

Homework 1

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

Name	Email
Raaj Atulkumar Patel	Pate2682@pnw.edu

Table of Contents

Abstract	4
1. Installing an virtual machine in your laptop	4
Step 1: Download VirtualBox	4
Step 2: Install VirtualBox.....	4
2. Installing the Ubuntu 25.04 Desktop	6
Step 1: Download the ubuntu 25.04 iso	6
Step 2: Start the VM and load 25.04 version Ubuntu iso file	7
Step 3: Run VM and setup necessary settings and update	10
3. Installing Python, Java, and C programming environments on the virtual machine	11
Step 1: Update repositories	11
Step 2: Install python environment	12
Step 3: Install java environment.....	13
Step 4: Install for C environment	14
4. Run "stack and queue" codes on the virtual machine, respectively.....	15
Step 1: Stack and Queue program written in C	15
Step 2: Stack and Queue program written in java	17
Step 3: Stack and Queue program written in python	19
Acknowledge	21
References.....	21
Appendix.....	21
1. Stack implementation in C.....	21
2. Queue implementation in C	23
3. Stack implementation in java.....	25
4. Queue implementation in java	26
5. Stack implementation in python	28
6. Queue implementation in python.....	29

Table of Figures

Figure 1 - Downloading the VirtualBox installer package from the official Oracle website ...	4
Figure 2 - Opening the downloaded file	5
Figure 3 - Step-by-step installation wizard for VirtualBox on macOS installing the Ubuntu 25.04 Desktop	5
Figure 4 - The official Ubuntu downloads page showing the option to download Ubuntu 25.04 Desktop	6
Figure 5 - Confirmation page after starting the Ubuntu 25.04 download.....	6
Figure 6 - Open virtual box.....	7
Figure 7 - Creating a new virtual machine in VirtualBox	7
Figure 8 - Setting up the unattended guest OS installation.....	8
Figure 9 - Configuring hardware resources for the VM.	8
Figure 10 - Creating a new virtual hard disk for the VM	9
Figure 11 - Final summary of the VM configuration	9
Figure 12 - VirtualBox Manager showing the newly created virtual machine ubuntu25.04v.10	10
Figure 13 – The VM booting process from the Ubuntu ISO	10
Figure 14 - Ubuntu startup screen with the logo, followed by the login page.....	11
Figure 15 – Ubuntu finishing setup	11
Figure 16 - Updating repositories in Ubuntu	12
Figure 17 – Installing Python 3 and pip with the command	13
Figure 18 – Verifying installation by checking Python and pip versions in the terminal	13
Figure 19 – Installing java with the command.....	14
Figure 20 – Verifying installation for java	14
Figure 21– Installing C and Verifying installation for C	15
Figure 22 - Screenshot of create folders	15
Figure 23 – Screenshot of code for stack in C	16
Figure 24 – Screenshot of code for queue in C	16
Figure 25 – Screenshot of running both code in C	17
Figure 26– Screenshot of code for stack in java	18
Figure 27 – Screenshot of code for queue in java.....	18
Figure 28 – Screenshot of running both code in java	19
Figure 29 – Screenshot of code for stack in python.....	19
Figure 30 – Screenshot of code for queue in python	20
Figure 31- Screenshot of running both code in python.....	20

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.

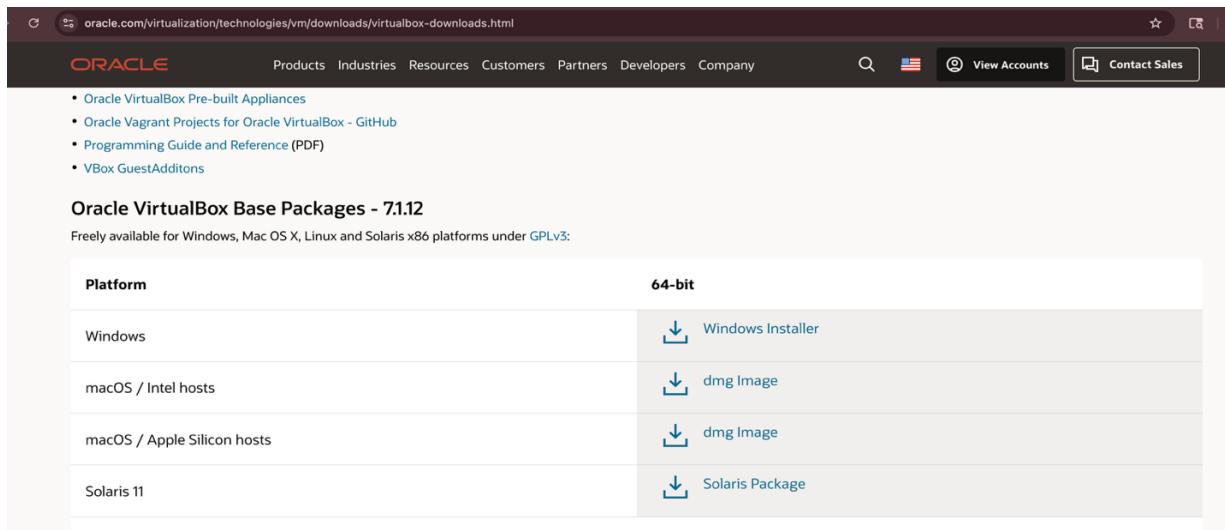


Figure 1 - Downloading the VirtualBox installer package from the official Oracle website

