Assignment #3

Q#1:

#include<iostream>

#include<string>

using namespace std;

class Node

{

public:

int data;

Node\* next;

Node()

{

this->data = data;

this->next = NULL;

}

};

class Stack

{

Node\* Top;

public:

Stack()

{

Top == NULL;

}

bool isEmpty()

{

return(Top == NULL);

}

bool push(char c)

{

Node\* newnode = new Node;

newnode->data = c;

newnode->next = Top;

Top = newnode;

return Top;

}

char Pop()

{

char v;

if (isEmpty())

{

cout << "Stack is Empty " << endl;

return false;

}

Node\* tempptr = Top;

v = Top->data;

Top = Top->next;

delete tempptr;

return true;

/\* cout << "The Poped Elements are " << Top->data << " " << endl;

Top = Top->next;\*/

}

bool ispalindrome()

{

int length;

int flag = 0;

char st[20];

cout << "Enter the String " << endl;

cin >> st;

length = strlen(st);

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

{

if (st[i]!=st[length-i-1])

{

flag = 1;

break;

}

}

if (flag)

{

cout << st << " String is Not palindrome " << endl;

}

else

{

cout <<st<< " String is palindrome " << endl;

return true;

}

}

void display()

{

Node\* temp = Top;

if (Top==NULL)

{

cout << "Stack is Empty " << endl;

}

else

{

cout << "Stack Elements are " << endl;

while (temp!=NULL)

{

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

temp = temp->next;

}

}

}

};

int main()

{

Stack obj;

obj.push('a');

obj.Pop();

obj.push('b');

obj.Pop();

obj.push('c');

obj.Pop();

obj.push('d');

obj.Pop();

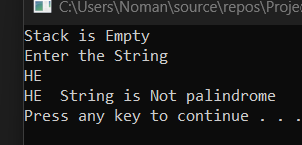
obj.display();

obj.ispalindrome();

system("pause");

}

Output:



Question #3:

#include<iostream>

#include<string>

using namespace std;

class Node

{

public:

char Data;

Node\* Next;

};

class Queue {

Node\* front;

Node\* rear;

public:

Queue() {

this->front = nullptr;

this->rear = nullptr;

}

bool isEmpty() {

return (this->front == nullptr || this->rear == nullptr);

}

void enqueue(char data) {

Node\* newNode;

newNode = new Node;

newNode->Data = data;

newNode->Next = NULL;

if (isEmpty())

{

this->front = newNode;

this->rear = newNode;

}

else {

this->rear->Next = newNode;

this->rear = newNode;

}

}

char dequeue() {

Node\* temp;

if (isEmpty()) {

return -1;

}

char data = this->front->Data;

temp = this->front->Next;

delete front;

front = temp;

return data;

}

char Top() {

if (!isEmpty()) {

return this->front->Data;

}

}

};

int main()

{

Queue obj;

string input;

int i = 0;

char temp;

int SoftwareEngineering = 0;

int ElectricalEngineering = 0;

int count = 0;

int count1 = 0;

cout << "Input: ";

cin >> input;

while (input[i] != '\0') {

obj.enqueue(input[i]);

i++;

}

while (!(obj.isEmpty())) {

temp = obj.dequeue();

if (count > 0 && temp == 'E') {

count--;

}

else if (count1 > 0 && temp == 'S') {

count1--;

}

else

{

if (temp == 'S') {

SoftwareEngineering++;

if (obj.Top() == 'E') {

obj.dequeue();

}

else

{

count++;

}

}

else {

ElectricalEngineering++;

if (obj.Top() == 'S') {

obj.dequeue();

}

else

{

count1++;

}

}

}

}

if (SoftwareEngineering > ElectricalEngineering) {

cout << "\nOutput: Software Engineering\n";

}

else if (SoftwareEngineering < ElectricalEngineering) {

cout << "\nOutput: Electrical Engineering\n";

}

else {

if (temp == 'S') {

cout << "\nOutput: Software Engineering\n";

}

else {

cout << "\nOutput: Electrical Engineering\n";

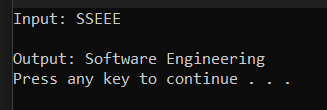
}

}

system("pause");

