Homework 1

CS51510-001, Fall 2025
Purdue University Northwest
09/05/2025

Name	Email
Raaj Atulkumar Patel	**Pate2682@pnw.edu**

Table of Contents

Table of Figures

ABSTRACT

This homework focused on practicing how to create a virtual machine, install Ubuntu 25.04 Desktop, and set up coding environments for Python, Java, and C. After finishing the setup, I practiced running basic **stack and queue programs** in all three languages to check if everything was working properly.

The main goal of the task was to show that I could install the required software, write the codes, and run them successfully inside the virtual machine. I added installation notes, example programs, and screenshots to show the results.

By completing this assignment, I gained hands-on experience with using a virtual machine, installing an operating system, and running programs in different languages. It also helped me understand stack and queue operations better in Python, Java, and C.

1. INSTALLING AN VIRTUAL MACHINE IN YOUR LAPTOP

Step 1: Download VirtualBox

- went to the official Oracle VirtualBox website.
- downloaded the version suitable for macOS / Intel hosts.

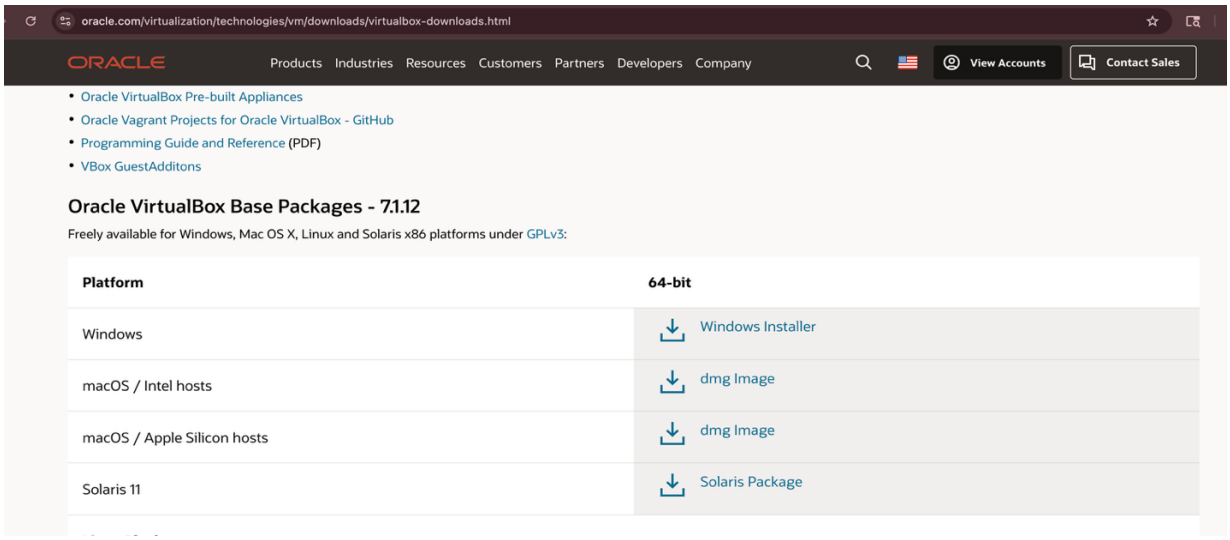


Figure 1 - Downloading the VirtualBox installer package from the official Oracle website

Step 2: Install VirtualBox

- Opened the downloaded installed dmg Image.
- Double clicked on “VirtualBox.pkg”.

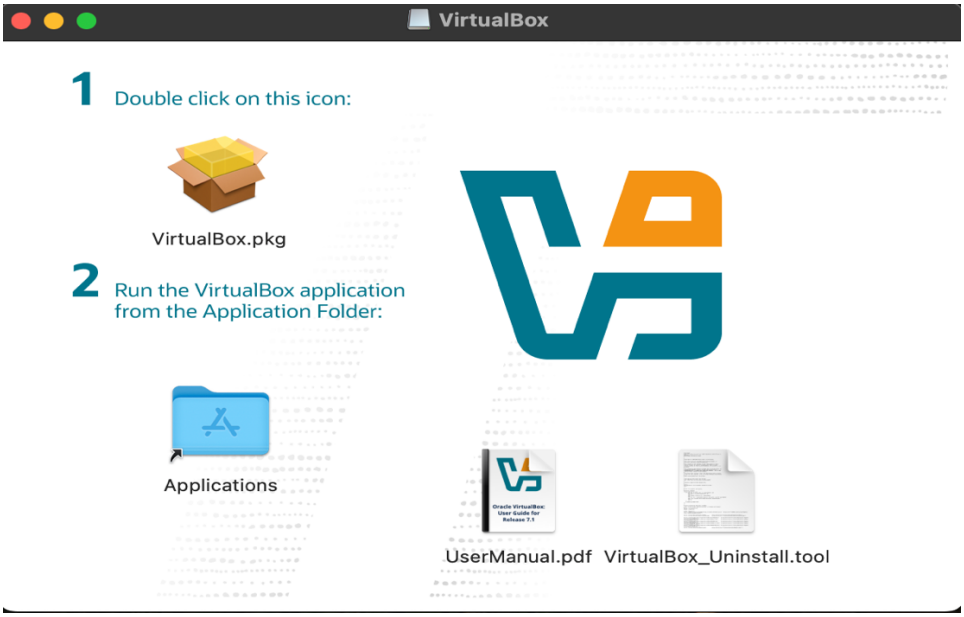


Figure 2 - Opening the downloaded file

- Complete install process step by step and VirtualBox successfully installed .

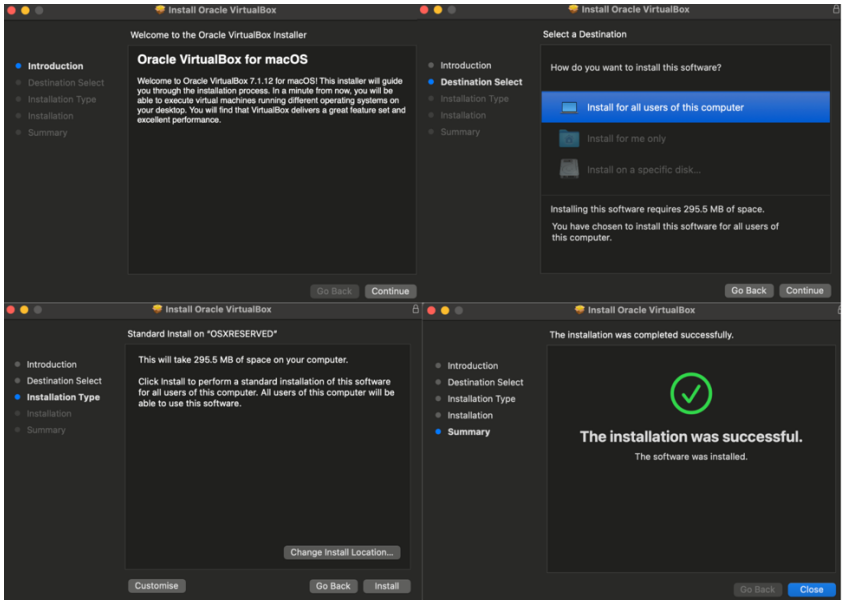


Figure 3 - Step-by-step installation wizard for VirtualBox on macOS installing the Ubuntu 25.04 Desktop

2. INSTALLING THE UBUNTU 25.04 DESKTOP

Step 1: Download the ubuntu 25.04 iso

- Go to the official Ubuntu website.
- I selected Ubuntu 25.04 Desktop (64-bit) and downloaded the ISO file called "ubuntu-25.04-desktop-amd64.iso".

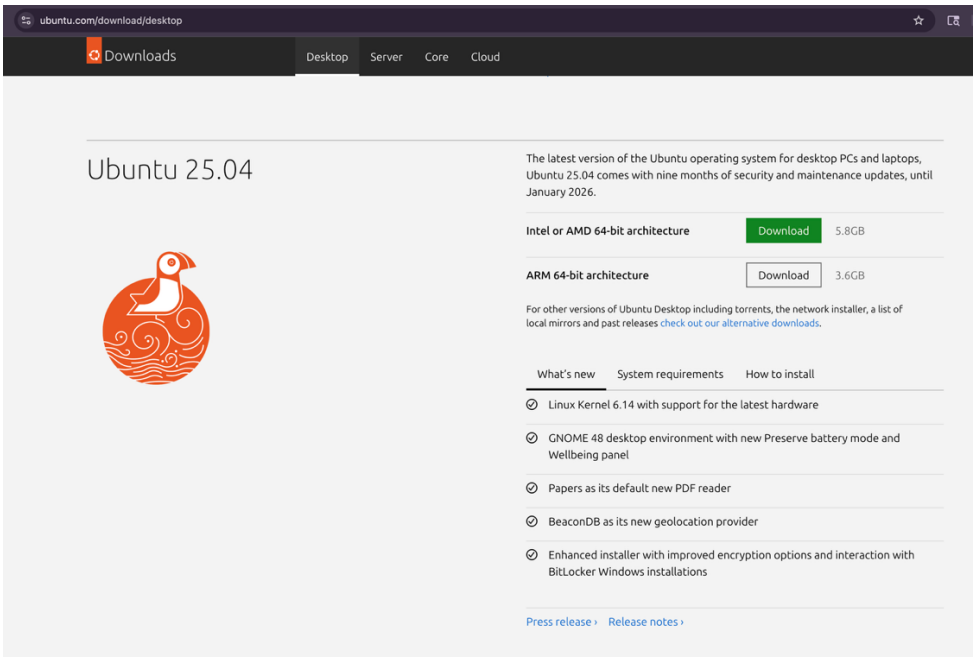


Figure 4 - The official Ubuntu downloads page showing the option to download Ubuntu 25.04 Desktop

- Successfully Downloaded of 5.8 GB iso file.

