Operating System Lab
PC/CSE/61-P

Operating Systems Lab. (UNIX/LINUX) General Course Information

General Course Information

Course Code: PC/CSE/61-P Course Credits: 1 Type: Professional Core Lab. Course Contact Hours: 2 hours/week Mode: Lab practice and assignments	**Course Assessment Methods (internal: 50; external: 50)** The internal and external assessment is based on the level of participation in lab. sessions and the timely submission of lab experiments/assignments, the quality of solutions designed for the assignments, the performance in VIVA- VOCE, the quality of lab. file and ethical practices followed. The internal examination is conducted by the course coordinator. The external examination is conducted by external examiner appointed by the Controller of Examination in association with the internal examiner appointed by the Chairperson of the Department.

Pre-requisites: Basic programming skills.

About the Course:

This lab. course on data science involves a rigorous training on R programming. It incorporates solving problems related to data science in statistical and predictive modelling framework. The objective of the lab course is to equip the students to solve the practical data science problems related to intelligent data analysis using R.

Course Outcomes: By the end of the course students will be able to:

CO1. **apply** commands related to vi and Emacs editors, general utilities and file systems.

(LOTS: Level 3: **Apply**)

CO2. **write** basic shell scripts and use sed commands as well as awk programming.

(LOTS: Level 3: **Apply**)

CO3. **analyse** the results of memory management and disk management commands.

(LOTS: Level 4: **Analyse**)

CO4. **evaluate** solutions for different operating system problems such as scheduling, memory management and file management. (LOTS: Level 5: **Evaluate**)

CO5. create lab record for assignments that includes problem definitions, design of solutions and conclusions. (LOTS: Level 6: **Create**)

CO6. demonstrate use of ethical practices, self-learning and team spirit.

(LOTS: Level 3: **Apply**)

List of experiments/assignments:

- Study of WINDOWS and Linux operating system (Linux kernel, shell, basic commands pipe & filter commands).
- Study vi editor.
- Administration of LINUX Operating System.
- Writing of Shell Scripts (Shell programming).
- AWK programming.
- Write a C program to simulate different scheduling algorithms
- Write a C program to simulate different file allocation strategies

Note: The actual experiments/assignments will be designed by the course coordinator. One assignment should be designed to be done in groups of two or three students. The assignments must meet the objective of the course and the levels of the given course outcomes. The list of assignments and schedule of submission will be prepared by the course coordinator at the beginning of the semester. CO-PO Articulation Matrix Operating System Lab. (PC/CSE/61-P)

Course Outcomes	PO 1	PO 2	PO 3	PO 4	PO 5	PO 6	PO 7	PO 8	PO 9	PO 10	PO 1 1	PO 12	PSO 13	PSO 14	PSO 15
CO1	1	_	_	_	_	_	_	_	_	_	_	_	1	_	_
CO2	2	_	_	_	_	_	_	_	_	_	_	_	3	_	_
CO3	2	2		2	2	_	_	_	_	_	_	_	3	_	_
CO4	2	2		2	2	_	_	_	_	_	_	_	3	_	_
CO5	_	_	_	_	_	_	_	_	_	3	_	_	_	_	_
CO6	_	_	_	_	_	_	_	3	3	_	_	3	_	_	_
3-High 2-Medium 1-Low															

PROGRAM 1 : Study windows operating system.

Introduction to Operating System
An operating system (OS) is system software that acts as an intermediary between the user and the computer hardware. It manages hardware resources, provides a platform for application execution, and ensures that programs run efficiently and securely. The OS kernel remains permanently loaded in main memory and handles core operations such as process scheduling, memory management, and I/O control.

Windows Operating System
Windows is a graphical operating system developed by Microsoft Corporation. It provides users with a visual interface to manage files, run applications, access the internet, and configure system settings. Microsoft first introduced Windows on November 10, 1983, and has since released numerous versions catering to both home and enterprise environments.

Key milestones in Windows evolution:
- **Windows NT 3.1 (1993)** – First business-oriented version of Windows
- **Windows XP (2001)** – Unified home and professional editions on a standard x86 architecture
- **Windows 10 (2015)** – Introduced a universal platform model
- **Windows 11 (2021)** – Latest release featuring a redesigned UI and enhanced security

Features of Windows
- Graphical User Interface (GUI) with taskbar, start menu, and window management
- Multitasking and multiprocessing support
- Built-in security through Windows Defender and BitLocker
- Regular system updates via Windows Update
- Extensive hardware and software compatibility
- Supports protected mode and supervisor mode for process isolation
- Memory management enabling multiprogramming

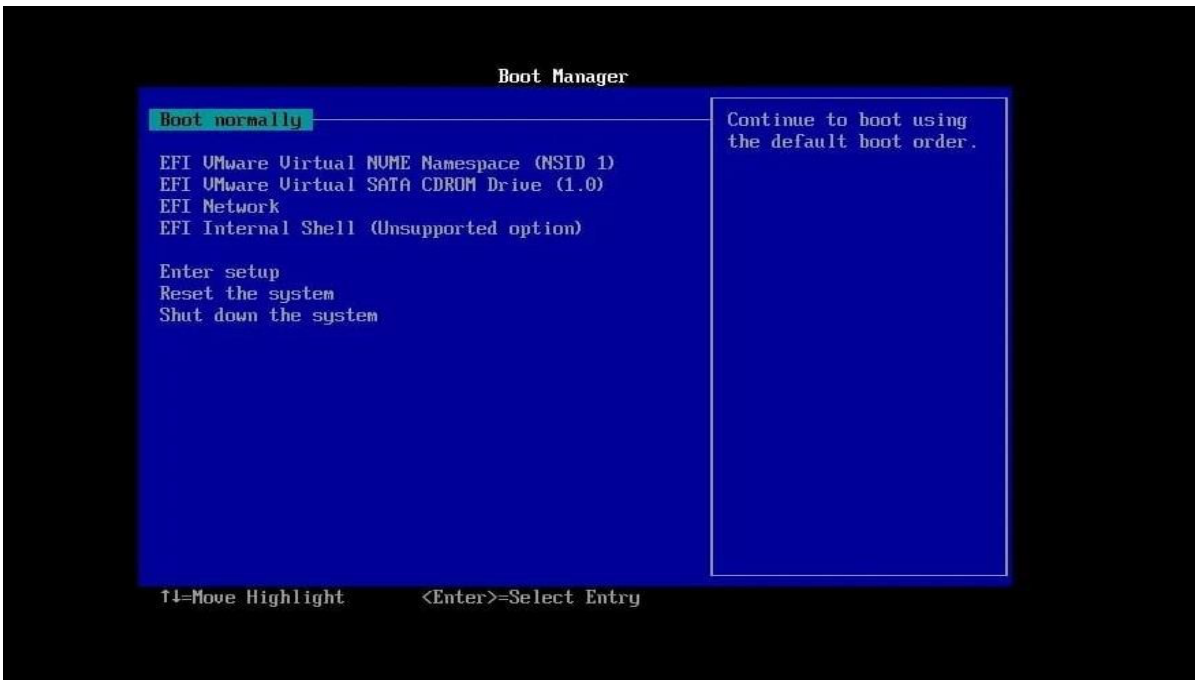
Hardware Requirements for Windows 11

Component	Minimum Requirement
Processor	1 GHz or faster (64-bit), compatible SoC
RAM	4 GB
Storage	64 GB or more
Graphics	DirectX 12 compatible, WDDM 2.0 driver
Display	720p, 9" or larger
Firmware	UEFI, Secure Boot capable
TPM	Trusted Platform Module version 2.0

