**Practical 5**

**Aim:** Queue Implementation using Array and Linked list.

**Description:**

**Queue:** Queue is a linear data structure that follows a particular order in which the operations are performed for storing data. The order is First In First Out (FIFO).

1. enqueue(): Inserts an element at the end of the queue i.e. at the rear end.
2. dequeue(): removes and returns an element that is at the front end of the queue.
3. isEmpty(): This operation indicates whether the queue is empty or not.
4. isFull(): This operation indicates whether the queue is full or not.

* Application:

1. When a resource is shared among multiple consumers. Examples include CPU scheduling, Disk Scheduling.
2. When data is transferred asynchronously (data not necessarily received at the same rate as sent) between two processes. Examples include IO Buffers, pipes, file IO, etc.
3. A queue can be used as an essential component in various other data structures.

* The complexity of Queue:

**Time complexity -**

|  | Access | Search | Insert | Delete |
| --- | --- | --- | --- | --- |
| Average | O(n) | O(n) | O(1) | O(1) |
| Worst | O(n) | O(n) | O(1) | O(1) |

* Space Complexity **-** O(n)
* Simple Queue: it’s also known as a linear queue is the most basic version of a queue. Here, insertion of an element i.e. the Enqueue operation takes place at the rear end and removal of an element i.e. the Dequeue operation takes place at the front end

Code:

#include <iostream>

using namespace std;

template<typename T>

class Queue{

class Node{

public:

T data;

Node \*prev;

Node \*next;

Node(T data){

this->data = data;

this->prev = NULL;

this->next = NULL;

}

}\*front, \*back;

int cnt = 0;

public:

bool isEmpty(){

return this->cnt == 0?true:false;

}

int size(){

return this->cnt;

}

void enqueue(T data){

Node\* n = new Node(data);

if(this->isEmpty()){

this->front = n;

this->back = n;

}else{

(this->back)->next = n;

n->prev = this->back;

this->back = n;

}

cnt++;

}

bool dequeue(){

if(isEmpty())

return false;

Node \*n = this->front;

front = front->next;

if(this->front != NULL)

(this->front)->prev = NULL;

delete n;

cnt--;

return true;

}

string toString(){

if(this->isEmpty()){

return "Queue is empty!\n";

}

Node \*temp = this->front;

string str = "Queue [";

while(temp != NULL){

str = str.append(to\_string(temp->data));

str = str.append(", ");

temp = temp -> next;

}

str = str.substr(0, str.length()-2);

str.append("]\n");

return str;

}

};

void utilQueue(){

int choice = 1;

Queue<int> \*queue = new Queue<int>();

while(choice == 1 || choice == 2 || choice == 3 ){

cout << "Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}\nchoice : ";

cin >> choice;

switch(choice){

case 1:

int data;

cout << "enter element : ";

cin >> data;

queue->enqueue(data);

break;

case 2: if(!queue->dequeue())

cout <<"underflow!" << endl;

break;

case 3: cout << queue->toString(); break;

}

}

cout << "closing the application..." << endl;

}

int main(){

utilQueue();

return 0;

}

Output:

| Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 1  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 2  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 3  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [1, 2, 3]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  underflow!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 2  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [2]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 6  closing the application... |
| --- |

* Circular Queue: In a circular queue, the element of the queue acts as a circular ring. The working of a circular queue is similar to the linear queue except for the fact that the last element is connected to the first element.

Code:

#include <iostream>

using namespace std;

template<typename T>

class Queue{

T \*queue;

int front = -1, rear = -1, capacity, cnt = 0;

public:

Queue(int capacity){

queue = new T[capacity]{0};

this->capacity = capacity;

}

bool isEmpty(){

return this->cnt == 0?true:false;

}

bool isFull(){

return this->cnt == this->capacity?true:false;

}

int size(){

return this->capacity;

}

bool enqueue(){

if(this->isFull()){

return false;