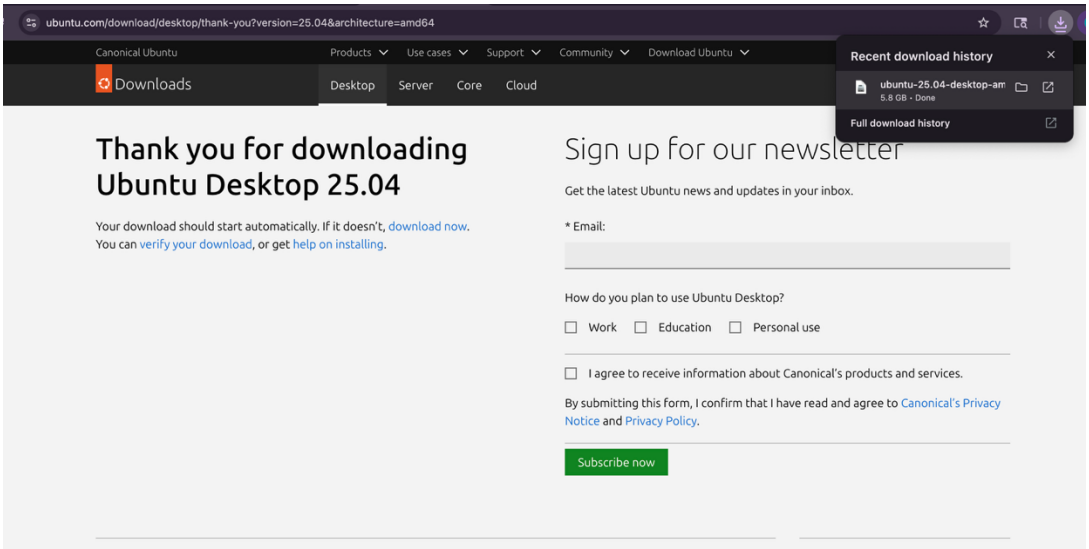


Figure 5 - Confirmation page after starting the Ubuntu 25.04 download

Step 2: Start the VM and load 25.04 version Ubuntu iso file

- Open VirtualBox and select new.

Figure 6 - Open virtual box

- Give name to virtual machine to "ubuntu v25.04" and choose folder to create and save and download the ISO file named "`ubuntu-25.04-desktop-amd64 .iso`".

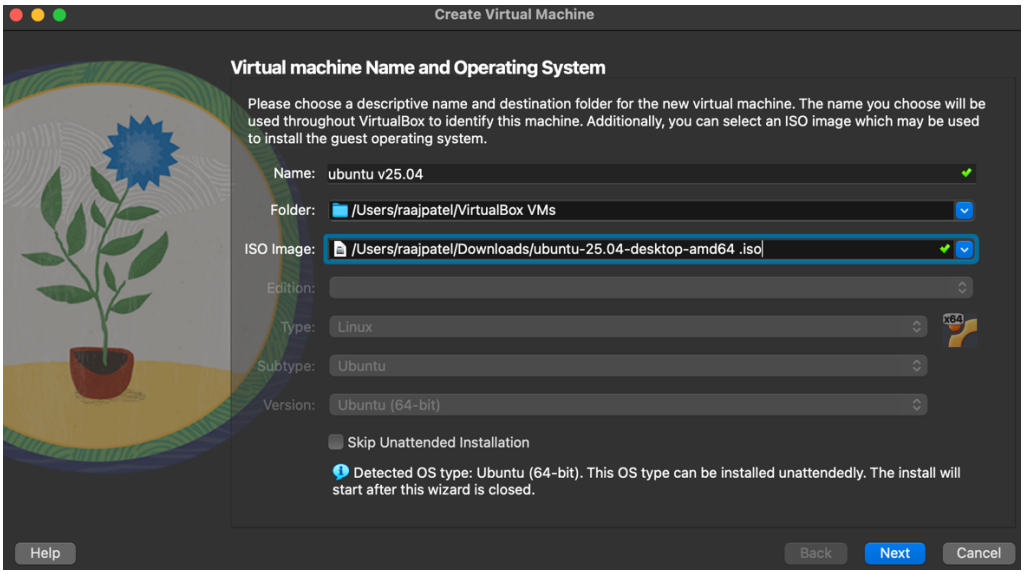


Figure 7 - Creating a new virtual machine in VirtualBox

- In the installation setup, I added my **username** and **password**. I also gave a **hostname** to the virtual machine so it could be identified on the network.

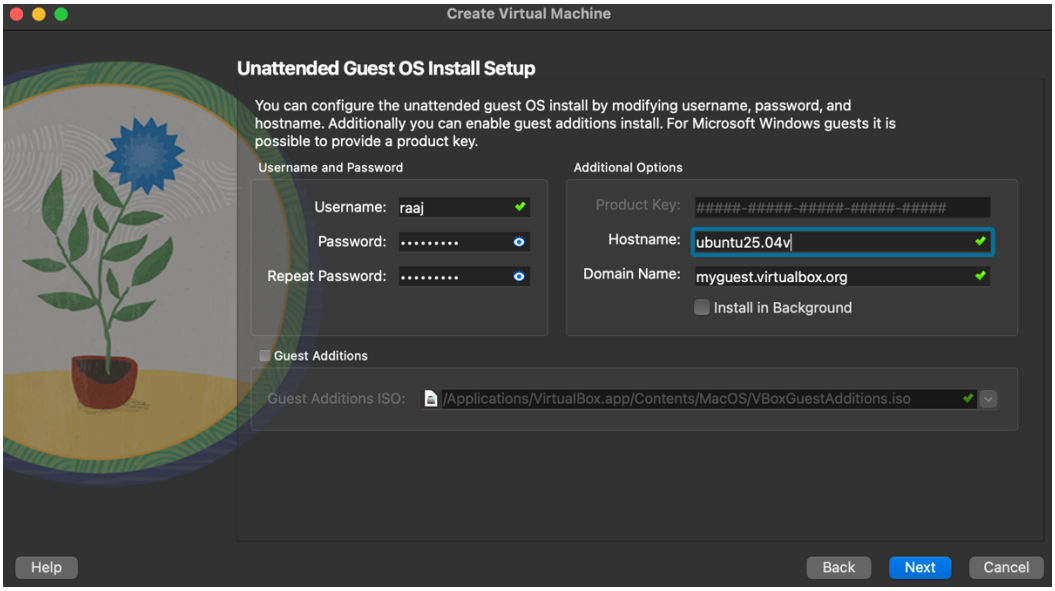


Figure 8 - Setting up the unattended guest OS installation

- I allocated **4 GB of RAM** and **2 processor cores** to the VM. This setup gives enough resources for Ubuntu to run smoothly without slowing down my main computer.

Figure 9 - Configuring hardware resources for the VM.

- I assigned **virtual hard disk** for the VM with a size of **40 GB**.

