|  |
| --- |
| C++ Assignments |

|  |
| --- |
| Name: Arka Das  Roll No: 002210503046 |

**­­­Exercise 1: Design your own Stack class. The program should contain the following**

* **A constructor to initialize the stack**
* **peek(), pop(), push() as their usual meaning**
* **overloaded display() as**
  + **Display the whole contents of the stack**
  + **Pass a parameter providing the depth and display only that element.**

**Source Code:**

**CustomStack header file (CustomStack.hpp)**

#include <iostream>

constexpr int MAX\_SIZE= 100;

class CustomStack {

private:

int stack[MAX\_SIZE];

int top;

public:

CustomStack() { top = -1; }

int getTop() { return top; }

void push(const int& value);

void peek(void);

int getPeek(void);

int pop(void);

void display(void);

void display(const int& depth);

};

void CustomStack::push(const int& value) {

if(top == MAX\_SIZE - 1) {

std::cout<<"\nStack overflow";

return;

}

top++;

stack[top] = value;

}

void CustomStack::peek() {

if(top == -1) {

std::cout<<"\nStack underflow";

return;

}

std::cout<<"\nTop of stack is: "<<stack[top];

}

int CustomStack::getPeek() {

if(top == -1)

return INT32\_MAX;

return stack[top];

}

int CustomStack::pop() {

if(top < 0) {

std::cout<<"\nStack underflow";

return INT32\_MAX;

}

int x = stack[top];

top--;

return x;

}

void CustomStack::display() {

if(top < 0) {

std::cout<<"Stack Empty";

return;

}

// std::cout<<"\nStack elements are:\n";

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

std::cout<<stack[i]<<" ,";

}

void CustomStack::display(const int& depth) {

if(depth < 0 || depth > top) {

std::cout<<"\nInvalid depth";

return;

}

std::cout<<"\nElement at depth "<<depth<<" is: "<<stack[top - depth];

}

**Driver code (CustomStack\_Driver.cpp)**

#include "CustomStack.hpp"

int main() {

CustomStack \*myStack = new CustomStack();

int choice, temp;

while(1) {

std::cout<<"\n1 -> push item";

std::cout<<"\n2 -> pop item";

std::cout<<"\n3 -> peek";

std::cout<<"\n4 -> display all items";

std::cout<<"\n5 -> element at given depth";

std::cout<<"\n6 -> exit";

std::cout<<"\nEnter your choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

std::cout<<"\nEnter element to push: ";

std::cin >> temp;

myStack->push(temp);

break;

case 2:

temp = myStack->pop();

if(temp != INT32\_MAX)

std::cout<<"\nPopped element is: "<<temp;

break;

case 3:

myStack->peek();

break;

case 4:

myStack->display();

break;

case 5:

std::cout<<"\nEnter depth of element: ";

std::cin >> temp;

myStack->display(temp);

break;

case 6:

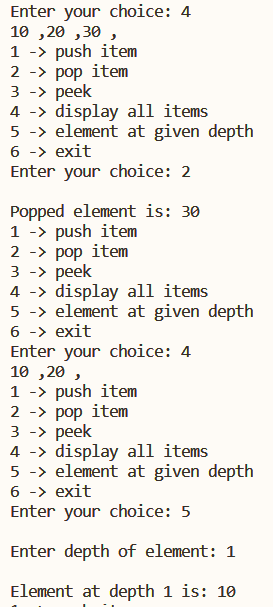
return 0;

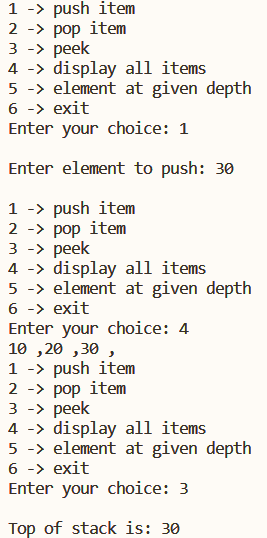
default:

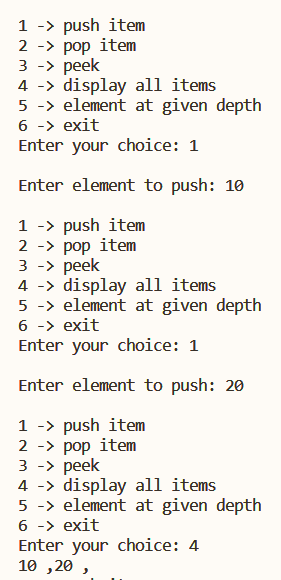
break;

}

}

****}

****

**Output:**

**Exercise 2: Inherit the custom stack created before and the inherited class should allow the following –**

* **A number is to be pushed in if and only if it is less than the one currently at the top**

**With the help of such stack objects, solve the Tower Of Hanoi problem most economically, by arranging a sequence of haphazard numbers in the original base stack in ascending order for the final stack.**

**Source Code:**

**CustomStack header file (CustomStack.hpp)**

#include <iostream>

constexpr int MAX\_SIZE= 100;

class CustomStack {

private:

int stack[MAX\_SIZE];

int top;

public:

CustomStack() { top = -1; }

int getTop() { return top; }

void push(const int& value);

void peek(void);

int getPeek(void);

int pop(void);

void display(void);

void display(const int& depth);

};

void CustomStack::push(const int& value) {

if(top == MAX\_SIZE - 1) {

std::cout<<"\nStack overflow";

return;

}

top++;

stack[top] = value;

}

void CustomStack::peek() {

if(top == -1) {

std::cout<<"\nStack underflow";

return;

}

std::cout<<"\nTop of stack is: "<<stack[top];

}

int CustomStack::getPeek() {

if(top == -1)

return INT32\_MAX;

return stack[top];

}

int CustomStack::pop() {

if(top < 0) {

std::cout<<"\nStack underflow";

return INT32\_MAX;

}

int x = stack[top];

top--;

return x;

}

void CustomStack::display() {

if(top < 0) {

std::cout<<"Stack Empty";

return;

}

// std::cout<<"\nStack elements are:\n";

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

std::cout<<stack[i]<<" ,";

}

void CustomStack::display(const int& depth) {

if(depth < 0 || depth > top) {

std::cout<<"\nInvalid depth";

return;

}

std::cout<<"\nElement at depth "<<depth<<" is: "<<stack[top - depth];

}

**DescendingStack header file (DescendingStack.hpp) Inherits Custom stack previously made**

#include "CustomStack.hpp"

class DescendingStack : public CustomStack {

public:

DescendingStack() { }

DescendingStack(CustomStack s) {

//constructor for creating sorted stack from normal custom stack

int a[MAX\_SIZE], k=0;

while(s.getTop() >= 0) {

int x = s.pop();

a[k] = x;

k++;

}

//now sort and push in new Descending stack

for(int i=0;i<k;i++)

for(int j=0;j<k-i-1;j++)

if(a[j] < a[j+1])

std::swap(a[j], a[j+1]);

for(int i=0;i<k;i++)

this->push(a[i]);

}

void push(const int& value) {

if(getTop() == -1)

CustomStack::push(value);

else if(getPeek() > value)

CustomStack::push(value);

else

std::cout<<"\nInvalid input";

}

};

**Driver code (TowerOfHanoi.cpp)**

#include "DescendingStack.hpp"

void towerOfHanoi(int , DescendingStack& , DescendingStack& , DescendingStack& );

int main()

{

int numberOfDisks;

int temp;

CustomStack s;

std::cout<<"\nEnter number of disks: ";

std::cin>>numberOfDisks;

std::cout<<"\nEnter the size of disks: ";

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

std::cin>>temp;

s.push(temp);

}

DescendingStack from\_rod(s), to\_rod, aux\_rod;

towerOfHanoi(numberOfDisks, from\_rod, to\_rod, aux\_rod);

}

void towerOfHanoi(int n, DescendingStack& from\_rod, DescendingStack& to\_rod, DescendingStack& aux\_rod)