Step 2: Install VirtualBox

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

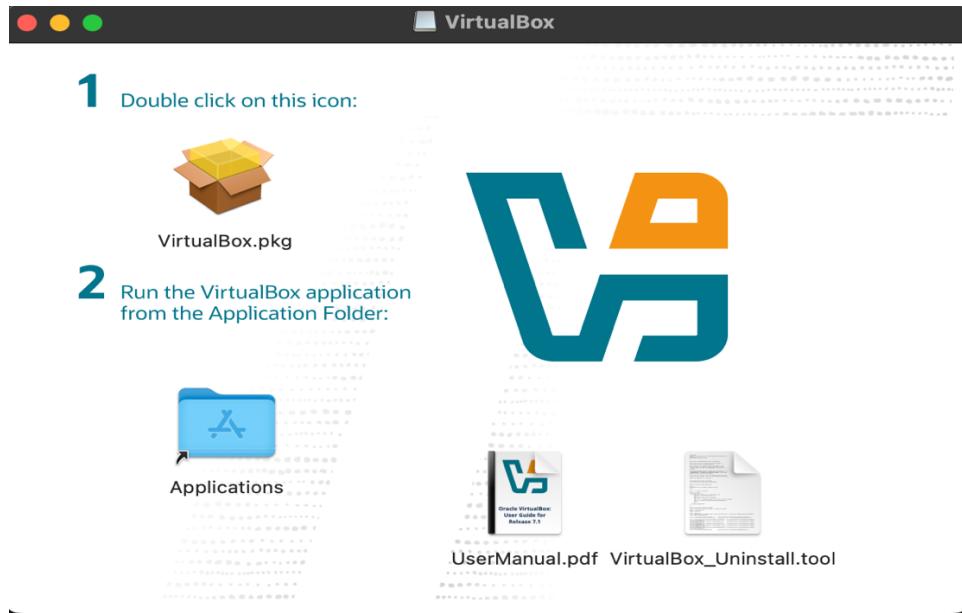


Figure 2 - Opening the downloaded file

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

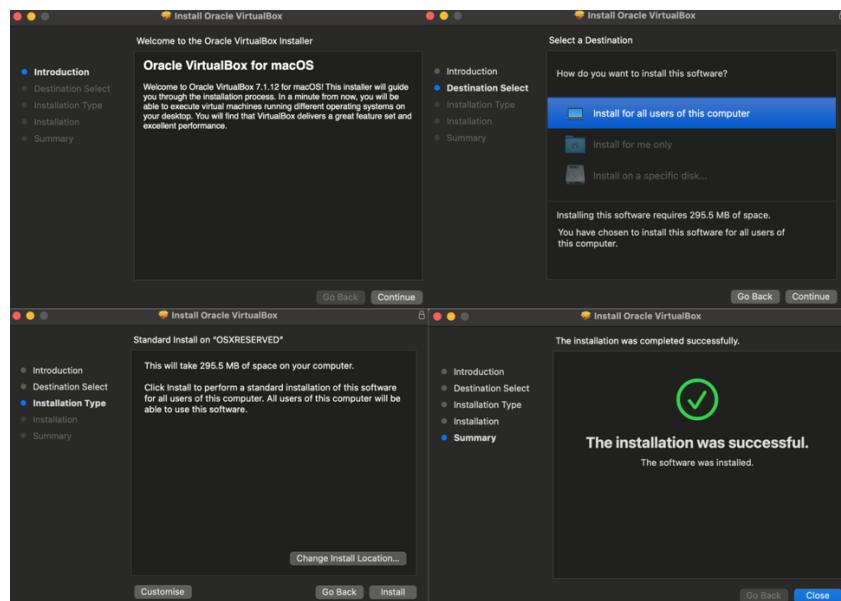


Figure 3 - Step-by-step installation wizard for VirtualBox on macOS installing the Ubuntu 25.04 Desktop

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”.

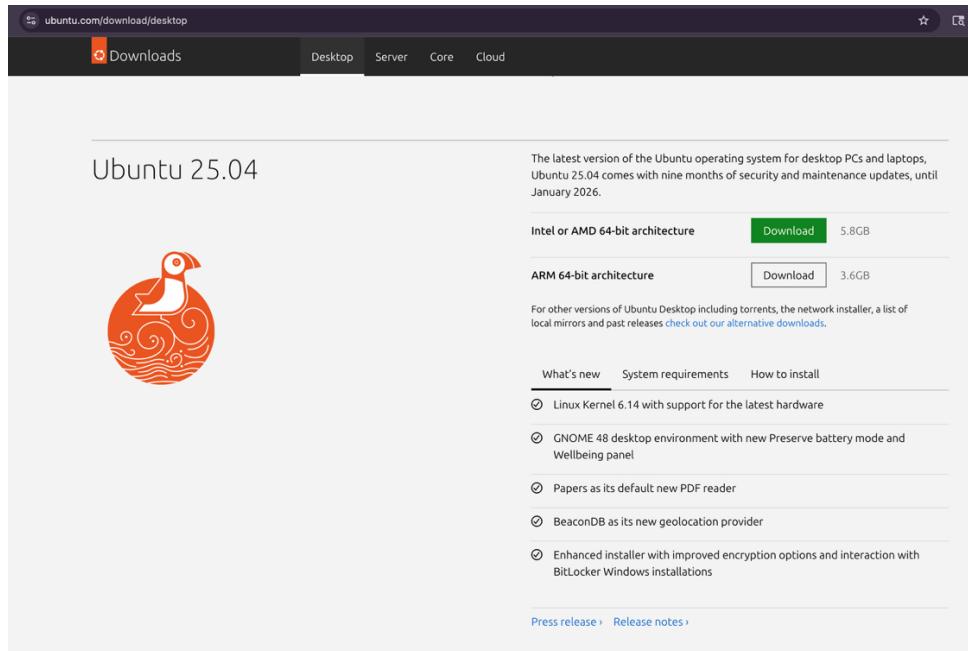


Figure 4 - The official Ubuntu downloads page showing the option to download Ubuntu 25.04 Desktop

- Successfully Downloaded of 5.8 GB iso file.

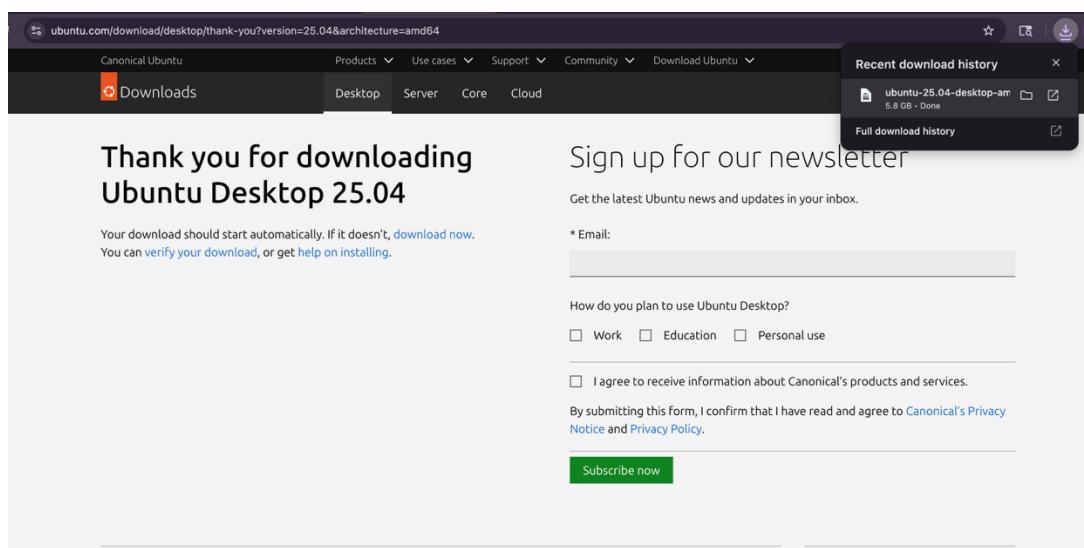


Figure 5 - Confirmation page after starting the Ubuntu 25.04 download

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

- Open VirtualBox and select new.

