# Task 1:

## Code:

#include<iostream>

#include<string>

using namespace std;

int priority(char a);

template <class type>

struct Node

{

type data;

Node\* next;

Node()

{

next = NULL;

}

};

class Stack

{

Node<char>\* top;

public:

Stack()

{

top = NULL;

}

bool push(char value)

{

Node<char>\* newnode = new Node<char>;

newnode->data = value;

newnode->next = top;

top = newnode;

return true;

}

char pop()

{

if (!(is\_Empty()))

{

char a;

Node<char>\* temp = top;

top = top->next;

a = temp->data;

delete temp;

temp = NULL;

return a;

}

else {

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

return 0;

}

}

bool is\_Empty()

{

if (top == NULL)

{

return true;

}

return false;

}

void display()

{

Node<char>\* temp = top;

cout << "Displaying all the Stack!" << endl;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl << endl;

}

int calculator(string a)

{

a = postfix(a);

char temp;

int num1, num2;

char temp1;

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

{

temp = a[i];

if (temp == '^' || temp == '/' || temp == '\*' || temp == '+' || temp == '-' && top != NULL)

{

num2 = int(pop()) - 48;

num1 = int(pop()) - 48;

if (temp == '^')

{

num1 = num1 ^ num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '/')

{

num1 = num1 / num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '\*')

{

num1 = num1 \* num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '+')

{

num1 = num1 + num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '-')

{

num1 = num1 - num2;

temp1 = char(num1) + 48;

push(temp1);

}

}

else

{

push(temp);

}

}

return num1;

}

string postfix(string x)

{

char temp;

string postfix;

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

{

temp = x[i];

if (temp >= '1' && temp <= '9')

{

postfix += temp;

}

else if (temp == '(')

{

push(temp);

}

else if (temp == ')')

{

if (is\_Empty() != true) {

while (top->data != '(')

{

postfix += pop();

}

pop();

}

}

else {

if (is\_Empty() != true) {

while (priority(temp) >= priority(top->data) && top != NULL)

{

if (top->data == '(')

{

pop();

}

else {

postfix += pop();

}

}

}

if (temp != ')')

{

push(temp);

}

}

}

while (!is\_Empty())

{

if (top->data == '(')

{

pop();

}

else {

postfix += pop();

}

}

return postfix;

}

};

int main()

{

Stack obj;

string x;

int ans;

cout << "Enter The Expression : ";

getline(cin, x);

ans = obj.calculator(x);

cout << ans << endl;

system("pause");

return 0;

}

int priority(char a)

{

if (a == '^') {

return 1;

}

else if (a == '/')

{

return 2;

}

else if (a == '\*')

{

return 3;

}

else if (a == '+')

{

return 4;

}

else if (a == '-')

{

return 5;

}

else if (a == '(')

{

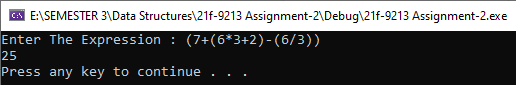
return 6;

}

return 0;

}

## Output:



# Task 2:

## Code:

#include<iostream>

#include<string>

using namespace std;

int priority(char a);

template <class type>

struct Node

{

type data;

Node\* next;

Node()

{

next = NULL;

}

};

class Stack

{

Node<char>\* top;

public:

Stack()

{

top = NULL;

}

bool push(char value)

{

Node<char>\* newnode = new Node<char>;

newnode->data = value;

newnode->next = top;

top = newnode;

return true;

}

char pop()

{

if (!(is\_Empty()))

{

char a;

Node<char>\* temp = top;

top = top->next;

a = temp->data;

delete temp;

temp = NULL;

return a;

}

else {

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

return 0;

}

}

bool is\_Empty()

{

if (top == NULL)

{

return true;

}

return false;

}

void display()

{

Node<char>\* temp = top;

cout << "Displaying all the Stack!" << endl;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl << endl;

}

int calculator(string a)

{

a = postfix(a);

char temp;

int num1, num2;

char temp1;