{

if (n == 0)

return;

towerOfHanoi(n - 1, from\_rod, aux\_rod, to\_rod);

int currentDisk = from\_rod.pop();

to\_rod.push(currentDisk);

std::cout<<"\nSource: ";

from\_rod.display();

std::cout<<"\nDest: ";

to\_rod.display();

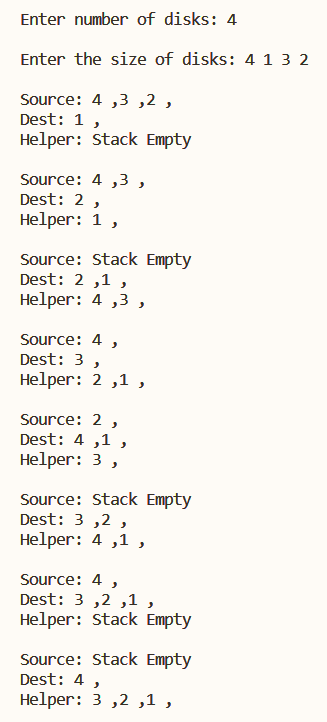
std::cout<<"\nHelper: ";

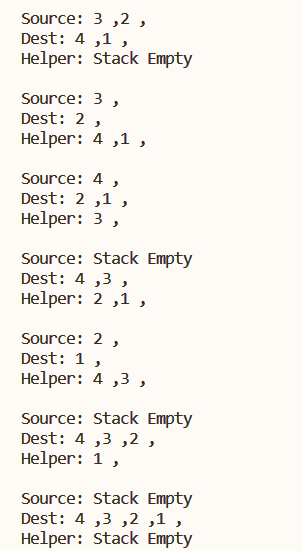
aux\_rod.display();

std::cout<<"\n";

towerOfHanoi(n - 1, aux\_rod, to\_rod, from\_rod);

}

**Output:**



**Exercise 3A: Modify the custom stack class created above with templates to store objects of own class of own design while keeping all the constraints same.**

**Source Code:**

**Generic\_Stack header file (Generic\_Stack.hpp)**

#include <iostream>

template <typename T, int N>

class Generic\_Stack {

private:

T stack[N];

int top;

public:

Generic\_Stack() { top = -1; }

int getTop() { return top; }

void push(const T& value);

void peek(void);

T pop(void);

void display(void);

void display(const int& depth);

};

template <typename T, int N>

void Generic\_Stack<T, N>::push(const T& value) {

if(top == N - 1) {

std::cout<<"\nStack overflow";

return;

}

top++;

stack[top] = value;

}

template <typename T, int N>

void Generic\_Stack<T, N>::peek() {

if(top == -1) {

std::cout<<"\nStack underflow";

return;

}

std::cout<<"\nTop of stack is: "<<stack[top];

}

template <typename T, int N>

T Generic\_Stack<T, N>::pop() {

T x = stack[top];

top--;

return x;

}

template <typename T, int N>

void Generic\_Stack<T, N>::display() {

if(top < 0) {

std::cout<<"\nStack underflow";

return;

}

std::cout<<"\nStack elements are:\n";

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

std::cout<<stack[i]<<" ,";

}

template <typename T, int N>

void Generic\_Stack<T, N>::display(const int& depth) {

if(depth < 0 || depth > top) {

std::cout<<"\nInvalid depth";

return;

}

std::cout<<"\nElement at depth "<<depth<<" is: "<<stack[top - depth];

}

**TIME class header file (TIME.hpp) the custom class to be stored:**

#pragma once

#include <iostream>

class TIME {

private:

int hour, min, sec;

public:

TIME() {}

TIME(int hour, int min, int sec) {

this->hour = hour;

this->min = min;

this->sec = sec;

}

friend std::ostream& operator << (std::ostream& out, const TIME& time) {

out << time.hour << ":" << time.min << ":" << time.sec;

return out;

