

Hands-on Activity 5.1

Queues

Course Code: CPE010	Program: Computer Engineering
Course Title: Data Structures and Algorithms	Date Performed: 9/10/25
Section: CPE21S4	Date Submitted: 9/11/25
Name(s): Avila, Vince Gabriel V.	Instructor: Engr. Jimlord Quejado

1. Output

ILO1

```
1 #include <iostream>
2 #include <queue>
3 #include <string>
4
5 int main() {
6     std::queue<std::string> heroQueue;
7     std::string heroList[] = {"Pau", "Alexcy", "Rose Ann", "Nievelyn"};
8
9     // Enqueue heroes
10    for (auto& hero : heroList) {
11        std::cout << "Enqueue: " << hero << std::endl;
12        heroQueue.push(hero);
13    }
14
15    // Display queue content
16    std::cout << "Queue after enqueues: ";
17    std::queue<std::string> copyQueue = heroQueue;
18    while (!copyQueue.empty()) {
19        std::cout << copyQueue.front() << " ";
20        copyQueue.pop();
21    }
22    std::cout << "\n";
23
24    // Dequeue heroes
25    while (!heroQueue.empty()) {
26        std::cout << "Dequeue: " << heroQueue.front() << std::endl;
27        heroQueue.pop();
28    }
29
30    return 0;
31 }
```

C:\Users\Olaco\Downloads\avila 5.1.exe
Enqueue: Pau
Enqueue: Alexcy
Enqueue: Rose Ann
Enqueue: Nievelyn
Queue after enqueues: Pau Alexcy Rose Ann Nievelyn
Dequeue: Pau
Dequeue: Alexcy
Dequeue: Rose Ann
Dequeue: Nievelyn

Process exited after 0.2727 seconds with return value 0
Press any key to continue . . .

Analysis:

This program adds hero names to a queue, displays them in order, and then removes and prints each hero following the first in, first out rule.

ILO2

The screenshot shows a C++ development environment with multiple tabs open. The active tab is 'avila 5.1.cpp'. The code implements a circular queue with methods for enqueueing, dequeuing, and displaying the contents. The main function creates a queue, enqueues four integers (10, 20, 30, 40), and then displays the queue's contents.

```
41     cout << "Enqueued: " << data << endl;
42 }
43
44 void dequeue() {
45     if (isEmpty()) {
46         cout << "Queue is empty! " << endl;
47         return;
48     }
49     cout << "Dequeued: " << buffer[head] << endl;
50     head = (head + 1) % maxSize;
51     itemCount--;
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 < itemCount; i++) {
61         cout << buffer[(head + i) % maxSize] << " ";
62     }
63     cout << endl;
64 }
65 }
66
67 int main() {
68     LoopQueue queue;
69
70     queue.enqueue(10);
71     queue.enqueue(20);
72     queue.enqueue(30);
73     queue.enqueue(40);
74     queue.display();
75 }
```

The right side of the interface shows the execution output of the program. It shows the queue being enqueued with values 10, 20, 30, and 40. Then it dequeues 10 and 20, and finally displays the queue's contents again after the last enqueue operation.

```
C:\Users\Olaco\Downloads\avila 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 0.2543 seconds with return value 0
Press any key to continue . . .
```

Below the code editor, there is a 'Compile Log' tab which shows the compilation results:

```
Compiling results...
-----
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\Olaco\Downloads\avila 5.2.exe
```

Analysis:

This program implements a circular queue that adds, removes, and displays integers while efficiently using a fixed size array in a looped manner.

ILO3

```
C:\Users\Olaco\Downloads\avila 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 0.2294 seconds with return value 0
Press any key to continue . . .
```

```
lobals) avila 5.1.cpp X avila 5.2.cpp X avila 5.3.cpp X supplementary avila.cpp X
37     }
38     end = (end + 1) % maxLength;
39     elements[end] = item;
40     count++;
41     cout << "Enqueued: " << item << endl;
42 }
43
44 void dequeue() {
45     if (isEmpty()) {
46         cout << "Queue is empty\n";
47         return;
48     }
49     cout << "Dequeued: " << elements[start] << endl;
50     start = (start + 1) % maxLength;
51     count--;
52 }
53
54 void display() {
55     if (isEmpty()) {
56         cout << "Queue is empty\n";
57         return;
58     }
59     cout << "Queue: ";
60     for (int i = 0; i < count; i++) {
61         cout << elements[(start + i) % maxLength] << " ";
62     }
63     cout << endl;
64 }
65 }
66
67 int main() {
68     BasicQueue queue(5);
69     queue.enqueue(5);
70 }
```

Analysis:

This program implements a circular queue that adds, removes, and displays integer elements using a fixed size array with wrap around indexing.

2. Supplementary Activity

The screenshot shows a C++ development environment with multiple tabs open. The active tab is 'avila 5.1.cpp'. The code implements a queue to manage print jobs. It defines a 'Job' class with attributes 'id', 'user', and 'pages'. A 'Printer' class has methods to add jobs to a queue and process them. The 'processJobs' method prints each job's details and removes it from the queue. The 'main' function creates six jobs (Job 1 to Job 6) and processes them. The output window shows the program's execution and the results of the processing.

```
26     void addJob(Job j) {
27         jobList.push(j);
28         cout << "Job " << j.id << " submitted by " << j.user
29         << " (" << j.pages << " pages)" << endl;
30     }
31
32     void processJobs() {
33         while (!jobList.empty()) {
34             Job cur = jobList.front();
35             cout << "Printing Job " << cur.id << " from "
36             << cur.user << " with " << cur.pages << " pages..." <<
37             jobList.pop();
38         }
39         cout << "All print jobs are done." << endl;
40     }
41 }
42
43 int main() {
44     Printer printer;
45
46
47     printer.addJob(Job(1, "Alex", 8));
48     printer.addJob(Job(2, "Maya", 12));
49     printer.addJob(Job(3, "Jordan", 5));
50     printer.addJob(Job(4, "Ella", 9));
51     printer.addJob(Job(5, "Noah", 6));
52     printer.addJob(Job(6, "Liam", 15));
53
54     cout << "\n--- Processing Queue ---\n";
55     printer.processJobs();
56
57
58     return 0;
59 }
```

```
C:\Users\Olaco\Downloads\supplementary avila.exe
Job 1 submitted by Alex (8 pages)
Job 2 submitted by Maya (12 pages)
Job 3 submitted by Jordan (5 pages)
Job 4 submitted by Ella (9 pages)
Job 5 submitted by Noah (6 pages)
Job 6 submitted by Liam (15 pages)

--- Processing Queue ---
Printing Job 1 from Alex with 8 pages...
Printing Job 2 from Maya with 12 pages...
Printing Job 3 from Jordan with 5 pages...
Printing Job 4 from Ella with 9 pages...
Printing Job 5 from Noah with 6 pages...
Printing Job 6 from Liam with 15 pages...
All print jobs are done.

-----
Process exited after 0.2389 seconds with return value 0
Press any key to continue . . .
```

Analysis:

I made a Job class that stores an ID, the user's name, and how many pages to print. Then, I created a Printer class with two main jobs adding print tasks and processing them. I used a queue because it prints jobs in the order they come, just like a real printer.

3. Conclusion

For my conclusion. I learned that queues are implemented with structures that store elements in a specified order and follow the rule of first come, first serve. These procedures were good for me to learn how to insert and delete things, gradually performing extra coding I improved my concept. I believed I performed satisfactorily but was convinced that with more practice complications can be avoided and codes simplified.

4. Assessment Rubric