}

int data;

cout << "enter element : ";

cin >> data;

if(this->front == -1 && this->rear == -1){

this->front = this->front + 1;

this->rear = this->rear + 1;

this->cnt = this->cnt + 1;

this->queue[this->rear] = data;

}else{

this->rear = (this->rear+1)%capacity;

this->queue[this->rear] = data;

this->cnt = this->cnt + 1;

}

return true;

}

bool dequeue(){

if(isEmpty())

return false;

cout << this->queue[this->front] <<" removed from queue."<<endl;

this->queue[this->front] = 0;

this->front = (this->front+1)%capacity;

this->cnt = this->cnt-1;

return true;

}

string toString(){

if(this->isEmpty()){

return "Queue is empty!\n";

}

string str = "Queue [";

int index;

for(int i = 0; i < this->capacity; i++){

index = (i+this->front)%this->capacity;

str.append(to\_string(this->queue[index]));

if(i != capacity-1)

str.append(", ");

}

return str.append("]\n");

}

};

void utilQueue(){

int choice;

cout << "enter queue capacity : ";

cin >> choice;

Queue<int> \*queue = new Queue<int>(choice);

choice = 1;

while(choice == 1 || choice == 2 || choice == 3 ){

cout << "Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}\nchoice : ";

cin >> choice;

switch(choice){

case 1:

cout << (queue->enqueue()?"inserted!":"overflow")<<endl;

break;

case 2: if(!queue->dequeue())

cout <<"underflow!" << endl;

break;

case 3: cout << queue->toString(); break;

}

}

cout << "closing the application..." << endl;

}

int main(){

utilQueue();

return 0;

}

Output:

| enter queue capacity : 4  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 34  inserted!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 43  inserted!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 67  inserted!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [34, 43, 67, 0]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  34 removed from queue.  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [43, 67, 0, 0]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 34  inserted!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [43, 67, 34, 0]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 45  inserted!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [43, 67, 34, 45]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  overflow  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  43 removed from queue.  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  67 removed from queue.  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  34 removed from queue.  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  45 removed from queue.  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  underflow!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue is empty!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 5  closing the application... |
| --- |

* Priority Queue: an abstract data type that is similar to a queue, and every element has some priority value associated with it. The priority of the elements in a priority queue determines the order in which elements are served (i.e., the order in which they are removed). If in any case the elements have the same priority, they are served as per their ordering in the queue.

Code:

#include <iostream>

using namespace std;

template<typename T>

class Priority\_Queue{

class Node{

public:

T data;

int priority;

Node \*next;

Node(T data, int priority){

this->priority = priority;

this->data = data;

this->next = NULL;

}

}\*front;

int cnt = 0;

public:

bool isEmpty(){

return this->cnt == 0?true:false;

}

int size(){

return this->cnt;

}

bool getPriority(int p1, int p2){

return (p1 < p2) ? true: false;

}

void enqueue(T data, int priority){

Node \*curr, \*newNode;

newNode = new Node(data, priority);

if(front == NULL || getPriority(priority, front->priority)){

newNode->next = front;

front = newNode;

}else{

curr = this->front;

while(curr->next != NULL && getPriority(curr->next->priority, priority))

curr = curr->next;

newNode->next = curr->next;

curr->next = newNode;

}

cnt++;

}

bool dequeue(){

if(isEmpty())

return false;

Node \*n = this->front;

cout << n->data << " removed from the queue"<<endl;

this->front = (this->front)->next;

delete n;

cnt--;

return true;

}

string toString(){

if(this->isEmpty()){

return "Priority Queue is empty!\n";

}

Node \*temp = this->front;

string str = "Priority Queue [";

while(temp != NULL){

str = str.append(to\_string(temp->data));

if(temp->next != NULL)

str = str.append(", ");

temp = temp -> next;

}

str.append("]\n");

return str;

}

};

void utilQueue(){

int choice = 1;