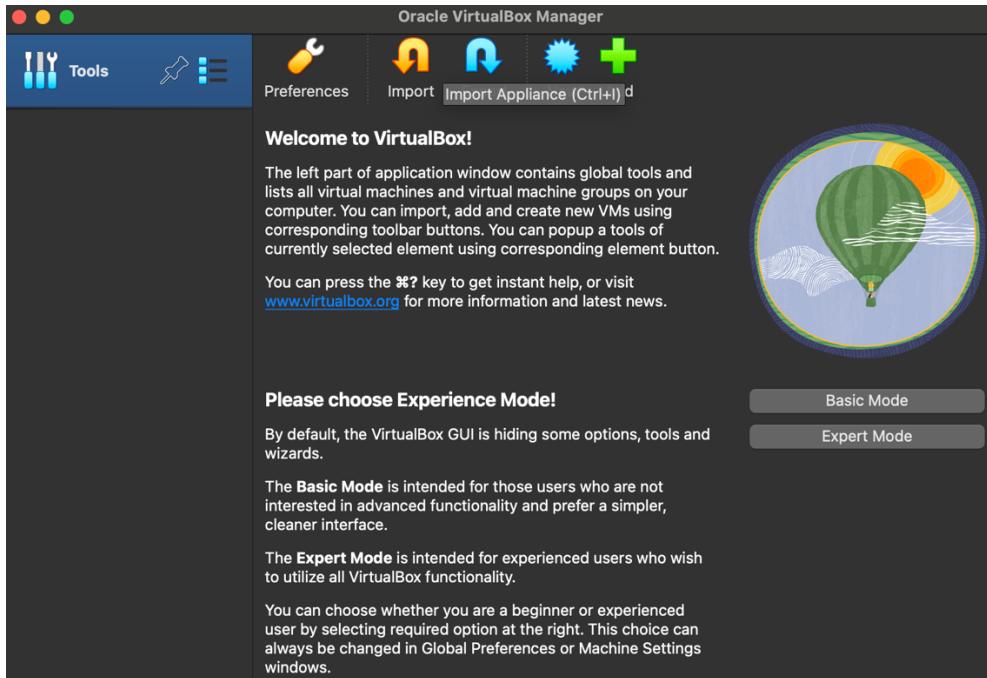


Figure 6 - Open virtual box

- 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”.

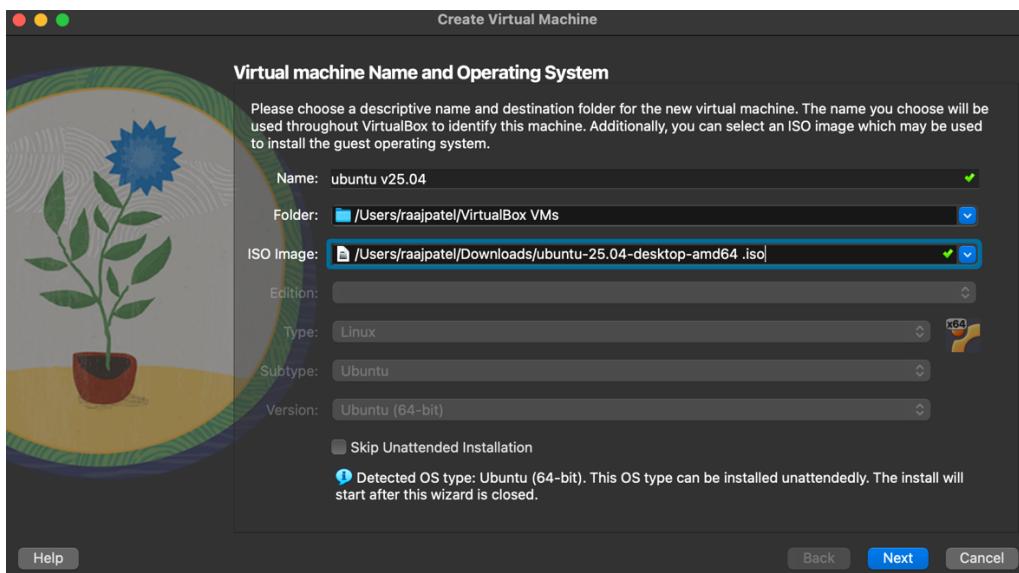


Figure 7 - Creating a new virtual machine in VirtualBox

- 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.

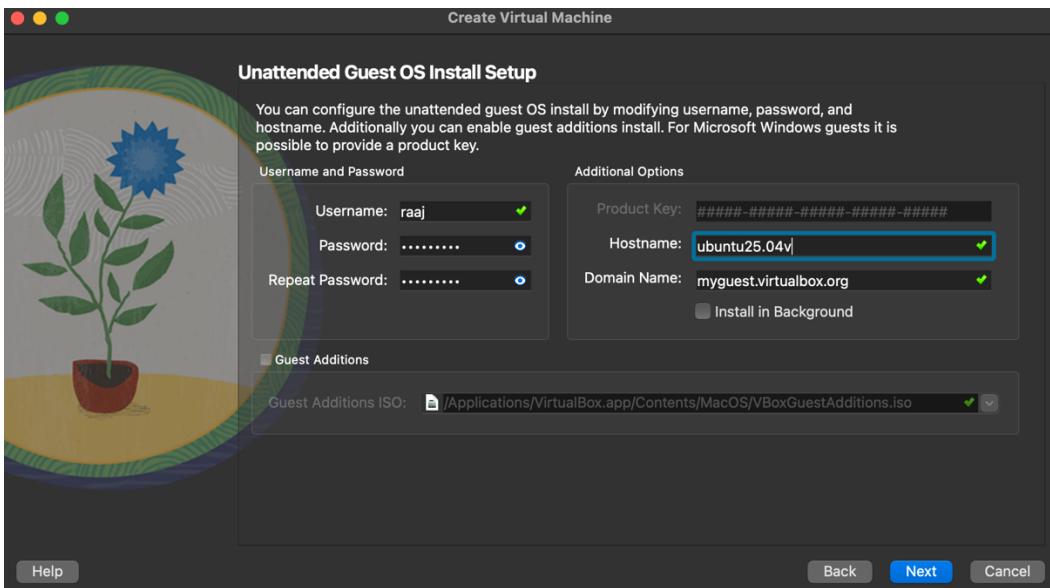


Figure 8 - Setting up the unattended guest OS installation

- 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.

