**Assignment# 2**

**TASK# 2: [sequence mmutation]**

**Code:**

#include<iostream>

#include<string>

using namespace std;

struct Node

{

char character;

Node\* next;

Node(char character)

{

this->character = character;

this->next = nullptr;

}

};

class Stack

{

private:

Node\* top, \* tail;

public:

//push function

bool push(char character)

{

Node\* newNode = new Node(character);

newNode->next = top;

top = newNode;

return true;

}

// pop function

char pop()

{

if (!isEmpty())

{

char charac = top->character;

Node\* temp = top;

top = top->next;

delete temp;

return charac;;

}

return NULL;

}

// isEmpty function

bool isEmpty()

{

if (top == nullptr)

return true;

return false;

}

// string sequence mutation function

string sequenceMutation(string str)

{

Stack s2,s3;

char character;

// loop for pushing values in the stack 1 without punctuation

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

{

if (int(str[i]) == 32 ||int(str[i]) == 65 || int(str[i]) == 66 || int(str[i]) == 67 || int(str[i]) == 68 || int(str[i]) == 69 || int(str[i]) == 70 || int(str[i]) == 71 || int(str[i]) == 72 || int(str[i]) == 73 || int(str[i]) == 74 || int(str[i]) == 75 || int(str[i]) == 76 || int(str[i]) == 77 || int(str[i]) == 78 || int(str[i]) == 79 || int(str[i]) == 80 || int(str[i]) == 81 || int(str[i]) == 82 || int(str[i]) == 83 || int(str[i]) == 84 || int(str[i]) == 85 || int(str[i]) == 86 || int(str[i]) == 87 || int(str[i]) == 88 || int(str[i]) == 89 || int(str[i]) == 90|| int(str[i]) == 97|| int(str[i]) == 98|| int(str[i]) == 99|| int(str[i]) == 100|| int(str[i]) == 101|| int(str[i]) == 102|| int(str[i]) == 103|| int(str[i]) == 104|| int(str[i]) == 105|| int(str[i]) == 106|| int(str[i]) == 107|| int(str[i]) == 108|| int(str[i]) == 109|| int(str[i]) == 110|| int(str[i]) == 111|| int(str[i]) == 112|| int(str[i]) == 113|| int(str[i]) == 114|| int(str[i]) == 115|| int(str[i]) == 116|| int(str[i]) == 117|| int(str[i]) == 118|| int(str[i]) == 119|| int(str[i]) == 120|| int(str[i]) == 121|| int(str[i]) == 122)

{

this->push(str[i]);

}

}

this->push(' ');

// loop for popping out values from the stack 1 and push in the stack 2

while (this->top)

{

s2.push(this->pop());

}

Stack tempStack;

// loop that reverse the string

while (s2.top)

{

character=s2.pop();

if (character == ' ')

{

s3.push(' ');

while (tempStack.top)

{

s3.push(tempStack.pop());

}

}

else

{

tempStack.push(character);

}

}

Node\* current = s3.top;

// this is string stores all charaters in a string to return to main function

string ReturnedString;

while (current)

{

ReturnedString = ReturnedString + current->character;

current = current->next;

}

return ReturnedString;

}

// display function

void display()

{

Node\* current = top;

while (current)

{

cout << current->character;

current = current->next;

}

}

// constructor

Stack()

{

this->tail = this->top = nullptr;

}

// destructor for deallocating the memory

~Stack()

{

while (this->top)

{

this->pop();

}

}

};

// main function

int main()

{

Stack s1;

string str;

// grtting dtring from user

cout << "Enter a string:";

getline(cin, str);

// passing string to sequenceMutation function of stack

cout << "Given String: " << endl << str << endl;

str = s1.sequenceMutation(str);

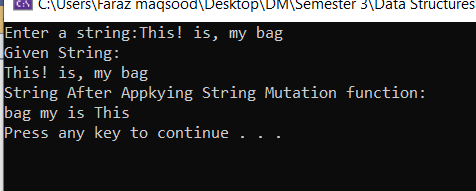
cout << "String After Appkying String Mutation function: "<<endl << str << endl;;

system("pause");

return 0;

}

**Output:**



**TASK# 4: (Traffic Jam)**

**Code:**

#include<iostream>

using namespace std;

struct Node

{

int carNumber;

Node\* next;

Node(int data)

{

this->carNumber = data;

this->next = nullptr;

}

};

class Dequeue

{

private:

Node\* front, \* rare,\*tail;

public:

// insertion at front

bool enqueueFront(int data)

{

Node\* newNode = new Node(data);

if (front == nullptr)

{

tail = newNode;

}

newNode->next = front;

front = newNode;

cout << "Car "<<data<<" is Entered from front" << endl;

return true;

}

// insertion at end

bool enqueueEnd(int data)

{

Node\* newNode = new Node(data);

tail->next = newNode;

tail = newNode;

cout << "Car "<<data<<" entered from end" << endl;

return true;

}

// insertion at middle

bool enqueueMiddle(int key, int data)

{

int count = 0;

Node\* current = front;

// for counting nodes

while (current)

{

count++;

current = current->next;

}

// if given key is greater than Nodes then insert at end

if (count < key)

{

enqueueEnd(data);

return false;

}

else if (key <=0)

{

cout<<"Invalid KEY"<<endl;

return false;

}

else if (key == 1)

{

enqueueFront(data);

return true;

}

// if given key is valid and approperiate number of nodes are exist than insert nodes at right place

else if (count >= key)

{

Node\* newNode = new Node(data);

count = 2;

current = front;

while (current)

{

if (count == key)

{

newNode->next = current->next;

current->next = newNode;

cout << "Car "<<data <<" Entered From middel" << endl;

return true;

}

count++;

current = current->next;

}

}

return false;

}

// deletion from front

bool dequeueFront()

{

if (front != nullptr)

{

Node\* temp = front;

cout << "car "<<temp->carNumber <<" leaves the traffic from front" << endl;

front = front->next;

delete temp;

return true;

}

cout << "There is no node to delete" << endl;

return false;

}

// deletion from end

bool dequeueEnd()

{

Node\* current = front;

while (current->next!=tail)

{

current = current->next;

}

Node\* temp = tail;

tail = current;

cout << "car number "<<tail->carNumber<<" leaves the traffic from end" << endl;

delete temp;

tail->next = nullptr;

return true;

}

// display function

void display()

{

Node\* current = front;

if (current == nullptr)

cout << "There is no traffic jam" << endl;

else

while (current)

{

cout << current->carNumber << endl;

current = current->next;

}

}

// constructor

Dequeue()

{

this->front = this->rare = this->tail = nullptr;

}

// destructor that also deallocate the memory

~Dequeue()

{

while (front)

{

Node\* temp = front;

front = front->next;

delete temp;

}

}

};

int main()

{

Dequeue d;

d.enqueueFront(1);

d.enqueueFront(2);

d.enqueueFront(3);

d.enqueueFront(0);

d.enqueueEnd(4);

d.enqueueMiddle(7, 23);

d.dequeueFront();

d.dequeueEnd();

d.enqueueMiddle(3, 223);

cout << "Remaining cars Stuck in Traffic are" << endl;

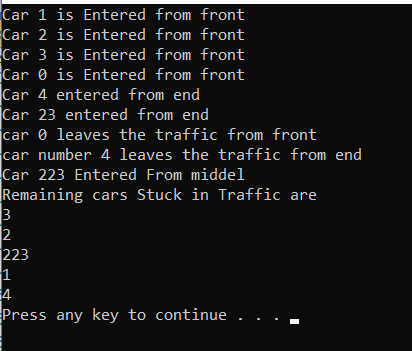
d.display();

system("pause");

return 0;

}

**Output:**



**TASK# 4: (Senate Election)**

**Code:**

#include<iostream>

#include<string>

using namespace std;

struct Node

{

char symbol;

Node\* next;

Node(char symbol)

{

this->next = nullptr;

this->symbol = symbol;

}

};

class Queue

{

private:

Node\* front,\*tail;

public:

void push(char symbol)

{

Node\* newNode = new Node(symbol);

if (front == nullptr)

{

front = newNode;

tail = newNode;

}

else

{

tail->next = newNode;

tail = newNode;

}

tail->next = front;

}

//voting function for decidng mendeate in law

bool voting()

{

// condition for confirming that senetors are participate

if (front != nullptr) {

char choice;

Node\* current = front;

do

{

// to ask the senetor if he wants to ban the next senetor

if (current->next->symbol == 'o' || current->next->symbol == 'O')

{

cout << "The next senetor is of opposition to ban him/her press y/Y else press any key:";

cin >> choice;

if (choice == 'y' || choice == 'Y')

{

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

}

}

else

{

cout << "The next senetor is of Government to ban him/her press y/Y else press any key:";

cin >> choice;

if (choice == 'y' || choice == 'Y')