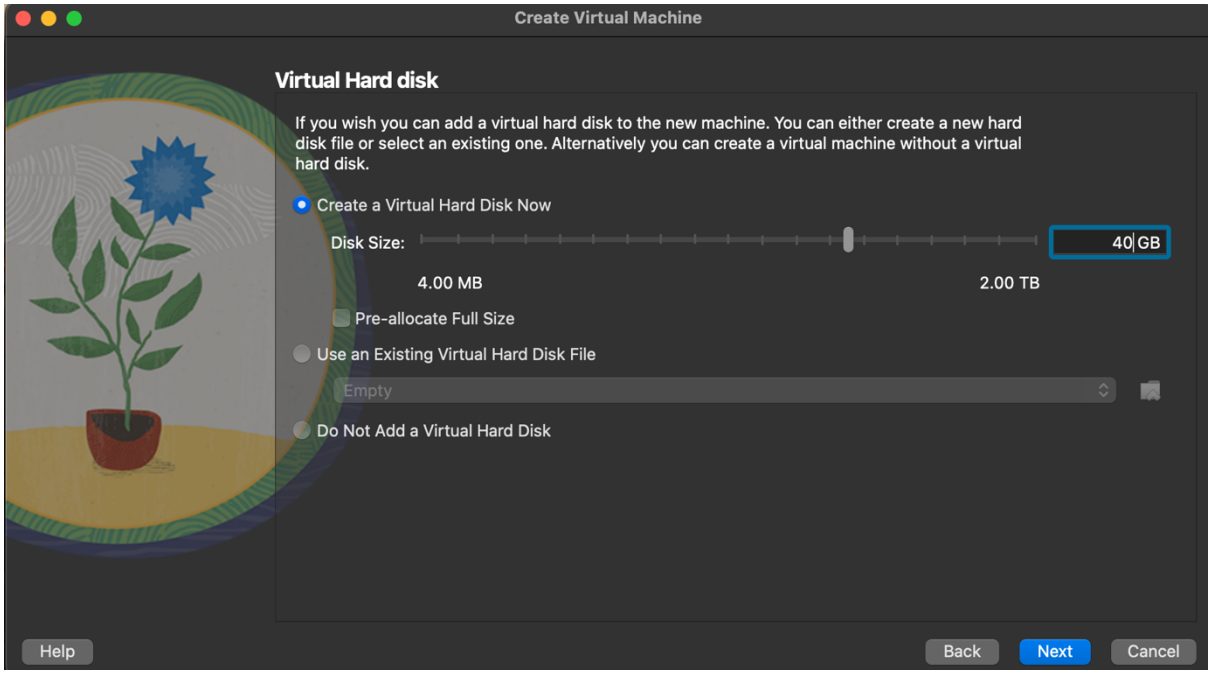


Figure 10 - Creating a new virtual hard disk for the VM

- I reviewed the **summary screen**, which showed all the settings I had selected (VM name, ISO path, memory, CPU, and hard disk size). Once everything looked correct, I clicked **Finish**.

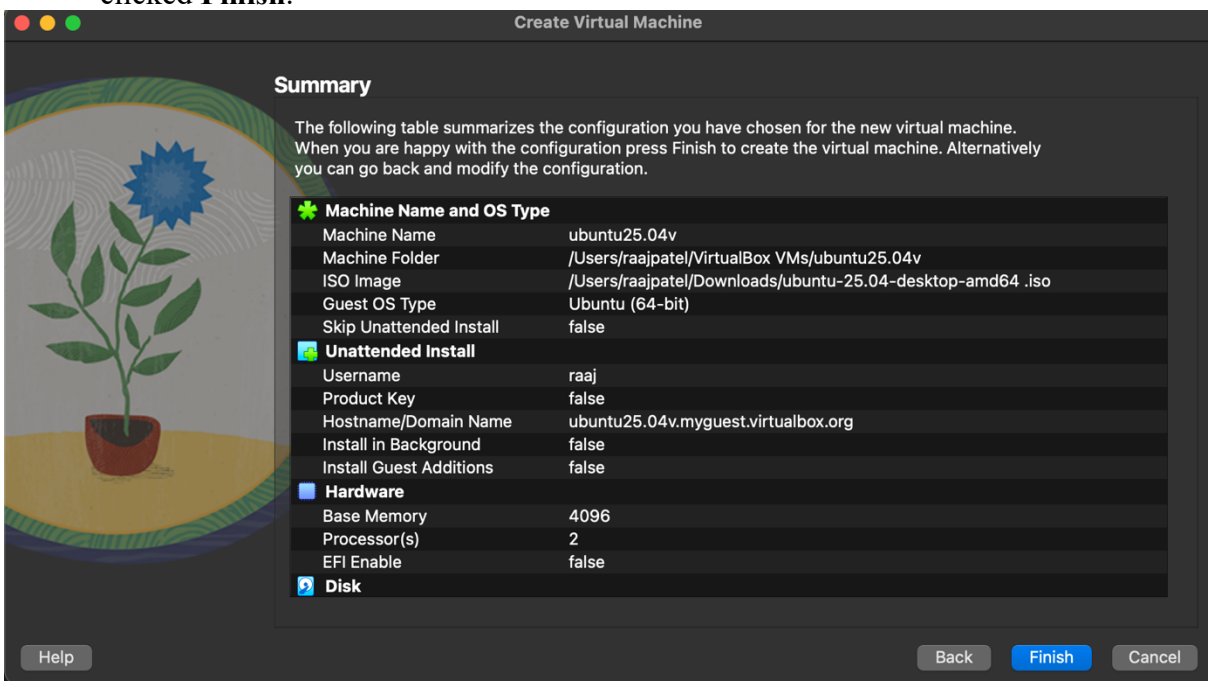


Figure 11 - Final summary of the VM configuration

- Finally, the new virtual machine appeared in the **VirtualBox Manager**. At this point, the VM was ready to be started and used for installing Ubuntu.

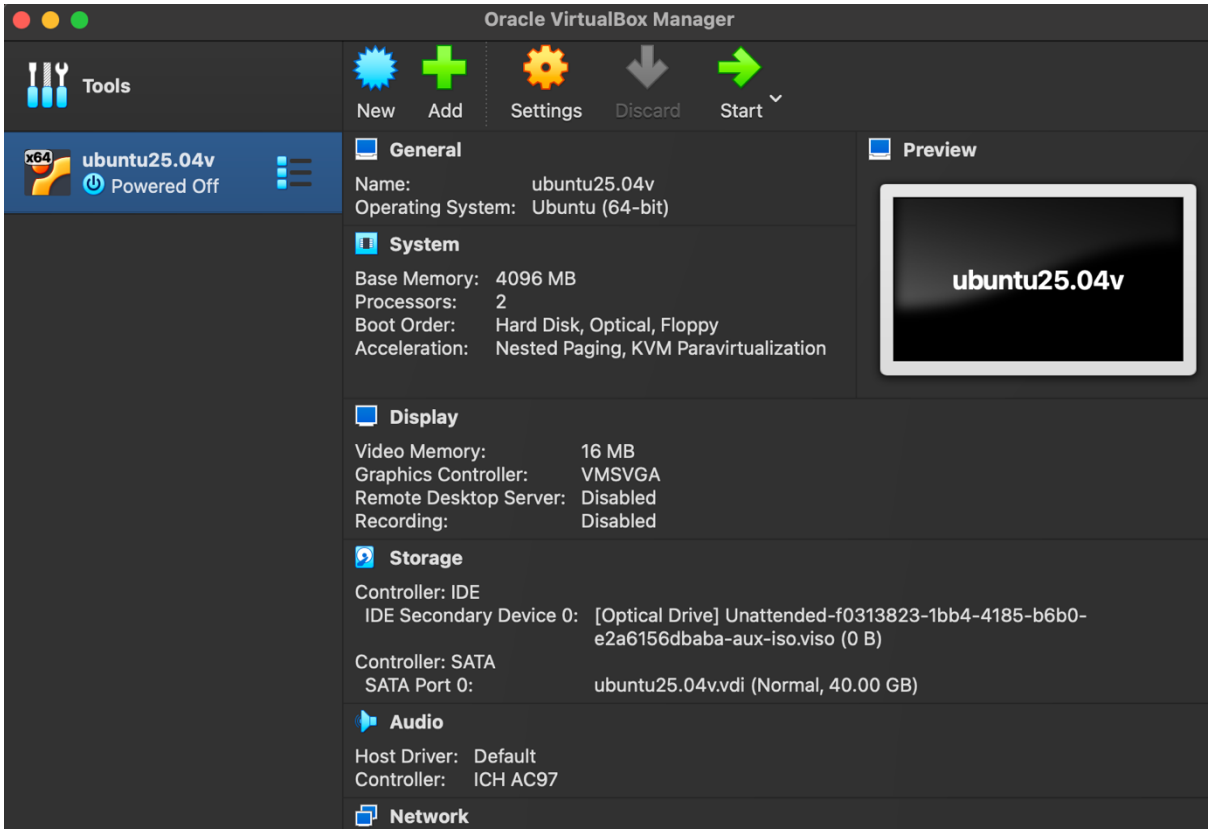


Figure 12 - VirtualBox Manager showing the newly created virtual machine ubuntu25.04v

Step 3: Run VM and setup necessary settings and update

- I launched the VM, and it opened into the GRUB menu. From there, I chose the "Install Ubuntu.