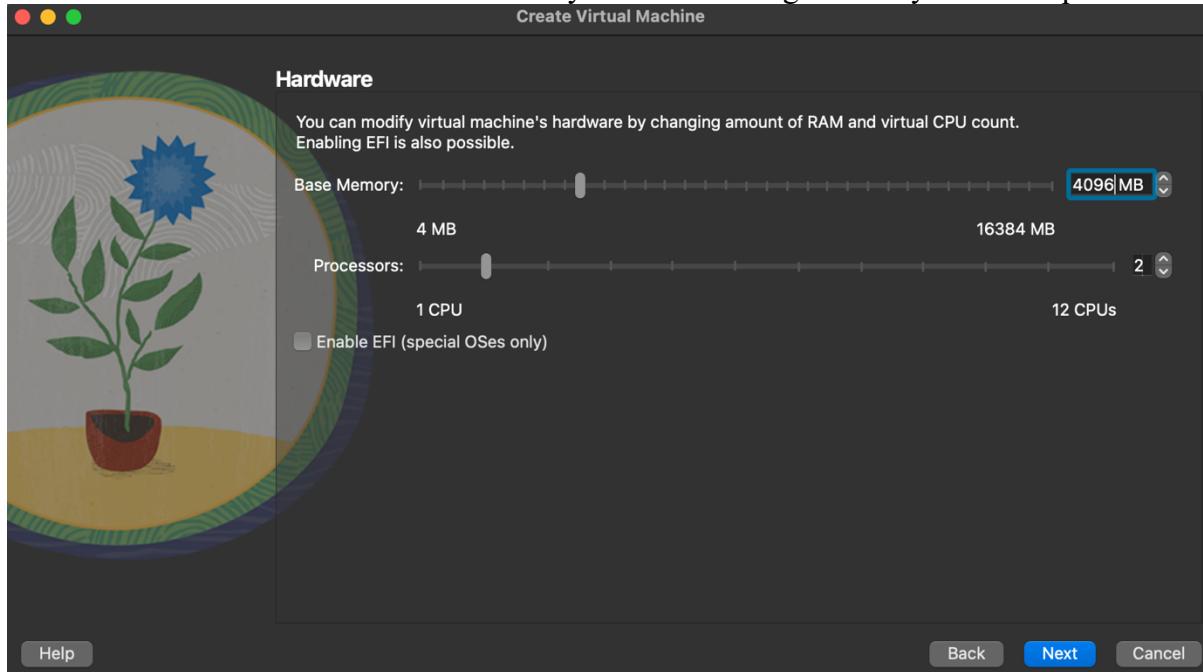


Figure 9 - Configuring hardware resources for the VM.

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

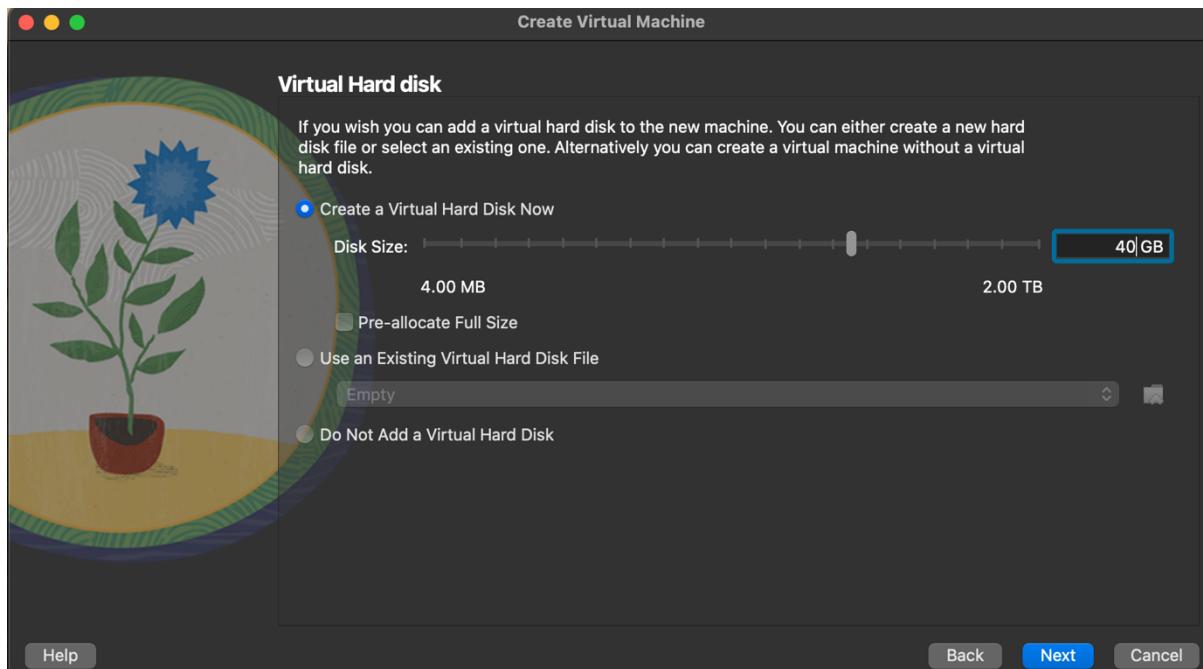


Figure 10 - Creating a new virtual hard disk for the VM

- 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.

