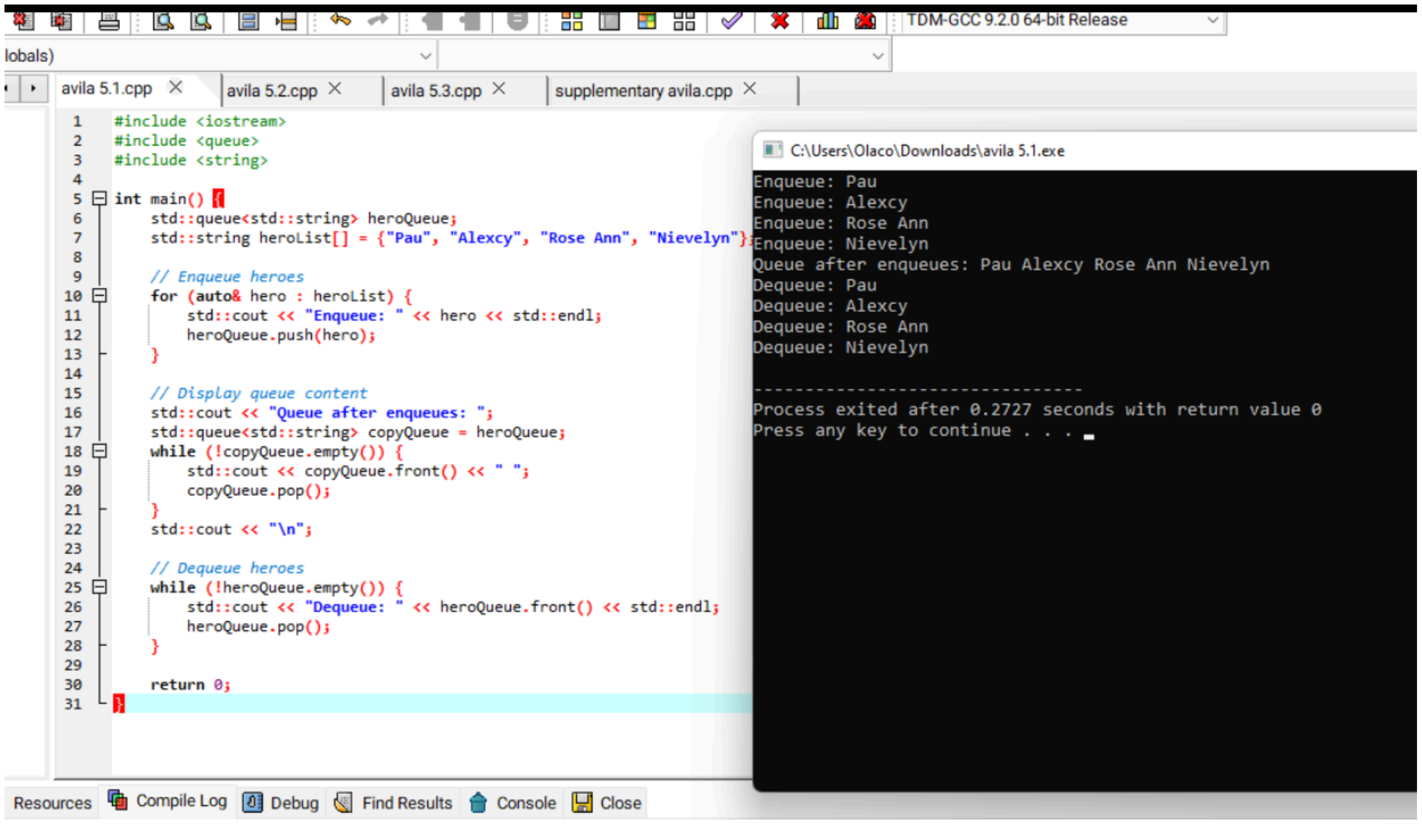
Hands-on Activity 5.1	
Queues	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 9/10/25
Section: CPE21S4	**Date Submitted:** 9/11/25
Name(s): Avila, Vince Gabriel V.	**Instructor:** Engr. Jimlord Quejado

