TASK 1:

#include <iostream>

using namespace std;

struct node {

char data;

node\* next;

};

int precedence(char ch){

switch (ch){

case '+':

case '-':

return 1; //low precedence

case '\*':

case '/':

return 2; //medium precidence

case '^':

return 3; //high precidence

}

return -1;

}

class StackLL {

node\* top;

node\* curr;

public:

StackLL(){

top = nullptr; //null pointer

curr = nullptr;

}

void Push(char elem){

if (IsEmpty()){

node\* newNode = new node; //dynamically creating new node

top = newNode; //if stack is empty then this new node we just created is the top of stack

top->data = elem; //top's data is elem

top->next = NULL; //next of top is null

}

else{

node\* curr;

curr = top;

top = new node;

top->data = elem;

top->next = curr;

}

}

int Pop(){

if (IsEmpty()){

cout << "Stack is empty" << endl; //printed if stack is empty then nothing can be popped out

}

else{ //if stack has some data

int val;

val = top->data;

node\* curr;

curr = top;

top = top->next;

delete curr; //deleting the curr

return val; //val has top of stack and it is popped out

}

}

bool IsEmpty(){

if (top == NULL){ //if top is NULL it means stack is empty

return true;

}

else{ //else not empty

return false;

}

}

void display(){

curr = top; //we initalize curr with top of stack

cout << "Stack: " << endl;

while (curr != nullptr){ //until curr does not get equal to nullptr or until stack is not finsihed we keep printing data of stack

cout << curr->data << endl; //print data

curr = curr->next; //making curr point to the next node

}

cout << endl;

}

friend string convertToPostfix(string infix);

friend char calculate(string infix);

};

string convertToPostfix(string infix){

string post = "";

StackLL S;

for (int i = 0; i < infix.size(); i++){

if (infix[i] >= '0' && infix[i] <= '9'){

post += infix[i];

continue;

}

else if (infix[i] == '('){

S.Push(infix[i]);

}

else if (infix[i] == ')'){

while (!S.IsEmpty() && S.top->data != '('){

post += S.Pop();

}

if (!S.IsEmpty() && S.top->data != '('){

continue;

}

else{

S.Pop();

}

}

else{

while (!S.IsEmpty() && precedence(infix[i]) <= precedence(S.top->data)){

post += S.Pop();

}

S.Push(infix[i]);

}

}

while (!S.IsEmpty()){

post += S.Pop();

}

return post;

}

char calculate(string postfix){

char post;

StackLL S;

for (int i = 0; i < postfix.size(); i++){

if (postfix[i] >= '0' && postfix[i] <= '9'){

S.Push(postfix[i]);

continue;

}

else{

int temp1 = S.top->data - 48;

S.Pop();

int temp2 = S.top->data - 48;

S.Pop();

switch (postfix[i]){

case '+': { temp2 = temp2 + temp1; break; }

case '-': {temp2 = temp2 - temp1; break; }

case '\*': {temp2 = temp2 \* temp1; break; }

case '/': {temp2 = temp2 / temp1; break; }

}

S.Push(temp2 + 48);

}

}

while (!S.IsEmpty()){

post = S.Pop();

}

return post - 48;

}

int main(){

string infix = "(7+(6\*3+2)-(6/3))";

string post = convertToPostfix(infix);

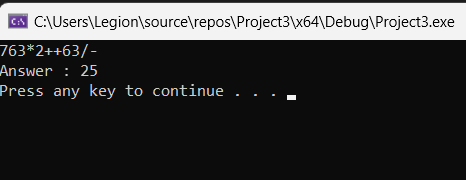
cout << post << endl;

int temp = calculate(post);

cout << "Answer : " << temp << endl;

system("pause");

}



TASK 2:

#include <iostream>

#include <string>

using namespace std;

class Node {

private:

string data;

Node\* next;

public:

Node() {

data = ""; //initally data string is empty

next = NULL; //next is NULL

}

Node(string d) {

data = d;

next = NULL;

}

void setData(string d) {

data = d; //data now has the string that we passed as parameter

}

string getData() {

return data; //return the data string

}

void setNext(Node\* n) {

next = n;

}

Node\* getNext() {

return next;

}

};

class Stack {

private:

Node\* head;

public:

Stack() {

head = NULL; //initally head is NULL because stack is empty

}

bool isEmpty() {

if (head == NULL) { //if head is NULL it means that the stack is empty

return true;

}

else //else not empty

return false;

}

string pop() {

if (isEmpty()) {

return "Stack is empty"; //if stack is empty nothing can be popped out

}

Node\* temp = head; //temprory is equal to head

string re = temp->getData();

head = head->getNext();

delete temp; //deallocating dynamic memory

return re; //returning the string

}

void push(string s) {

if (isEmpty()) {

head = new Node(s); //if stack is empty then the new node is the head node

return;

}

Node\* n = new Node(s);

n->setNext(head);

head = n;

}

string top() {

if (isEmpty()) {

return "Empty stack!!!";

}

return head->getData(); //returning the string

}

};

string convert(string msg) {

Stack s;

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

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

continue;

}

if (msg[i] == '.' || msg[i] == ',' || msg[i] == ':' || msg[i] == ';' || msg[i] == '?' || msg[i] == '!') {

//if any of these signs are found then

s.push(".");

continue;

}

string temp = "";

while (msg[i] != ' ' && i < msg.length()) {

if (msg[i] == '.' || msg[i] == ',' || msg[i] == ':' || msg[i] == ';' || msg[i] == '?' || msg[i] == '!') {

break;

}

temp.push\_back(msg[i]);

i++;

}

s.push(temp);

}

string re = "";

while (!s.isEmpty()) {

string temp = s.pop();

if (temp == ".") {

continue;

}

re = re + temp + " ";

}

return re;

}

int main() {

string msg;

cout << "enter the sentence: ";

getline(cin, msg);

cout << convert(msg) << endl;

return 0;

}

Text

Description automatically generated

TASK 3:

#include <iostream>

using namespace std;

class Node {

public:

int data;

Node\* next;

};

class stackLinkedList {

private:

Node\* top, \* curr;

public:

stackLinkedList() {

top = nullptr;

curr = nullptr; //initially top and curr are nullptr as stack is empty

}

void Push(char c) {

if (isEmpty()) {

Node\* newNode = new Node; //dynamically creating a new node

top = newNode; //if stack is empty then new node is the top of stack

top->data = c; //the data in top is the parameter we passed 'c'

top->next = NULL; //next of top is NULL i.e. end of stack

}

else {

Node\* curr;

curr = top;

top = new Node;

top->data = c;

top->next = curr;

}

}

int Pop() {

if (isEmpty()) {

cout << "Stack is empty" << endl;

return -1; //if stack is empty nothing can be popped out

}

else {

int var;

var = top->data; //var has top data of stack

Node\* curr;

curr = top;

top = top->next;

delete curr; //deallocation

return var; //else return var which now has the upmost element of stack which is to be popped out

}

}

bool isEmpty() {

if (top == NULL) { //if top is NULL then stack is empty

return true;

}

else { //else not empty

return false;

}

}

void display(int num) {

curr = top; //initalizing curr with top

cout << "Cupboard : " << num << ": " << endl;

while (curr != nullptr) { //printing data till we reach nullptr i.e. end of stack

cout << curr->data << " "; //printing

curr = curr->next; //moving to next address

}

cout << endl;

}

friend void Move(stackLinkedList& from, stackLinkedList& to);

friend void TowerOfHanoi(int plates\_num, stackLinkedList& from, stackLinkedList& mid, stackLinkedList& to);

};

void Move(stackLinkedList& from, stackLinkedList& to) {

int var1 = from.Pop();

int var2 = to.Pop();

if (var1 == -1) {

from.Push(var2);

}

else if (var2 == -1) {

to.Push(var1);

}

else if (var1 > var2) {

from.Push(var1);

from.Push(var2);

}

else {

to.Push(var2);

to.Push(var1);

}

}

void TowerOfHanoi(int plates\_num, stackLinkedList& from, stackLinkedList& mid,

stackLinkedList& to) {

int Moves = pow(2, plates\_num) - 1;

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

if (i % 3 == 0) {

Move(mid, to);

cout << endl << "Move NO : " << i << ": " << endl;

from.display(1);

mid.display(2);

to.display(3);

}

else if (i % 3 == 1) {

Move(from, to);

cout << endl << "Move NO : " << i << ": " << endl;

from.display(1);

mid.display(2);

to.display(3);

