

Activity No. <5>

< QUEUES >

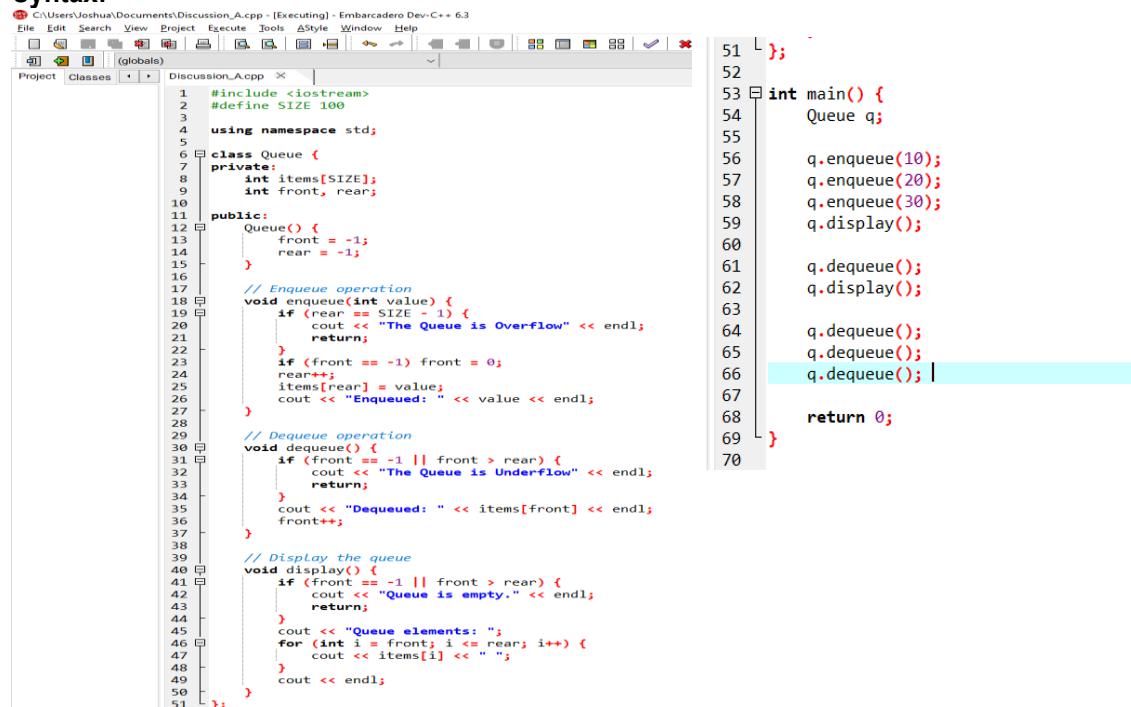
Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 9/11/25
Section: CPE21S4	Date Submitted: 9/11/25
Name(s): Alcantara, Jason P.	Instructor: Engr. Jimlord Quejado

6. Output:

Discussion:

Part: A

Syntax:

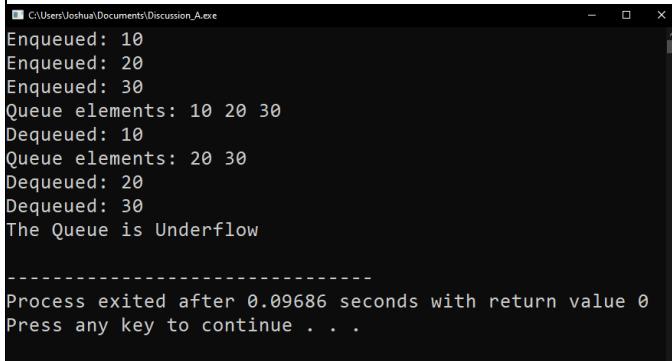


```

1 //include <iostream>
2 #define SIZE 100
3
4 using namespace std;
5
6 class Queue {
7 private:
8     int items[SIZE];
9     int front, rear;
10 public:
11     Queue() {
12         front = -1;
13         rear = -1;
14     }
15
16     // Enqueue operation
17     void enqueue(int value) {
18         if (rear == SIZE - 1) {
19             cout << "The Queue is Overflow" << endl;
20             return;
21         }
22         if (front == -1) front = 0;
23         rear++;
24         items[rear] = value;
25         cout << "Enqueued: " << value << endl;
26     }
27
28     // Dequeue operation
29     void dequeue() {
30         if (front == -1 || front > rear) {
31             cout << "The Queue is Underflow" << endl;
32             return;
33         }
34         cout << "Dequeued: " << items[front] << endl;
35         front++;
36     }
37
38     // Display the queue
39     void display() {
40         if (front == -1 || front > rear) {
41             cout << "Queue is empty." << endl;
42             return;
43         }
44         cout << "Queue elements: ";
45         for (int i = front; i <= rear; i++) {
46             cout << items[i] << " ";
47         }
48         cout << endl;
49     }
50 };
51
52
53 int main() {
54     Queue q;
55
56     q.enqueue(10);
57     q.enqueue(20);
58     q.enqueue(30);
59     q.display();
60
61     q.dequeue();
62     q.display();
63
64     q.dequeue();
65     q.dequeue();
66     q.dequeue();
67
68
69 }
70