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

{

temp = a[i];

if (temp == '^' || temp == '/' || temp == '\*' || temp == '+' || temp == '-' && top != NULL)

{

num2 = int(pop()) - 48;

num1 = int(pop()) - 48;

if (temp == '^')

{

num1 = num1 ^ num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '/')

{

num1 = num1 / num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '\*')

{

num1 = num1 \* num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '+')

{

num1 = num1 + num2;

temp1 = char(num1) + 48;

push(temp1);

}

else if (temp == '-')

{

num1 = num1 - num2;

temp1 = char(num1) + 48;

push(temp1);

}

}

else

{

push(temp);

}

}

return num1;

}

string postfix(string x)

{

char temp;

string postfix;

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

{

temp = x[i];

if (temp >= '1' && temp <= '9')

{

postfix += temp;

}

else if (temp == '(')

{

push(temp);

}

else if (temp == ')')

{

if (is\_Empty() != true) {

while (top->data != '(')

{

postfix += pop();

}

pop();

}

}

else {

if (is\_Empty() != true) {

while (priority(temp) >= priority(top->data) && top != NULL)

{

if (top->data == '(')

{

pop();

}

else {

postfix += pop();

}

}

}

if (temp != ')')

{

push(temp);

}

}

}

while (!is\_Empty())

{

if (top->data == '(')

{

pop();

}

else {

postfix += pop();

}

}

return postfix;

}

string sequenceMutation(string x)

{

char temp = '/0';

string mutated;

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

{

temp = x[i];

if ((temp >= 'a' && temp <= 'z') || (temp >= 'A' && temp <= 'Z') || temp == ' ')

{

push(temp);

}

}

x = "";

while (is\_Empty() != true)

{

temp = '\0';

while (temp != ' ' && is\_Empty() != true) {

temp = pop();

mutated = temp + mutated;

}

mutated += ' ';

x = x + mutated;

mutated = "";

}

return x;

}

};

int main()

{

Stack obj;

string x;

cout << "Enter The Expression : ";

getline(cin, x);

x = obj.sequenceMutation(x);

cout << x << endl;

system("pause");

return 0;

}

int priority(char a)

{

if (a == '^') {

return 1;

}

else if (a == '/')

{

return 2;

}

else if (a == '\*')

{

return 3;

}

else if (a == '+')

{

return 4;

}

else if (a == '-')

{

return 5;

}

else if (a == '(')

{

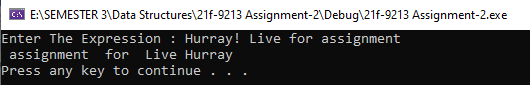
return 6;

}

return 0;

}

## Output:



# Task 3:

## Code:

#include<iostream>

using namespace std;

struct Node

{

int data;

Node\* next;

Node()

{

next = NULL;

}

};

class Stack {

Node\* top;

int count = 0;

public:

Stack()

{

top = NULL;

}

bool Push(int value)

{

Node\* newnode = new Node;

newnode->data = value;

newnode->next = top;

top = newnode;

count++;

return true;

}

bool is\_Empty()

{

if (top == NULL)

{

return true;

}

return false;

}

bool pop()

{

if (is\_Empty())

{

cout << "The Stack is Already Empty!" << endl;

return false;

}

else

{

Node\* temp = top;

top = top->next;

delete temp;

temp = NULL;

count--;

return true;

}

}

int topval()

{

if (top == NULL)

{

return 6;

}

else return top->data;

}

void display()

{

Node\* temp = top;

while (temp != NULL)

{

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

temp = temp->next;

}

}

bool tower(int i1, int i2, Stack\* ptr)

{

bool x,j;

if (ptr[i1 - 1].topval() < ptr[i2 - 1].topval())

{

ptr[i2 - 1].Push(ptr[i1 - 1].topval());

ptr[i1 - 1].pop();

x = ptr[2].checkend();

j = ptr[1].checkend();

if (x == true||j==true)

{

return true;

