Worksheet 3

Task 1

1.

a.

#include <iostream>

#include <list>

using namespace std;

int main() {

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

myList.reverse();

for (int n : myList) cout << n << ' ';

cout << '\n';

return 0;

}

b.

#include <iostream>

#include <stack>

using namespace std;

void decimalToBinary(int n) {

stack<int> binary;

while (n > 0) {

binary.push(n % 2);

n /= 2;

}

while (!binary.empty()) {

cout << binary.top();

binary.pop();

}

cout << endl;

}

int main() {

int number = 10;

decimalToBinary(number);

return 0;

}

c.

#include <iostream>

#include <queue>

using namespace std;

int main() {

queue<int> q;

// Enqueue elements

q.push(1);

q.push(2);

q.push(3);

// Dequeue elements

while (!q.empty()) {

cout << q.front() << endl;

q.pop();

}

return 0;

}

Task 2

1.void insert (int x)

{

if (rear+1==size)

cout<<”queue overflow”<<endl;

++rear=x;

}

Ans-

void insert(int x) {

if (rear + 1 == size) {

cout << "Queue overflow" << endl;

} else {

rear += 1; // Increment rear first

queue[rear] = x; // Insert x into the queue at the new rear position

}

}

2. void insert (int x)

{

if (rear+1==size)

cout<<”queue overflow”<<endl;

rear+1=x;

}

Ans-

void insert(int x)

{

// Check if queue is full

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

{

cout << "Queue overflow" << endl;

} else

{

// Check if inserting the first element

if (front == -1) front = 0;

// Calculate new rear position and insert the element

rear = (rear + 1) % size;

queue[rear] = x;

}

}

3.

void display()

{

int beg=0;

while(beg<=top)

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

}

void display()

{

int t=top;

while(t>=0)

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

}

Ans- Corrected version

1. void display()

{

int beg = 0;

while (beg <= top)

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

}

1. void display()

{

int t = top;

while (t >= 0)

{

cout << stack[t--] << endl; // Display stack from top to bottom

}

}

-🡪 second is generally the preferred approach for displaying stack contents in accordance with LIFO.

4.

t pop()

{

t x=top->data;

if(top==NULL)

cout<<” Stack underflow”<<endl;

delete top; // Remove the top node

top=top->next; // move the top to the previous position

}

Ans-

T pop()

{

if (top == NULL)

{

cout << "Stack underflow" << endl;

// Handle error, e.g., throw an exception or return a default value

throw runtime\_error("Attempted to pop from an empty stack.");

}

T x = top->data; // Save the data to return

Node<T>\* toDelete = top; // Save the current top node to delete it

top = top->next; // Move the top to the next node

delete toDelete; // Delete the old top node

return x; // Return the saved data

}

Task 3

Ans-

#include <iostream>

#include <string>

using namespace std;

const int MAX\_PATIENTS = 100;

const int MAX\_DAYS = 30;

struct Patient

{

string id;

char disease;

char prescribedTreatment;

bool newPatient;

int tokenNumber;

string dateOfVisit;

};

Patient patientQueue[MAX\_PATIENTS];

int patientCounts[MAX\_DAYS] = {0};

int totalPatients = 0;

int globalTokenNumber = 1;

void addPatient (const string& id, char disease, char prescribedTreatment, bool newPatient, const string& dateOfVisit)

{

if (totalPatients >= MAX\_PATIENTS)

{

cout << "Hospital is full. Cannot add more patients." << endl;

return;

}

int dayIndex = stoi(dateOfVisit.substr(8, 2)) - 1; // Extract day from date

patientQueue[totalPatients] = {id, disease, prescribedTreatment, newPatient, globalTokenNumber++, dateOfVisit};

patientCounts[dayIndex]++;

totalPatients++;

cout << "Patient " << id << " added with token number " << globalTokenNumber - 1 << endl;

}

int getDailyPatientCount(int day)

{

if(day < 1 || day > MAX\_DAYS)

{

cout << "Invalid day." << endl;

return 0;

}

return patientCounts[day - 1];

}

double getAveragePatientsPerDay()

{

int daysWithPatients = 0;

for(int i = 0; i < MAX\_DAYS; i++)

{

if(patientCounts[i] > 0) daysWithPatients++;

}

return daysWithPatients > 0 ? static\_cast<double>(totalPatients) / daysWithPatients : 0;

}

int getTotalPatientsWaiting()

{

return totalPatients;

}

int main()

{

addPatient("ID1", 'C', 'A', true, "2023-04-01");

addPatient("ID2", 'D', 'B', false, "2023-04-02");

cout << "Total patients: " << totalPatients << endl;

cout << "Patients on 2023-04-01: " << getDailyPatientCount(1) << endl;

cout << "Average patients per day: " << getAveragePatientsPerDay() << endl;

cout << "Total patients waiting: " << getTotalPatientsWaiting() << endl;

return 0;

}

* The choice of arrays and structs is well-suited for the given scenario, balancing simplicity with the need to organize and process patient data efficiently.