}

else if (i % 3 == 2) {

Move(from, mid);

cout << endl << "Move NO : " << i << ": " << endl;

from.display(1);

mid.display(2);

to.display(3);

}

}

}

int main() {

stackLinkedList sLL1, sLL2, sLL3;

sLL1.Push(5);

sLL1.Push(4);

sLL1.Push(3);

sLL1.Push(2);

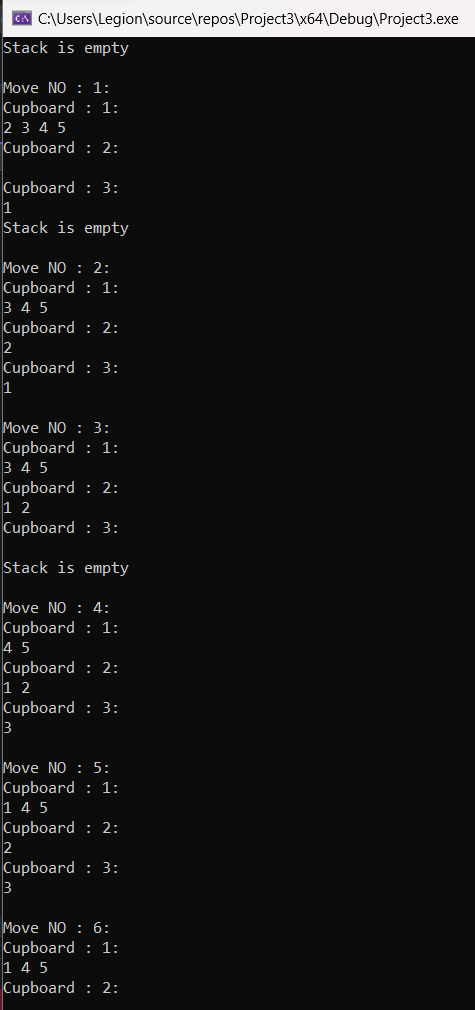
sLL1.Push(1);

TowerOfHanoi(5, sLL1, sLL2, sLL3);

system("pause");

return 0;

}



Text

Description automatically generated

A picture containing text

Description automatically generated

Text

Description automatically generated

TASK 4:

#include <iostream>

using namespace std;

#define SIZE 20 //defining queue size as 5

class Queue {

private:

int A[SIZE]; //making array of size 5

int front = -1; //front is -1 because we have to enter in queue at index 0 at first

int rear = -1; //rear is -1 because we have to enter in queue at index 0 at first

public:

void insertInMiddle(int key) {

int mid = SIZE / 2; //going to middle of queue

if (A[mid] == NULL) { //if mid of queue is NULL

A[mid] = key;

}

else {

cout << "Traffic already exists and car is already there"<<endl; //means something already exists there

}

}

void insertAtRear(int key) {

if (isFull()) {

cout << "Queue is already full\n";

return;

}

if (isEmpty()) {

front = rear = 0;

}

rear = (rear + 1) % SIZE;

A[rear] = key;

}

void enqueue(int value) { //adding value in queue

if (rear == SIZE - 1)

cout << "Queue is full" << endl; //if the rear of queue is size-1 then it means queue is full

else {

if (front == -1)

front = 0;

rear++; //else rear++

A[rear] = value; //add value in that index

}

}

void dequeue() { //to pop out of queue

if (isEmpty()) //if queue is empty nothing will be popped out

cout << "Queue is empty\n";

else

if (front == rear)

front = rear = -1;

else

front++;

}

void displayQueue() {

if (isEmpty()) //if queue is empty nothing will be displayed

cout << "Queue is empty\n";

else {

for (int i = front; i <= rear; i++) //else run for loop till all the data in the queue is printed

cout << A[i] << " ";

cout << endl;

}

}

bool isFull() {

return ((front == 0 && rear == SIZE - 1) || front == rear + 1);

}

bool isEmpty() {

return (front == -1); //if front is -1 it indicates that stack is empty

}

int getFront() {

return (A[front]); //returns the front element of queue

}

int getRear() {

return (A[rear]); //returns rear element

}

int deleteFront(){

if (isEmpty()) {

cout << "Queue is Empty"<<endl; //if queue is empty nothing can be deleted

return -1;