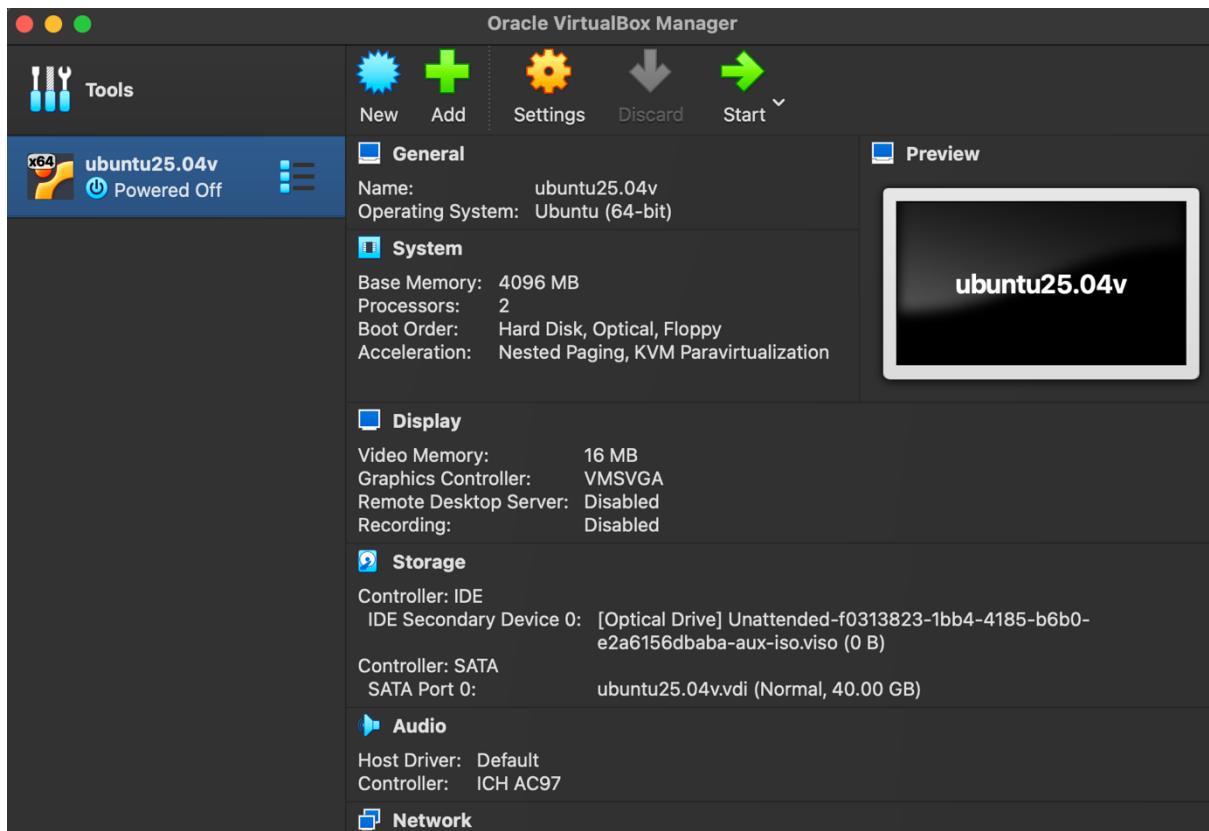


Figure 12 - VirtualBox Manager showing the newly created virtual machine ubuntu25.04v

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”.

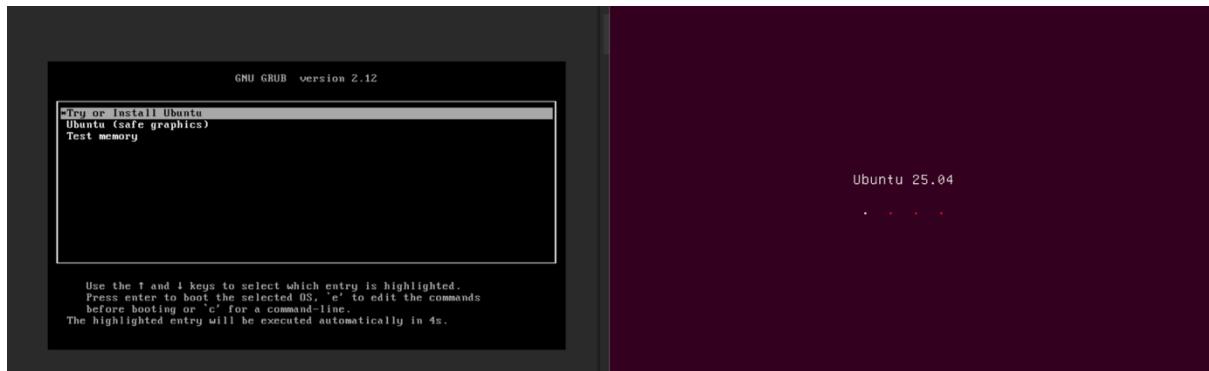


Figure 13 – The VM booting process from the Ubuntu ISO

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

- Logged in using the username and password created earlier.

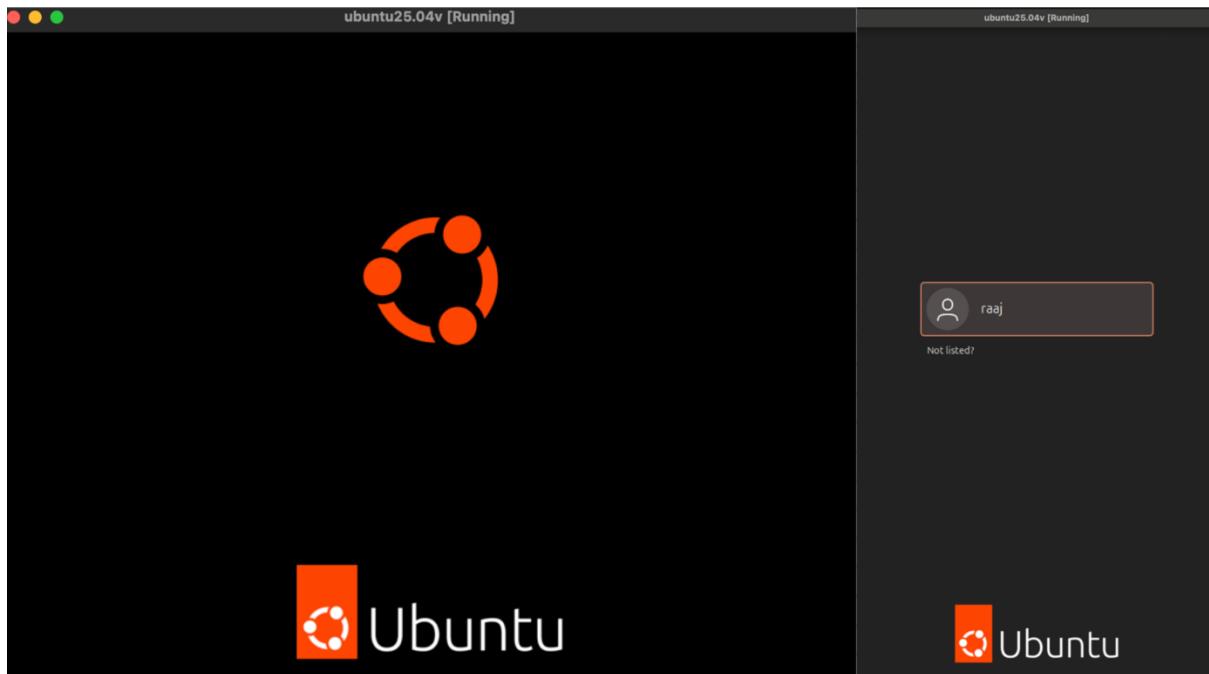


Figure 14 - Ubuntu startup screen with the logo, followed by the login page

- 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.