```

Output:



```

C:\Users\Joshua\Documents\Discussion_A.exe
Enqueued: 10
Enqueued: 20
Enqueued: 30
Queue elements: 10 20 30
Dequeued: 10
Queue elements: 20 30
Dequeued: 20
Dequeued: 30
The Queue is Underflow

-----
Process exited after 0.09686 seconds with return value 0
Press any key to continue . . .

```

Part B:

Main.cpp:

Output:

```
Discussion_B.cpp Queue.h
1 #include <iostream>
2 #include <string>
3 #include "Queue.h"
4
5
6 int main() {
7     Queue<std::string> CPE2154;
8
9     CPE2154.enqueue("Spiderman");
10    CPE2154.enqueue("Batman");
11    CPE2154.enqueue("Hulk");
12    CPE2154.enqueue("Thanos");
13    CPE2154.enqueue("Naruto");
14    CPE2154.enqueue("Hikaru");
15
16    CPE2154.getRear();
17    CPE2154.getFront();
18
19    CPE2154.dequeue();
20    CPE2154.getFront();
21
22    CPE2154.display();
23
24
25    return 0;
26 }
```

```
C:\Users\Joshua\Documents\Discussion_B.exe
An empty queue has been created.
Enqueued: Spiderman
Enqueued: Batman
Enqueued: Hulk
Enqueued: Thanos
Enqueued: Naruto
Enqueued: Hikaru
Current Rear: Hikaru
Current Front: Spiderman
Current Front: Batman
Queue contents: Batman Hulk Thanos Naruto Hikaru
-----
Process exited after 0.09919 seconds with return value 0
Press any key to continue . . .
```

Header File:

```
Discussion_B.cpp Queue.h
1 #ifndef QUEUE_H
2 #define QUEUE_H
3
4 #include <iostream>
5
6 template <typename T>
7 class Queue {
8 private:
9     struct Node {
10         T data;
11         Node* next;
12     };
13
14     Node* front;
15     Node* rear;
16
17 public:
18     Queue() : front(nullptr), rear(nullptr) {
19         std::cout << "An empty queue has been created.\n";
20     }
21
22     // Check if queue is empty
23     bool isEmpty() const {
24         return front == nullptr;
25     }
26
27     // Enqueue
28     void enqueue(const T& item) {
29         Node* newNode = new Node{item, nullptr};
30
31         if (isEmpty()) {
32             front = rear = newNode;
33         } else {
34             rear->next = newNode;
35             rear = newNode;
36         }
37
38         std::cout << "Enqueued: " << item << std::endl;
39     }
40
41     // Dequeue
42     void dequeue() {
43         if (isEmpty()) {
44             std::cout << "The queue is empty.\n";
45             return;
46         }
47
48         Node* temp = front;
49         front = front->next;
50
51         delete temp;
52
53         if (front == nullptr)
54             rear = nullptr;
55     }
56 }
```

```
Discussion_B.cpp Queue.h
47
48     Node* temp = front;
49     front = front->next;
50
51     delete temp;
52
53     if (front == nullptr)
54         rear = nullptr;
55
56
57     // Get front
58     void getFront() const {
59         if (isEmpty())
60             std::cout << "The queue is empty.\n";
61         else
62             std::cout << "Current Front: " << front->data << std::endl;
63     }
64
65     // Get rear
66     void getRear() const {
67         if (isEmpty())
68             std::cout << "The queue is empty.\n";
69         else
70             std::cout << "Current Rear: " << rear->data << std::endl;
71     }
72
73     // Display
74     void display() const {
75         if (isEmpty())
76             std::cout << "The queue is empty.\n";
77         else
78             std::cout << "Queue contents: ";
79             Node* temp = front;
80             while (temp != nullptr) {
81                 std::cout << temp->data << " ";
82                 temp = temp->next;
83             }
84             std::cout << std::endl;
85     }
86
87
88     // Destructor
89     ~Queue() {
90         while (!isEmpty())
91             dequeue();
92     }
93
94
95
96
97
98
99
100 #endif
```