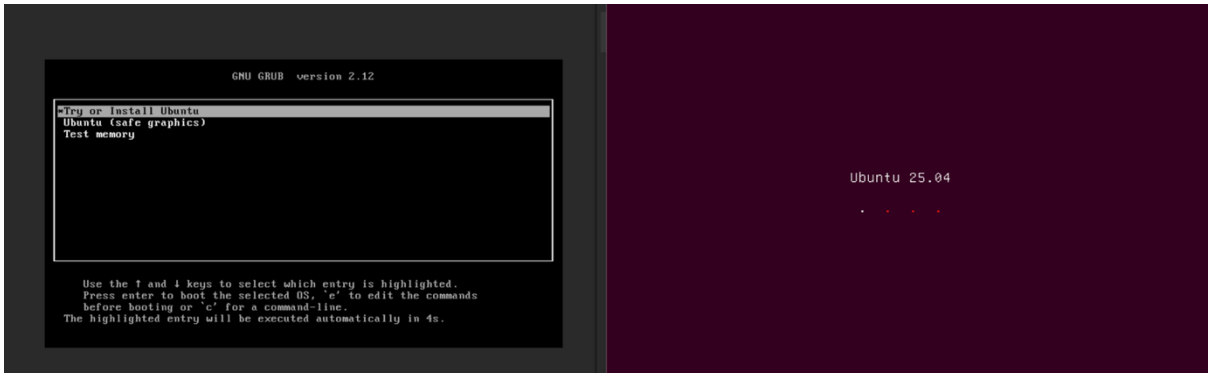


Figure 13 – The VM booting process from the Ubuntu ISO

- The system loaded the required files and displayed the Ubuntu logo while booting

- Logged in using the username and password created earlier.

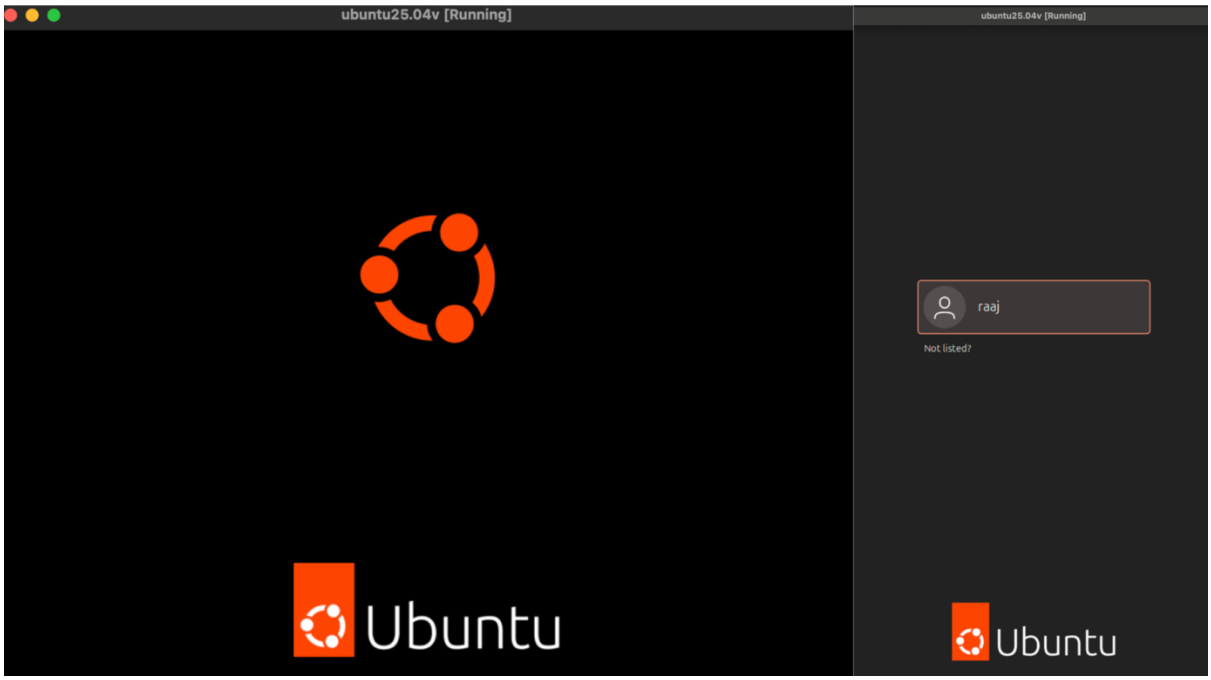


Figure 14 - Ubuntu startup screen with the logo, followed by the login page

- After the first login, Ubuntu displayed the **Welcome screen** with options to update the system and configure basic settings.
- The desktop was fully functional, and Ubuntu 25.04 was ready to use.

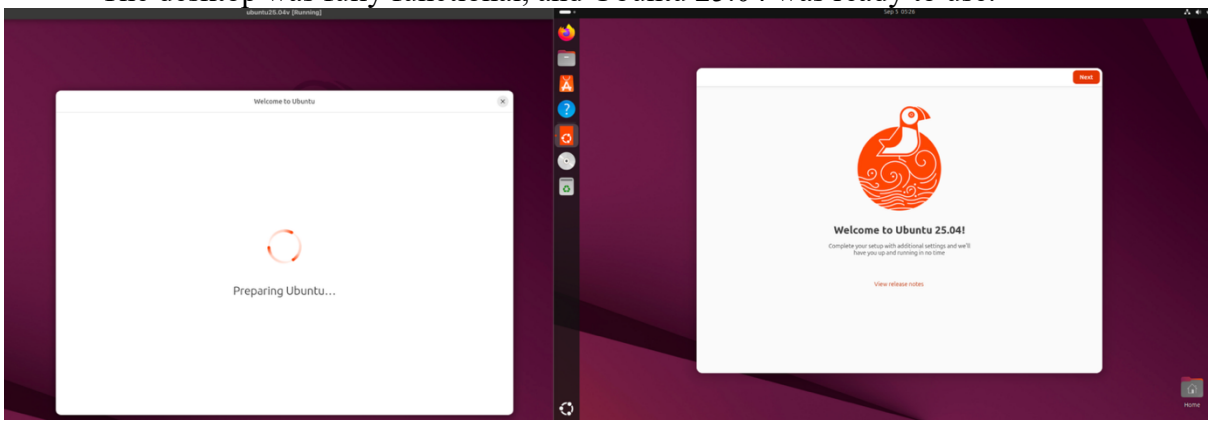


Figure 15 – Ubuntu finishing setup

3. INSTALLING PYTHON, JAVA, AND C PROGRAMMING ENVIRONMENTS ON THE VIRTUAL MACHINE

Step 1: Update repositories

- Opened the terminal in Ubuntu and ran "`sudo apt-get update`" to update all repositories.

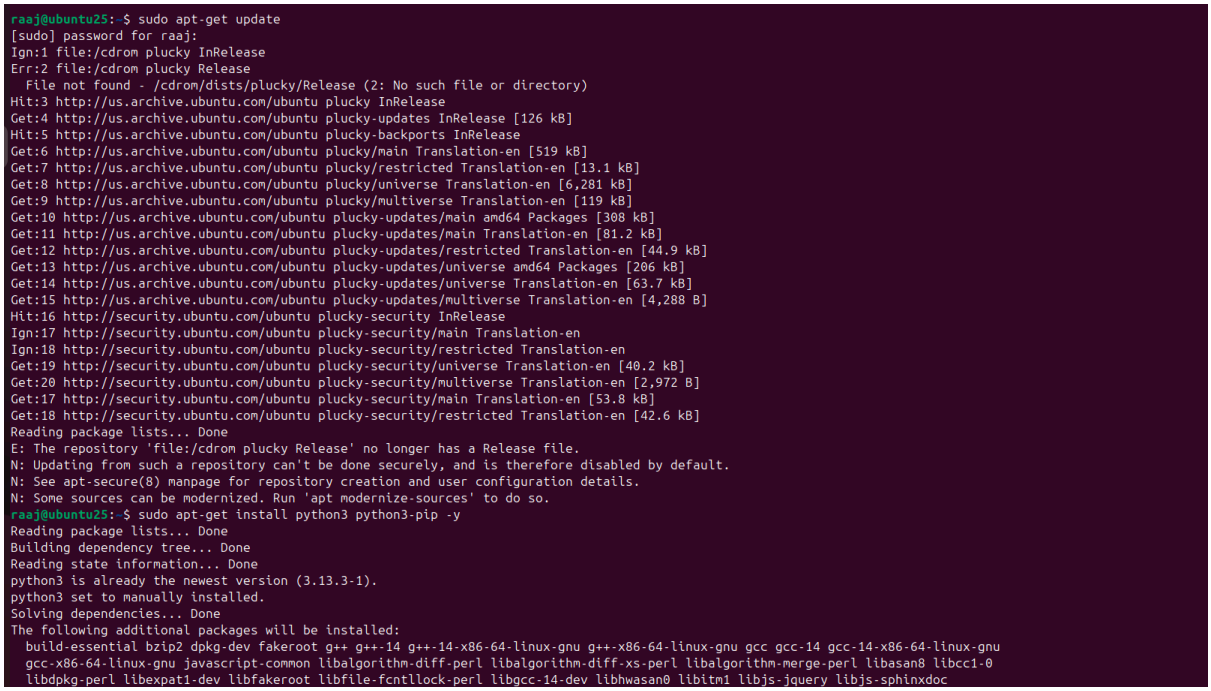


Figure 16 - Updating repositories in Ubuntu

Step 2: Install python environment

- I installed Python 3 along with pip using the following command "`sudo apt-get install python3 python3-pip -y`"

Figure 17 – Installing Python 3 and pip with the command

- Verified installation using command "`python3 --version`" that showed python 3.13.3 and "`pip3 --version`" that showed pip 25.0.