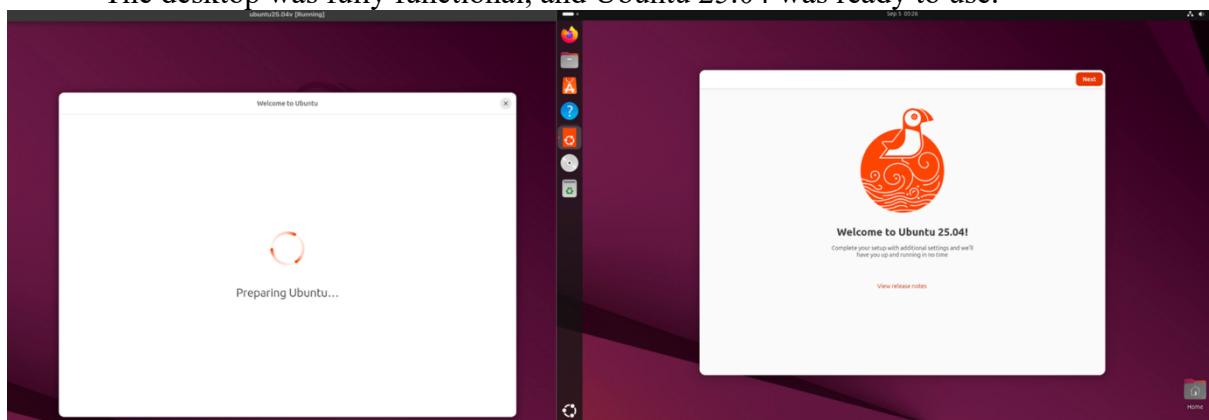


Figure 15 – Ubuntu finishing setup

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.

```

raaj@ubuntu25: $ sudo apt-get update
[sudo] password for raaj:
Ign:1 file:/cdrom plucky InRelease
Err:2 file:/cdrom plucky Release
  File not found - /cdrom/dists/plucky/Release (2: No such file or directory)
Hit:3 http://us.archive.ubuntu.com/ubuntu plucky InRelease
Get:4 http://us.archive.ubuntu.com/ubuntu plucky-updates InRelease [126 kB]
Hit:5 http://us.archive.ubuntu.com/ubuntu plucky-backports InRelease
Get:6 http://us.archive.ubuntu.com/ubuntu plucky/main Translation-en [519 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu plucky/restricted Translation-en [13.1 kB]
Get:8 http://us.archive.ubuntu.com/ubuntu plucky/universe Translation-en [6,281 kB]
Get:9 http://us.archive.ubuntu.com/ubuntu plucky/multiverse Translation-en [119 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 Packages [308 kB]
Get:11 http://us.archive.ubuntu.com/ubuntu plucky-updates/main Translation-en [81.2 kB]
Get:12 http://us.archive.ubuntu.com/ubuntu plucky-updates/restricted Translation-en [44.9 kB]
Get:13 http://us.archive.ubuntu.com/ubuntu plucky-updates/universe amd64 Packages [206 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu plucky-updates/universe Translation-en [63.7 kB]
Get:15 http://us.archive.ubuntu.com/ubuntu plucky-updates/multiverse Translation-en [4,288 B]
Hit:16 http://security.ubuntu.com/ubuntu plucky-security InRelease
Ign:17 http://security.ubuntu.com/ubuntu plucky-security/main Translation-en
Ign:18 http://security.ubuntu.com/ubuntu plucky-security/restricted Translation-en
Get:19 http://security.ubuntu.com/ubuntu plucky-security/universe Translation-en [40.2 kB]
Get:20 http://security.ubuntu.com/ubuntu plucky-security/multiverse Translation-en [2,972 B]
Get:17 http://security.ubuntu.com/ubuntu plucky-security/main Translation-en [53.8 kB]
Get:18 http://security.ubuntu.com/ubuntu plucky-security/restricted Translation-en [42.6 kB]
Reading package lists... Done
E: The repository 'file:/cdrom plucky Release' no longer has a Release file.
N: Updating from such a repository can't be done securely, and is therefore disabled by default.
N: See apt-secure(8) manpage for repository creation and user configuration details.
N: Some sources can be modernized. Run 'apt modernize-sources' to do so.
raaj@ubuntu25: $ sudo apt-get install python3 python3-pip -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.13.3-1).
python3 set to manually installed.
Solving dependencies... Done
The following additional packages will be installed:
  build-essential bzip2 dpkg-dev fakeroot g++ g++-14 g++-14-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-14 gcc-14-x86-64-linux-gnu
  gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libcc1-0
  libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libgcc-14-dev libhwasan0 libitm1 libjs-jquery libjs-sphinxdoc