}

int item = A[front];

if (front = rear) {

front = rear = -1;

}

else

if (front == SIZE - 1)

front = 0;

else

front = front + 1;

return item;

}

int deleteRear() {

if (isEmpty()) {

cout << "Underflow\n";

return -1;

}

int item = A[front];

if (rear == front)

front = rear = -1;

else if (rear == 0)

rear = SIZE - 1;

else

rear = rear - 1;

return item;

}

};

int main() {

Queue queue;

int car\_number;

int ch = 0;

do {

cout << "1- Insert car at start of traffic"<<endl;

cout << "2- Insert car at end of traffic" << endl;

cout << "3- Insert car at mid of traffic" << endl;

cout << "4- Remove car from start of traffic" << endl;

cout << "5- Remove car from end of traffic" << endl;

cout << "6- check traffic situation" << endl;

cout << "7- Check if traffic jam is cleared" << endl;

cout << "8- Check if the traffic jam is worse" << endl;

cout << "9- Get the front car number plate" << endl;

cout << "10- Get the rear car number plate" << endl;

cout << "11- exit" << endl;

cin >> ch;

switch (ch) {

case 1: {

cout << "Enter the car number: ";

cin >> car\_number;

queue.enqueue(car\_number);

break;

}

case 2: {

cout << "Enter the car number: ";

cin >> car\_number;

queue.insertAtRear(car\_number);

break;

}

case 3: {

cout << "Enter the car number:";

cin >> car\_number;

queue.insertInMiddle(car\_number);

break;

}

case 4: {

cout << "Deleted Car is : " << queue.deleteFront() << endl;

break;

}

case 5: {

cout << "Deleted Car is : " << queue.deleteRear() << endl;

break;

}

case 6: {

queue.displayQueue();

break;

}

case 7: {

if (queue.isEmpty()) {

cout << "Traffic is cleared"<<endl;

}

else

cout << "Traffic is not cleard"<<endl;

break;

}

case 8: {

if (queue.isFull()) {

cout << "Traffic is out of control"<<endl;

}

else

cout << "Traffic is is under control"<<endl;

break; }

case 9: {

cout << "Number plate of car at front is : " << queue.getFront() <<endl;

break;

}

case 10: {

cout << "Number plate of car at rear is " << queue.getRear() <<endl;

break;

}

case 11: {

break;

}

default:

cout << "Invalid input"<<endl;

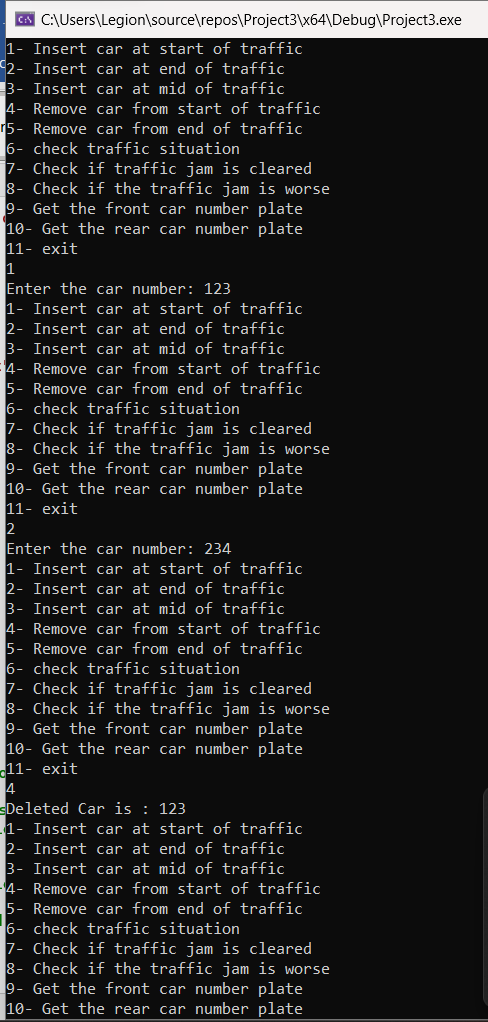
break;

}

} while (ch != 11);

return 0;

}



Text

Description automatically generated

TASK 5:

#include <iostream>

using namespace std;

class Node{

public:

char data;

Node\* next;

};

