UNIVERSITY OF THE WEST OF ENGLAND





Submitted by: - Albin Pokharel

Module: - Programming in C++

**Write the code for using STL containers and STL algorithms for the following:**

**(a) To reverse a list**

#include<iostream>

#include <list>

#include <algorithm>

using namespace std;

int main() {

// Create a list

list<int> myList = {1, 2, 3, 4, 5};

// Print original list

cout << "Original List: ";

for (const auto& num : myList) {

cout << num << " ";

}

cout << endl;

// Reverse the list using std::reverse

reverse(myList.begin(), myList.end());

// Print reversed list

cout << "Reversed List: ";

for (const auto& num : myList) {

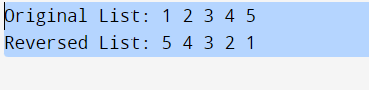
cout << num << " ";

}

cout << endl;

return 0;

}



**(b) To convert a decimal to binary form using stack**

#include <iostream>

#include <stack>

using namespace std;

int main() {

int decimal;

// Get input from the user

cout << "Enter a decimal number: ";

cin >> decimal;

stack<int> binaryStack;

// Convert decimal to binary using a stack

while (decimal > 0) {

int remainder = decimal % 2;

binaryStack.push(remainder);

decimal /= 2;

}

// Print binary representation

cout << "Binary representation: ";

while (!binaryStack.empty()) {

cout << binaryStack.top();

binaryStack.pop();

}

cout << endl;

return 0;

}



**(c) Queue operations**

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> myQueue;

// Inserting elements into the queue

myQueue.push(1);

myQueue.push(2);

myQueue.push(3);

// Accessing the front element of the queue

cout << "Front element: " << myQueue.front() << endl;

// Accessing the back element of the queue

cout << "Back element: " << myQueue.back() << endl;

// Removing elements from the queue

myQueue.pop();

cout << "Front element after pop: " << myQueue.front() << endl;

// Checking if the queue is empty

if (myQueue.empty()) {

cout << "Queue is empty" << endl;

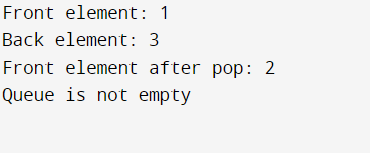
} else {

cout << "Queue is not empty" << endl;

}

return 0;

}



TASK 2

**Correct the code, if any problems noticed**

**1. In the given queue, front and rear has been initialized with 0. Insert func ion of the queue has been given as:**

**void insert (int x)**

**{**

**if (rear+1==size)**

**cout<<”queue overflow”<<endl;**

**++rear=x;**

**}**

void insert(int x) {

if (rear + 1 == size) {

cout << "Queue overflow" << endl;

} else {

++rear;

queue[rear] = x;

}

}

**2. In the circular queue, front and rear has been initialized with -1. Insert function of the**

**queue has been given as:**

**void insert (int x)**

**{**

**if (rear+1==size)**

**cout<<”queue overflow”<<endl;**

**rear+1=x;**

**}**

void insert(int x) {

if ((rear + 1) % size == front) {

cout << "Queue overflow" << endl;

} else {

rear = (rear + 1) % size;

queue[rear] = x;

}

}

**3. In the display function, following code has been written:**

**void display()**

**{**

**int beg=0;**

**while(beg<=top)**

**cout<<stack[beg++]<<endl;**

**}**

**void display()**

**{**

**int t=top;**

**while(t>=0)**

**cout<<stack[t++]<<endl;**

**}**

**After correcting error from both the functions (if any), which version of them will be preferred?**

Ans: the 2nd version should be preferred.ie the display function .

void display() {

int t = top;

while (t >= 0)

cout << stack[t--] << endl;}

**4. In a stack based template, pop function is written; find out the possible problem that**

**may occur in this code.**

t pop() {

if (top == NULL) {

cout << "Stack underflow" << endl;

return default\_value; // Return a default value indicating underflow

}

t x = top->data;

Node<t>\* temp = top;

top = top->next;

delete temp; // Remove the top node

return x;

}

**Task 3: 60 marks**

**In a hospital there are a number of department based on disease. Major department exist are Dermatology, Cardiology, Gynecology, Pathology, etc. Each department is capable to serve the patient in increasingly fair manner. To avail the treatment facility, patient visits to the relevant department where (s)he is allotted a token number. Allotment of token number is purely on the basis of arrival time. Separate queue is maintained for each department.**

**Hospital maintains the following information about the patient.**

**Patient Information**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Information** | **Type of Information** |
| **1** | **Patientid** | **Stores in alphanumeric** |
| **2** | **Social\_id** | **Numeric value** |
| **3** | **Occupation** | **Character value** |
| **4** | **Disease reported** | **Character value** |
| **5** | **Treatment prescribed** | **Character value** |
| **6** | **Previous treated** | **Yes/No** |
| **7** | **Token number** | **Number** |
| **8** | **dateofVisit** | **Date** |