1. Output

ILO1



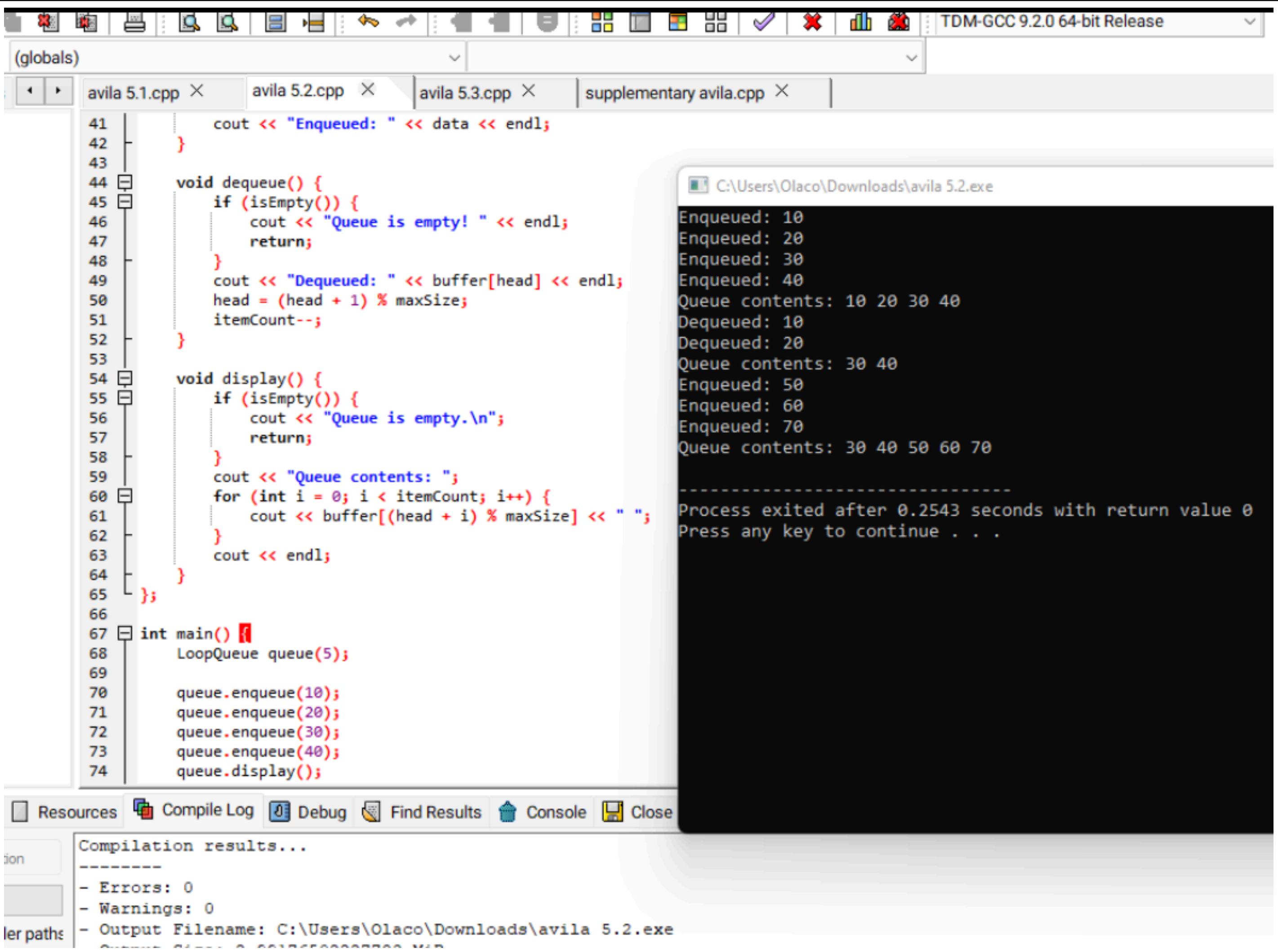
```cpp
#include <iostream>
#include <queue>
#include <string>

int main() {
    std::queue<std::string> heroQueue;
    std::string heroList[] = {"Pau", "Alexcy", "Rose Ann", "Nievelyn"};

    // Enqueue heroes
    for (auto& hero : heroList) {
        std::cout << "Enqueue: " << hero << std::endl;
        heroQueue.push(hero);
    }

    // Display queue content
    std::cout << "Queue after enqueues: ";
    std::queue<std::string> copyQueue = heroQueue;
    while (!copyQueue.empty()) {
        std::cout << copyQueue.front() << " ";
        copyQueue.pop();
    }
    std::cout << "\n";

    // Dequeue heroes
    while (!heroQueue.empty()) {
        std::cout << "Dequeue: " << heroQueue.front() << std::endl;
        heroQueue.pop();
    }

    return 0;
}
```

```
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