}

**Output:**

****

**Question #4:**

#include<iostream>

#include<fstream>

#include<string>

#include<cstdlib>

#include<Windows.h>

using namespace std;

class Queue

{

private:

string\* queueArray;

int queueSize;

int front;

int rear;

int counter = 0;

public:

int count = 0;

Queue(int size) {

this->queueSize = size;

this->queueArray = new string[size];

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

this->queueArray[i] = "-";

}

this->front = this->rear = -1;

}

bool isEmpty() {

return (this->front == -1 && this->rear == -1);

}

bool isFull() {

return (this->front == ((this->rear + 1) % this->queueSize));

}

bool enqueue(string data) {

if (isFull()) {

return false;

}

if (isEmpty()) {

front = rear = 0;

queueArray[rear] = data;

}

else

{

rear = (rear + 1) % queueSize;

queueArray[rear] = data;

}

count++;

return true;

}

string dequeue() {

if (isEmpty()) {

return "";

}

front = (front + 1) % queueSize;

string data = queueArray[front];

queueArray[front] = "-";

count--;

return data;

}

string getELement() {

if (isEmpty()) {

counter++;

return this->queueArray[counter - 1];

}

else

{

counter++;

int index = (this->front + (counter - 1)) % queueSize;

return this->queueArray[index];

}

}

};

void Display(Queue EGaming, Queue Technical, Queue Literature, Queue Socials,

Queue Creative, Queue Sports)

{

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

cout << "| EGaming\t| Technical\t| Literature\t| Socials\t|Creative\t | Sports\t | " << endl;

for (int i = 0; i < 15; i++)

{

cout << "|\t" << EGaming.getELement() << "\t|\t" <<

Technical.getELement() << "\t|\t" << Literature.getELement() << "\t|\t"

<< Socials.getELement() << "\t|\t" << Creative.getELement() <<

"\t|\t" << Sports.getELement() << "\t|" << endl;

}

}

int main()

{

Queue EGaming(15);

Queue Technical(15);

Queue Literature(15);

Queue Socials(15);

Queue Creative(15);

Queue Sports(15);

string EG[15] = { "EG 01","EG 02","EG 03","EG 04","EG 05","EG 06","EG 07" ,"EG 08","EG 09","EG 10","EG 11","EG 12","EG 13","EG 14","EG 15" };

string Tech[15] = { "Tech 01","Tech 02","Tech 03","Tech 04","Tech 05","Tech 06","Tech 07" ,"Tech 08","Tech 09","Tech 10","Tech 11","Tech 12","Tech 13","Tech14","Tech 15" };

string Lit[15] = { "Lit 01","Lit 02","Lit 03","Lit 04","Lit 05","Lit06","Lit 07" ,"Lit 08","Lit 09","Lit 10","Lit 11","Lit 12","Lit 13","Lit 14","Lit 15" };

string Soc[15] = { "Soc 01","Soc 02","Soc 03","Soc 04","Soc 05","Soc06","Soc 07","Soc 08","Soc 09","Soc 10","Soc 11","Soc 12","Soc 13","Soc 14","So 15" };

string Cre[15] = { "Cre 01","Cre 02","Cre 03","Cre 04","Cre 05","Cre06","Cre 07" ,"Cre 08","Cre 09","Cre 10","Cre 11","Cre 12","Cre 13","Cre 14","Cre15" };

string Spo[15] = { "Spo 01","Spo 02","Spo 03","Spo 04","Spo 05","Spo06","Spo 07" ,"Spo 08","Spo 09","Spo 10","Spo 11","Spo 12","Spo 13","Spo 14","Spo15" };

int choice = 0;

while (choice != 3) {

system("cls");

cout << "1.New Entry\n";

cout << "2.Display\n";

cout << "3.Exit\n\n";

cin >> choice;

switch (choice)

{

case 1:

{

system("cls");

string name;

string rollno;

string data;

cout << "Name: ";

cin >> name;

cout << "RollNo: ";

cin >> rollno;

system("cls");

int choice1 = 0;

cout << "1.E-Gaming\n";

cout << "2.Technical\n";

cout << "3.Literature\n";

cout << "4.Socials\n";

cout << "5.Creative\n";

cout << "6.Sports\n";

cin >> choice1;

switch (choice1)

{

case 1:

{

data = EG[EGaming.count];

EGaming.enqueue(data);

}

break;

case 2:

{

data = Tech[Technical.count];

Technical.enqueue(data);

}

break;

case 3:

{

data = Lit[Literature.count];

Literature.enqueue(data);

}

break;

case 4:

{

data = Soc[Socials.count];

Socials.enqueue(data);

}

break;

case 5:

{

data = Cre[Creative.count];

Creative.enqueue(data);

}

break;

case 6:

{

data = Spo[Sports.count];

Sports.enqueue(data);

}

break;

}

}

break;

case 2:

{

system("cls");

Display(EGaming, Technical, Literature, Socials, Creative,

Sports);

Sleep(2000);

if (!EGaming.isEmpty())

EGaming.dequeue();

if (!Technical.isEmpty())

Technical.dequeue();

if (!Literature.isEmpty())

Literature.dequeue();

if (!Socials.isEmpty())

Socials.dequeue();

if (!Creative.isEmpty())

Creative.dequeue();

if (!Sports.isEmpty())

Sports.dequeue();

}

break;

case 3:

{

exit(0);

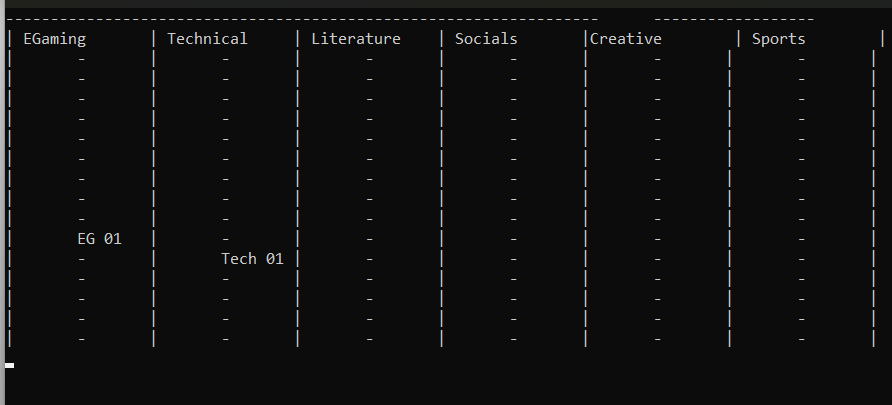
}

}

}

}

**Output:**

****

**Question #5:**

#include<iostream>

#include<cstdlib>

#include<Windows.h>

using namespace std;

template <class T>

class Node

{

public:

T Data;

Node\* Next;

};

template <class U>

class Queue

{

Node<U>\* front;

Node<U>\* rear;

public:

Queue() {

this->front = this->rear = nullptr;

}

bool isEmpty() {

return (this->front == NULL || this->rear == NULL);

}

void enqueue(int data) {

Node<U>\* newNode = new Node<U>;

newNode->Data = data;

newNode->Next = NULL;

if (isEmpty()) {

this->front = newNode;

this->rear = newNode;

}

else {

this->rear->Next = newNode;

this->rear = newNode;

}

}

int dequeue() {

int data;

Node<U>\* temp;

if (!isEmpty()) {

data = front->Data;

temp = front->Next;

delete front;

front = temp;

return data;

}

}

void display() {

Node<U>\* temp = this->front;

while (temp) {

cout << temp->Data << " ";

temp = temp->Next;

}

cout << endl;

}

};

void displayAll(Queue<int> obj[], int size) {

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

if (!obj[i].isEmpty()) {

cout << "Queue " << i + 1 << ": ";

obj[i].display();

}

}

}

void createQueue(Queue<int> obj[], int size) {

obj[0].dequeue();

int i = size - 1;

while (i > 0)

{

if (!obj[i].isEmpty()) {

obj[i - 1].enqueue(obj[i].dequeue());

}

i--;

}

}

int main()

{

Queue<int> obj[100];

int size;

cout << "No of Queue: ";

cin >> size;

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

for (int j = 0; j < 10; j++) {

obj[i].enqueue(j);

}

}

while (!obj[0].isEmpty()) {

system("cls");

displayAll(obj, size);

Sleep(1000);

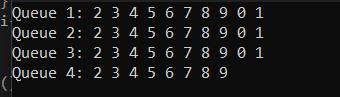
createQueue(obj, size);

}

system("cls");

}

**Output:**

****