}

else {

return false;

}

}

else {

cout << "Operation can not be performed!" << endl;

return false;

}

}

bool checkend()

{

if (count == 5)

{

return true;

}

return false;

}

};

int main()

{

Stack\* obj = new Stack[3];

bool c = false;

for (int i = 5; i > 0; i--)

{

obj[0].Push(i);

}

int ch1, ch2;

do {

for (int i = 0; i < 3; i++)

{

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

obj[i].display();

cout << endl;

}

cout << "Enter From Whcih Stack you want to move:";

cin >> ch1;

cout << "Enter Destination ";

cin >> ch2;

c = obj[0].tower(ch1, ch2, obj);

} while (c == false);

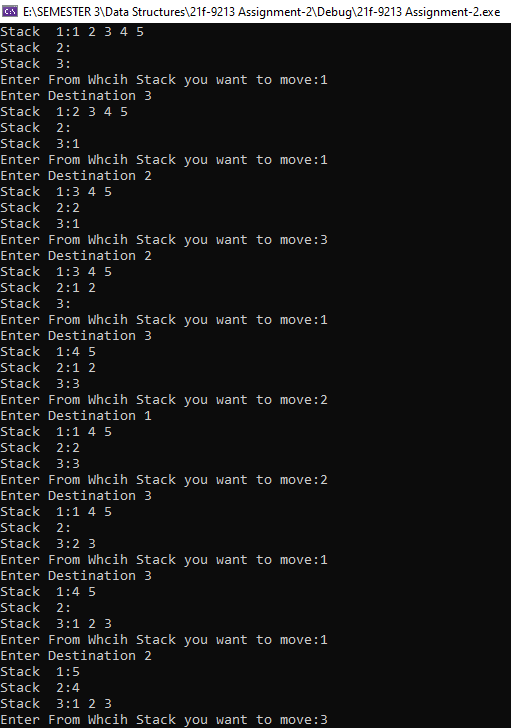
cout << "You Have Completed The Game!" << endl;

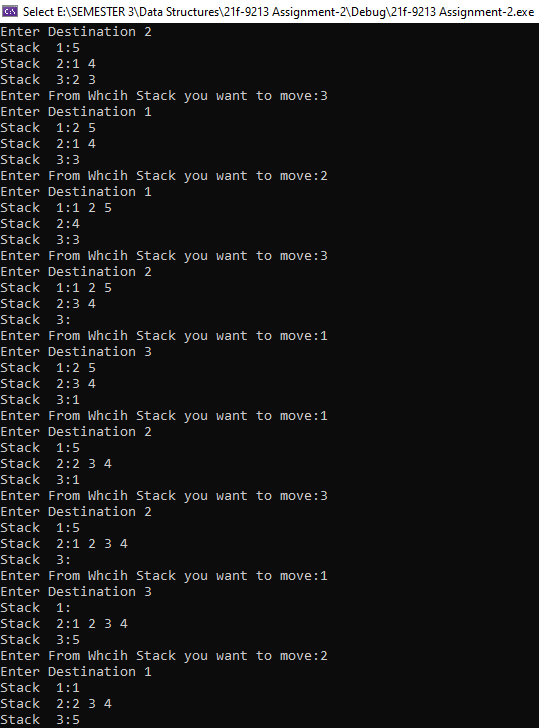
system("Pause");