```
        cout << "Enqueued: " << data << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty! " << endl;
            return;
        }
        cout << "Dequeued: " << buffer[head] << endl;
        head = (head + 1) % maxSize;
        itemCount--;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty.\n";
            return;
        }
        cout << "Queue contents: ";
        for (int i = 0; i < itemCount; i++) {
            cout << buffer[(head + i) % maxSize] << " ";
        }
        cout << endl;
    }
};

int main() {
    LoopQueue queue(5);

    queue.enqueue(10);
    queue.enqueue(20);
    queue.enqueue(30);
    queue.enqueue(40);
    queue.display();
```

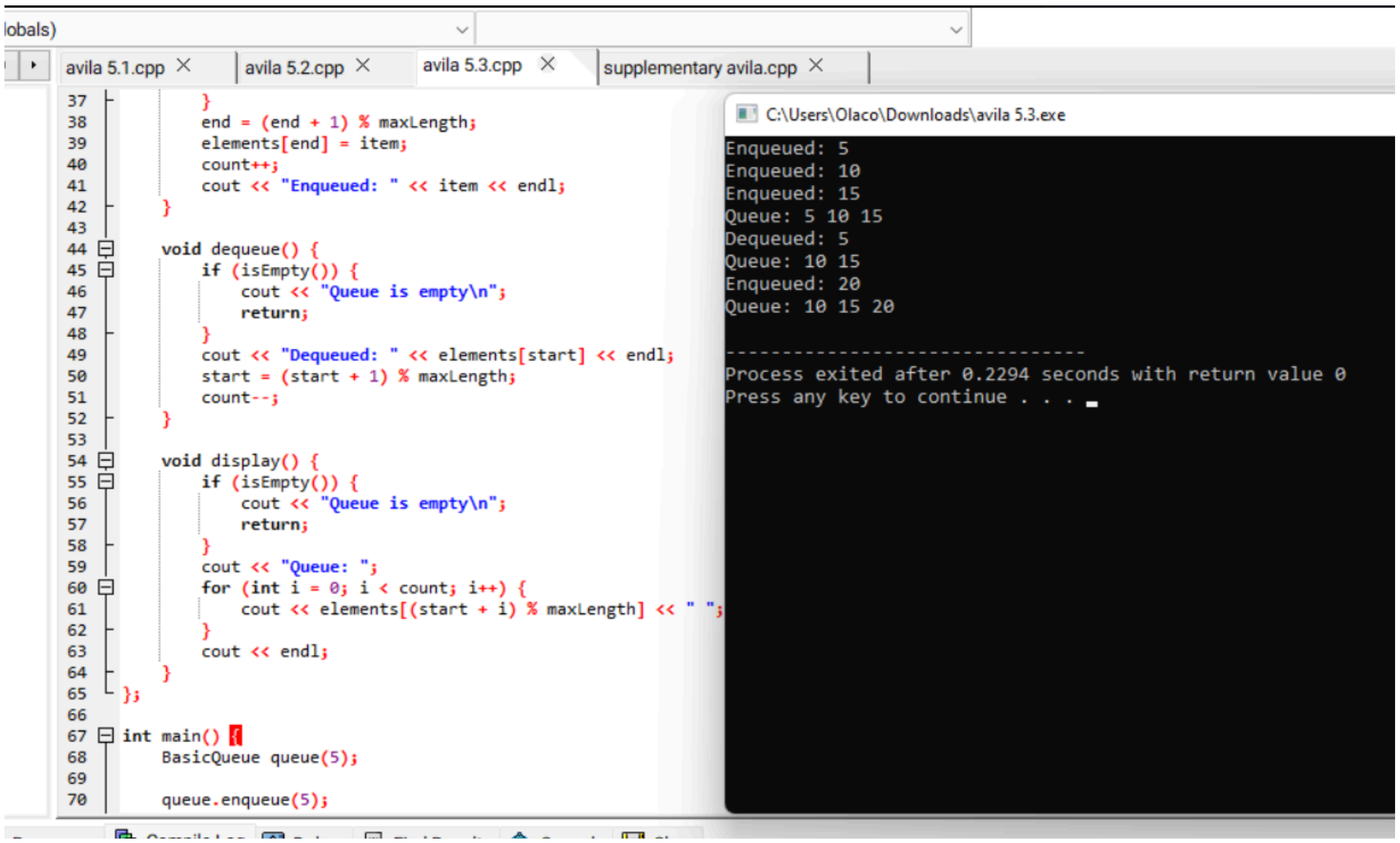
```
Enqueued: 10
Enqueued: 20
Enqueued: 30
Enqueued: 40
Queue contents: 10 20 30 40
Dequeued: 10
Dequeued: 20
Queue contents: 30 40
Enqueued: 50
Enqueued: 60
Enqueued: 70
Queue contents: 30 40 50 60 70

