

Hands-on Activity 5.1	
Queues	
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 09/11/2025
Section: CPE21S4	Date Submitted: 09/11/2025
Name(s): Cabrera, Gabriel A.	Instructor: Engr. Jimlord Quejado
6. Output	
<pre>Queue5.1.cpp > ⌂ displayAll<T>(std::queue<T>) 1 #include <iostream> 2 #include <string> 3 #include <queue> 4 template <typename T> 5 void displayAll(std::queue<T> n); 6 7 int main() { 8 std::string list1[5] = {"Gabriel", "Base", "Lynn", "Tamayo", "Ycay"}; 9 std::queue<std::string> queue1; 10 11 for(int i = 0; i < 5; i++){ 12 queue1.push(list1[i]); 13 } 14 15 displayAll(queue1); 16 17 } 18 19 template <typename T> 20 void displayAll(std::queue<T> n){ 21 while(!n.empty()){ 22 std::cout << n.front() << std::endl; 23 n.pop(); 24 } 25 }</pre>	
<pre>* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe Gabriel Base Lynn Tamayo Ycay * Terminal will be reused by tasks, press any key to close it.</pre>	

Table 5-1. Queues using C++ STL

Very simple implementation of queues using the STL, I made an array of names to display and used a loop to enqueue all the names and display them after. Using a template type T for more flexibility in the data entered.

```
Queue5.2.cpp > Queue5.2.cpp
1 #include <iostream>
2 #include "queue.h"
3 #include <string.h>
4
5
6 int main (){
7     std::string CPE21S4[5] = {"Gabriel", "Base", "Lynn", "Tamayo", "Ycay"};
8     Queue <std::string> queue1;
9     for(int i = 0; i < 5; i++){
10         queue1.enqueue(CPE21S4[i]);
11     }
12     queue1.display();
13     return 0;
14 }
15
```

Queue5.1.cpp

Queue5.2.cpp

queue.h X

```
C queue.h > Node<T> > Node(T)
1 #ifndef QUEUE_H
2 #define QUEUE_H
3 #include <iostream>
4
5
6
7 template <typename T>
8 class Node{
9     public:
10         T data;
11         Node* next;
12
13         Node(T new_data){
14             data = new_data;
15             next = nullptr;
16         }
17
18 };
19
20 template <typename T>
21 class Queue{
22     private:
23         Node<T> *front;
24         Node<T> *rear;
25     public:
26         //creates an empty queue
27         Queue(){
28             front = rear = nullptr;
29             std::cout<<"A queue has been created.\n";
30         }
31
32         //isEmpty
33         bool isEmpty(){
34             return front == nullptr;
35         }
36 }
```

Queue5.1.cpp

Queue5.2.cpp

queue.h X

```
C queue.h > Node<T> > Node(T)
1  #ifndef QUEUE_H
21 class Queue{
33     bool isEmpty(){
34         return front == nullptr;
35     }
36
37     //enqueue
38     void enqueue(T new_data){
39         Node<T> *new_node=new Node<T> (new_data);
40         if (isEmpty()){
41             front = rear = new_node;
42             std::cout<<"Enqueued to an empty queue.\n";
43             return;
44         }
45         rear->next=new_node;
46         rear=new_node;
47         std::cout<<"Successfully enqueued.\n";
48     }
49
50     //dequeue
51     void dequeue(){
52         if (isEmpty()){
53             return;
54         }
55
56         Node<T>* temp=front;
57
58         if (front==nullptr) {
59             rear=nullptr;
60         }
61         else{
62             front=front->next;
63         }
64         delete temp;
```