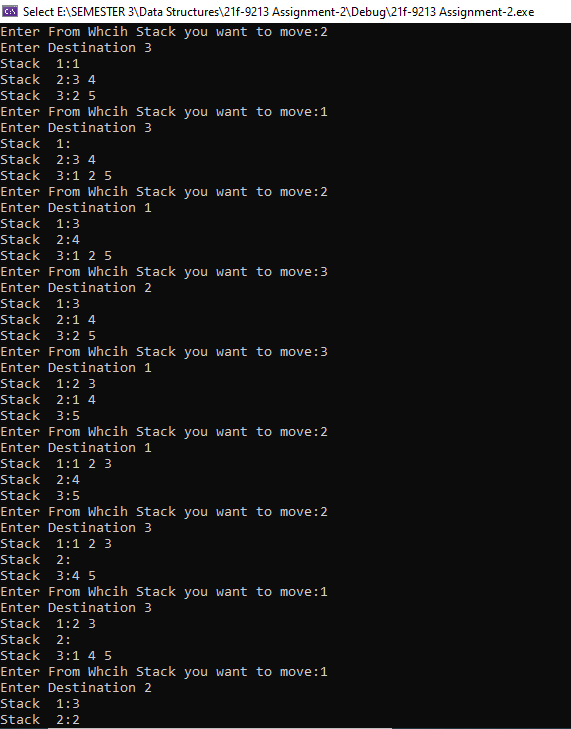
return 0;

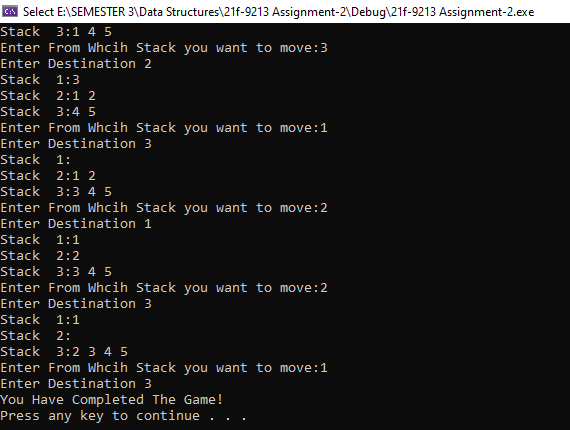
}

## Output:









# Task 4:

## Code:

#include<iostream>

#include<string>

#include<fstream>

using namespace std;

template<class t>

class De\_Queue

{

private:

t\* arr;

int Front, Rear, Q\_Size;

public:

De\_Queue(int s)

{

Front = Rear = -1;

arr = new t[s];

Q\_Size = s;

for (int i = 0; i < Q\_Size; i++)

{

arr[i] = -1;

}

}

bool is\_Full()

{

if ((Rear + 1 % Q\_Size) == Front) {

return true;

}

return false;

}

bool is\_Empty()

{

if (Front == -1 && Rear == -1)

{

return true;

}

return false;

}

bool Insertion\_Last(t value)

{

if (is\_Full())

{

cout << "The Queue is already Full!" << endl;

return false;

}

else if (is\_Empty())

{

Front = Rear = 0;

arr[Rear] = value;

FIle\_Incoming(value, "Last");

return true;

}

else if (arr[(Rear + 1) % Q\_Size] == -1) {

Rear = (Rear + 1) % Q\_Size;

arr[Rear] = value;

FIle\_Incoming(value, "Last");

return true;

}

else {

int j = 0;

while (arr[(Rear) % Q\_Size] != -1 && j < Q\_Size)

{

Rear = (Rear + 1) % Q\_Size;

j++;

}

if (arr[(Rear) % Q\_Size] == -1)

{

arr[Rear] = value;

FIle\_Incoming(value, "Last");

return true;

}

else {

cout << "Queue is Full" << endl;

return false;

}

}

}

bool Insertion\_Front(t value)

{

if (is\_Full())

{

cout << "Queue Is Already Full!" << endl;

return false;

}

else if (is\_Empty())

{

Front = Rear = 0;

arr[Front] = value;

FIle\_Incoming(value, "Front");

return true;

}

else if (Front == 0) {

Front = Q\_Size - 1;

arr[Front] = value;

FIle\_Incoming(value, "Front");

return true;

}

else {

Front--;

arr[Front] = value;

FIle\_Incoming(value, "Front");

return true;

}

}

bool Insertion\_Middle(t value, int x)

{

if (is\_Full())

{

cout << "Queue Is Already Full!" << endl;

return false;

}

else if (is\_Empty())

{

Front = Rear = x - 1;

arr[Front] = value;