Figure 18 – Verifying installation by checking Python and pip versions in the terminal

Step 3: Install java environment

- installed java with "`sudo apt-get install default-jdk -y`".

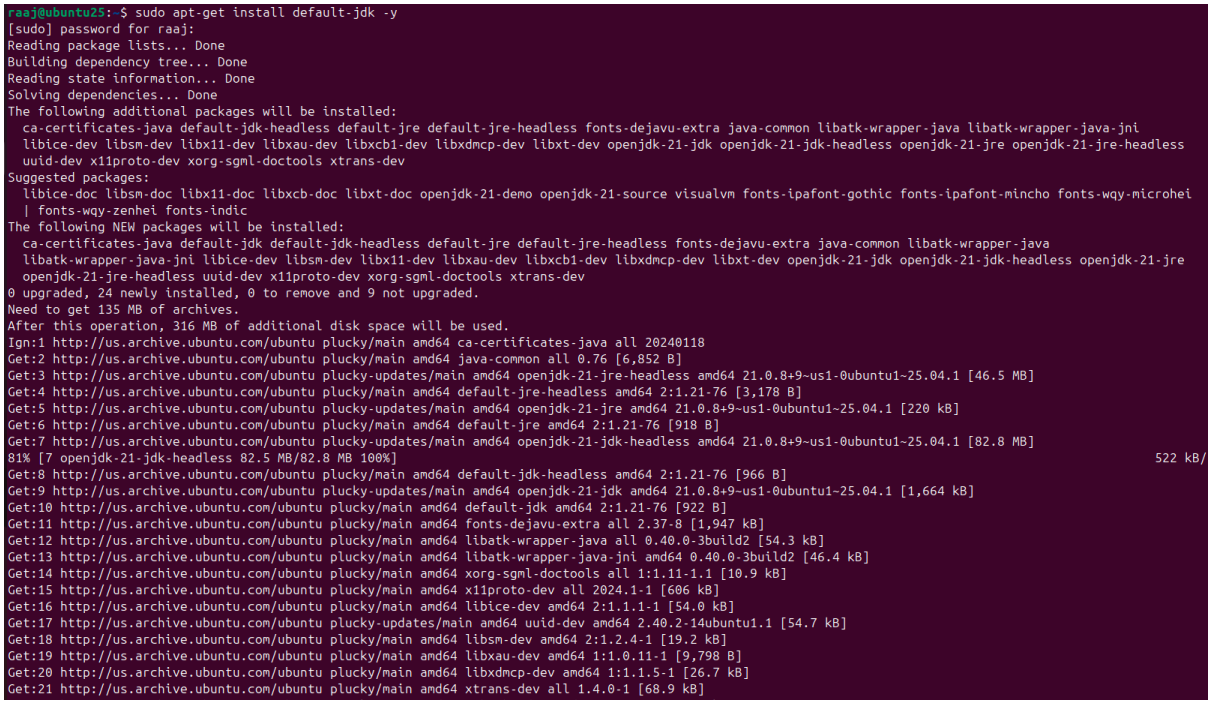


Figure 19 – Installing java with the command

- After running `java --version`, the system displayed OpenJDK 21.0.8, confirming the installation.

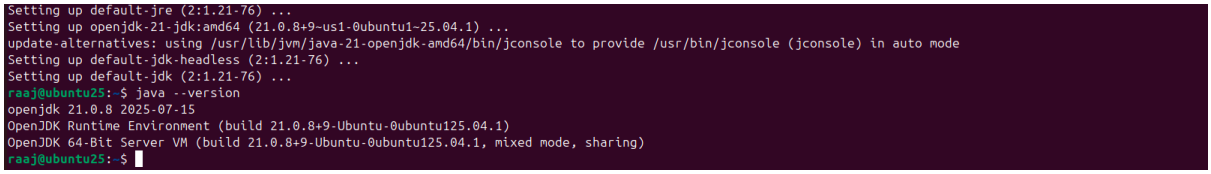


Figure 20 – Verifying installation for java

Step 4: Install for C environment

• Installed C by "`sudo apt-get install build-essential -y`" and the GNU Compiler Collection (GCC) and after installation, I checked the version of GCC by typing "`gcc --version`". Running `gcc --version` confirmed that GCC 14.2.0 was successfully installed.

```
raaj@ubuntu25:~$ sudo apt-get install build-essential -y
[sudo] password for raaj:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
build-essential is already the newest version (12.12ubuntu1).
build-essential set to manually installed.
Solving dependencies... Done
0 upgraded, 0 newly installed, 0 to remove and 9 not upgraded.
raaj@ubuntu25:~$ gcc --version
gcc (Ubuntu 14.2.0-19ubuntu2) 14.2.0
Copyright (C) 2024 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

raaj@ubuntu25:~$
```

Figure 21– Installing C and Verifying installation for C

4. RUN "STACK AND QUEUE" CODES ON THE VIRTUAL MACHINE, RESPECTIVELY.

- After setting up the programming environments, I tested the installation by running STACK AND QUEUE PROGRAMS in python, java, and c. This confirmed that all languages were working correctly inside the virtual machine.
- For that I create directory called “Algorithm” and create three folder named “C”, “Java” and “python”.

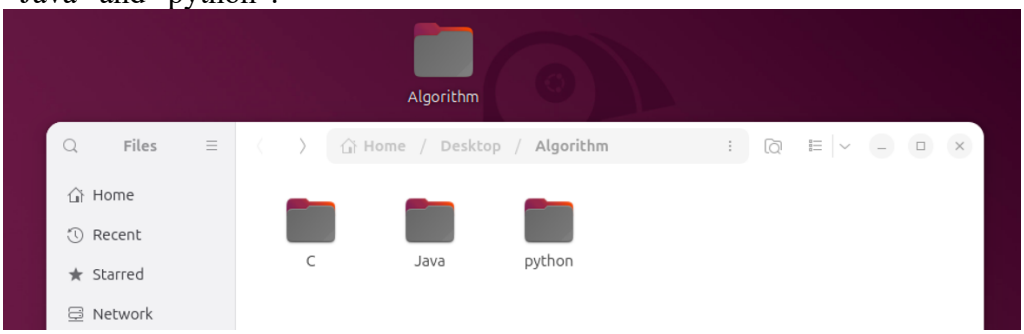


Figure 22 - Screenshot of create folders

Step 1: Stack and Queue program written in C

- In the **C folder**, I wrote two programs:
    - stack.c → stack implementation with `push` and `pop` functions.
    - queue.c → queue implementation with `enqueue` and `dequeue` functions.

```c
// Stack implementation in C

#include <stdio.h>
#include <stdlib.h>

#define MAX 10

int count = 0;

// Creating a stack
struct stack {
  int items[MAX];
  int top;
};
typedef struct stack st;

void createEmptyStack(st *s) {
  s->top = -1;
}

// Check if the stack is full
int isfull(st *s) {
  if (s->top == MAX - 1)
    return 1;
  else
    return 0;
}

// Check if the stack is empty
int isempty(st *s) {
  if (s->top == -1)
    return 1;
  else
    return 0;
}
```

```c
}

// Add elements into stack
void push(st *s, int newitem) {
  if (isfull(s)) {
    printf("STACK FULL");
  } else {
    s->top++;
    s->items[s->top] = newitem;
  }
  count++;
}

// Remove element from stack
void pop(st *s) {
  if (isempty(s)) {
    printf("\n STACK EMPTY \n");
  } else {
    printf("Item popped= %d", s->items[s->top]);
    s->top--;
  }
  count--;
  printf("\n");
}

// Print elements of stack
void printStack(st *s) {
  printf("Stack: ");
  for (int i = 0; i < count; i++) {
    printf("%d ", s->items[i]);
  }
  printf("\n");
}
```

```c
  printf("\n");
}

// Driver code
int main() {
  int ch;
  st *s = (st *)malloc(sizeof(st));

  createEmptyStack(s);

  push(s, 1);
  push(s, 2);
  push(s, 3);
  push(s, 4);

  printStack(s);

  pop(s);

  printf("\nAfter popping out\n");
  printStack(s);
}
```