Queue5.1.cpp

Queue5.2.cpp

queue.h X

```
C queue.h > Node<T> > Node(T)
1 #ifndef QUEUE_H
21 class Queue{
51     void dequeue(){
52         if (isEmpty()){
54             }
55
56             Node<T>* temp=front;
57
58             if (front==nullptr) {
59                 rear=nullptr;
60             }
61             else{
62                 front=front->next;
63             }
64             delete temp;
65         }
66
67         //getfront
68         void getFront(){
69             if (isEmpty()){
70                 std::cout<<"The queue is empty.\n";
71                 return;
72             }
73             std::cout<<"Current Front: "<<front->data<<std::endl;
74         }
75
76         //getrear
77         void getRear(){
78             if (isEmpty()){
79                 std::cout<<"The queue is empty.\n";
80                 return;
81             }
82             std::cout<<"Current Rear: "<<rear->data<<std::endl;
83         }
}
```

```

Queue5.1.cpp Queue5.2.cpp queue.h X
C queue.h > Node<T> > Node(T)
1 #ifndef QUEUE_H
21 class Queue{
67     //getfront
68     void getFront(){
69         if (isEmpty()){
70             std::cout<<"The queue is empty.\n";
71             return;
72         }
73         std::cout<<"Current Front: "<<front->data<<std::endl;
74     }
75
76     //getrear
77     void getRear(){
78         if (isEmpty()){
79             std::cout<<"The queue is empty.\n";
80             return;
81         }
82         std::cout<<"Current Rear: "<<rear->data<<std::endl;
83     }
84
85     //display
86     void display(){
87         if (isEmpty()){
88             std::cout<<"The queue is empty.\n";
89             return;
90         }
91         Node<T> *temp=front;
92         while (temp!=nullptr){
93             std::cout<<temp->data<<" ";
94             temp=temp->next;
95         }
96         std::cout<<std::endl;
97     }
98 }
```

* Executing task: C:/Windows/System32/cmd.exe /d /c .\build\Debug\outDebug.exe

A queue has been created.

Equeued to an empty queue.

Successfully enqueued.

Successfully enqueued.

Successfully enqueued.

Successfully enqueued.

Gabriel Base Lynn Tamayo Ycay

* Terminal will be reused by tasks, press any key to close it.

Table 5-2. Queues using Linked List Implementation

The program uses a header file with a linked list implementation to use the queue structure. The isEmpty function checks if the queue is empty by verifying if the front pointer is a null pointer. The enqueue function adds a new element to the rear of the queue by creating a new node and updating the rear pointer. The getFront function retrieves the data from the front of the queue, while the getRear function retrieves the data from the rear. Finally, the Display function prints all the elements in the queue, starting from the front and moving towards the rear, without altering the queue's structure.

```
Queue5.1.cpp Queue5.2.cpp queue.h Queue5.3.cpp X queueArr.h queue_supplementary.h
Queue5.3.cpp > displayAll<T>(queueArr<T>)
1 #include "queueArr.h"
2 #include <string>
3
4 template <typename T>
5 void displayAll(queueArr<T> queue1);
6
7 int main(){
8
9     std::string studentArr[5] = {"A", "B", "C", "D", "E"};
10    queueArr<std::string> studentQueue(5);
11
12
13    int i = 0;
14    while(!studentQueue.isFull()){
15        studentQueue.enqueue(studentArr[i]);
16        i++;
17    }
18
19    studentQueue.Front();
20    studentQueue.Back();
21
22    displayAll(studentQueue);
23
24    return 0;
25 }
26
27 template <typename T>
28 void displayAll(queueArr<T> queue1){
29     while(!queue1.isEmpty()){
30         queue1.Front();
31         queue1.dequeue();
32     }
33 }
```

Queue5.1.cpp Queue5.2.cpp queue.h Queue5.3.cpp queueArr.h X queue

C queueArr.h > queueArr<T> > isEmpty()

```
1
2 #ifndef QUEUE_ARR_H
3 #define QUEUE_ARR_H
4 #include<iostream>
5
6 template <typename T>
7 class queueArr{
8     //data members
9     T *q_array;
10    int q_capacity;
11    int q_size = 0, q_front = 0, q_back = 0;
12 public:
13     queueArr(int newCap){
14         q_capacity = newCap;
15         q_array = new T[q_capacity];
16     }
17
18     bool isEmpty(){
19         return q_size==0;
20     }
21
22     bool isFull(){
23         return q_size==q_capacity;
24     }
25
26     int qSize(){
27         return q_size;
28     }
29
30     void clear(){
31         q_size = 0; q_front = 0; q_back = 0;
32     }
33
34     void Front(){
35         //check if empty
36         if(isEmpty()){
37             std::cout << "Queue is Empty.\n";
38             return;
39         }
40         std::cout << "Front: " << q_array[q_front%q_capacity] << std::endl;
41     }
42
43     void Back(){
44         //check if empty
45         if(isEmpty()){
46             std::cout << "Queue is Empty.\n";
47             return;
48         }
}
```