FIle\_Incoming(value, "Middle");

return true;

}

else if (arr[x - 1] == -1)

{

arr[x - 1] = value;

FIle\_Incoming(value, "Middle");

return true;

}

else {

cout << "ALready Element is There!" << endl;

return false;

}

}

t Dequeue\_Last()

{

t x;

if (is\_Empty())

{

cout << "Queue is Already Empty!" << endl;

return 0;

}

else if (Front == Rear)

{

x = arr[Rear];

Front = Rear = -1;

File\_Outgoing(x, "Last");

return x;

}

else if (Rear == 0) {

x = arr[Rear];

arr[Rear] = -1;

Rear = Q\_Size - 1;

File\_Outgoing(x, "Last");

return x;

}

else {

x = arr[Rear];

arr[Rear] = -1;

Rear--;

File\_Outgoing(x, "Last");

return x;

}

}

t Dequeue\_Front()

{

t x;

if (is\_Empty())

{

cout << "Queue is Already Empty!" << endl;

return 0;

}

else if (Front == Rear)

{

x = arr[Front];

Front = Rear = -1;

File\_Outgoing(x, "Front");

return x;

}

else {

x = arr[Front];

arr[Front] = -1;

Front = (Front + 1) % Q\_Size;

File\_Outgoing(x, "Front");

return x;

}

}

void display()

{

int x = Front;

if (!is\_Empty()) {

do

{

cout << arr[x] << " ";

x = (x + 1) % Q\_Size;

} while (x != Front);

}

cout << endl;

}

bool FIle\_Incoming(t value, string end)

{

ofstream Write;

Write.open("Incoming.txt", ios::app);

Write << value << " " << end << endl;

Write.close();

return true;

}

bool File\_Outgoing(t value, string end)

{

ofstream Write;

Write.open("Outgoing.txt", ios::app);

Write << value << " " << end << endl;

Write.close();

return true;

}

void Display\_Incoming()

{

ifstream read;

string x;

read.open("Incoming.txt");

while (!read.eof()) {

getline(read, x);

cout << x << endl;

}

read.close();

}

void Display\_Outgoing()

{

ifstream read;

string x;

read.open("Outgoing.txt");

while (!read.eof()) {

getline(read, x);

cout << x << endl;

}

read.close();

}

};

int main()

{

int x;

cout << "Enter the Size Of Queue ";

cin >> x;

De\_Queue<int> obj(x);

int val;

do {

cout << "Press 0 to exit" << endl;

cout << "Press 1 TO Enqueue At Front :" << endl;

cout << "Press 2 TO Enqueue At End :" << endl;

cout << "Press 3 TO Enqueue At Middle :" << endl;

cout << "Press 4 TO DEqueue At Front :" << endl;

cout << "Press 5 TO DEqueue At Last :" << endl;

cout << "Press 6 TO Display Queue :" << endl;

cout << "Press 7 TO Display Incomings :" << endl;

cout << "Press 8 TO Display Outgoings :" << endl;

cin >> x;

switch (x)

{

case 0: {

exit(0);

break;

}

case 1: {

cout << "Enter The Value";

cin >> val;

obj.Insertion\_Front(val);

break;

}

case 2: {

cout << "Enter The Value";

cin >> val;

obj.Insertion\_Last(val);

break;

}

case 3:

{

int i = 0;

cout << "Enter The Value";

cin >> val;

cout << "Enter the Index ";

cin >> i;

obj.Insertion\_Middle(val, i);

break;

}

case 4: {

obj.Dequeue\_Front();

break;

}

case 5: {

obj.Dequeue\_Last();

break;

}

case 6: {

obj.display();

break;

}

case 7: {

obj.Display\_Incoming();

break;

}

case 8: {

obj.Display\_Outgoing();

break;

}

}

} while (x != 0);

system("pause");

return 0;

}

## Output:

|  |  |
| --- | --- |
|  |  |

|  |  |
| --- | --- |
|  |  |

# Task 5:

## Code:

#include<iostream>

#include<string>

using namespace std;

struct Node

{

char data;