Part C:

Main.cpp:

```
Discussion_Cpp X QUEUE_H.h X
1 #include <iostream>
2 #include "QUEUE_H.h"
3
4 int main() {
5     Queue<int> x; // CreateQueue()
6
7     x.enqueue(5);
8     x.enqueue(3);
9     x.enqueue(2);
10
11    x.dequeue(); // Removes 5
12
13    x.enqueue(7);
14
15    // Dequeue the variable a
16    if (!x.isEmpty()) {
17        int a = x.getFrontValue();
18        x.dequeue();
19        std::cout << "Dequeued a: " << a << std::endl;
20    }
21
22    // Dequeue the variable b
23    if (!x.isEmpty()) {
24        int b = x.getFrontValue();
25        x.dequeue();
26        std::cout << "Dequeued b: " << b << std::endl;
27    }
28
29    x.display();
30
31    return 0;
32 }
```

Header File:

```
Discussion_Cpp X QUEUE_H.h X
1 #ifndef QUEUE_H
2 #define QUEUE_H
3
4 #include <iostream>
5 #include <stdexcept>
6
7 template <typename T>
8 class Queue {
9 private:
10     struct Node {
11         T data;
12         Node* next;
13     };
14
15     Node* front;
16     Node* rear;
17
18 public:
19     // Constructor
20     Queue() : front(nullptr), rear(nullptr) {
21         std::cout << "An empty queue has been created.\n";
22     }
23
24     // Check if empty
25     bool isEmpty() const {
26         return front == nullptr;
27     }
28
29     // Enqueue
30     void enqueue(const T& item) {
31         Node* newNode = new Node{item, nullptr};
32         if (isEmpty()) {
33             front = rear = newNode;
34         } else {
35             rear->next = newNode;
36             rear = newNode;
37         }
38         std::cout << "Enqueued: " << item << std::endl;
39     }
40
41     // Dequeue
42     void dequeue() {
43         if (isEmpty()) {
44             std::cout << "Queue is empty.\n";
45             return;
46         }
47         Node* temp = front;
48         front = front->next;
49         delete temp;
50
51         if (front == nullptr)
52             rear = nullptr;
53     }
54
55     // Get front value (without removing)
56     T getFrontValue() const {
57         if (isEmpty()) {
58             throw std::runtime_error("Queue is empty.");
59         }
60         return front->data;
61     }
62
63     // Display queue
64     void display() const {
65         if (isEmpty()) {
66             std::cout << "[Queue is empty]" << std::endl;
67             return;
68         }
69
70         Node* temp = front;
71         std::cout << "Queue: ";
72         while (temp != nullptr) {
73             std::cout << temp->data << " ";
74             temp = temp->next;
75         }
76         std::cout << "(Front > Back)" << std::endl;
77     }
78
79     // Destructor
80     ~Queue() {
81         while (!isEmpty()) {
82             dequeue();
83         }
84     }
85 }
86
87 #endif // QUEUE_H
88
```

Output:

```
C:\Users\Joshua\Documents\Discussion_Cpp
An empty queue has been created.
Enqueued: 5
Enqueued: 3
Enqueued: 2
Enqueued: 7
Dequeued a: 3
Dequeued b: 2
Queue: 7 (Front ? Back)

-----
Process exited after 0.08925 seconds with return value 0
Press any key to continue . . .
```

