {

if (stack.pop(val)) {

cout << "Popped element: " << val << endl;

}

break;

}

case 3: {

if (stack.isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "The stack is not empty." << endl;

}

break;

}

case 4: {

if (stack.isFull()) {

cout << "The stack is full." << endl;

}

else {

cout << "The stack is not full." << endl;

}

break;

}

case 5: {

val = stack.top();

if (val != -1) {

cout << "Top element: " << val << endl;

}

break;

}

case 6: {

val = stack.getMin();

if (val != -1) {

cout << "Minimum element: " << val << endl;

}

break;

}

case 7: {

stack.printStack();

break;

}

case 0: {

cout << "Exiting..." << endl;

return 0;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

}

return 0;

}

**Task 3**

#include<iostream>

using namespace std;

class IntStack {

private:

int\* StackArray;

int StackSize;

int Top;

public:

IntStack(int);

~IntStack();

bool push(int);

bool pop(int&);

bool isFull();

bool isempty();

int top();

void displayStack();

};

IntStack::IntStack(int size) {

StackArray = new int[size];

StackSize = size;

Top = -1;

}

IntStack::~IntStack() {

delete[] StackArray;

}

bool IntStack::push(int val) {

if (isFull()) {

cout << "The stack is full." << endl;

return false;

}

Top++;

StackArray[Top] = val;

return true;

}

bool IntStack::pop(int& val) {

if (isempty())

{

cout << "The stack is empty.\n";

return false;

}

val = StackArray[Top];

Top--;

return true;

}

bool IntStack::isFull() {

if (Top == StackSize - 1)

return true;

else

return false;

}

bool IntStack::isempty() {

return (Top == -1);

}

int IntStack::top() {

if (isempty())

{

cout << "The stack is empty.\n";

return -1;

}

return StackArray[Top];

}

void IntStack::displayStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

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

cout << StackArray[i] << " ";

}

cout << endl;

}

}

void reverseStack(IntStack& stack) {

IntStack tempStack(stack.StackSize);

while (!stack.isempty()) {

int val;

stack.pop(val);

tempStack.push(val);

}

stack = tempStack;

}

int main() {

int val, choice;

IntStack stack(5);

while (true) {

cout << "Menu:" << endl;

cout << "1. Push an element onto the stack" << endl;

cout << "2. Pop an element from the stack" << endl;

cout << "3. Check if the stack is empty" << endl;

cout << "4. Check if the stack is full" << endl;

cout << "5. Get the top element of the stack" << endl;

cout << "6. Reverse the stack" << endl;

cout << "7. Display the stack" << endl;

cout << "0. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1: {

cout << "Enter an integer to push onto the stack: ";

cin >> val;

if (stack.push(val)) {

cout << "Element pushed successfully." << endl;

}

break;

}

case 2: {

if (stack.pop(val)) {

cout << "Popped element: " << val << endl;

}

break;

}

case 3: {

if (stack.isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "The stack is not empty." << endl;

}

break;

}

case 4: {

if (stack.isFull()) {

cout << "The stack is full." << endl;

}

else {

cout << "The stack is not full." << endl;

}

break;

}

case 5: {

val = stack.top();

if (val != -1) {

cout << "Top element: " << val << endl;

}

break;

}

case 6: {

reverseStack(stack);

cout << "Stack reversed successfully." << endl;

break;

}

case 7: {

stack.displayStack();

break;

}

case 0: {

cout << "Exiting..." << endl;

return 0;

}

default: {

cout << "Invalid choice. Please try again." << endl;

break;

}

}

}

return 0;

}

**Task 4 (1)**

#include<iostream>

#include<string>

using namespace std;

class IntStack {

private:

int\* StackArray;

int StackSize;

int Top;

public:

IntStack(int);

~IntStack();

bool push(int);

bool pop(int&);

bool isFull();

bool isempty();

int top();

void printStack();

};

IntStack::IntStack(int size) {

StackArray = new int[size];

StackSize = size;

Top = -1;

}

IntStack::~IntStack() {

delete[] StackArray;

}

bool IntStack::push(int val) {

if (isFull()) {

cout << "The stack is full." << endl;

return false;

}

Top++;

StackArray[Top] = val;

return true;

}

bool IntStack::pop(int& val) {

if (isempty())

{

cout << "The stack is empty.\n";

return false;

}

val = StackArray[Top];

Top--;

return true;

}

bool IntStack::isFull() {

if (Top == StackSize - 1)

return true;

else

return false;

}

bool IntStack::isempty() {

return (Top == -1);

}

int IntStack::top() {

if (isempty())

{

cout << "The stack is empty.\n";

return -1;

}

return StackArray[Top];

}

void IntStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

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

cout << StackArray[i] << " ";

}

cout << endl;

}

}

bool isBalanced(string str) {

IntStack stack(100);

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

if (str[i] == '(' || str[i] == '{' || str[i] == '[') {

stack.push(str[i]);

}

else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {

if (stack.isempty()) {

return false;

}

int top;

stack.pop(top);

if ((str[i] == ')' && top != '(') || (str[i] == '}' && top != '{') || (str[i] == ']' && top != '[')) {

return false;

}

}

}

if (stack.isempty()) {

return true;

}

else {

return false;

}

}

int main() {

string str;

cout << "Enter a string: ";

getline(cin, str);

if (isBalanced(str)) {

cout << "The string is balanced." << endl;

}

else {

cout << "The string is not balanced." << endl;

}

return 0;

}

**Task 4 (2)**

#include<iostream>

#include<string>

using namespace std;

struct Node {

char data;

Node\* next;