Node\* next;

Node()

{

next = NULL;

}

};

class Queue

{

Node\* Front, \* Rear;

public:

Queue()

{

Front = Rear = NULL;

}

bool Enqueue(char value)

{

if (Front == NULL)

{

Node\* newnode = new Node;

newnode->data = value;

newnode->next = Front;

Front = newnode;

Rear = newnode;

return true;

}

else {

Node\* newnode = new Node;

newnode->data = value;

Rear->next = newnode;

Rear = newnode;

return true;

}

}

char De\_Queue()

{

if (Front == Rear && (!(is\_Empty())))

{

char a;

Node\* temp = Front;

Front = Rear = NULL;

a = temp->data;

delete temp;

temp = NULL;

return a;

}

else if (!(is\_Empty()))

{

char a;

Node\* temp = Front;

Front = Front->next;

a = temp->data;

delete temp;

temp = NULL;

return a;

}

else {

cout << "Queue is already Empty!" << endl;

return 0;

}

}

bool is\_Empty()

{

if (Front == NULL && Rear == NULL)

{

return true;

}

return false;

}

void display()

{

Node\* temp = Front;

cout << "Displaying all the Queue!" << endl;

while (temp != NULL)

{

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

temp = temp->next;

}

cout << endl << endl;

}

bool senetor(string x)

{

int count = 0;

char temp1, temp2;

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

{

Enqueue(x[i]);

}

temp1 = De\_Queue();

temp2 = De\_Queue();

while (Front != NULL && count <= x.length())

{

count++;

if (temp1 != temp2)

{

Enqueue(temp1);

temp1 = De\_Queue();

temp2 = De\_Queue();

}

else

{

Enqueue(temp1);

temp1 = temp2;

temp2 = De\_Queue();

}

}

if (temp1 == 'g' || temp1 == 'G')

{

cout << "Government wins!" << endl;

}

else if (temp1 == 'O' || temp1 == 'o') {

cout << "Opposition wins!" << endl;

}

return true;

}

};

int main()

{

Queue obj;

string x;

cout << "Enter The Expression : ";

getline(cin, x);

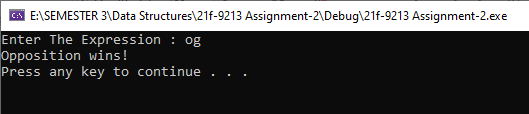
obj.senetor(x);

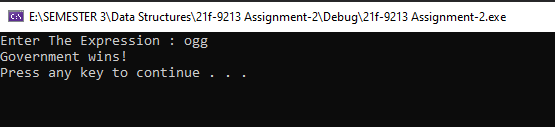
system("pause");

return 0;

}

## Output:





# Task 6:

## Code:

#include<iostream>

#include<string>

#include<fstream>

#include <chrono>

#include <thread>

#include<iomanip>

using namespace std;

string t;

template<class type>

struct Node

{

type data;

Node\* next;

Node()

{

next = NULL;

}

};

class Queue

{

Node<string>\* Front, \* Rear;

Node<string>\* dis;

int count;

public:

Queue()

{

Front = Rear = NULL;

count = 0;

}

bool Enqueue(string value)

{

if (count < 15)

{

count++;

assigncode(value);

if (Front == NULL)

{

Node<string>\* newnode = new Node<string>;

newnode->data = value;

newnode->next = Front;

Front = newnode;

Rear = newnode;

dis = newnode;

return true;

}

else {

Node<string>\* newnode = new Node<string>;

newnode->data = value;

Rear->next = newnode;

Rear = newnode;

return true;

}

}

else {

cout << "Currrently Queue is full!" << endl;

return false;

}

}

string De\_Queue()

{

count--;

if (Front == Rear && (!(is\_Empty())))

{

string a;

Node<string>\* temp = Front;

Front = Rear = NULL;

a = temp->data;

delete temp;

temp = NULL;

return a;

}

else if (!(is\_Empty()))

{

string a;