Priority\_Queue<int> \*queue = new Priority\_Queue<int>();

while(choice == 1 || choice == 2 || choice == 3 ){

cout << "Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}\nchoice : ";

cin >> choice;

switch(choice){

case 1:

int data, priority;

cout << "enter element : ";

cin >> data;

cout << "enter priority : ";

cin >> priority;

queue->enqueue(data, priority);

break;

case 2:

if(!queue->dequeue())

cout <<"underflow!" << endl;

break;

case 3:

cout << queue->toString();

Break;

}

}

cout << "closing the application..." << endl;

}

int main(){

utilQueue();

return 0;

}

Output:

| Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 234  enter priority : 1  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 78  enter priority : 3  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 89  enter priority : 2  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 1  enter priority : 99  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Priority Queue [234, 89, 78, 1]  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  234 removed from the queue  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Priority Queue [89, 78, 1]  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : 56  enter priority : 1  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Priority Queue [56, 89, 78, 1]  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  56 removed from the queue  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  89 removed from the queue  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  78 removed from the queue  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  1 removed from the queue  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  underflow!  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Priority Queue is empty!  Priority Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 5  closing the application... |
| --- |

* **Restricted Queue**

1. **Input-restricted Queue:** In an input restricted queue, insertion operation can be performed at only one end, while deletion can be performed from both ends.
2. **Output-restricted Queue** In an output-restricted queue, deletion operation can be performed at only one end, while insertion can be performed from both ends.

Code:

#include <iostream>

using namespace std;

template<typename T>

class Output\_resQueue{

class Node{

public:

T data;

Node \*prev;

Node \*next;

Node(T data){

this->data = data;

this->prev = NULL;

this->next = NULL;

}

}\*front, \*back;

int cnt = 0;

public:

bool isEmpty(){

return this->cnt == 0?true:false;

}

int size(){

return this->cnt;

}

void enqueue(T data, char insertFrom){

Node\* newNode = new Node(data);

if(this->isEmpty()){

this->front = newNode;

this->back = newNode;

}else{

if(insertFrom == 'r'){

(this->back)->next = newNode;

newNode->prev = this->back;

this->back = newNode;

}else{

newNode->next = front;

front = newNode;

}

}

cnt++;

}

bool dequeue(){

if(isEmpty())

return false;

Node \*n = this->front;

front = front->next;

if(this->front != NULL)

(this->front)->prev = NULL;

cout << n->data << " removed from the queue"<<endl;

delete n;

cnt--;

return true;

}

string toString(){

if(this->isEmpty()){

return "Queue is empty!\n";

}

Node \*temp = this->front;

string str = "Queue [";

while(temp != NULL){

str = str.append(temp->data);

if(temp->next != NULL)

str = str.append(", ");

temp = temp -> next;

}

str.append("]\n");

return str;

}

};

void utilQueue(){

int choice = 1;

Output\_resQueue<string> \*queue = new Output\_resQueue<string>();

while(choice == 1 || choice == 2 || choice == 3 ){

cout << "Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}\nchoice : ";

cin >> choice;

switch(choice){

case 1:{

string data;

char insertFrom;

cout << "enter element : ";

cin >> data;

cout << "enter place where you want to insert a element(r : back) : ";

cin >> insertFrom;

queue->enqueue(data, insertFrom);

}break;

case 2: if(!queue->dequeue())

cout <<"underflow!" << endl;

break;

case 3: cout << queue->toString(); break;

}

}

cout << "closing the application..." << endl;

}

int main(){

utilQueue();

return 0;

}

Output:

| Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : krushna  enter place where you want to insert a element(r : back) :  d  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : amit  enter place where you want to insert a element(r : back) : r  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : saurabh  enter place where you want to insert a element(r : back) : d  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [saurabh, krushna, amit]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  saurabh removed from the queue  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 1  enter element : gaurav  enter place where you want to insert a element(r : back) : r  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 3  Queue [krushna, amit, gaurav]  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  krushna removed from the queue  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  amit removed from the queue  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  gaurav removed from the queue  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 2  underflow!  Queue Operations {1 : enqueue, 2 : dequeue, 3 : display, other : exit}  choice : 5  closing the application... |
| --- |

**Conclusion:** Queue and its type implemented successfully.