Procedure A:

```
Discussion_Cpp X | QUEUE_H.h X | Procedure_A.cpp X |
1 #include <iostream>
2 #include <queue>
3
4 int main() {
5     std::queue<int> x;
6     int a, b;
7
8     x.push(5);
9     x.push(3);
10    x.push(2);
11    x.pop();
12
13    x.push(7);
14
15    if (!x.empty()) {
16        a = x.front();
17        x.pop();
18        std::cout << "Dequeued a = " << a << std::endl;
19    }
20
21    if (!x.empty()) {
22        b = x.front();
23        x.pop();
24        std::cout << "Dequeued b = " << b << std::endl;
25    }
26
27    // Display remaining queue
28    std::cout << "Final queue: ";
29    std::queue<int> temp = x;
30    while (!temp.empty()) {
31        std::cout << temp.front() << " ";
32        temp.pop();
33    }
34    std::cout << "(Front → Back)" << std::endl;
35
36    return 0;
37 }
```

Output:

```
C:\Users\Joshua\Documents\Procedure_A.exe
Dequeued a = 3
Dequeued b = 2
Final queue: 7 (Front ? Back)
-----
Process exited after 0.1039 seconds with return value 0
Press any key to continue . . .
```

ILO B:

5.1

Syntax:

```
STL.cpp X |
1 #include <iostream>
2 #include <queue>
3 #include <string>
4
5 int main() {
6     std::queue<std::string> q;
7     std::string students[] = {"Spiderman", "Batman", "Hulk", "Superman"};
8
9     // Insert students
10    for (auto& name : students) {
11        std::cout << "Enqueue: " << name << std::endl;
12        q.push(name);
13    }
14
15    // Display
16    std::cout << "Queue after enqueues: ";
17    std::queue<std::string> temp = q;
18    while (!temp.empty()) {
19        std::cout << temp.front() << " ";
20        temp.pop();
21    }
22    std::cout << "\n";
23
24    // Dequeue the students one by one
25    while (!q.empty()) {
26        std::cout << "Dequeue: " << q.front() << std::endl;
27        q.pop();
28    }
29
30    return 0;
31 }
```

Output:

```
C:\Users\Joshua\Documents\STL.exe
Enqueue: Spiderman
Enqueue: Batman
Enqueue: Hulk
Enqueue: Superman
Queue after enqueues: Spiderman Batman Hulk Superman
Dequeue: Spiderman
Dequeue: Batman
Dequeue: Hulk
Dequeue: Superman
-----
Process exited after 0.08972 seconds with return value 0
Press any key to continue . . .
```

5.2

Syntax:

```
1 #include <iostream>
2 using namespace std;
3
4 class CircularQueue {
5 private:
6     int* q_array;
7     int capacity;
8     int front;
9     int back;
10    int size;
11
12 public:
13     CircularQueue(int cap) {
14         capacity = cap;
15         q_array = new int[capacity];
16         front = 0;
17         back = capacity - 1;
18         size = 0;
19     }
20
21     ~CircularQueue() {
22         delete[] q_array;
23     }
24
25     bool isEmpty() {
26         return size == 0;
27     }
28
29     bool isFull() {
30         return size == capacity;
31     }
32
33     void enqueue(int value) {
34         if (isFull()) {
35             cout << "Queue is full! " << value << endl;
36             return;
37         }
38         back = (back + 1) % capacity;
39         q_array[back] = value;
40         size++;
41         cout << "Enqueued: " << value << endl;
42     }
43
44     void dequeue() {
45         if (isEmpty()) {
46             cout << "Queue is empty! " << endl;
47             return;
48         }
49         cout << "Dequeued: " << q_array[front] << endl;
50         front = (front + 1) % capacity;
51         size--;
52     }
53 }
```

```
54     void display() {
55         if (isEmpty()) {
56             cout << "Queue is empty.\n";
57             return;
58         }
59         cout << "Queue contents: ";
60         for (int i = 0; i < size; i++) {
61             cout << q_array[(front + i) % capacity] << " ";
62         }
63         cout << endl;
64     }
65 }
66
67 int main() {
68     CircularQueue q(5);
69
70     q.enqueue(10);
71     q.enqueue(20);
72     q.enqueue(30);
73     q.enqueue(40);
74     q.display();
75
76     q.dequeue();
77     q.dequeue();
78     q.display();
79
80     q.enqueue(50);
81     q.enqueue(60);
82     q.enqueue(70);
83     q.display();
84
85     return 0;
86 }
87 }
```