Node<string>\* temp = Front;

Front = Front->next;

a = temp->data;

dis = Front;

delete temp;

temp = NULL;

return a;

}

return "";

}

bool is\_Empty()

{

if (Front == NULL && Rear == NULL)

{

return true;

}

return false;

}

string display()

{

if (dis != NULL) {

t = dis->data;

dis = dis->next;

return t;

}

return "-";

}

void assigncode(string& a)

{

int x = count % 16;

a = a + to\_string(x);

}

};

bool writeFile(string x)

{

ofstream write;

write.open("Students Visited.txt", ios::app);

write << x << endl;

write.close();

return true;

}

int main()

{

Queue acc, academics, admin, exam, lost, sports, studenaffairs;

int option;

do {

cout << "Press 0 to Exit!" << endl;

cout << "Press 1 For New Entry " << endl;

cout << "Press 2 to Display" << endl;

cin >> option;

switch (option)

{

case 0: {

exit(0);

}

case 1: {

string details, rollno, name;

int user;

cout << "Enter Your Name : ";

cin.ignore();

getline(cin, name);

cout << "Enter Your Roll no : ";

cin >> rollno;

details = name + " " + rollno;

cout << "Press 1 for Accounts" << endl;

cout << "Press 2 for Academics" << endl;

cout << "Press 3 for Admin" << endl;

cout << "Press 4 for Examinations" << endl;

cout << "Press 5 for Lost And Found" << endl;

cout << "Press 6 for Sports" << endl;

cout << "Press 7 for Student Affairs" << endl;

cin >> user;

switch (user)

{

case 1: {

details += " Accounts";

acc.Enqueue("Acc");

writeFile(details);

break;

}

case 2: {

details += " Academics";

academics.Enqueue("Acd");

writeFile(details);

break;

}

case 3: {

details += " Admin";

admin.Enqueue("Adm");

writeFile(details);

break;

}

case 4: {

details += " Examinations";

exam.Enqueue("Exm");

writeFile(details);

break;

}

case 5: {

details += " Lost&Found";

lost.Enqueue("L&F");

writeFile(details);

break;

}

case 6: {

details += " Sports";

sports.Enqueue("Sp");

writeFile(details);

break;

}

case 7: {

details += " Student Affairs";

academics.Enqueue("Sa");

writeFile(details);

break;

}

default:

cout << "Invalid input!" << endl;

break;

}

break;

}

case 2: {

system("cls");

for (int i = 0; i < 70; i++)

{

cout << "- ";

}

cout << endl << "| Accounts | Academics | Admin | Examinations| Loast&Found | Sports | Student Affairs | " << endl;

for (int i = 0; i < 70; i++)

{

cout << "- ";

}

cout << endl;

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

{

cout << "|" << acc.display() << setw(11) << "|";

cout << academics.display() << setw(11) << "|";

cout << admin.display() << setw(11) << "|";

cout << exam.display() << setw(11) << "|";

cout << lost.display() << setw(11) << "|";

cout << sports.display() << setw(11) << "|";

cout << studenaffairs.display() << setw(11) << "|" << endl;

}

this\_thread::sleep\_for(chrono::seconds(2s));

acc.De\_Queue();

academics.De\_Queue();

admin.De\_Queue();

exam.De\_Queue();

lost.De\_Queue();

sports.De\_Queue();

studenaffairs.De\_Queue();

break;

}

default:

cout << "Invalid Input!" << endl;

break;