```

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`”

```

raaj@ubuntu25:~$ sudo apt-get install python3 python3-pip -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3 is already the newest version (3.13.3-1).
python3 set to manually installed.
Solving dependencies... Done
The following additional packages will be installed:
  build-essential bzip2 dpkg-dev fakeroot g++-14 g++-14-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-14 gcc-14-x86-64-linux-gnu
  gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libgcc1-0
  libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libgcc-14-dev libhwasan0 libitm1 libjs-jquery libjs-sphinxdoc
  libjs-underscore liblsan0 libpython3-dev libpython3.13-dev libquadmath0 libstdc++-14-dev libtsan2 libubsan1 lto-disabled-list make
  python3-dev python3-wheel python3.13-dev zlibg-dev
Suggested packages:
  bzip2-doc debian-keyring g++-multilib g++-14-multilib gcc-14-doc gcc-multilib autoconf automake libtool flex gcc-doc
  gcc-14-multilib gcc-14-locales gdb-x86-64-linux-gnu apache2 | lighttpd git bzip2 libstdc++-14-doc make-doc python3-setuptools
The following NEW packages will be installed:
  build-essential bzip2 dpkg-dev fakeroot g++-14 g++-14-x86-64-linux-gnu g++-x86-64-linux-gnu gcc gcc-14 gcc-14-x86-64-linux-gnu
  gcc-x86-64-linux-gnu javascript-common libalgorithm-diff-perl libalgorithm-diff-xs-perl libalgorithm-merge-perl libasan8 libgcc1-0
  libdpkg-perl libexpat1-dev libfakeroot libfile-fcntllock-perl libgcc-14-dev libhwasan0 libitm1 libjs-jquery libjs-sphinxdoc
  libjs-underscore liblsan0 libpython3-dev libpython3.13-dev libquadmath0 libstdc++-14-dev libtsan2 libubsan1 lto-disabled-list make
  python3-dev python3-pip python3-wheel python3.13-dev zlibg-dev
0 upgraded, 42 newly installed, 0 to remove and 9 not upgraded.
Need to get 24.7 MB/63.8 MB of archives.
After this operation, 231 MB of additional disk space will be used.
Ign:1 file:/cdrom/plucky/main amd64 libgcc1-0 amd64 15-20250404.0ubuntu1
Ign:2 file:/cdrom/plucky/main amd64 gcc-14-x86-64-linux-gnu amd64 14.2.0-19ubuntu2
Ign:3 file:/cdrom/plucky/main amd64 gcc-14 amd64 14.2.0-19ubuntu1
Ign:4 file:/cdrom/plucky/main amd64 gcc-x86-64-linux-gnu amd64 4:14.2.0-1ubuntu1
Ign:5 file:/cdrom/plucky/main amd64 gcc amd64 4:14.2.0-1ubuntu1
Ign:6 file:/cdrom/plucky/main amd64 g++-14-x86-64-linux-gnu amd64 14.2.0-19ubuntu2
Ign:7 file:/cdrom/plucky/main amd64 g++-14 amd64 14.2.0-19ubuntu2
Ign:8 file:/cdrom/plucky/main amd64 g++-x86-64-linux-gnu amd64 4:14.2.0-1ubuntu1
Ign:9 file:/cdrom/plucky/main amd64 g++ amd64 4:14.2.0-1ubuntu1
Ign:10 file:/cdrom/plucky/main amd64 make amd64 4.4.1-1
Ign:11 file:/cdrom/plucky/main amd64 libdpkg-perl all 1.22.18ubuntu2
Ign:12 file:/cdrom/plucky/main amd64 bzip2 amd64 1.0.8-6
Ign:13 file:/cdrom/plucky/main amd64 lto-disabled-list all 57
Ign:14 file:/cdrom/plucky/main amd64 dpkg-dev all 1.22.18ubuntu2
Ign:15 file:/cdrom/plucky/main amd64 build-essential amd64 12.12ubuntu1
Ign:16 file:/cdrom/plucky/main amd64 libfakeroot amd64 1.37.1-1

```

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.

```

Processing triggers for man-db (2.13.0-1) ...
Processing triggers for libc-bin (2.41-6ubuntu1.1) ...
raaj@ubuntu25:~$ python3 --version
Python 3.13.3
raaj@ubuntu25:~$ pip3 --version
pip 25.0 from /usr/lib/python3/dist-packages/pip (python 3.13)
raaj@ubuntu25:~$ 

```

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`”.

```

raaj@ubuntu25:~$ sudo apt-get install default-jdk -y
[sudo] password for raaj:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Solving dependencies... Done
The following additional packages will be installed:
  ca-certificates-java default-jdk-headless default-jre default-jre-headless fonts-dejavu-extra java-common libatk-wrapper-java libatk-wrapper-java-jni
  libice-dev libsm-dev libxi1-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-21-jdk openjdk-21-jdk-headless openjdk-21-jre openjdk-21-jre-headless
  uuid-dev x1proto-dev xorg-sgml-doctools xtrans-dev
Suggested packages:
  libbice-doc libsm-doc libxi1-doc libxcb-doc libxt-doc openjdk-21-demo openjdk-21-source visualvm fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
  | fonts-wqy-zenhei fonts-indic
The following NEW packages will be installed:
  ca-certificates-java default-jdk default-jdk-headless default-jre default-jre-headless fonts-dejavu-extra java-common libatk-wrapper-java
  libatk-wrapper-java-jni libice-dev libsm-dev libxi1-dev libxau-dev libxcb1-dev libxdmcp-dev libxt-dev openjdk-21-jdk openjdk-21-jdk-headless openjdk-21-jre
  openjdk-21-jre-headless uuid-dev x1proto-dev xorg-sgml-doctools xtrans-dev
0 upgraded, 24 newly installed, 0 to remove and 9 not upgraded.
Need to get 135 MB of archives.
After this operation, 316 MB of additional disk space will be used.
Ign:1 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 ca-certificates-java all 20240118
Get:2 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 java-common all 0.76 [6,852 B]
Get:3 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 openjdk-21-jre-headless amd64 21.0.8+9~us1~0ubuntu1~25.04.1 [46.5 MB]
Get:4 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 default-jdk-headless amd64 2:1.21-76 [3,178 B]
Get:5 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 openjdk-21-jre amd64 21.0.8+9~us1~0ubuntu1~25.04.1 [220 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 default-jre amd64 2:1.21-76 [918 B]
Get:7 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 openjdk-21-jdk-headless amd64 21.0.8+9~us1~0ubuntu1~25.04.1 [82.8 MB]
81% [7 openjdk-21-jdk-headless 82.5 MB/82.8 MB 100%] 522 kB/
Get:8 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 default-jdk-headless amd64 2:1.21-76 [966 B]
Get:9 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 openjdk-21-jdk amd64 21.0.8+9~us1~0ubuntu1~25.04.1 [1,664 kB]
Get:10 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 default-jdk amd64 2:1.21-76 [922 B]
Get:11 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 fonts-dejavu-extra all 2.37-8 [1,947 kB]
Get:12 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libatk-wrapper-java all 0.40.0-3build2 [54.3 kB]
Get:13 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libatk-wrapper-java-jni amd64 0.40.0-3build2 [46.4 kB]
Get:14 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 xorg-sgml-doctools all 1:1.11-1.1 [10.9 kB]
Get:15 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 x1proto-dev all 2024.1-1 [606 kB]
Get:16 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libice-dev amd64 2:1.1.1-1 [54.6 kB]
Get:17 http://us.archive.ubuntu.com/ubuntu plucky-updates/main amd64 uuid-dev amd64 2.40.2-14ubuntu1.1 [54.7 kB]
Get:18 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libsm-dev amd64 2:1.2.4-1 [19.2 kB]
Get:19 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libxau-dev amd64 1:1.0.11-1 [9,798 B]
Get:20 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 libxdmcp-dev amd64 1:1.1.5-1 [26.7 kB]
Get:21 http://us.archive.ubuntu.com/ubuntu plucky/main amd64 xtrans-dev all 1.4.0-1 [68.9 kB]