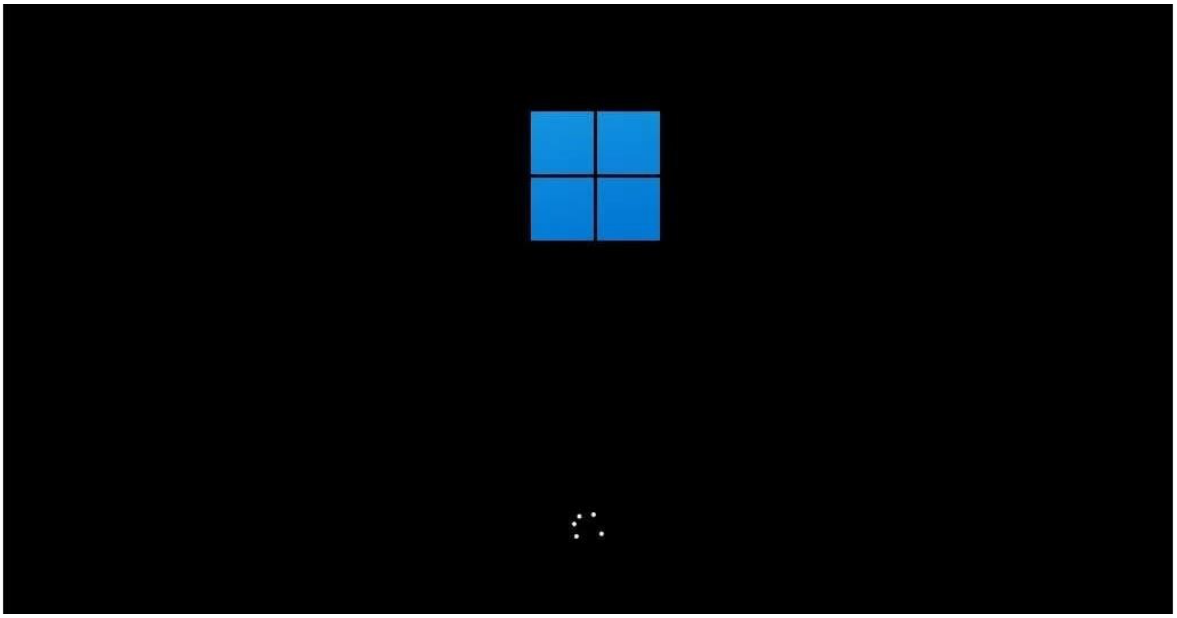
Step-by-Step Installation of Windows 11

I. Insert the bootable USB or DVD containing the Windows 11 installer and restart the system.

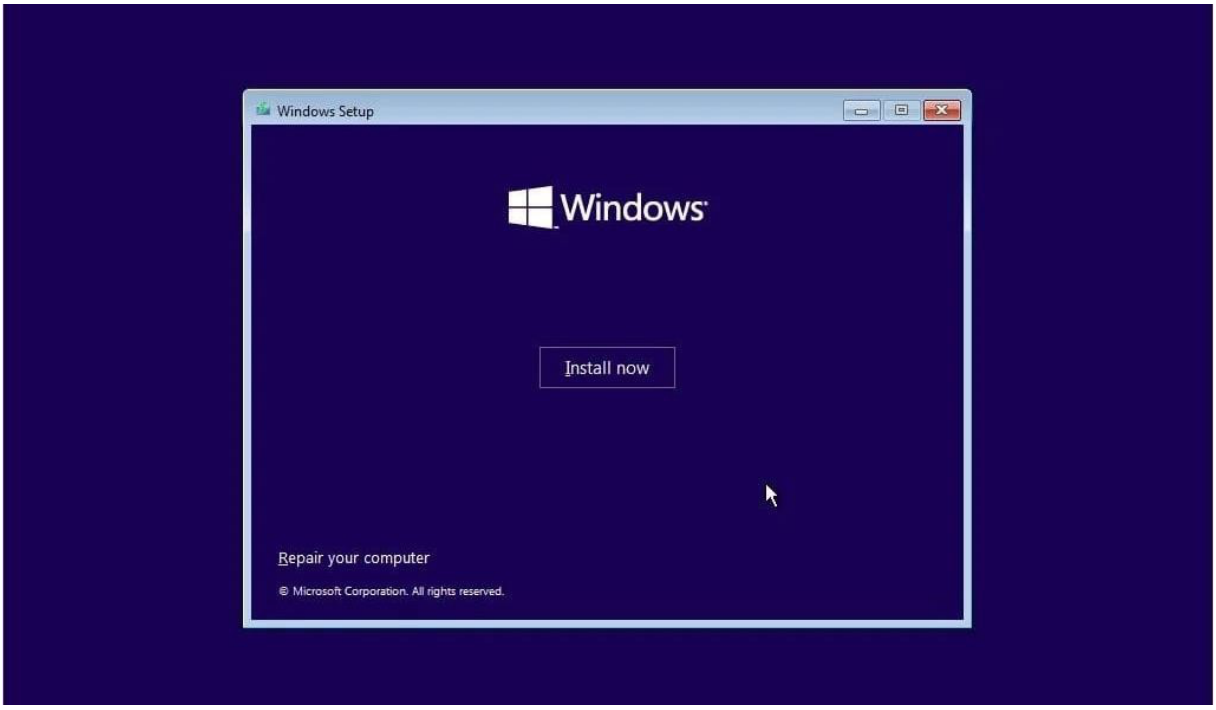
II. Press F8 (or the designated key) repeatedly during startup to open the Boot Manager. Select the installation drive — prefer the UEFI option for a modern boot setup.



III. When prompted with "Press any key to boot from CD or DVD," do so. The Windows 11 logo will appear indicating the installer has launched.