Queue5.1.cpp Queue5.2.cpp queue.h Queue5.3.cpp queueArr.h X queue_supplementary.h

```
C queueArr.h > queueArr<T> > isEmpty()
2 #ifndef QUEUE_ARR_H
7 class queueArr{
58     queueArr(const queueArr& obj){
62         q_array = obj.q_array;
63         q_front = obj.q_front;
64         q_back = obj.q_back;
65     }
66
67     //Copy Assignment Operator
68     queueArr& operator=(const queueArr& obj){
69         std::cout << "\nAssignment Operator is Called" << std::endl;
70         q_capacity = obj.q_capacity;
71         q_size = obj.q_size;
72         q_array = obj.q_array;
73         q_front = obj.q_front;
74         q_back = obj.q_back;
75         return *this;
76     }
77
78     //Destructor
79     ~queueArr(){
80         std::cout << "\nDestructor is called." << std::endl;
81         Front();
82         Back();
83     }
84 };
85
86 template<typename T>
87 void queueArr<T>::enqueue(T newData){
88     //check if full
89     if(isFull()){
90         std::cout << "Cannot add: Queue is Full.\n";
91         return;
92     }
93     //check if it's empty
94     if(isEmpty()){
95         q_array[q_front%q_capacity] = newData;
96         //Front();
97         //Back();
98         //if front == back in display, correct operation!
99         q_size++;
100        return;
101    }
102    //normal enqueue
103    q_back++;
104    q_array[q_back%q_capacity] = newData;
105    //Front();
106    //Back();
```

```
Queue5.1.cpp Queue5.2.cpp queue.h Queue5.3.cpp queueArr.h X queue_supplementary.h
C queueArr.h > queueArr<T> > isEmpty()
2 #ifndef QUEUE_ARR_H
85
86     template<typename T>
87     void queueArr<T>::enqueue(T newData){
88         //check if full
89         if(isFull()){
90             std::cout << "Cannot add: Queue is Full.\n";
91             return;
92         }
93         //check if it's empty
94         if(isEmpty()){
95             q_array[q_front%q_capacity] = newData;
96             //Front();
97             //Back();
98             //if front == back in display, correct operation!
99             q_size++;
100            return;
101        }
102        //normal enqueue
103        q_back++;
104        q_array[q_back%q_capacity] = newData;
105        //Front();
106        //Back();
107        q_size++;
108    }
109
110    template<typename T>
111    void queueArr<T>::dequeue() {
112        //check if it's empty
113        if(isEmpty()){
114            std::cout << "Cannot dequeue: Queue is empty.\n";
115            return;
116        }
117        //check if there's only 1 element
118        if(q_size==1){
119            std::cout << q_array[q_front%q_capacity] << " was successfully dequeued.\n";
120            clear();
121            //Front();
122            return;
123        }
124        //normal dequeue
125        std::cout << q_array[q_front%q_capacity] << " was successfully dequeued.\n";
126        q_front++;
127        q_size--;
128    }
129 #endif
```

```
Front: A
Back: E

Copy Constructor is called
Front: A
A was successfully dequeued.
Front: B
B was successfully dequeued.
Front: C
C was successfully dequeued.
Front: D
D was successfully dequeued.
Front: E
E was successfully dequeued.

Destructor is called.
Queue is Empty.
Queue is Empty.

Destructor is called.
Front: A
Back: E
```

Table 5-3. Queues using Array Implementation

The implementation is almost similar to linked lists but with the added capacity variable since the activity requires stacks. The core functionalities are handled by the enqueue and dequeue methods, which add and remove elements from the queue, respectively, while managing the front and back pointers to handle circular behavior. Utility functions like isEmpty and isFull allow for checks on the queue's state, while qSize returns the current number of elements. Additionally, Front and Back methods provide access to the front and rear elements without modifying the queue, and a clear function resets the queue to its initial state.