**Based on the above information, solve the following challenges the hospital has to encounter.[Write functions with appropriate data structures for each case]**

* + - * **Allot the unique token number to each patient.**
      * **Store all the information related to the above field.**
      * **Total number of patient in a particular queue.**
      * **How many patients waiting in the hospital.**
      * **How many patients arrive in a day (you can take the average of week or month)?**
      * **Which type of data structure is preferred by you? Explain its advantages over any other data structure which you may consider near appropriate.**

**[Hint: queue Data structure can be considered]**

**#include <queue>**

**#include <vector>**

**#include <algorithm>**

**#include <string>**

**#include <iostream>**

**#include <map>**

**using namespace std;**

**struct Patient**

**{**

**string patientid;**

**int social\_id;**

**string occupation;**

**string disease;**

**string treatment;**

**bool previousTreated;**

**int tokenNumber;**

**string dateofvisit;**

**};**

**class Department**

**{**

**public:**

**queue<Patient> patientQueue;**

**void AddPatient(Patient patient)**

**{**

**patientQueue.push(patient);**

**}**

**Patient TreatNextPatient()**

**{**

**Patient nextPatient = patientQueue.front();**

**patientQueue.pop();**

**return nextPatient;**

**}**

**int NumWaitingPatients()**

**{**

**return patientQueue.size();**

**}**

**};**

**class Hospital**

**{**

**public:**

**map<string, Department> departments;**

**int nextTokenNumber;**

**int numTreatedPatient;**

**int numPatientToday;**

**Hospital() : nextTokenNumber(1), numTreatedPatient(0), numPatientToday(0) {}**

**void AddPatient(string departmentName, Patient patient)**

**{**

**vector<string> validDepartments = {"Dermatology", "Cardiology", "Gynecology", "Pathology"};**

**for (auto &x : validDepartments)**

**{**

**transform(x.begin(), x.end(), x.begin(), ::tolower);**

**}**

**if (count(validDepartments.begin(), validDepartments.end(), departmentName) == 0)**

**{**

**throw invalid\_argument("Invalid department name.");**

**}**

**patient.tokenNumber = nextTokenNumber;**

**nextTokenNumber++;**

**numPatientToday++;**

**departments[departmentName].AddPatient(patient);**

**cout << "----------------------------------" << endl;**

**cout << "Patient added to department " << departmentName << " with token number " << patient.tokenNumber << endl;**

**cout << "----------------------------------" << endl;**

**}**

**Patient TreatNextPatient(string departmentName)**

**{**

**Patient patient = departments[departmentName].TreatNextPatient();**

**numTreatedPatient++;**

**return patient;**

**}**

**int NumWaitingPatients(string departmentName)**

**{**

**auto it = departments.find(departmentName);**

**if (it == departments.end())**

**{**

**return 0;**

**}**

**return it->second.NumWaitingPatients();**

**}**

**int NumWaitingPatients()**

**{**

**int totalWaiting = 0;**

**for (auto department : departments)**

**{**

**totalWaiting += department.second.NumWaitingPatients();**

**}**

**return totalWaiting;**

**}**

**int NumTreatedPatients()**

**{**

**return numTreatedPatient;**

**}**

**int NumPatientToday()**

**{**

**return numPatientToday;**

**}**

**void generatePatientReport(string patientId)**

**{**

**cout << "------------------------" << endl;**

**for (auto department : departments)**

**{**

**queue<Patient> patientQueue = department.second.patientQueue;**

**while (!patientQueue.empty())**

**{**

**Patient patient = patientQueue.front();**

**if (patient.patientid == patientId)**