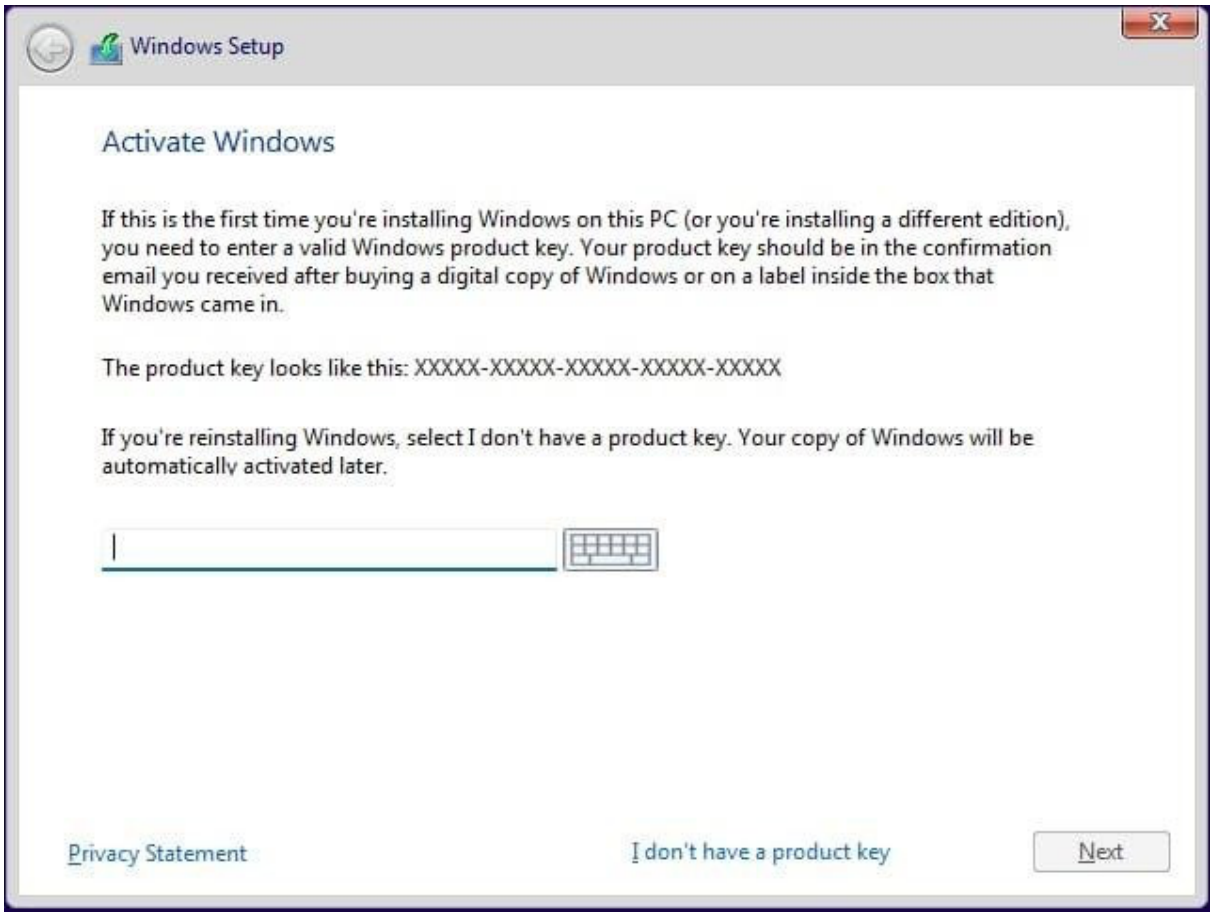


IV. Click **Install Now** on the setup screen.

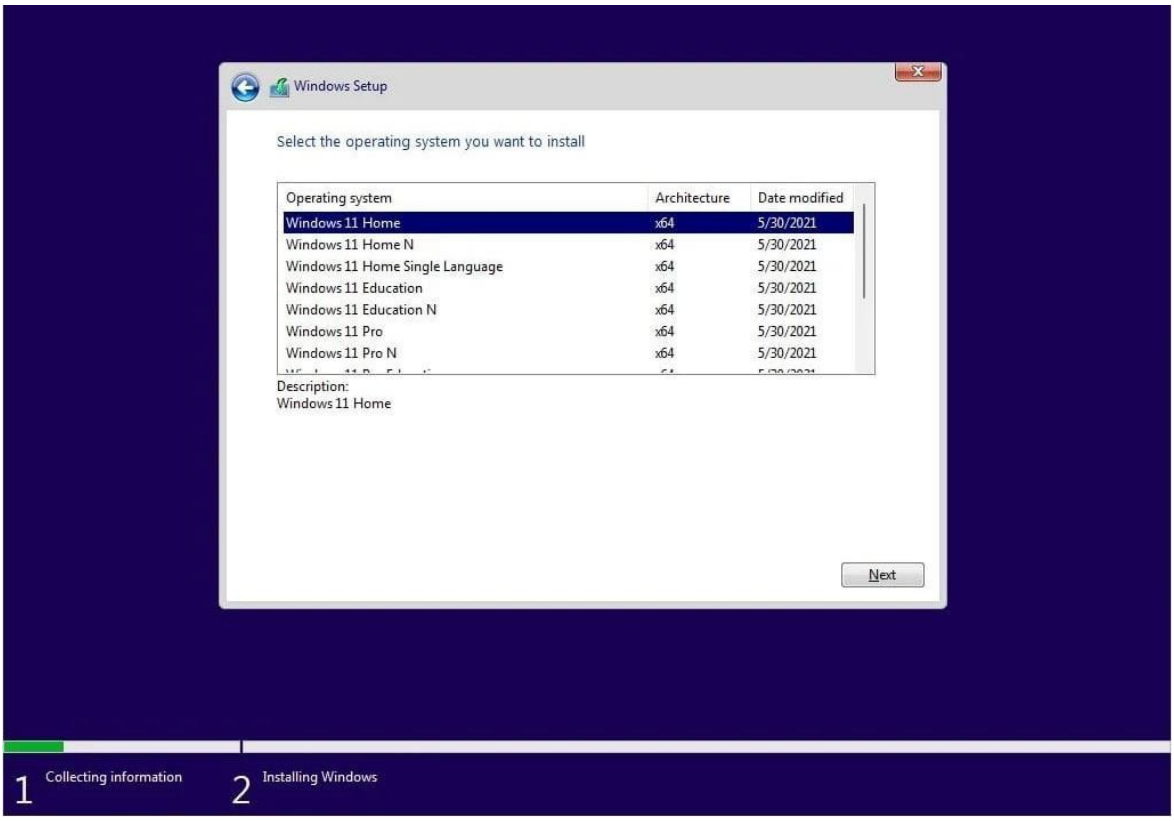


V. Choose your preferred language, time/currency format, and keyboard input method, then click **Continue**.

VI. Enter your product key when prompted. If unavailable, click **I don't have a product key** to activate later.



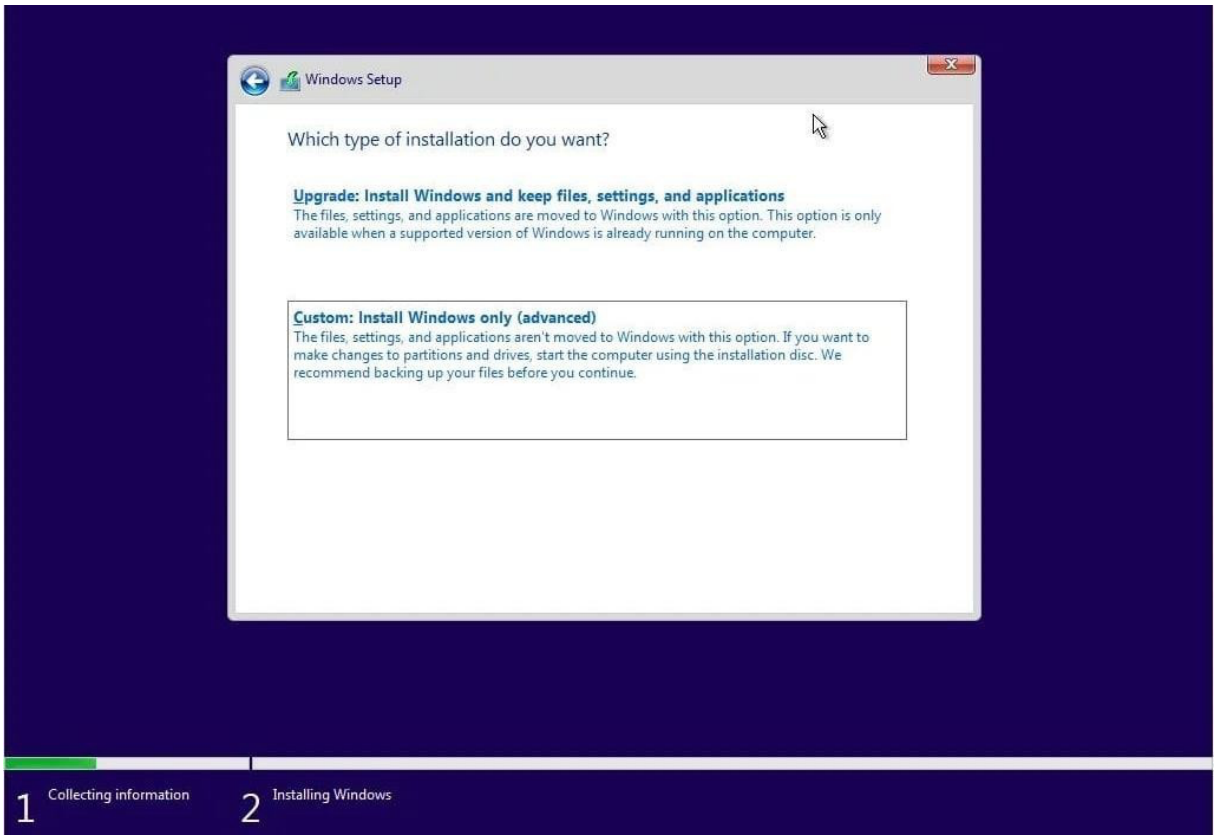
VII. Select the desired Windows 11 edition (Home, Pro, or Education) and accept the License Terms.