```

Figure 19 – Installing java with the command

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

```

Setting up default-jre (2:1.21-76) ...
Setting up openjdk-21-jdk:amd64 (21.0.8+9~us1~0ubuntu1~25.04.1) ...
update-alternatives: using /usr/lib/jvm/java-21-openjdk-amd64/bin/jconsole to provide /usr/bin/jconsole (jconsole) in auto mode
Setting up default-jdk-headless (2:1.21-76) ...
Setting up default-jdk (2:1.21-76) ...
raaj@ubuntu25:~$ java --version
openjdk 21.0.8 2025-07-15
OpenJDK Runtime Environment (build 21.0.8+9-Ubuntu-0ubuntu125.04.1)
OpenJDK 64-Bit Server VM (build 21.0.8+9-Ubuntu-0ubuntu125.04.1, mixed mode, sharing)
raaj@ubuntu25:~$ 

```

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”.

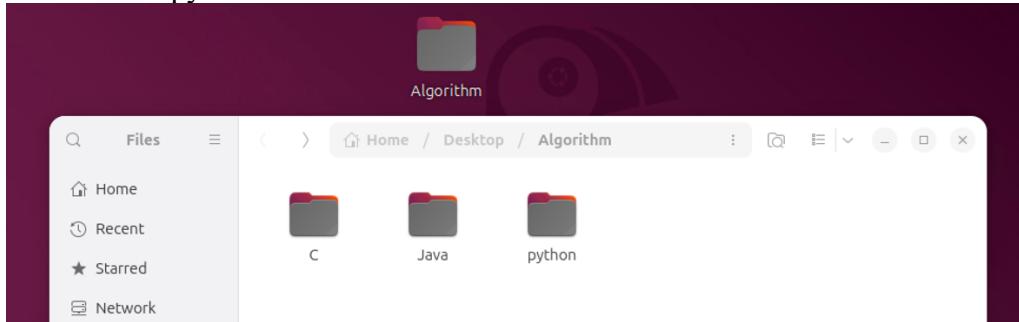


Figure 22 - Screenshot of create folders

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.

The screenshot shows three separate code editors side-by-side:

- stack.c**: A C program for a stack implementation. It includes headers for stdio.h and stdlib.h, defines MAX as 10, and creates a stack structure with items and top pointers. Functions include createEmptyStack, isfull, isempty, push, pop, and printStack.
- Algorithm**: A C program for a stack algorithm. It includes a main function that creates an empty stack, pushes elements 1 through 4, prints the stack, pops elements, and prints the stack again after popping.
- Algorithm**: A C program for a stack algorithm. It includes a main function that creates an empty stack, pushes elements 1 through 4, prints the stack, pops elements, and prints the stack again after popping.

Figure 23 – Screenshot of code for stack in C

The screenshot shows three separate code editors side-by-side:

- queue.c**: A C program for a queue implementation. It includes headers for stdio.h, defines SIZE as 5, and creates a queue structure with items and front/rear pointers. Functions include enqueue, dequeue, and display.
- queue.c**: A C program for a queue implementation. It includes headers for stdio.h, defines SIZE as 5, and creates a queue structure with items and front/rear pointers. Functions include enqueue, dequeue, and display.
- queue.c**: A C program for a queue implementation. It includes headers for stdio.h, defines SIZE as 5, and creates a queue structure with items and front/rear pointers. Functions include enqueue, dequeue, and display.

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..

The screenshot shows a terminal window with three tabs open, all titled 'raaj@ubuntu25: ~/Desktop/Algorithm/C'. The tabs are represented by small windows at the top of the main terminal area. The terminal output is as follows:

```
raaj@ubuntu25: ~/Desktop/Algorithm/python x      raaj@ubuntu25: ~/Desktop/Algorithm/Java x      raaj@ubuntu25: ~/Desktop/Algorithm/C x
raaj@ubuntu25:~/Desktop/Algorithm/Java$ cd ..
raaj@ubuntu25:~/Desktop/Algorithm$ cd C
raaj@ubuntu25:~/Desktop/Algorithm/C$ gcc stack.c -o stack
raaj@ubuntu25:~/Desktop/Algorithm/C$ ./stack
Stack: 1 2 3 4
Item popped= 4

After popping out
Stack: 1 2 3
raaj@ubuntu25:~/Desktop/Algorithm/C$ gcc queue.c -o queue
raaj@ubuntu25:~/Desktop/Algorithm/C$ ./queue

Queue is Empty!!
Inserted -> 1
Inserted -> 2
Inserted -> 3
Inserted -> 4
Inserted -> 5
Queue is Full!!
Queue elements are:
1 2 3 4 5

Deleted : 1
Queue elements are:
2 3 4 5
raaj@ubuntu25:~/Desktop/Algorithm/C$
```

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()`.

Stack.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");
            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;
    }
}

```

StackTest.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

Queue.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;
        else
            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");
        else {
            element = items[front];
            if (front >= rear)
                front = -1;
            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);
        }
    }
}

```

QueueTest.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.
    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.

```

raaj@ubuntu25: ~/Desktop/Algorithm/python  ×      raaj@ubuntu25: ~/Desktop/Algorithm/Java  ×      ▾
raaj@ubuntu25:~/Desktop/Algorithm/python$ cd ..
raaj@ubuntu25:~/Desktop/Algorithm$ cd Java/
raaj@ubuntu25:~/Desktop/Algorithm/Java$ javac stack.java
raaj@ubuntu25:~/Desktop/Algorithm/Java$ java Stack
Inserting 1
Inserting 2
Inserting 3
Inserting 4

After popping out
1
2
3

```

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



```

stack.py
-/Desktop/Algorithm/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))

```

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.

The image shows two side-by-side code editors. The left editor contains the `queue.py` script, which defines a Queue class with methods for enqueueing and dequeuing elements. The right editor contains the `stack.py` script, which defines a Stack class with methods for pushing and popping elements. Both scripts include examples of how to use their respective classes.

```

# 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

The image shows a terminal window with a dark background. It displays the execution of two Python scripts. First, `stack.py` is run, showing a stack of numbers being pushed and popped. Then, `queue.py` is run, demonstrating a queue's FIFO behavior.

```

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

```

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()

```