**{**

**cout << "Patient ID: " << patient.patientid << endl;**

**cout << "Social ID: " << patient.social\_id << endl;**

**cout << "Occupation: " << patient.occupation << endl;**

**cout << "Disease: " << patient.disease << endl;**

**cout << "Treatment: " << patient.treatment << endl;**

**cout << "Previously treated: " << patient.previousTreated << endl;**

**cout << "Token Number: " << patient.tokenNumber << endl;**

**cout << "Date of Visit: " << patient.dateofvisit << endl;**

**cout << "------------------------" << endl;**

**return;**

**}**

**patientQueue.pop();**

**}**

**}**

**cout << "Patient not found." << endl;**

**cout << "------------------------" << endl;**

**}**

**};**

**int main()**

**{**

**Hospital hospital;**

**while (true)**

**{**

**cout << "Menu:" << endl;**

**cout << "1. Add patient" << endl;**

**cout << "2. Treat next patient" << endl;**

**cout << "3. Get number of waiting patients for a department" << endl;**

**cout << "4. Get total number of waiting patients" << endl;**

**cout << "5. Get total number of treated patients" << endl;**

**cout << "6. Get total number of patients today" << endl;**

**cout << "7. Generate patient report" << endl;**

**cout << "8. Quit" << endl;**

**int choice;**

**cout << "Enter your choice: ";**

**cin >> choice;**

**if (choice == 1)**

**{**

**string departmentName;**

**cout << "Enter department name: ";**

**cin >> departmentName;**

**Patient patient;**

**cout << "Enter patient ID: ";**

**cin >> patient.patientid;**

**cout << "Enter social ID: ";**

**cin >> patient.social\_id;**

**cout << "Enter occupation: ";**

**cin >> patient.occupation;**

**cout << "Enter disease: ";**

**cin >> patient.disease;**

**cout << "Enter treatment: ";**

**cin >> patient.treatment;**

**cout << "Enter previous treated (0 or 1): ";**

**cin >> patient.previousTreated;**

**cout << "Enter date of visit: ";**

**cin >> patient.dateofvisit;**

**try**

**{**

**hospital.AddPatient(departmentName, patient);**

**}**

**catch (invalid\_argument &e)**

**{**

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

**}**

**}**

**else if (choice == 2)**

**{**

**string departmentName;**

**cout << "Enter department name: ";**

**cin >> departmentName;**

**if (hospital.NumWaitingPatients(departmentName) == 0)**

**{**

**cout << "No patients waiting in " << departmentName << endl;**

**}**

**else**

**{**

**Patient patient = hospital.TreatNextPatient(departmentName);**

**cout << "Patient treated: " << patient.patientid << endl;**

**}**

**}**

**else if (choice == 3)**

**{**

**string departmentName;**

**cout << "Enter department name: ";**

**cin >> departmentName;**

**int numWaiting = hospital.NumWaitingPatients(departmentName);**

**cout << "Number of waiting patients in " << departmentName << ": " << numWaiting << endl;**

**}**

**else if (choice == 4)**

**{**

**int totalWaiting = hospital.NumWaitingPatients();**

**cout << "Total number of waiting patients: " << totalWaiting << endl;**

**}**

**else if (choice == 5)**

**{**

**int totalTreated = hospital.NumTreatedPatients();**

**cout << "Total number of treated patients: " << totalTreated << endl;**

**}**

**else if (choice == 6)**

**{**

**int totalToday = hospital.NumPatientToday();**

**cout << "Total number of patients today: " << totalToday << endl;**

**}**

**else if (choice == 7)**

**{**

**string patientId;**

**cout << "Enter patient ID: ";**

**cin >> patientId;**

**hospital.generatePatientReport(patientId);**

**}**

**else if (choice == 8)**

**{**

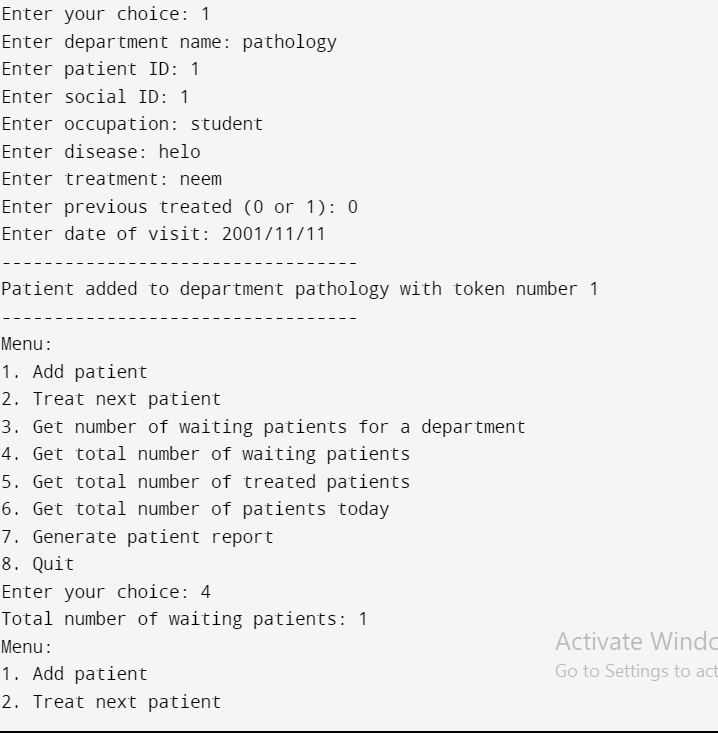
**break;**

**}**

**}**

**return 0;**

**}**

****