}

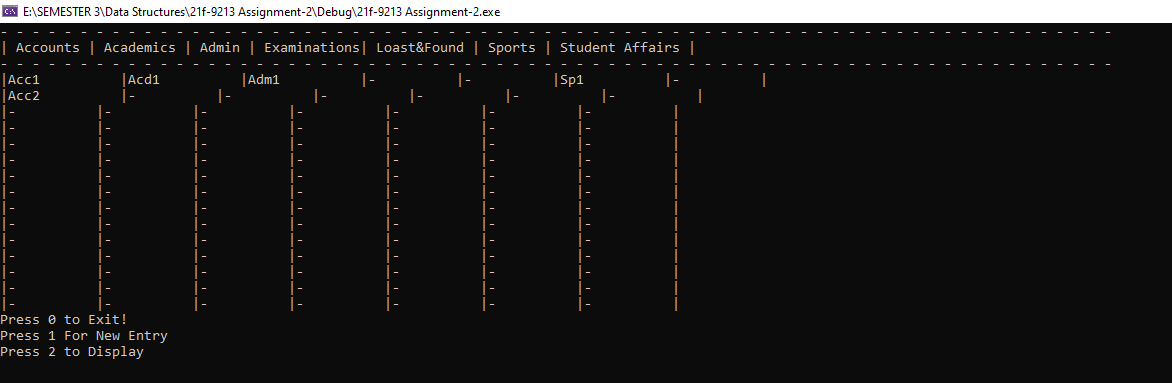
} while (option != 0);

system("pause");

return 0;

}

## Output:



## After 2sec:



# Task 7:

## Code:

#include<iostream>

#include<string>

#include<thread>

#include<chrono>

using namespace std;

template<class type>

struct Node

{

int data;

Node\* next;

Node()

{

next = NULL;

}

};

template<class t>

class Queue

{

Node<t>\* Front;

Node<t>\* Rear;

int count;

public:

Queue()

{

Front = Rear = NULL;

count = 0;

}

bool is\_Empty()

{

if (Front == NULL && Rear == NULL)

{

return true;

}

return false;

}

bool is\_Full()

{

if (count >= 10)

{

return true;

}

return false;

}

bool Enqueue(t value)

{

if (is\_Full())

{

cout << "The Queue is Already Full !" << endl;

return false;

}

else if (is\_Empty())

{

count++;

Node<t>\* newnode = new Node<t>;

newnode->data = value;

newnode->next = Front;

Front = newnode;

Rear = newnode;

return true;

}

else {

count++;

Node<t>\* newnode = new Node<t>;

newnode->data = value;

Rear->next = newnode;

Rear = newnode;

return true;

}

}

t Dequeue()

{

if (is\_Empty())

{

return -1;

}

else if (Front == Rear)

{

count--;

Node<t>\* temp = Front;

Front = Rear = NULL;

t var;

var = temp->data;

delete temp;

temp = NULL;

return var;

}

else {

count--;

Node<t>\* temp = Front;

Front = Front->next;

t var = temp->data;

delete temp;

temp = NULL;

return var;

}

}

void Arrange(Queue\* ptr, int s)

{

t var;

ptr[0].Dequeue();

for (int i = 1; i < s; i++)

{

var = ptr[i].Dequeue();

if (var != -1) {

ptr[i - 1].Enqueue(var);

}

}

}

void Display()

{

Node<t>\* ptr = Front;

while (ptr != NULL)

{

cout << ptr->data << " ";

ptr = ptr->next;

}

cout << endl;

}

};

int main()

{

int user, option;

cout << "Enter the Number of Queues : ";

cin >> user;

Queue<int>\* obj = new Queue<int>[user];

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

{

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

{

obj[i].Enqueue(j);

}

}

do {

cout << "Press0 to Exit" << endl;

cout << "Press1 to Enqueue" << endl;

cout << "Press2 to Dequeue" << endl;

cout << "Press3 to Display" << endl;

cin >> option;

switch (option)

{

case 0: {

exit(0);

break;

}

case 1: {

int x;

cout << "Enter the value You Want to Enter :";

cin >> x;

obj[user - 1].Enqueue(x);

break;

}

case 2: {

obj[0].Arrange(obj, user);

break;

}

case 3: {

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

{

obj[i].Display();

}

break;

}

default: {

cout << "Invalid Choice!" << endl;

break;

}

}

this\_thread::sleep\_for(chrono::seconds(2));

obj[0].Arrange(obj, user);

} while (option != 0);

system("pause");

return 0;

}

## Output:

