1. Implement queue using array

int cap = 100005;

//Function to push an element x in a queue.

void MyQueue :: push(int x)

{

arr[rear] = x;

rear = (rear+1)%cap;

}

//Function to pop an element from queue and return that element.

int MyQueue :: pop()

{

if(front==rear) return -1;

int temp = front;

front = (front+1)%cap;

return arr[temp];

}

1. Operations of queue

//Function to push an element in queue.

void enqueue(queue<int> &q,int x)

{

q.push(x);

}

//Function to remove front element from queue.

void dequeue(queue<int> &q)

{

q.pop();

}

//Function to find the front element of queue.

int front(queue<int> &q)

{

return q.front();

}

//Function to find an element in the queue.

string find(queue<int> q, int x)

{

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

{

if(q.front()==x)

return "Yes";

q.push(q.front());

q.pop();

}

return "No";

}

1. Implement queue using linked list

//Function to push an element into the queue.

void MyQueue:: push(int x)

{

QueueNode \*temp = new QueueNode(x);

if(front==NULL)

{

front = temp;

rear = temp;

return;

}

rear->next = temp;

rear = temp;

}

//Function to pop front element from the queue.

int MyQueue :: pop()

{

if(front==NULL) return -1;

if(front==rear)

{

QueueNode \*temp = rear;

front=NULL;

rear=NULL;

return temp->data;

}

QueueNode \*temp = front;

front=front->next;

return temp->data;

}

1. Queue reversal

queue<int> rev(queue<int> q)

{

stack<int> s;

while(!q.empty())

{

s.push(q.front());

q.pop();

}

while(!s.empty())

{

q.push(s.top());

s.pop();

}

return q;

}

1. Queue using two stacks

//Function to push an element in queue by using 2 stacks.

void StackQueue :: push(int x)

{

while(!s1.empty())

{

s2.push(s1.top());

s1.pop();

}

s1.push(x);

while(!s2.empty())

{

s1.push(s2.top());

s2.pop();

}

}

//Function to pop an element from queue by using 2 stacks.

int StackQueue :: pop()

{

if(s1.empty()) return -1;

int temp = s1.top();

s1.pop();

return temp;

}

1. Stack using two queues

//Function to push an element into stack using two queues.

void QueueStack :: push(int x)

{

q1.push(x);

}

//Function to pop an element from stack using two queues.

int QueueStack :: pop()

{

if(q1.empty()) return -1;

while(q1.size()!=1)

{

q2.push(q1.front());

q1.pop();

}

int temp = q1.front();

q1.pop();

q1.swap(q2);

return temp;

}

1. Generate binary numbers

vector<string> generate(int N)

{

queue<string> q, temp;

vector<string> v;

q.push("1");

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

{

v.push\_back(q.front());

temp.push(q.front()+"0");

temp.push(q.front()+"1");

q.pop();

if(q.empty())

{

q.swap(temp);

}

}

return v;

}

1. Reverse first k elements of queue

queue<int> modifyQueue(queue<int> q, int k) {

stack<int> s;

int r = q.size()-k;

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

{

s.push(q.front());

q.pop();

}

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

{

q.push(s.top());

s.pop();

}

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

{

q.push(q.front());

q.pop();

}

return q;

}

1. Circular tour

int tour(petrolPump arr[],int n)

{

// Consider first petrol pump as a starting point

int start = 0;

int end = 1;

int curr\_petrol = arr[start].petrol - arr[start].distance;

/\* Run a loop while all petrol pumps are not visited.

And we have reached first petrol pump again with 0 or more petrol \*/

while (end != start || curr\_petrol < 0)

{

// If current amount of petrol in truck becomes less than 0, then

// remove the starting petrol pump from tour

while (curr\_petrol < 0 && start != end)

{

// Remove starting petrol pump. Change start

curr\_petrol -= arr[start].petrol - arr[start].distance;

start = (start + 1) % n;

// If 0 is being considered as start again, then there is no

// possible solution

if (start == 0)

return -1;

}

// Add a petrol pump to current tour

curr\_petrol += arr[end].petrol - arr[end].distance;

end = (end + 1) % n;

}

// Return starting point

return start;

}