Figure 23 – Screenshot of code for stack in C

```c
// Queue implementation in C

#include <stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int items[SIZE], front = -1, rear = -1;

int main() {
  //deQueue is not possible on empty queue
  deQueue();

  //enQueue 5 elements
  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);

  // 6th element can't be added to because the queue is full
  enQueue(6);

  display();

  //deQueue removes element entered first i.e. 1
  deQueue();

  //Now we have just 4 elements
  display();

  return 0;
}
```

```c
void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
    printf("\nInserted -> %d", value);
  }
}

void deQueue() {
  if (front == -1)
    printf("\nQueue is Empty!!");
  else {
    printf("\nDeleted : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}

// Function to print the queue
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    printf("\nQueue elements are:\n");
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```

Figure 24 – Screenshot of code for queue in C

- I compiled the programs using the gcc compiler inside the terminal and the stack showed last-in-first-out behavior, while the queue followed first-in-first-out..

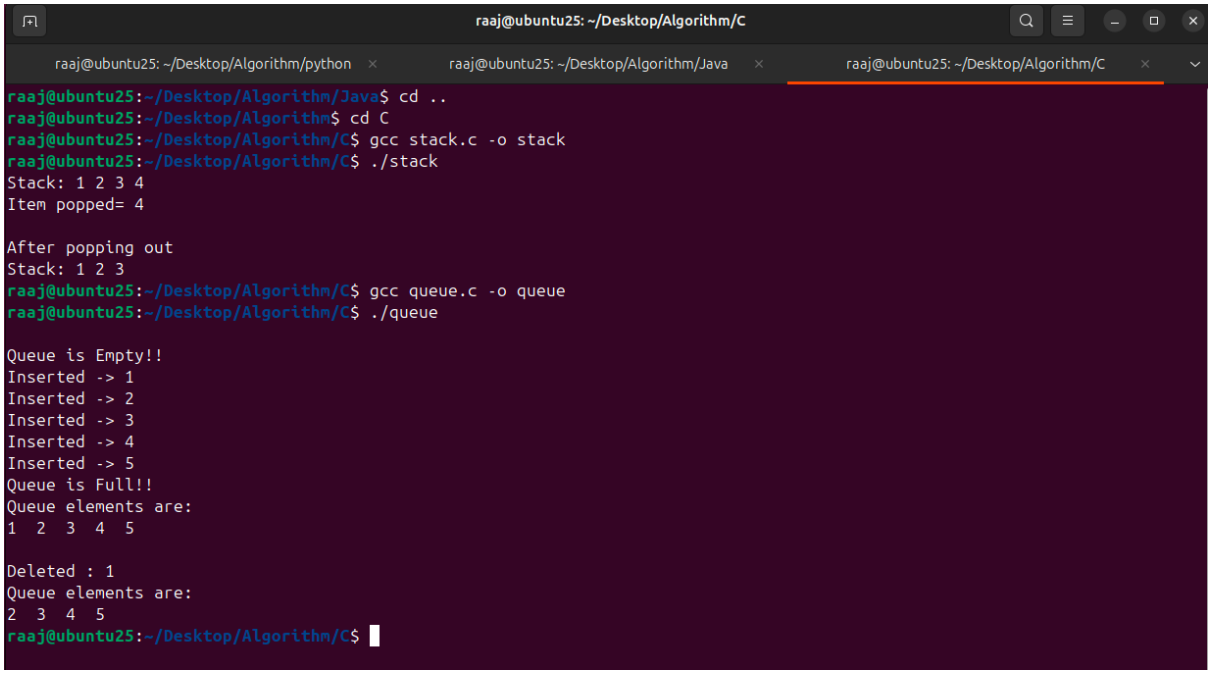


Figure 25 – Screenshot of running both code in C

Step 2: Stack and Queue program written in java

- In the **Java folder**, I created two classes:
  - Stack.java → stack program using Java methods like `push()` and `pop()`.
  - Queue.java → queue program using the `LinkedList` class for `add()` and `remove()`.

Open

Stack.java
~/Desktop/Algorithm/Java

```java
// Stack implementation in Java

class Stack {
  private int arr[];
  private int top;
  private int capacity;

  // Creating a stack
  Stack(int size) {
    arr = new int[size];
    capacity = size;
    top = -1;
  }

  // Add elements into stack
  public void push(int x) {
    if (isFull()) {
      System.out.println("OverFlow\nProgram Terminated\n");
      System.exit(1);
    }

    System.out.println("Inserting " + x);
    arr[++top] = x;
  }

  // Remove element from stack
  public int pop() {
    if (isEmpty()) {
      System.out.println("STACK EMPTY");
      System.exit(1);
    }
    return arr[top--];
  }

  // Utility function to return the size of the stack
  public int size() {
    return top + 1;
  }
```

Open

```java
  // Check if the stack is empty
  public Boolean isEmpty() {
    return top == -1;
  }

  // Check if the stack is full
  public Boolean isFull() {
    return top == capacity - 1;
  }

  public void printStack() {
    for (int i = 0; i <= top; i++) {
      System.out.println(arr[i]);
    }
  }

  public static void main(String[] args) {
    Stack stack = new Stack(5);
    stack.push(1);
    stack.push(2);
    stack.push(3);
    stack.push(4);
    stack.pop();
    System.out.println("\nAfter popping out");
    stack.printStack();
  }
}
```

Figure 26– Screenshot of code for stack in java

Open

Queue.java
~/Desktop/Algorithm/Java

```java
// Queue implementation in Java

public class Queue {
  int SIZE = 5;
  int items[] = new int[SIZE];
  int front, rear;

  Queue() {
    front = -1;
    rear = -1;
  }

  boolean isFull() {
    if (rear == SIZE - 1) {
      return true;
    }
    return false;
  }

  boolean isEmpty() {
    if (front == -1)
      return true;
    else
      return false;
  }

  void enQueue(int element) {
    if (isFull()) {
      System.out.println("Queue is full");
    } else {
      if (front == -1)
        front = 0;
      rear++;
      items[rear] = element;
      System.out.println("Inserted " + element);
    }
  }
```

Open

```java
    }
  }

  int deQueue() {
    int element;
    if (isEmpty()) {
      System.out.println("Queue is empty");
      return (-1);
    } else {
      element = items[front];
      if (front >= rear) {
        front = -1;
        rear = -1;
      } /* Q has only one element, so we reset the queue after deleting it. */
      else {
        front++;
      }
      System.out.println("Deleted -> " + element);
      return (element);
    }
  }

  void display() {
    /* Function to display elements of Queue */
    int i;
    if (isEmpty()) {
      System.out.println("Empty Queue");
    } else {
      System.out.println("\nFront index-> " + front);
      System.out.println("Items -> ");
      for (i = front; i <= rear; i++)
        System.out.print(items[i] + "  ");

      System.out.println("\nRear index-> " + rear);
    }
  }
```

Open

```java
        System.out.print(items[i] + "  ");

      System.out.println("\nRear index-> " + rear);
    }
  }

  public static void main(String[] args) {
    Queue q = new Queue();

    // deQueue is not possible on empty queue
    q.deQueue();

    // enQueue 5 elements
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);
    q.enQueue(4);
    q.enQueue(5);

    // 6th element can't be added to because the queue is full
    q.enQueue(6);

    q.display();

    // deQueue removes element entered first i.e. 1
    q.deQueue();

    // Now we have just 4 elements
    q.display();