};

class CharStack {

private:

Node\* Top;

public:

CharStack();

~CharStack();

bool push(char);

bool pop(char&);

bool isempty();

char top();

void printStack();

};

CharStack::CharStack() {

Top = nullptr;

}

CharStack::~CharStack() {

while (Top != nullptr) {

Node\* temp = Top;

Top = Top->next;

delete temp;

}

}

bool CharStack::push(char val) {

Node\* newNode = new Node;

newNode->data = val;

newNode->next = Top;

Top = newNode;

return true;

}

bool CharStack::pop(char& val) {

if (isempty()) {

cout << "The stack is empty.\n";

return false;

}

val = Top->data;

Node\* temp = Top;

Top = Top->next;

delete temp;

return true;

}

bool CharStack::isempty() {

return (Top == nullptr);

}

char CharStack::top() {

if (isempty()) {

throw std::runtime\_error("Stack is empty");

}

return Top->data;

}

void CharStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

Node\* temp = Top;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

bool isBalanced(string str) {

CharStack stack;

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

if (str[i] == '(' || str[i] == '{' || str[i] == '[') {

stack.push(str[i]);

}

else if (str[i] == ')' || str[i] == '}' || str[i] == ']') {

if (stack.isempty()) {

return false;

}

char top;

stack.pop(top);

if ((str[i] == ')' && top != '(') || (str[i] == '}' && top != '{') || (str[i] == ']' && top != '[')) {

return false;

}

}

}

if (stack.isempty()) {

return true;

}

else {

return false;

}

}

int main() {

string str;

cout << "Enter a string: ";

getline(cin, str);

if (isBalanced(str)) {

cout << "The string is balanced." << endl;

}

else {

cout << "The string is not balanced." << endl;

}

return 0;

}

**Task 5**

#include<iostream>

#include<string>

using namespace std;

struct Node {

int data;

Node\* next;

};

class IntStack {

private:

Node\* Top;

public:

IntStack();

~IntStack();

bool push(int);

bool pop(int&);

bool isempty();

int top();

void printStack();

};

IntStack::IntStack() {

Top = nullptr;

}

IntStack::~IntStack() {

while (Top != nullptr) {

Node\* temp = Top;

Top = Top->next;

delete temp;

}

}

bool IntStack::push(int val) {

Node\* newNode = new Node;

newNode->data = val;

newNode->next = Top;

Top = newNode;

return true;

}

bool IntStack::pop(int& val) {

if (isempty()) {

cout << "The stack is empty.\n";

return false;

}

val = Top->data;

Node\* temp = Top;

Top = Top->next;

delete temp;

return true;

}

bool IntStack::isempty() {

return (Top == nullptr);

}

int IntStack::top() {

if (isempty()) {

throw std::runtime\_error("Stack is empty");

}

return Top->data;

}

void IntStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

Node\* temp = Top;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

int evaluatePostfix(string str) {

IntStack stack;

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

if (str[i] == ' ') {

continue;

}

else if (isdigit(str[i])) {

int num = 0;

while (i < str.length() && isdigit(str[i])) {

num = num \* 10 + (str[i] - '0');

i++;

}

i--;

stack.push(num);

}

else if (str[i] == '+' || str[i] == '-' || str[i] == '\*' || str[i] == '/') {

int op1, op2;

stack.pop(op2);

stack.pop(op1);

if (str[i] == '+') {

stack.push(op1 + op2);

}

else if (str[i] == '-') {

stack.push(op1 - op2);

}

else if (str[i] == '\*') {

stack.push(op1 \* op2);

}

else if (str[i] == '/') {

stack.push(op1 / op2);

}

}

}

return stack.top();

}

int main() {

string str;

cout << "Enter a postfix expression: ";

getline(cin, str);

cout << "Result: " << evaluatePostfix(str) << endl;

return 0;

}

**Task 6**

#include<iostream>

#include<string>

using namespace std;

struct Node {

char data;

Node\* next;

};

class CharStack {

private:

Node\* Top;

public:

CharStack();

~CharStack();

bool push(char);

bool pop(char&);

bool isempty();

char top();

void printStack();

};

CharStack::CharStack() {

Top = nullptr;

}

CharStack::~CharStack() {

while (Top != nullptr) {

Node\* temp = Top;

Top = Top->next;

delete temp;

}

}

bool CharStack::push(char val) {

Node\* newNode = new Node;

newNode->data = val;

newNode->next = Top;

Top = newNode;

return true;

}

bool CharStack::pop(char& val) {

if (isempty()) {

cout << "The stack is empty.\n";

return false;

}

val = Top->data;

Node\* temp = Top;

Top = Top->next;

delete temp;

return true;

}

bool CharStack::isempty() {

return (Top == nullptr);

}

char CharStack::top() {

if (isempty()) {

throw std::runtime\_error("Stack is empty");

}

return Top->data;

}

void CharStack::printStack() {

if (isempty()) {

cout << "The stack is empty." << endl;

}

else {

cout << "Stack elements: ";

Node\* temp = Top;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

}

bool isPalindrome(string str) {

CharStack stack;

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

if (isalnum(str[i])) {

stack.push(tolower(str[i]));

}

}

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

if (isalnum(str[i])) {

char top;

stack.pop(top);

if (top != tolower(str[i])) {

return false;

}

}

}

return true;

}

int main() {

string str;

cout << "Enter a string: ";

getline(cin, str);

if (isPalindrome(str)) {

cout << "The string is a palindrome." << endl;

}

else {

cout << "The string is not a palindrome." << endl;

}

return 0;

}

**Task 7**