{

Node\* temp = current->next;

current->next = current->next->next;

delete temp;

}

}

// counting the number of senetors

int oppositionSeats = 0, governmentsSeat = 0;

Node\* tempCurrent = current;

do

{

if (tempCurrent->symbol == 'o' || tempCurrent->symbol == 'O')

oppositionSeats++;

else

governmentsSeat++;

tempCurrent = tempCurrent->next;

} while (tempCurrent != current);

//announcing the result if all remaining senetors are of same party

if (oppositionSeats == 0 && governmentsSeat >= 1)

{

cout << "OutPut:" << "government" << endl;

break;

}

else if (governmentsSeat == 0 && oppositionSeats >= 1)

{

cout <<"OutPut:"<< "Opposition"<<endl;

break;

}

current = current->next;

} while (current != current->next);

}

else

cout << "There are no senetors";

return true;

}

// display

bool display()

{

Node\* current1 = front;

if (front != nullptr)

{

cout << "Input:";

do

{

cout << current1->symbol;

current1 = current1->next;

} while (current1!=front);

}

return true;

}

//constrcutor

Queue()

{

this->front = this->tail = nullptr;

}

};

// main function

int main()

{

Queue q;

string str;

cout << "Enter o/O for oppositionSenetors and g/G for govenmentSenetors others discard:";

getline(cin, str);

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

{

if (str[i] == 'o' || str[i] == 'O' || str[i] == 'g' || str[i] == 'G')

{

q.push(str[i]);

}

}

q.display();

cout << endl;

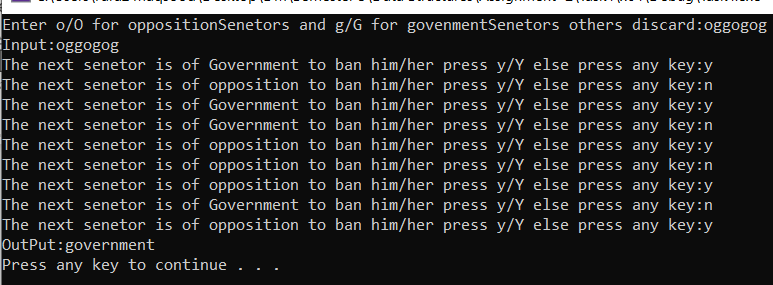
q.voting();

system("pause");

return 0;

}

**Output:**



**TASK# 6: (Bank queue management)**

**code:**

#include<iostream>

using namespace std;

template<class T>

class queue

{

private:

T name[10];

int front, rare;

public:

// enqueu function

bool enqueue(T data)

{

if (front == -1 && rare == -1)

{

front = rare = 0;

name[rare] = data;

return true;

}

else

{

if (isfull())

{

name[rare] = data;

return true;

}

}

return true;

}

// isfull function

bool isfull()

{

if (front == 0 && rare == 9)

return false;

// checks if rare is at the end but there are some free space available at the start of array

else

{

rare++;

if (rare == 10)

rare = 0;

if (rare == front)

return false;

return true;

}

}

//empty function

bool isEmpty()

{

if (rare == -1)

return false;

return true;

}

T dequeu(T data)

{

T temp = name[front];

name[front] = data;

front++;

return temp;

}

//main function

void functionality(queue <string>arr[],int size)

{

arr[size - 1].dequeu("0");

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

{

arr[size - i].enqueue(arr[size - (i + 1)].dequeu(" 0"));

}

}

// display function

void display(queue <string>arr[],int size)

{

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

cout << " " << i + 1 << " ";

cout << endl;

int temp = arr[1].front;

do

{

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

{

cout << arr[i].name[temp]<<" ";

}

cout << endl;

temp++;

if (temp == 10)

temp = 0;

} while (temp!=arr[1].front);

}

// constructor

queue()

{

front = rare = -1;

}

};

int main()

{

int num;

cout << "Enter number of queues:";

cin >> num;

queue<string> \*obj,temp;

obj = new queue<string>[num];

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

{

int c = 48;

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

{

string str = "person";

obj[i].enqueue(str + char(c));

}

}

cout << "You have following queues and person in that queues"<<endl;

temp.display(obj,num);

char choice;

do

{

cout << "If you want to process person press y/Y else press any key:";

cin >> choice;

system("cls");

if (choice == 'y' || choice == 'Y')

temp.functionality(obj, num);

else

break;

} while (1);

temp.display(obj, num);

cout << "0 means this is person is shifted:";

system("pause");

return 0;

}

**Output:**