  }
}
```

Figure 27 – Screenshot of code for queue in java

- I compiled them using javac and executed with java and the output confirmed correct behaviour for both stack and queue in Java.

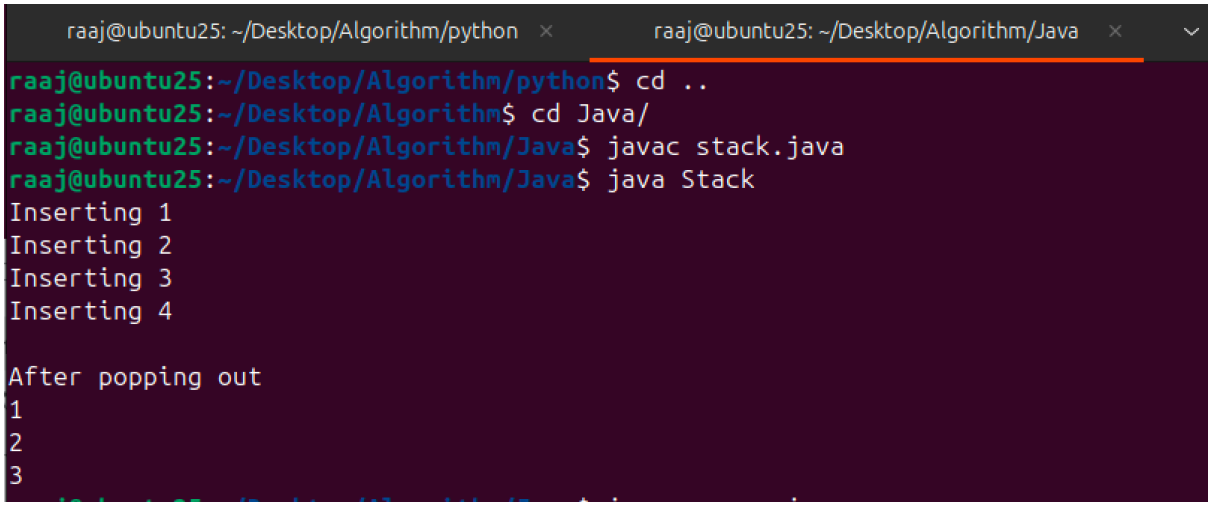


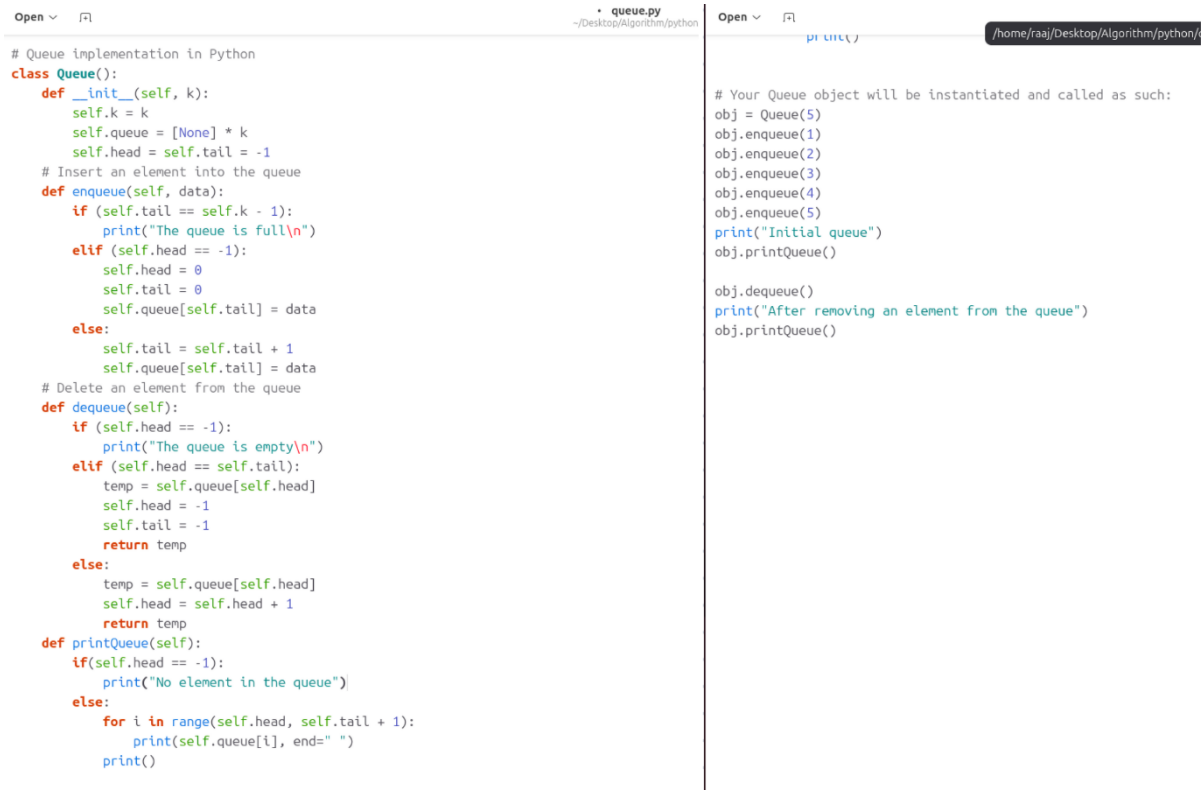
Figure 28 – Screenshot of running both code in java

Step 3: Stack and Queue program written in python

- In the **Python folder**, I wrote two scripts:
  - stack.py → stack program using a Python list with `append()` and `pop()`.
  - queue.py → queue program using the `collections.deque` class

Figure 29 – Screenshot of code for stack in python

- I ran the programs with the `python3` command in the terminal. The Python programs worked properly, producing the expected LIFO behaviour for stack and FIFO behaviour for queue.

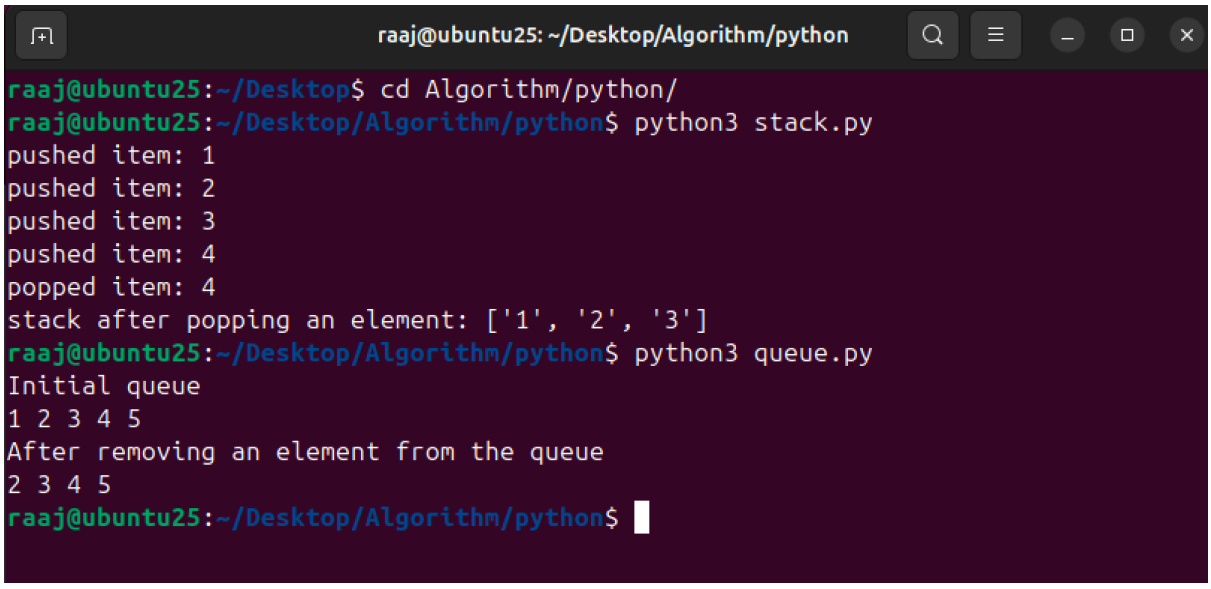


```
# Queue implementation in Python
class Queue():
    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1
    # Insert an element into the queue
    def enqueue(self, data):
        if (self.tail == self.k - 1):
            print("The queue is full\n")
        elif (self.head == -1):
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = self.tail + 1
            self.queue[self.tail] = data
    # Delete an element from the queue
    def dequeue(self):
        if (self.head == -1):
            print("The queue is empty\n")
        elif (self.head == self.tail):
            temp = self.queue[self.head]
            self.head = -1
            self.tail = -1
            return temp
        else:
            temp = self.queue[self.head]
            self.head = self.head + 1
            return temp
    def printQueue(self):
        if(self.head == -1):
            print("No element in the queue")
        else:
            for i in range(self.head, self.tail + 1):
                print(self.queue[i], end=" ")
            print()
```

```
# Your Queue object will be instantiated and called as such:
obj = Queue(5)
obj.enqueue(1)
obj.enqueue(2)
obj.enqueue(3)
obj.enqueue(4)
obj.enqueue(5)
print("Initial queue")
obj.printQueue()

obj.dequeue()
print("After removing an element from the queue")
obj.printQueue()
```

Figure 30 – Screenshot of code for queue in python



```
raaj@ubuntu25:~/Desktop$ cd Algorithm/python/
raaj@ubuntu25:~/Desktop/Algorithm/python$ python3 stack.py
pushed item: 1
pushed item: 2
pushed item: 3
pushed item: 4
popped item: 4
stack after popping an element: ['1', '2', '3']
raaj@ubuntu25:~/Desktop/Algorithm/python$ python3 queue.py
Initial queue
1 2 3 4 5
After removing an element from the queue
2 3 4 5
raaj@ubuntu25:~/Desktop/Algorithm/python$
```

Figure 31- Screenshot of running both code in python

Acknowledge

I am grateful for this homework because it gave me the opportunity to practice setting up a virtual machine and installing .Through this task, I also learned how to prepare environments for python, java, and c, and how to run stack and queue programs in each language. This work improved my practical skills and gave me more confidence in using different programming tools.

References

1. Programiz. (n.d.). *Stack Data Structure.* Retrieved September 5, 2025, from https://www.programiz.com/dsa/stack
2. Programiz. (n.d.). *Queue Data Structure.* Retrieved September 5, 2025, from https://www.programiz.com/dsa/queue
3. Ubuntu. (2025). *Download Ubuntu Desktop 25.04.* Canonical Ltd. Retrieved September 5, 2025, from https://ubuntu.com/download/desktop
4. Oracle. (2025). *VirtualBox Downloads.* Oracle Corporation. Retrieved September 5, 2025, from https://www.virtualbox.org/wiki/Downloads
5. Ubuntu Documentation. (2025). *Install Python on Ubuntu.* Canonical Ltd. Retrieved September 5, 2025, from https://ubuntu.com/server/docs/installing-python
6. Ubuntu Documentation. (2025). *Install Java on Ubuntu.* Canonical Ltd. Retrieved September 5, 2025, from https://ubuntu.com/tutorials/install-jdk-on-ubuntu
7. Ubuntu Documentation. (2025). *Install and Use GCC, the C Compiler.* Canonical Ltd. Retrieved September 5, 2025, from https://ubuntu.com/server/docs/programming-using-gcc

Appendix

1. Stack implementation in C

```
// Stack implementation in C

#include <stdio.h>
#include <stdlib.h>

#define MAX 10

int count = 0;

// Creating a stack
struct stack {
  int items[MAX];
  int top;
};
typedef struct stack st;

void createEmptyStack(st *s) {
```

```
  s->top = -1;
}

// Check if the stack is full
int isfull(st *s) {
  if (s->top == MAX - 1)
    return 1;
  else
    return 0;
}

// Check if the stack is empty
int isempty(st *s) {
  if (s->top == -1)
    return 1;
  else
    return 0;
}

// Add elements into stack
void push(st *s, int newitem) {
  if (isfull(s)) {
    printf("STACK FULL");
  } else {
    s->top++;
    s->items[s->top] = newitem;
  }
  count++;
}

// Remove element from stack
void pop(st *s) {
  if (isempty(s)) {
    printf("\n STACK EMPTY \n");
  } else {
    printf("Item popped= %d", s->items[s->top]);
    s->top--;
  }
  count--;
  printf("\n");
}

// Print elements of stack
void printStack(st *s) {
  printf("Stack: ");
  for (int i = 0; i < count; i++) {
    printf("%d ", s->items[i]);
  }
  printf("\n");
}
```

```
// Driver code
int main() {
  int ch;
  st *s = (st *)malloc(sizeof(st));

  createEmptyStack(s);

  push(s, 1);
  push(s, 2);
  push(s, 3);
  push(s, 4);

  printStack(s);

  pop(s);

  printf("\nAfter popping out\n");
  printStack(s);
}
```

2. Queue implementation in C

```
// Queue implementation in C

#include <stdio.h>
#define SIZE 5

void enQueue(int);
void deQueue();
void display();

int items[SIZE], front = -1, rear = -1;

int main() {
  //deQueue is not possible on empty queue
  deQueue();

  //enQueue 5 elements
  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);

  // 6th element can't be added to because the queue is full
  enQueue(6);
```

```
  display();

  //deQueue removes element entered first i.e. 1
  deQueue();

  //Now we have just 4 elements
  display();

  return 0;
}

void enQueue(int value) {
  if (rear == SIZE - 1)
    printf("\nQueue is Full!!");
  else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = value;
    printf("\nInserted -> %d", value);
  }
}

void deQueue() {
  if (front == -1)
    printf("\nQueue is Empty!!");
  else {
    printf("\nDeleted : %d", items[front]);
    front++;
    if (front > rear)
      front = rear = -1;
  }
}

// Function to print the queue
void display() {
  if (rear == -1)
    printf("\nQueue is Empty!!!");
  else {
    int i;
    printf("\nQueue elements are:\n");
    for (i = front; i <= rear; i++)
      printf("%d  ", items[i]);
  }
  printf("\n");
}
```

3. Stack implementation in java

```
// Stack implementation in Java

class Stack {
  private int arr[];
  private int top;
  private int capacity;

  // Creating a stack
  Stack(int size) {
    arr = new int[size];
    capacity = size;
    top = -1;
  }

  // Add elements into stack
  public void push(int x) {
    if (isFull()) {
      System.out.println("OverFlow\nProgram Terminated\n");
      System.exit(1);
    }

    System.out.println("Inserting " + x);
    arr[++top] = x;
  }

  // Remove element from stack
  public int pop() {
    if (isEmpty()) {
      System.out.println("STACK EMPTY");
      System.exit(1);
    }
    return arr[top--];
  }

  // Utility function to return the size of the stack
  public int size() {
    return top + 1;
  }

  // Check if the stack is empty
  public Boolean isEmpty() {
    return top == -1;
  }

  // Check if the stack is full
  public Boolean isFull() {
```

```
  return top == capacity - 1;
 }

 public void printStack() {
  for (int i = 0; i <= top; i++) {
   System.out.println(arr[i]);
  }
 }

 public static void main(String[] args) {
  Stack stack = new Stack(5);

  stack.push(1);
  stack.push(2);
  stack.push(3);
  stack.push(4);

  stack.pop();
  System.out.println("\nAfter popping out");

  stack.printStack();

 }
}
```

4. Queue implementation in java

```
// Queue implementation in Java

public class Queue {
 int SIZE = 5;
 int items[] = new int[SIZE];
 int front, rear;

 Queue() {
  front = -1;
  rear = -1;
 }

 boolean isFull() {
  if (rear == SIZE - 1) {
   return true;
  }
  return false;
 }

 boolean isEmpty() {
  if (front == -1)
```

```
    return true;
  else
    return false;
}

void enQueue(int element) {
  if (isFull()) {
    System.out.println("Queue is full");
  } else {
    if (front == -1)
      front = 0;
    rear++;
    items[rear] = element;
    System.out.println("Inserted " + element);
  }
}

int deQueue() {
  int element;
  if (isEmpty()) {
    System.out.println("Queue is empty");
    return (-1);
  } else {
    element = items[front];
    if (front >= rear) {
      front = -1;
      rear = -1;
    } /* Q has only one element, so we reset the queue after deleting it. */
    else {
      front++;
    }
    System.out.println("Deleted -> " + element);
    return (element);
  }
}

void display() {
  /* Function to display elements of Queue */
  int i;
  if (isEmpty()) {
    System.out.println("Empty Queue");
  } else {
    System.out.println("\nFront index-> " + front);
    System.out.println("Items -> ");
    for (i = front; i <= rear; i++)
      System.out.print(items[i] + "  ");

    System.out.println("\nRear index-> " + rear);
  }
}
```

```
  public static void main(String[] args) {
    Queue q = new Queue();

    // deQueue is not possible on empty queue
    q.deQueue();

    // enQueue 5 elements
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);
    q.enQueue(4);
    q.enQueue(5);

    // 6th element can't be added to because the queue is full
    q.enQueue(6);

    q.display();

    // deQueue removes element entered first i.e. 1
    q.deQueue();

    // Now we have just 4 elements
    q.display();

  }
}
```

5. Stack implementation in python

```
# Stack implementation in python

# Creating a stack
def create_stack():
    stack = []
    return stack


# Creating an empty stack
def check_empty(stack):
    return len(stack) == 0


# Adding items into the stack
def push(stack, item):
    stack.append(item)
    print("pushed item: " + item)
```

```
# Removing an element from the stack
def pop(stack):
    if (check_empty(stack)):
        return "stack is empty"

    return stack.pop()


stack = create_stack()
push(stack, str(1))
push(stack, str(2))
push(stack, str(3))
push(stack, str(4))
print("popped item: " + pop(stack))
print("stack after popping an element: " + str(stack))
```

6. Queue implementation in python

```
# Queue implementation in Python

class Queue():

    def __init__(self, k):
        self.k = k
        self.queue = [None] * k
        self.head = self.tail = -1

    # Insert an element into the queue
    def enqueue(self, data):

        if (self.tail == self.k - 1):
            print("The queue is full\n")

        elif (self.head == -1):
            self.head = 0
            self.tail = 0
            self.queue[self.tail] = data
        else:
            self.tail = self.tail + 1
            self.queue[self.tail] = data

    # Delete an element from the queue
    def dequeue(self):
        if (self.head == -1):
            print("The queue is empty\n")

        elif (self.head == self.tail):
            temp = self.queue[self.head]
            self.head = -1
```

```
            self.tail = -1
            return temp
        else:
            temp = self.queue[self.head]
            self.head = self.head + 1
            return temp

    def printQueue(self):
        if(self.head == -1):
            print("No element in the queue")

        else:
            for i in range(self.head, self.tail + 1):
                print(self.queue[i], end=" ")
            print()


# Your Queue object will be instantiated and called as such:
obj = Queue(5)
obj.enqueue(1)
obj.enqueue(2)
obj.enqueue(3)
obj.enqueue(4)
obj.enqueue(5)
print("Initial queue")
obj.printQueue()

obj.dequeue()
print("After removing an element from the queue")
obj.printQueue()
```