Node\* front = NULL; // Pointer to front node of the queue implemented as linked list

void push(char x){ // Function that pushes a new node to the queue from the rear

Node\* newnode = new Node; // Dynamic allocation of a new node

newnode->data = x;

newnode->next = NULL;

Node\* p = front; // A node p dynamically allocated for traversing through the linkedlist

if (front == NULL){ // Empty linked list

front = newnode;

return;

}

while (p->next != NULL){ // Traverse the linked till the last node

p = p->next;

}

p->next = newnode; // Assigning the newnode as last node in the linked list

}

char pop() // Function to pop out a element from the front and return the character

{

char ch = front->data; // Assigning the front character to a variable

front = front->next; // Front node is set to the next element thus deleting the first node

return ch; // Character of deleted first node is returned

}

int main(){

int n; // Number of senators

cout << "Enter the number of senators: ";

cin >> n;

getchar();

int gov = 0, nongov = 0; // cg -> Number of government senators , co -> Number of opposition senators

for (int i = 0; i < n; i++){ // Input into queue

char tmp;

cin >> tmp;

push(tmp); // tmp is pushed into queue from rear

}

char tmp; // Temporary variable to know which type of senator

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

tmp = pop(); // Front value is popped from the queue

if (tmp == 'G') // To increase count of government senators

gov++;

else // To increase count of opposition senators

nongov++;

}

if (gov > nongov) // If number of government senators are more than opposition senators then definitely government will win as government senators can ban all the opposition senators and still be left with atleast one government senator

cout << "Government\n";

else if (gov < nongov) // If number of opposition senators are more than government senators then definitely opposition will win as opposition senators can ban all the government senators and still be left with atleast one opposition senator

cout << "Opposition\n";

else{ // If number of government senators are equal to number of opposition senators

if (tmp == 'G') // Now tmp will be having the last senator as its value. So if tmp is a government senator, since number of both senators are equal the last opposition senator would have already banned this last government senator, so opposition wins

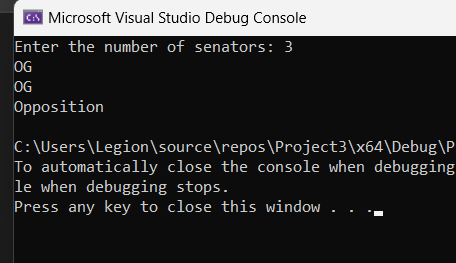
cout << "Opposition\n";

else // If tmp is a opposition senator, since number of both senators are equal the last government senator would have already banned this last opposition senator, so governmet wins

cout << "Government\n";

}

}



TASK 6:

#include <iostream>

#include <fstream>

#include <string>

#include <Windows.h>

using namespace std;

struct node {

string name, rollno;

int queueCode;

node\* next;

};

class Queue {

node\* front, \* rear;

int numOfItems, total;

public:

Queue() { //default constructor

front = rear = nullptr; //initalizing with nullptr

numOfItems = 0; // initalize with 0

total = 0;

}

Queue(int size) {

front = rear = nullptr;

numOfItems = 0;

total = size; //total is size that we pass as parameter

}

bool isEmpty() {

if (numOfItems == 0) //if numOfItems=0 it means that queue is empty

return true;

else

return false;

}

bool isFull() {

if (numOfItems == total) //if numOfItems=total or size it means that queue is full

return true;

return false;

}

void insert(int code, string nam, string roll) {

if (isFull()) { //if we try to insert in queue and it is already full it will overflow

cout << "Overflow\n";

return;

}

node\* temp = new node;

temp->queueCode = code;

temp->name = nam;

temp->rollno = roll;

temp->next = nullptr;

filing(temp);

if (isEmpty()) {

front = temp;

rear = temp;

}

else {

rear->next = temp;

rear = temp;

}

numOfItems++;

temp = nullptr;

}

void deleteFront() {

if (isEmpty()) {

return;

}

node\* temp = front->next;

delete front;

front = temp;

numOfItems--;

temp = nullptr;

}

void display() {

if (isEmpty()) {

cout << "empty" << endl;

}

else {

node\* temp = front;

while (temp != nullptr) {

cout << temp->queueCode << " |";

temp = temp->next;

}

cout << endl;

temp = nullptr;

}

}

void filing(node\* stud) {

ofstream outData;

outData.open("StudentData.txt");

outData << stud->name << ", " << stud->rollno << endl;