}

friend std::istream& operator >> (std::istream& in, TIME& time) {

std::cout<<"\nEnter time in hour min second: ";

in >> time.hour;

in >> time.min;

in >> time.sec;

return in;

}

};

**Driver code (Generic\_Stack\_Driver.cpp):**

#include "Generic\_Stack.hpp"

#include "TIME.hpp"

int main() {

Generic\_Stack<TIME, 10> myStack;

TIME temp;

int depth;

int choice;

std::cout<<"\nThis is a stack which sotres TIME type data";

while(1) {

std::cout<<"\n1 -> push item";

std::cout<<"\n2 -> pop item";

std::cout<<"\n3 -> peek";

std::cout<<"\n4 -> display all items";

std::cout<<"\n5 -> element at given depth";

std::cout<<"\n6 -> exit";

std::cout<<"\nEnter your choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

std::cout<<"\nEnter element to push: ";

std::cin >> temp;

myStack.push(temp);

break;

case 2:

temp = myStack.pop();

std::cout<<"\nPopped element is: "<<temp;

break;

case 3:

myStack.peek();

break;

case 4:

myStack.display();

break;

case 5:

std::cout<<"\nEnter depth of element: ";

std::cin >> depth;

myStack.display(depth);

break;

case 6:

return 0;

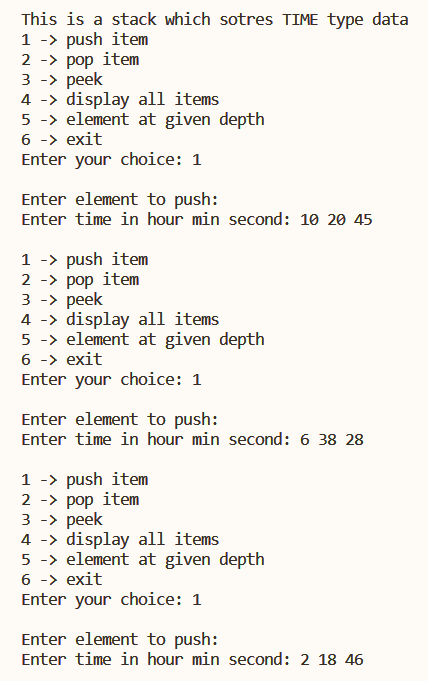
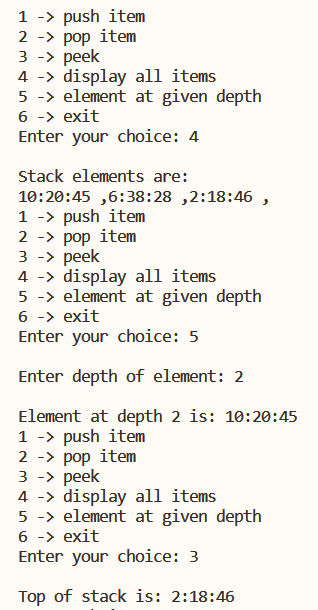
default:

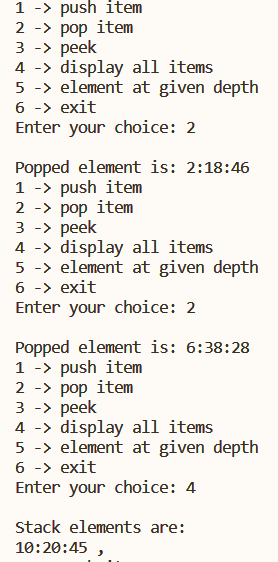
break;

}

}

}

**Output:**



**Exercise 3B: Modify the above stack class created using templates and use try-throw-catch in some form to accommodate exception handling.**

**Source Code:**

**Generic\_Stack\_Exp header file (Generic\_Stack\_Exp.hpp)**

#include <iostream>

#include <string>

class customStackException {

private:

std::string message;

public:

customStackException(const std::string& message) {

this->message = message;

}

std::string what() const {

return message;

}

};

//template stack class

template <typename T, int N>

class Generic\_Stack {

private:

T stack[N];

int top;

public:

Generic\_Stack() { top = -1; }

int getTop() { return top; }

void push(const T& value);

void peek(void);

T pop(void);

void display(void);

void display(const int& depth);

T getStackAvg(void);

};

template <typename T, int N>

T Generic\_Stack<T, N>::getStackAvg() {

if(top < 0)

throw customStackException("Stack underflow");

int temp = 0;

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

temp = temp + stack[i];

temp /= top+1;

return T(temp);

}

template <typename T, int N>

void Generic\_Stack<T, N>::push(const T& value) {

if(top == N - 1)

throw customStackException("Stack Overflow");

top++;

stack[top] = value;

}

template <typename T, int N>

void Generic\_Stack<T, N>::peek() {

if(top == -1)

throw customStackException("Stack underflow");

std::cout<<"\nTop of stack is: "<<stack[top];

}

template <typename T, int N>

T Generic\_Stack<T, N>::pop() {

if(top < 0)

throw customStackException("Stack underflow");

int x = stack[top];

top--;

return x;

}

template <typename T, int N>

void Generic\_Stack<T, N>::display() {

if(top < 0)

throw customStackException("Stack underflow");

std::cout<<"\nStack elements are:\n";

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

std::cout<<stack[i]<<" ,";

}

template <typename T, int N>

void Generic\_Stack<T, N>::display(const int& depth) {

if(depth < 0 || depth > top)

throw customStackException("Invalid Depth");

std::cout<<"\nElement at depth "<<depth<<" is: "<<stack[top - depth];

}

**Driver class (Generic\_Stack\_Excp.cpp):**

#include "Generic\_Stack\_Exp.hpp"

int main() {

Generic\_Stack<char, 10> myStack;

char temp;

int depth;

int choice;

while(1) {

std::cout<<"\n1 -> push item";

std::cout<<"\n2 -> pop item";

std::cout<<"\n3 -> peek";

std::cout<<"\n4 -> display all items";

std::cout<<"\n5 -> element at given depth";

std::cout<<"\n6 -> get stack average";

std::cout<<"\n7 -> exit";

std::cout<<"\nEnter your choice: ";

std::cin >> choice;

switch (choice)

{

case 1:

std::cout<<"\nEnter element to push: ";

std::cin >> temp;

try {

myStack.push(temp);

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 2:

try {

temp = myStack.pop();

std::cout<<"\nPopped element is: "<<temp;

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 3:

try {

myStack.peek();

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 4:

try {

myStack.display();

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 5:

std::cout<<"\nEnter depth of element: ";

std::cin >> depth;

try {

myStack.display(temp);

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 6:

try {

temp = myStack.getStackAvg();

std::cout<<"\n"<<temp;

}

catch (const customStackException& e) {

std::cout<<"\nException: "<<e.what();

}

break;

case 7:

return 0;

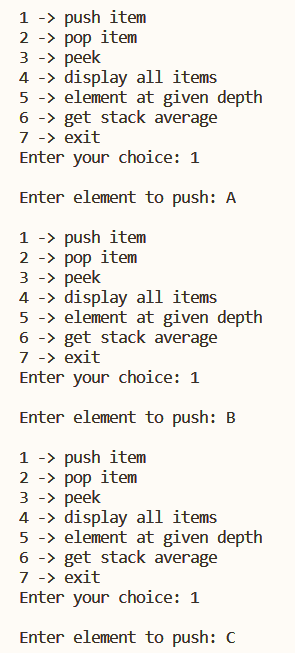
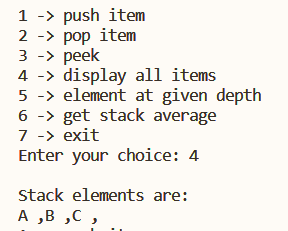
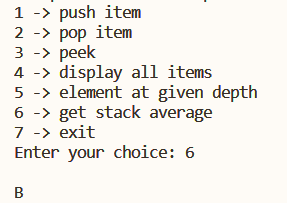
default:

break;

}

}

}

**Output:**