Output:

```
C:\Users\Joshua\Documents\ILO_5.2.exe
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Queue contents: 10 20 30 40
Dequeued: 10
Dequeued: 20
Queue contents: 30 40
Enqueued: 50
Enqueued: 60
Enqueued: 70
Queue contents: 30 40 50 60 70

-----
Process exited after 0.1101 seconds with return value 0
Press any key to continue . . .
```

5.3

Syntax:

```
1 #include <iostream>
2 using namespace std;
3
4 class SimpleQueue {
5 private:
6     int* arr;
7     int capacity;
8     int front;
9     int back;
10    int size;
11
12 public:
13     SimpleQueue(int cap) {
14         capacity = cap;
15         arr = new int[capacity];
16         front = 0;
17         back = -1;
18         size = 0;
19     }
20
21     ~SimpleQueue() {
22         delete[] arr;
23     }
24
25     bool isEmpty() {
26         return size == 0;
27     }
28
29     bool isFull() {
30         return size == capacity;
31     }
32
33     void enqueue(int value) {
34         if (isFull()) {
35             cout << "Queue is full\n";
36             return;
37         }
38         back = (back + 1) % capacity;
39         arr[back] = value;
40         size++;
41         cout << "Enqueued: " << value << endl;
42     }
43
44     void dequeue() {
45         if (isEmpty()) {
46             cout << "Queue is empty\n";
47             return;
48         }
49         cout << "Dequeued: " << arr[front] << endl;
50         front = (front + 1) % capacity;
51         size--;
52     }
53 }
```

Output:

```
C:\Users\Joshua\Documents\ILO_5.3.exe
Enqueued: 5
Enqueued: 10
Enqueued: 15
Queue: 5 10 15
Dequeued: 5
Queue: 10 15
Enqueued: 20
Queue: 10 15 20

-----
Process exited after 0.209 seconds with return value 0
Press any key to continue . . .
```

```
54     void display() {
55         if (isEmpty()) {
56             cout << "Queue is empty\n";
57             return;
58         }
59         cout << "Queue: ";
60         for (int i = 0; i < size; i++) {
61             cout << arr[(front + i) % capacity] << " ";
62         }
63         cout << endl;
64     }
65 }
66
67 int main() {
68     SimpleQueue q(5);
69
70     q.enqueue(5);
71     q.enqueue(10);
72     q.enqueue(15);
73     q.display();
74
75     q.dequeue();
76     q.display();
77
78     q.enqueue(20);
79     q.display();
80
81     return 0;
82 }
```

7. Supplementary Activity:

Syntax:

```
1 #include <iostream>
2 #include <queue>
3 #include <string>
4
5 int main() {
6     std::queue<std::string> printerQueue;
7
8     // Adding print jobs
9     printerQueue.push("Goku job");
10    printerQueue.push("Naruto job");
11    printerQueue.push("Sponge Bob job");
12
13    std::cout << "Printer queue started...\n";
14
15    // Process the queue
16    while (!printerQueue.empty()) {
17        std::cout << "Printing: " << printerQueue.front() << std::endl;
18        printerQueue.pop(); // Remove the job after printing
19    }
20
21    std::cout << "All print jobs done.\n";
22
23    return 0;
24 }
```

Output:

```
C:\Users\Joshua\Documents\Supplementary.exe
Printer queue started...
Printing: Goku job
Printing: Naruto job
Printing: Sponge Bob job
All print jobs done.

-----
Process exited after 0.08991 seconds with return value 0
Press any key to continue . . .
```

8. Conclusion:

In HOA activity I learned how to make a queue by using an array in C++. The queue followed the “FIFO” rule. The rule says that elements are added at the back and taken out at the front. I also learned important things in practice like enqueue, dequeue and is the queue empty or full. This exercise helped me a lot in understanding how queues work and how to take care of data efficiently using array.

9. Assessment Rubric