7. Supplementary Activity

```
Queue5.1.cpp Queue5.2.cpp queue.h Queue5.3.cpp Queue_Supplementary_Cabrera.cpp X queue_supplementary.h

Queue_Supplementary_Cabrera.cpp > ...
1 #include <iostream>
2 #include "queue_supplementary.h"
3 #include <string>
4
5 class Job{
6 public:
7     std::string Name;
8     int Pages;
9     int id = 0;
10 };
11
12 template <typename T>
13 class Printer{
14 public:
15     void add_Queue(){
16         Queue <std::string> q;
17         Job j;
18         char choice='Y';
19         while (choice == 'Y' || choice == 'y') {
20             std::cout << "Enter your name: ";
21             std::cin >> j.Name;
22             std::cout << "Number of pages you want to print: ";
23             std::cin >> j.Pages;
24             std::string concan = std::to_string(j.id + 1) + "\t\t\t" + j.Name + "\t\t\t" + std::to_string(j.Pages);
25             q.enqueue(concan);
26             j.id++;
27             std::cout << "Are you done? Do you want to add more? (Y/N)";
28             std::cin >> choice;
29         }
30         std::cout << "ID\t\tName\t\tPages" << std::endl;
31         q.display();
32     }
33 };
34
35
36
37 int main()
38 {
39     Printer<Job> print;
40     print.add_Queue();
41 }
42
```

```
C Queue_Supplementary_Cabrera.cpp      C queue_supplementary.h X
C queue_supplementary.h > Queue<T> > enqueue(T)
1 #ifndef QUEUE_SUPPLEMENTARY_H
2 #define QUEUE_SUPPLEMENTARY_H
3 #include <iostream>
4
5
6
7 template <typename T>
8 class Node{
9     public:
10         T data;
11         Node* next;
12
13         Node(T new_data){
14             data = new_data;
15             next = nullptr;
16         }
17
18
19 };
20 template <typename T>
21 class Queue{
22     private:
23         Node<T> *front;
24         Node<T> *rear;
25     public:
26         //creates an empty queue
27         Queue(){
28             front = rear = nullptr;
29         }
30
31         //isEmpty
32         bool isEmpty(){
33             return front == nullptr;
34         }
35
36         //enqueue
37         void enqueue(T new_data){
38             Node<T> *new_node=new Node<T> (new_data);
39             if (isEmpty()){
40                 front = rear = new_node;
41             }
42             rear->next=new_node;
43             rear=new_node;
44             std::cout << "Successfully added to the queue.\n";
45         }
46
47 }
```

Queue_Supplementary_Cabrera.cpp

queue_supplementary.h X

```
C queue_supplementary.h > Queue<T> > enqueue(T)
1 #ifndef QUEUE_SUPPLEMENTARY_H
21 class Queue{
37     void enqueue(T new_data){
39         if (isEmpty()){
40             front = rear = new_node;
41             return;
42         }
43         rear->next=new_node;
44         rear=new_node;
45         std::cout << "Successfully added to the queue.\n";
46     }
47
48     //dequeue
49     void dequeue(){
50         if (isEmpty()){
51             return;
52         }
53
54         Node<T>* temp=front;
55
56         if (front==nullptr) {
57             rear=nullptr;
58         }
59         else{
60             front=front->next;
61         }
62         delete temp;
63     }
64
65     //getfront
66     void getFront(){
67         if (isEmpty()){
68             std::cout<<"The queue is empty.\n";
69             return;
70         }
71         std::cout<<"Current Front: "<<front->data<<std::endl;
72     }
73
74     //getrear
75     void getRear(){
76         if (isEmpty()){
77             std::cout<<"The queue is empty.\n";
78             return;
79         }
80         std::cout<<"Current Rear: "<<rear->data<<std::endl;
81     }
82
83     //display
```

Queue_Supplementary_Cabrera.cpp

queue_supplementary.h X

```
C queue_supplementary.h > Queue<T> > enqueue(T)
1  #ifndef QUEUE_SUPPLEMENTARY_H
21 class Queue{
66     void getFront(){
67         if (isEmpty()){
71             std::cout<<"Current Front: "<<front->data<<std::endl;
72         }
73
74         //getrear
75         void getRear(){
76             if (isEmpty()){
77                 std::cout<<"The queue is empty.\n";
78                 return;
79             }
80             std::cout<<"Current Rear: "<<rear->data<<std::endl;
81         }
82
83         //display
84         void display(){
85             if (isEmpty()){
86                 std::cout<<"The queue is empty.\n";
87                 return;
88             }
89             Node<T> *temp=front;
90             while (temp!=nullptr){
91                 std::cout<<temp->data;
92                 temp=temp->next;
93             }
94             std::cout<<std::endl;
95         }
96
97         //to deallocate memory
98         ~Queue(){
99             while(!isEmpty()){
100                 dequeue();
101             }
102         }
103
104     };
105
106 #endif
107
```

```
Are you done? Do you want to add more? (Y/N)N
```

ID	Name	Pages
1	Angelo	5
2	Eric	3
3	Ricardo	4

8. Conclusion

The activity has shown the implementation of queues using STL, linked lists, and arrays. Now that I can understand the other structures and using headers better, I can say that this activity has expanded my knowledge on queues while reinforcing the things I learned on the previous data structures.

9. Assessment Rubric