}

};

int main() {

Queue Accounts(15), Academics(15), Admin(15), Examination(15), Lost\_Found(15), Sports(15), Student\_Affairs(15);

bool flag = false;

int choice = 0, id = 105;

string name, rollno;

while (!flag) {

cout << "1- New Entry" << endl;

cout << "2- Display" << endl;

cout << " Enter choice : ";

cin >> choice;

if (choice == 1) {

cout << "1- Accounts" << endl;

cout << "2- Academics" << endl;

cout << "3- Admin" << endl;

cout << "4- Examination" << endl;

cout << "6- Lost & Found" << endl;

cout << "6- Sports" << endl;

cout << "7- Student Affairs" << endl;

cout << "Enter choice: ";

cin >> choice;

switch (choice) {

case 1:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Accounts.insert(id++, name, rollno);

break;

case 2:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Academics.insert(id++, name, rollno);

break;

case 3:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Admin.insert(id++, name, rollno);

break;

case 4:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Examination.insert(id++, name, rollno);

break;

case 5:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Lost\_Found.insert(id++, name, rollno);

break;

case 6:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Sports.insert(id++, name, rollno);

break;

case 7:

cout << "Enter name: ";

cin >> name;

cout << "Enter roll no: ";

cin >> rollno;

Student\_Affairs.insert(id++, name, rollno);

break;

default:

cout << "Wrong choice, resetting" << endl;

Sleep(1000);

}

system("CLS");

}

else {

Accounts.display();

Academics.display();

Admin.display();

Examination.display();

Lost\_Found.display();

Sports.display();

Student\_Affairs.display();

}

cout << "Do you want to enter more students?(0 for yes)";

cin >> flag;

}

while (id != 105){

Accounts.display();

Academics.display();

Admin.display();

Examination.display();

Lost\_Found.display();

Sports.display();

Student\_Affairs.display();

Accounts.deleteFront();

Academics.deleteFront();

Admin.deleteFront();

Examination.deleteFront();

Lost\_Found.deleteFront();

Sports.deleteFront();

Student\_Affairs.deleteFront();

Sleep(2000);

system("cls");

id--;

}

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

system("pause");

return 0;

}

TASK 7:

#include<iostream>

#include<conio.h>

using namespace std;

template<typename T>

class queue{

private:

T\* data;

int front;

int back;

int size;

int capacity;

public:

queue(){

data = new T[10];

size = 0;

front = 0;

back = 0;

capacity = 10;

}

queue(int n){

data = new T[n];

size = 0;

front = 0;

back = 0;

capacity = n;

}

~queue(){

delete[] data;

}

void enqueue(T n){

if (size == capacity){

cout << endl << "Queue is full." << endl;

return;

}

else{

data[back] = n;

back = (back + 1) % capacity;

size++;

}

}

void dequeue(){

if (size == 0){

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

return;

}

else{

data[front] = 0;

front = (front + 1) % capacity;

size--;

}

}

T get\_front(){

return data[front];

}

int get\_size(){

return size;

}

int get\_capacity(){

return capacity;

}

};

int main(){

cout << "Enter number of queues: ";

int n;

cin >> n;

queue<int>\* queues = new queue<int>[n];

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

cout << "Enter number of persons in queue " << i + 1 << ": ";

int num;

cin >> num;

for (int j = 0; j < num; j++){

int p;

cout << "Enter person " << j + 1 << ": ";

cin >> p;

queues[i].enqueue(p);

}

}

cout << endl << "Enter number of seconds for each customer: ";

int seconds;

cin >> seconds;

int current = n - 1;

int current\_second = 0;

int num\_of\_people = 0;

while (1){

if (current == -1){

current++;

current\_second++;

continue;

}

if (queues[current].get\_size() == 0){

current--;

continue;

}

if (current\_second % seconds == 0){

cout << endl << "Person " << queues[current].get\_front() << " is processed.";

queues[current].dequeue();

num\_of\_people++;

if (current > 0){

queues[current - 1].enqueue(num\_of\_people);

}

}

current\_second++;

if (current == 0 && queues[current].get\_size() == 0){

break;

}

}

cout << endl << "Processing completed in " << current\_second << " seconds." << endl;

delete[] queues;

return 0;

}