--------------------------------
Process exited after 0.2543 seconds with return value 0
Press any key to continue . . .
```

```
Compilation 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


lobals)

avila 5.1.cpp ✕ | avila 5.2.cpp ✕ | avila 5.3.cpp ✕ | supplementary avila.cpp ✕

```cpp
        }
        end = (end + 1) % maxLength;
        elements[end] = item;
        count++;
        cout << "Enqueued: " << item << endl;
    }

    void dequeue() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        cout << "Dequeued: " << elements[start] << endl;
        start = (start + 1) % maxLength;
        count--;
    }

    void display() {
        if (isEmpty()) {
            cout << "Queue is empty\n";
            return;
        }
        cout << "Queue: ";
        for (int i = 0; i < count; i++) {
            cout << elements[(start + i) % maxLength] << " ";
        }
        cout << endl;
    }
};

int main() {
    BasicQueue queue(5);

    queue.enqueue(5);
```

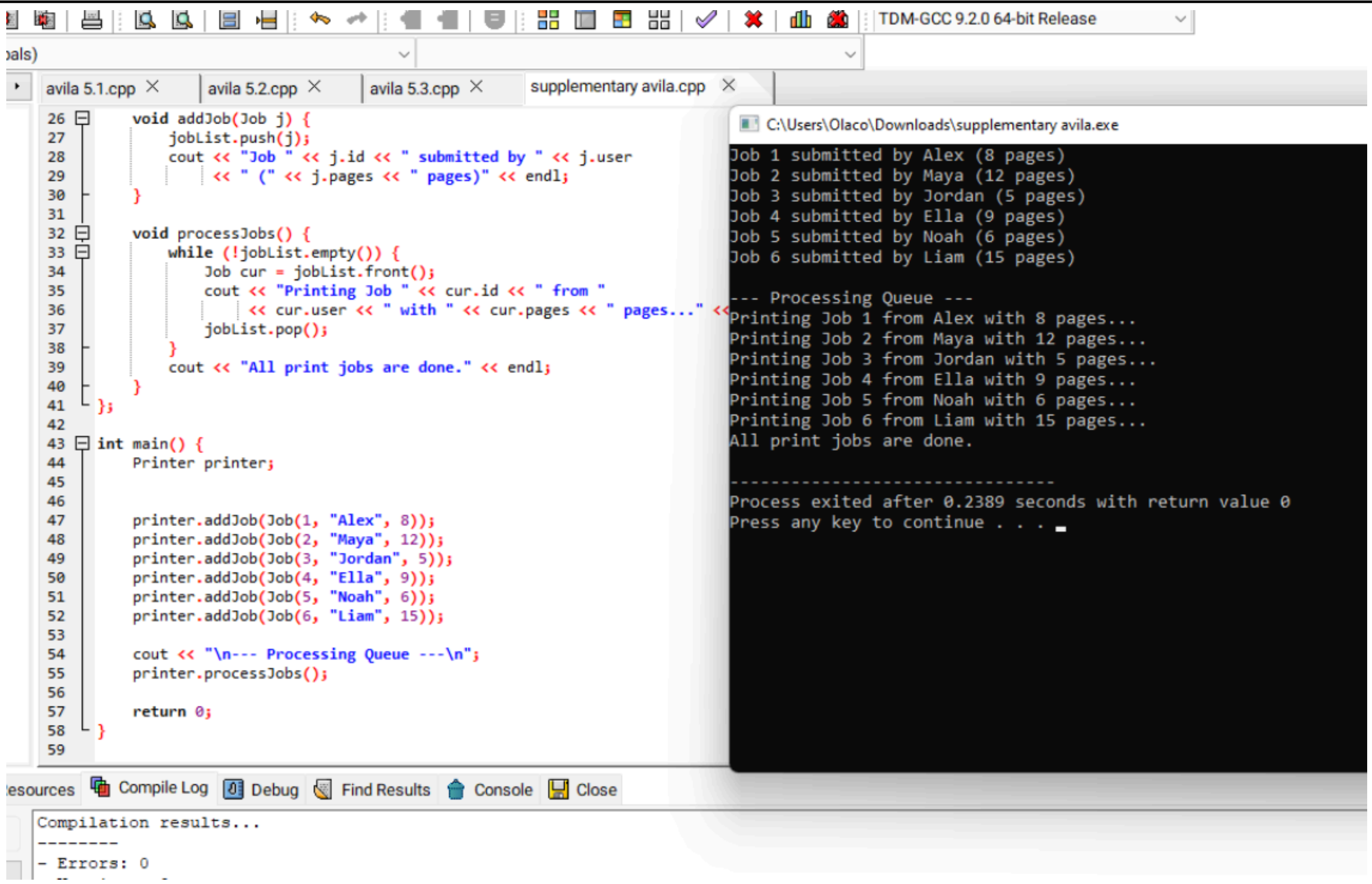
C:\Users\Olaco\Downloads\avila 5.3.exe

```
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 . . .
```

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



```cpp
    void addJob(Job j) {
        jobList.push(j);
        cout << "Job " << j.id << " submitted by " << j.user
             << " (" << j.pages << " pages)" << endl;
    }

    void processJobs() {
        while (!jobList.empty()) {
            Job cur = jobList.front();
            cout << "Printing Job " << cur.id << " from "
                 << cur.user << " with " << cur.pages << " pages..." <<
            jobList.pop();
        }
        cout << "All print jobs are done." << endl;
    }
};

int main() {
    Printer printer;


    printer.addJob(Job(1, "Alex", 8));
    printer.addJob(Job(2, "Maya", 12));
    printer.addJob(Job(3, "Jordan", 5));
    printer.addJob(Job(4, "Ella", 9));
    printer.addJob(Job(5, "Noah", 6));
    printer.addJob(Job(6, "Liam", 15));

    cout << "\n--- Processing Queue ---\n";
    printer.processJobs();

    return 0;
}
```

```
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 . . .
```

```
Compilation results...
--------
- Errors: 0
```

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