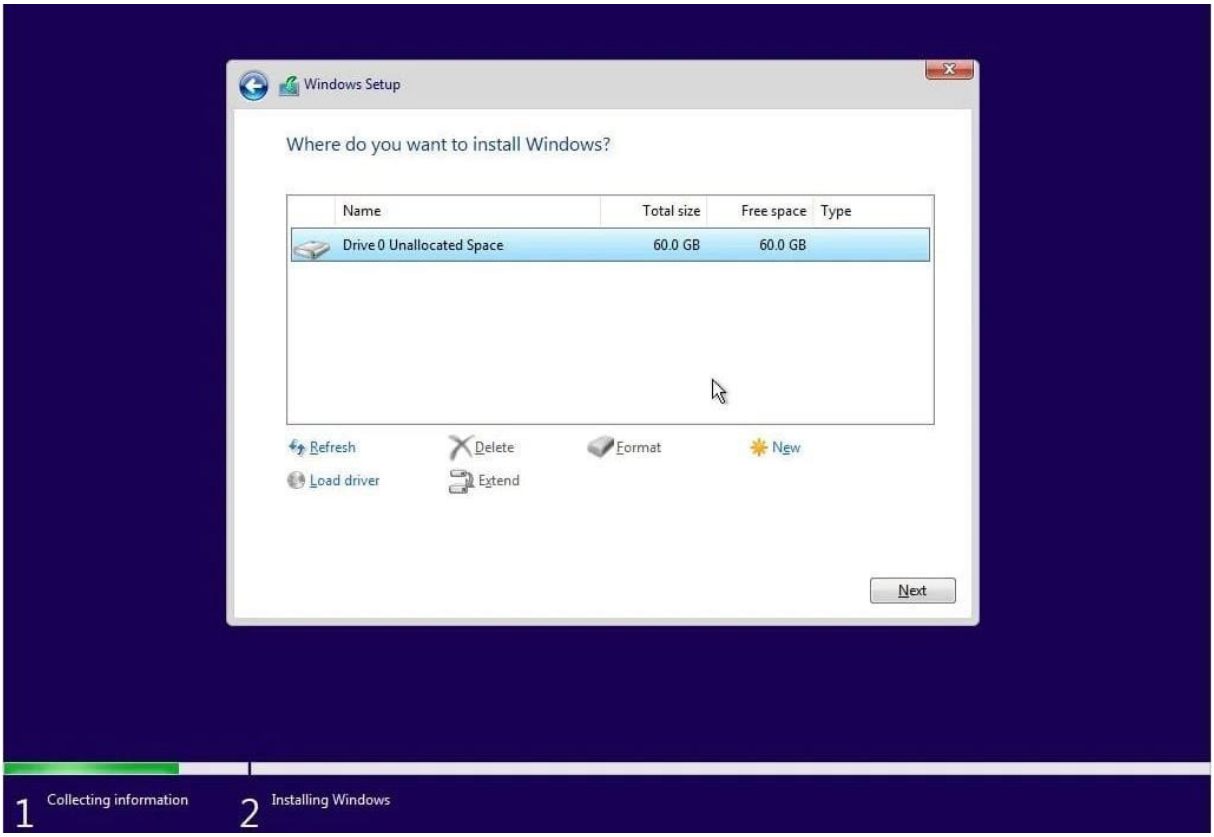
VIII. Choose the installation type:

a. **Upgrade** – Retains existing files, settings, and applications

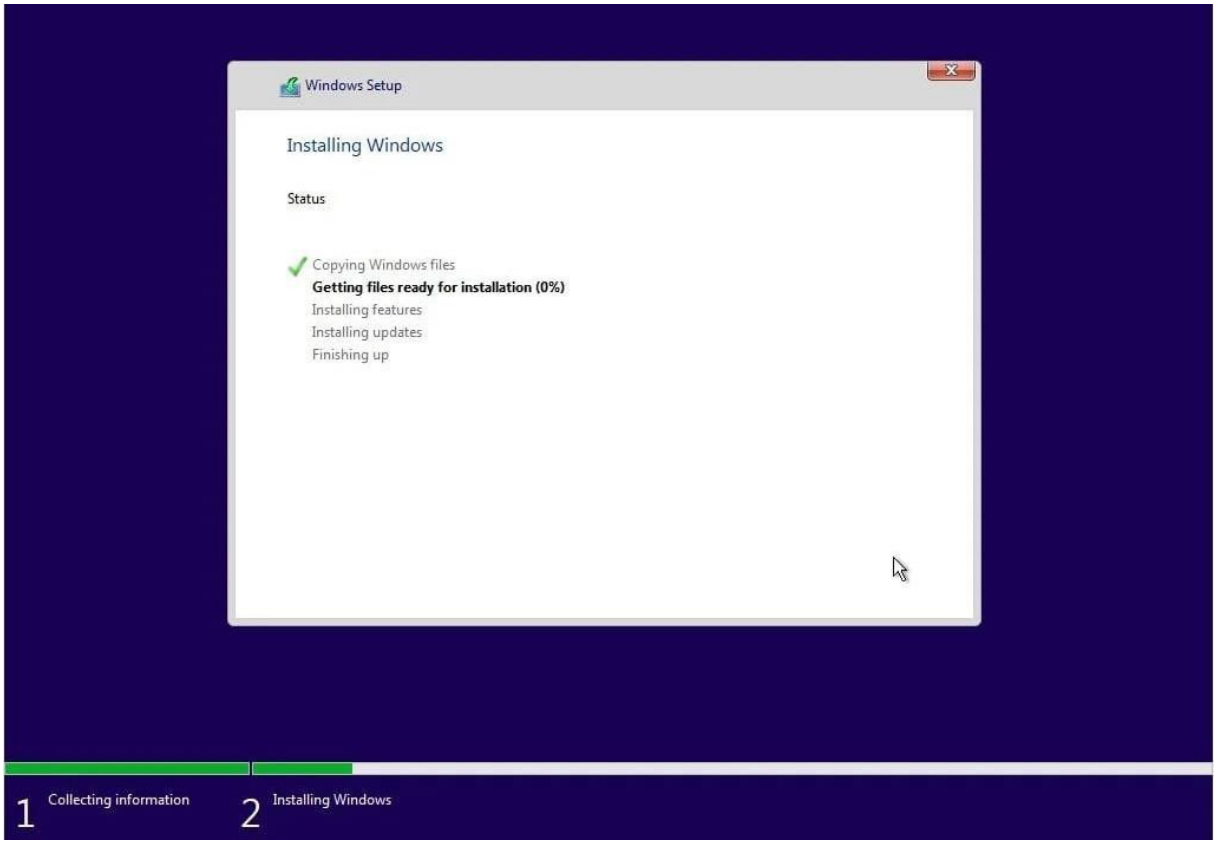
b. **Custom** – Clean installation; recommended for fresh setups



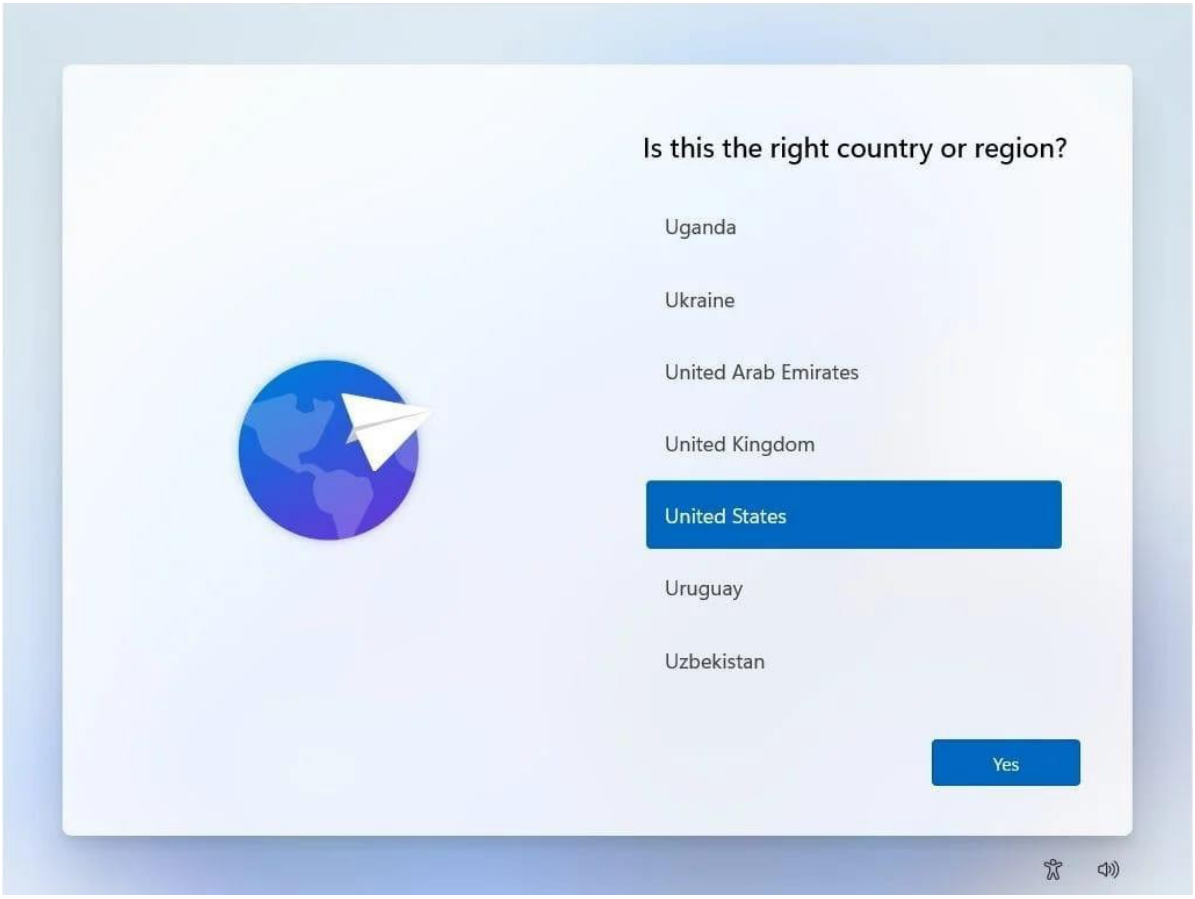
IX. Select the target disk/partition for installation and click **Next**.

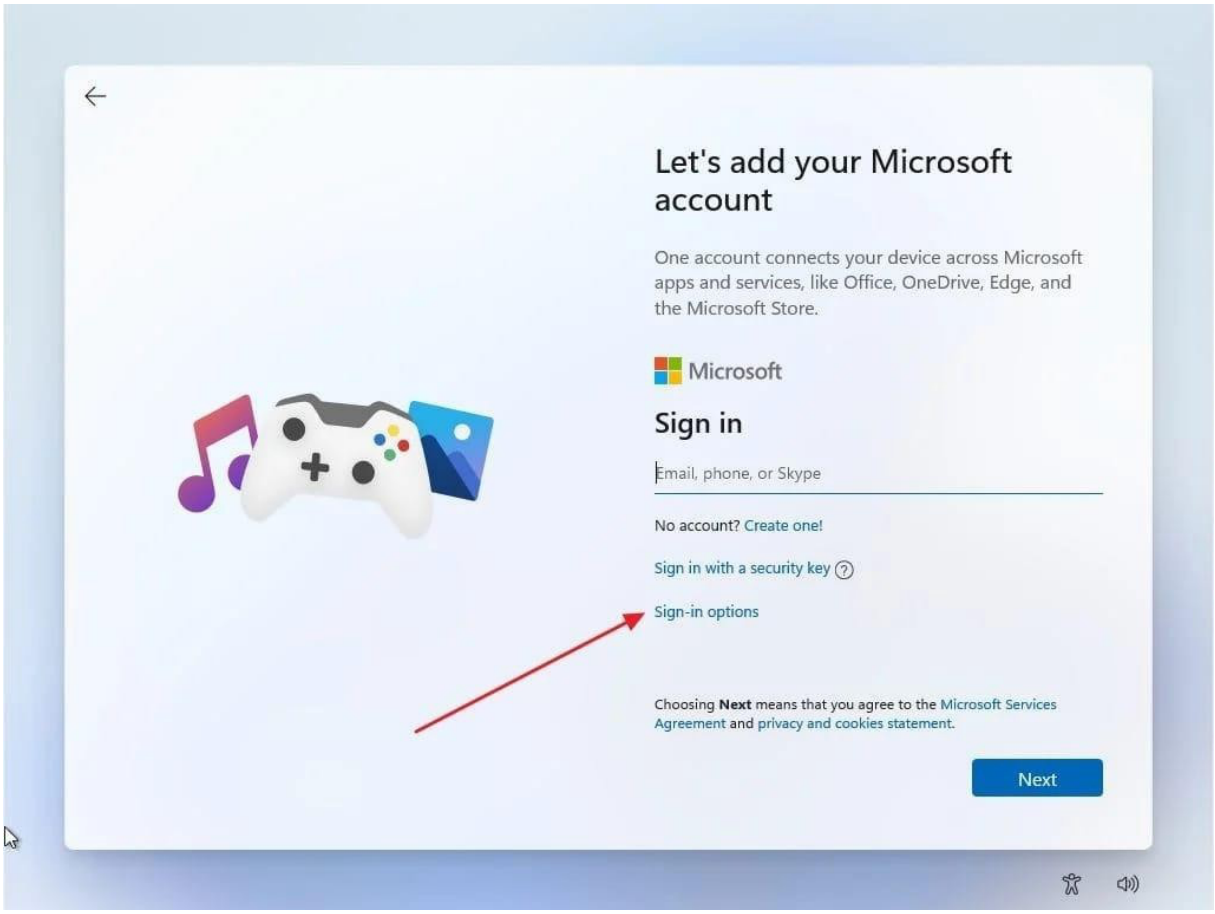


X. Wait for the installation to complete. The system will restart automatically.

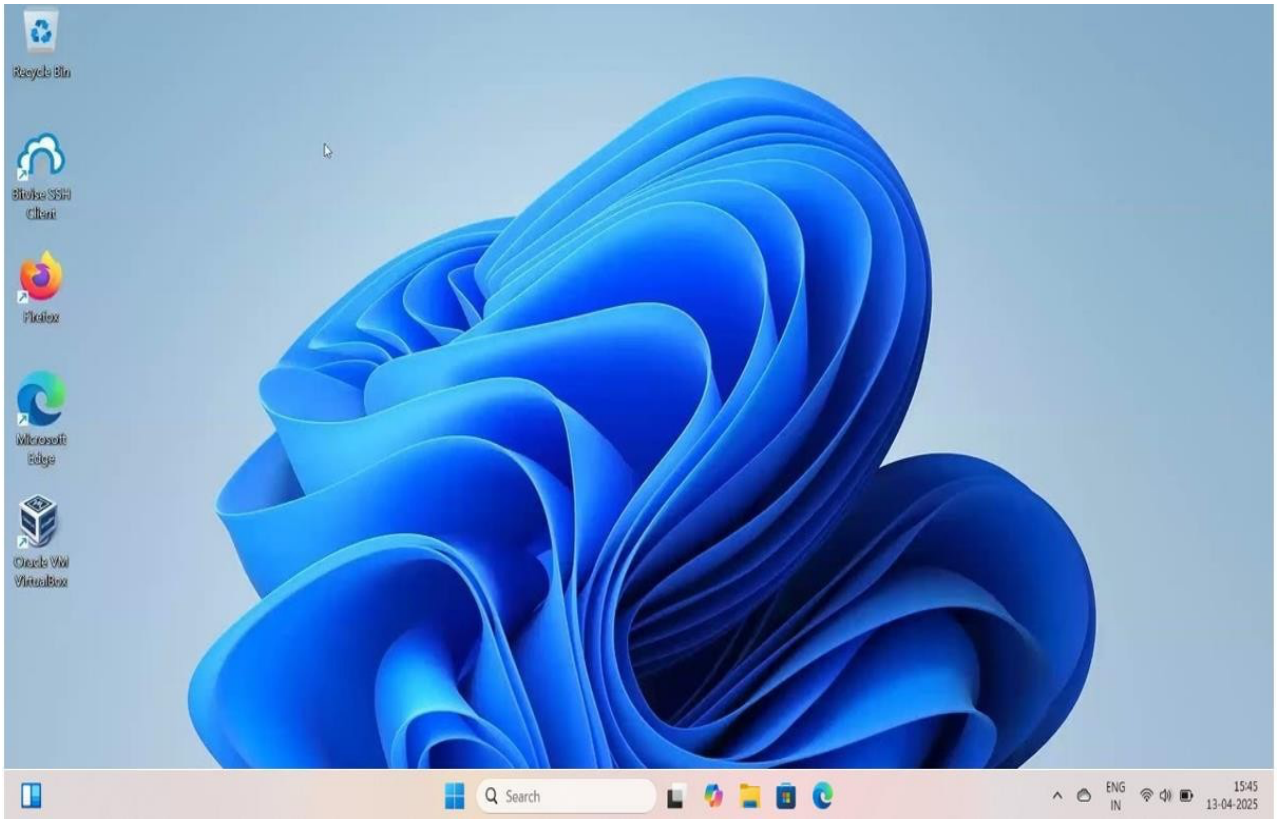


XI. During first boot, configure region, keyboard layout, and sign in with a Microsoft account.





XII. After final setup, the Windows 11 desktop will be ready for use.



PROGRAM 2 : Study of linux operating system.

Introduction to Linux
Linux is an open-source, Unix-like operating system kernel originally created by Linus Torvalds in 1991. It forms the foundation of numerous distributions (distros) used in desktops, servers, embedded systems, and supercomputers. Linux is known for its stability, security, and flexibility, and is freely available under the GNU General Public License (GPL).

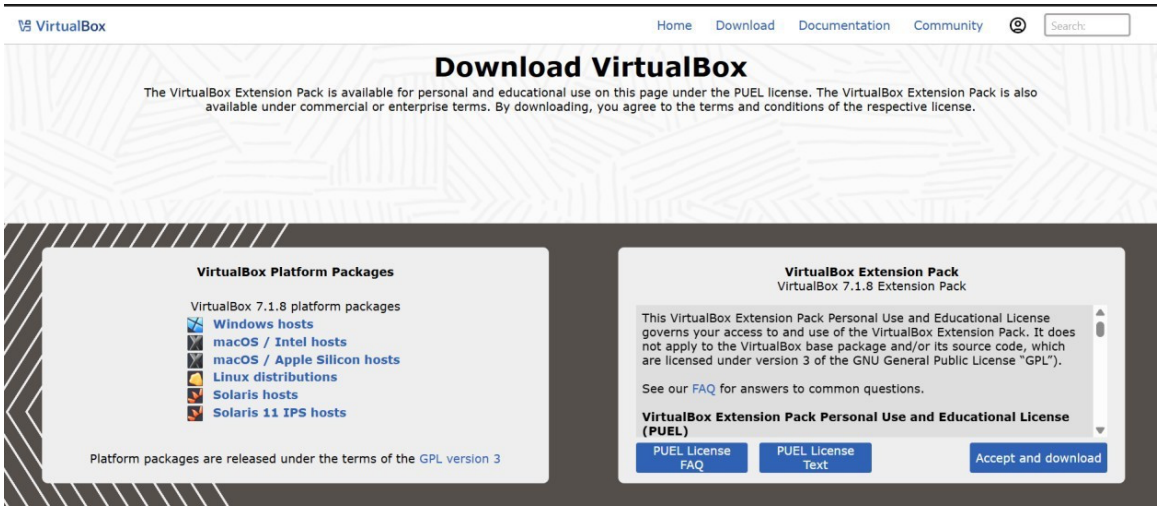
Ubuntu Linux
Ubuntu is one of the most widely used Linux distributions, developed and maintained by **Canonical Ltd.** It is based on Debian and targets both beginners and advanced users. First released in **October 2004**, Ubuntu derives its name from an African philosophy meaning "humanity to others," reflecting its community-driven open-source spirit.

Ubuntu ships with the **GNOME desktop environment** in its standard edition, offering a clean and modern GUI. For users preferring command-line access, the terminal remains fully functional. Long-Term Support (LTS) releases are issued every two years and receive five years of security and maintenance updates.

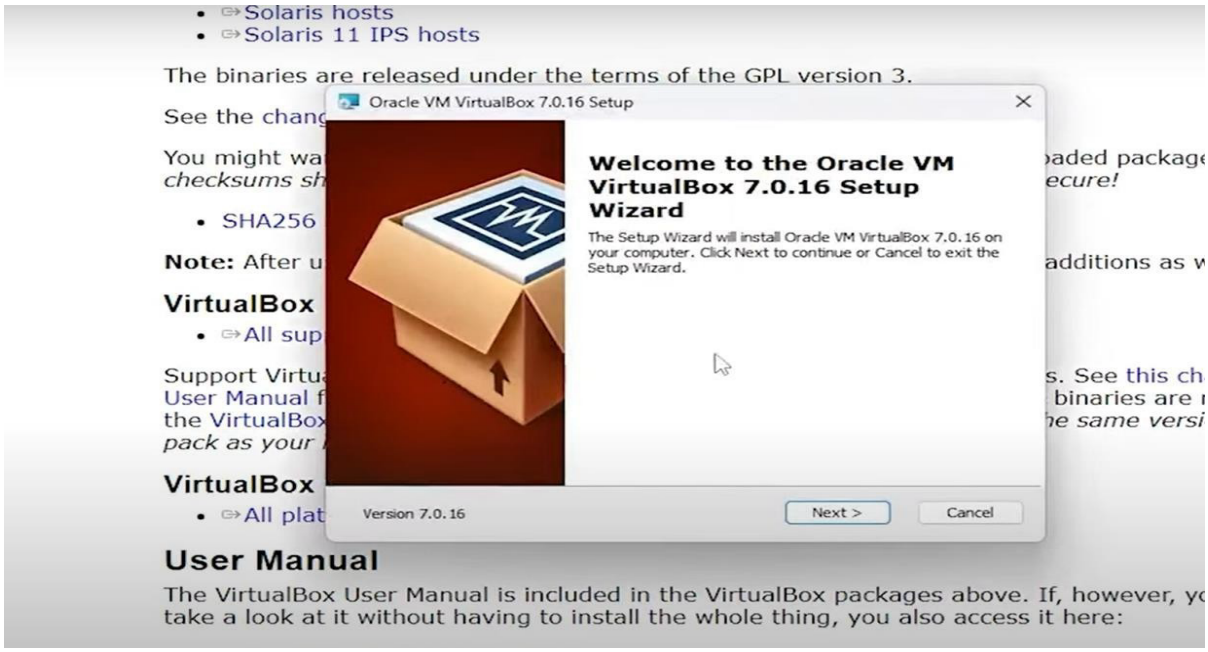
Features of Ubuntu
- Intuitive GNOME-based desktop interface
- Robust security with built-in UFW firewall and regular security patches
- APT and Snap package managers for easy software installation
- Wide hardware compatibility across modern systems
- Lightweight variants available: Xubuntu, Lubuntu, Ubuntu MATE
- Strong support for cloud, containerization (Docker, Kubernetes), and server deployments
- Regular LTS releases with long-term maintenance support

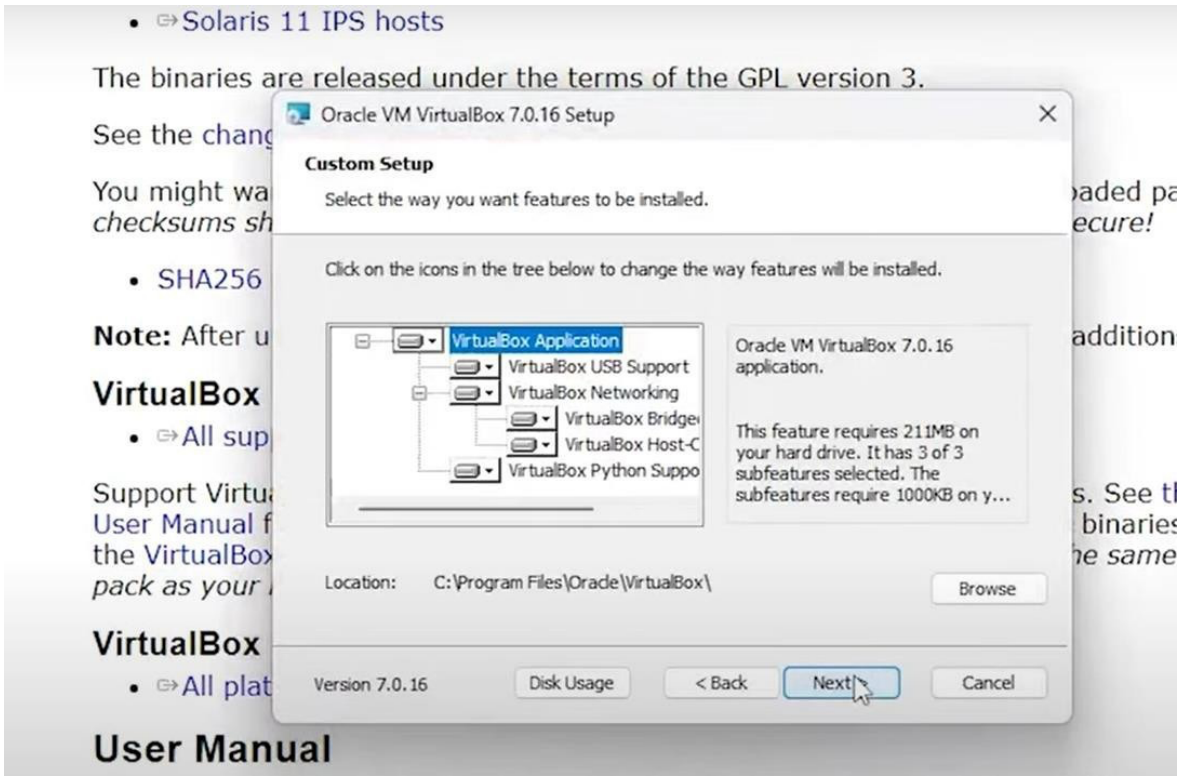
Step by Step Installation of Ubuntu on virtual box:

I. Download virtual box, Go to https://www.virtualbox.org/ and download VirtualBox for your operating system (Windows, macOS, or Linux).



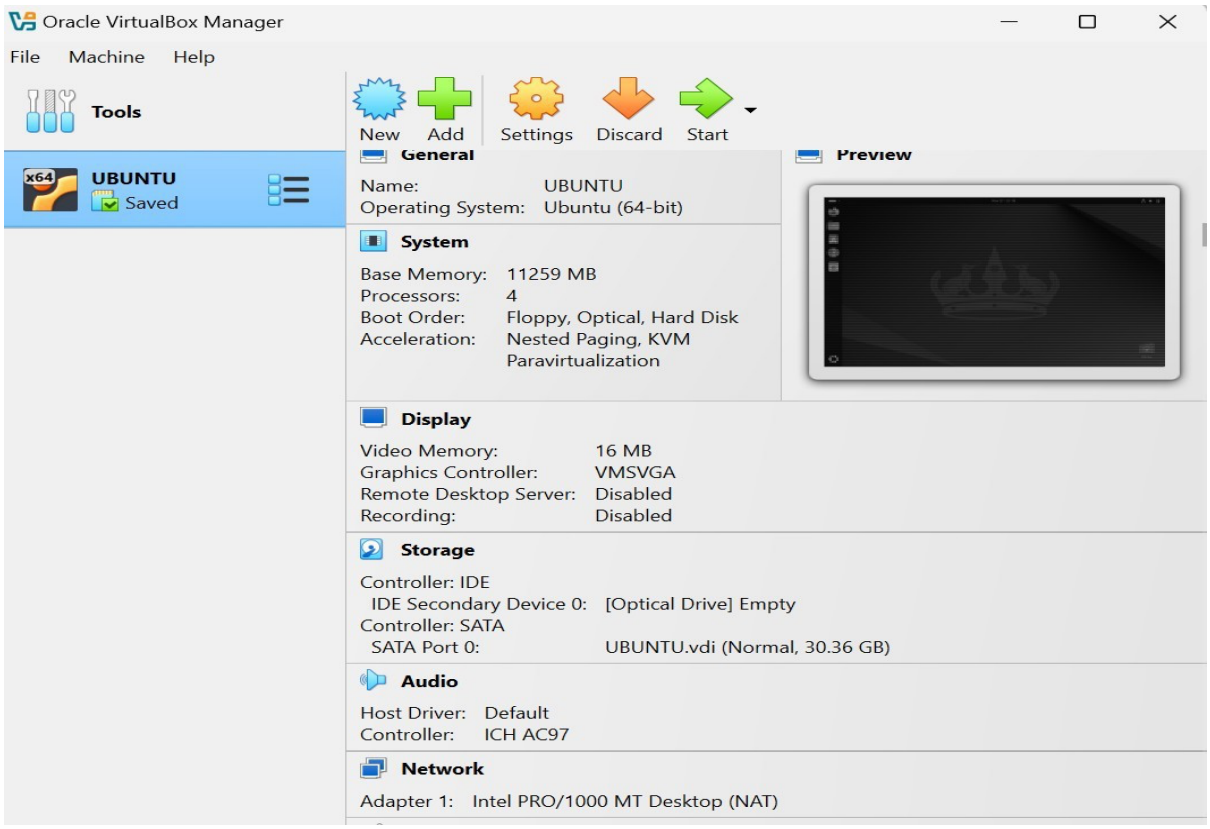
II. Install VirtualBox , Open the downloaded VirtualBox installer file.Follow the installation wizard. Click Next several times . Accept the default options. Click Install. Once installed, launch VirtualBox.



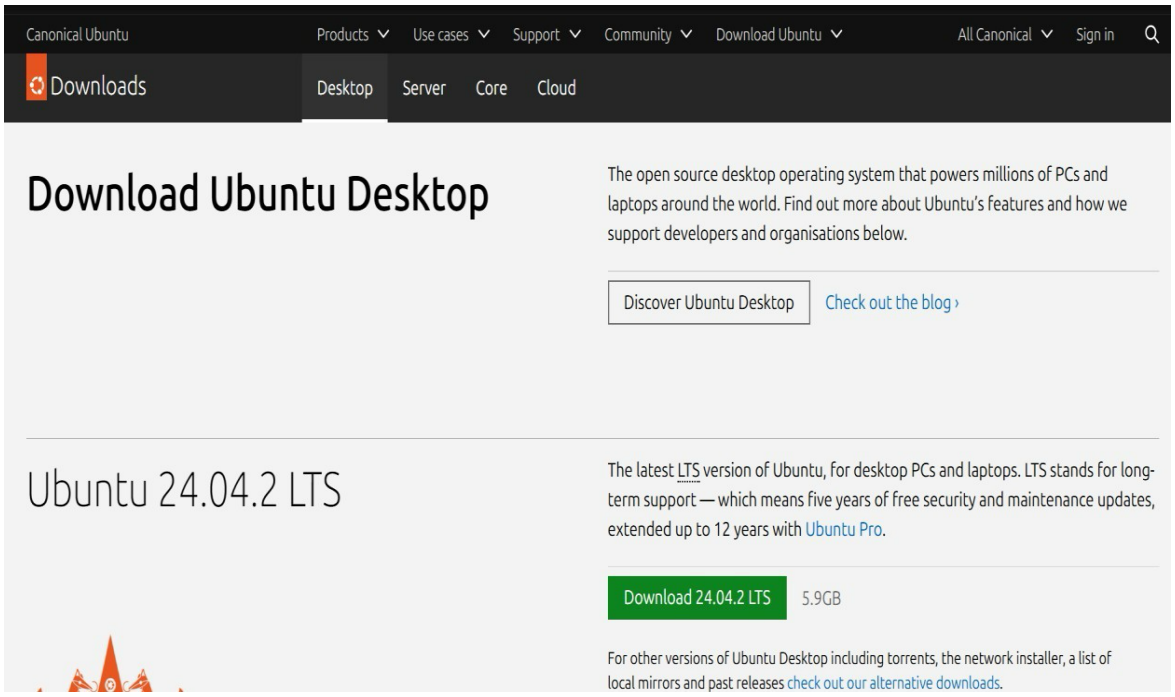


III. Create a New Virtual Machine – Click "New," enter a name (e.g., "Ubuntu"), set Type to Linux and Version to Ubuntu (64-bit), and allocate at least 3 GB RAM.

IV. Create a Virtual Hard Disk – Select "Create a virtual hard disk now," choose VDI format, and allocate at least 20 GB of space.

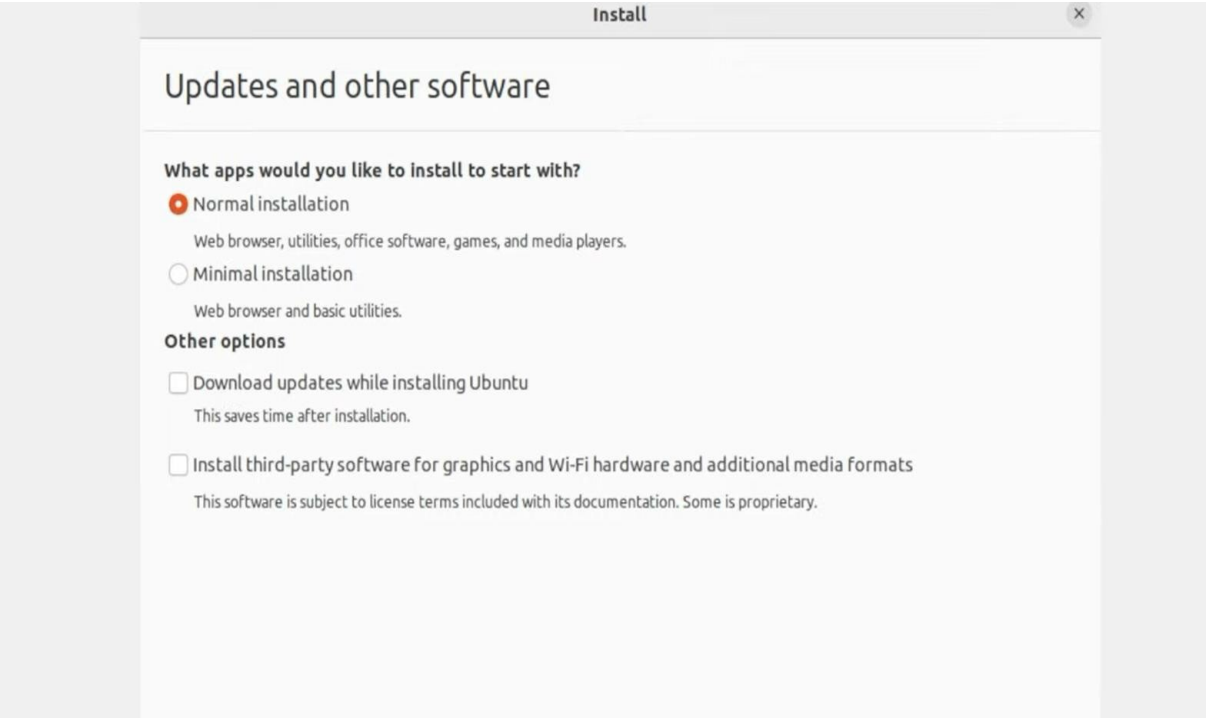


V. Download Ubuntu ISO – Visit https://ubuntu.com/download/desktop and download the latest LTS version.

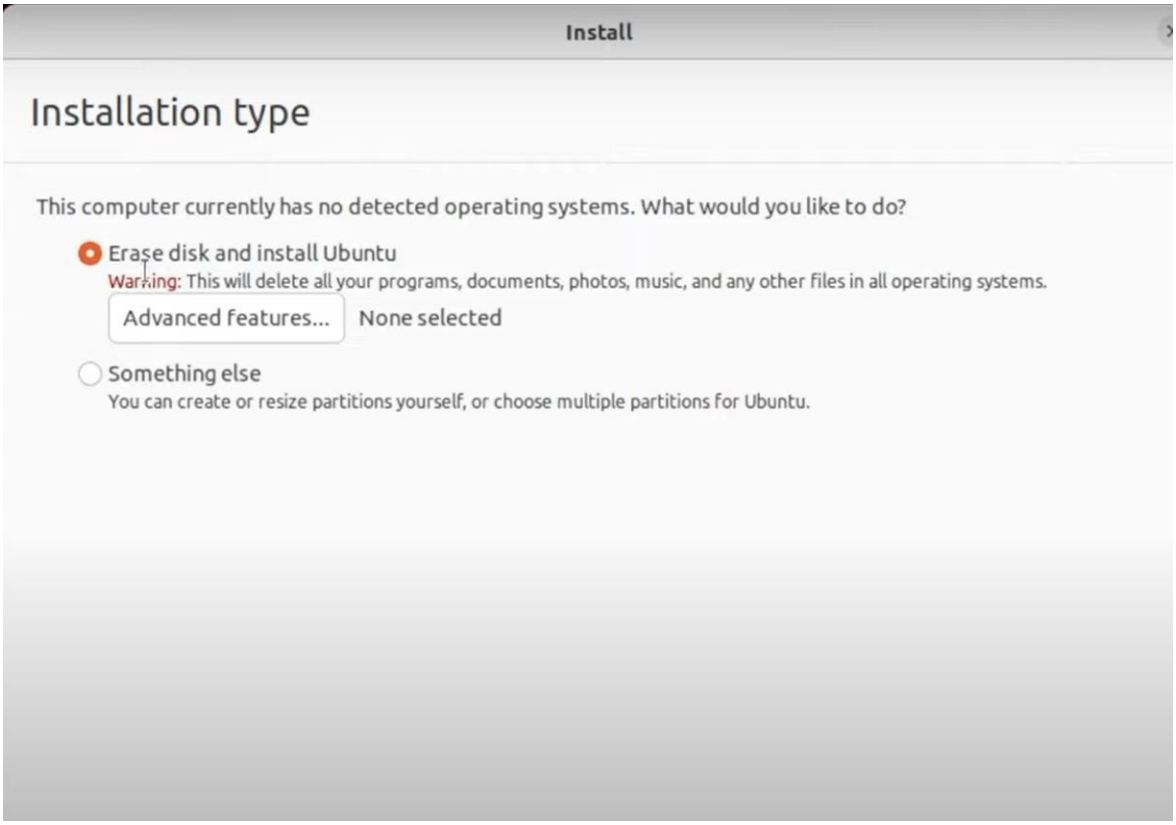


VI. Attach ISO to VM – In VirtualBox settings, go to Storage, click the empty optical drive, and select the downloaded Ubuntu ISO.

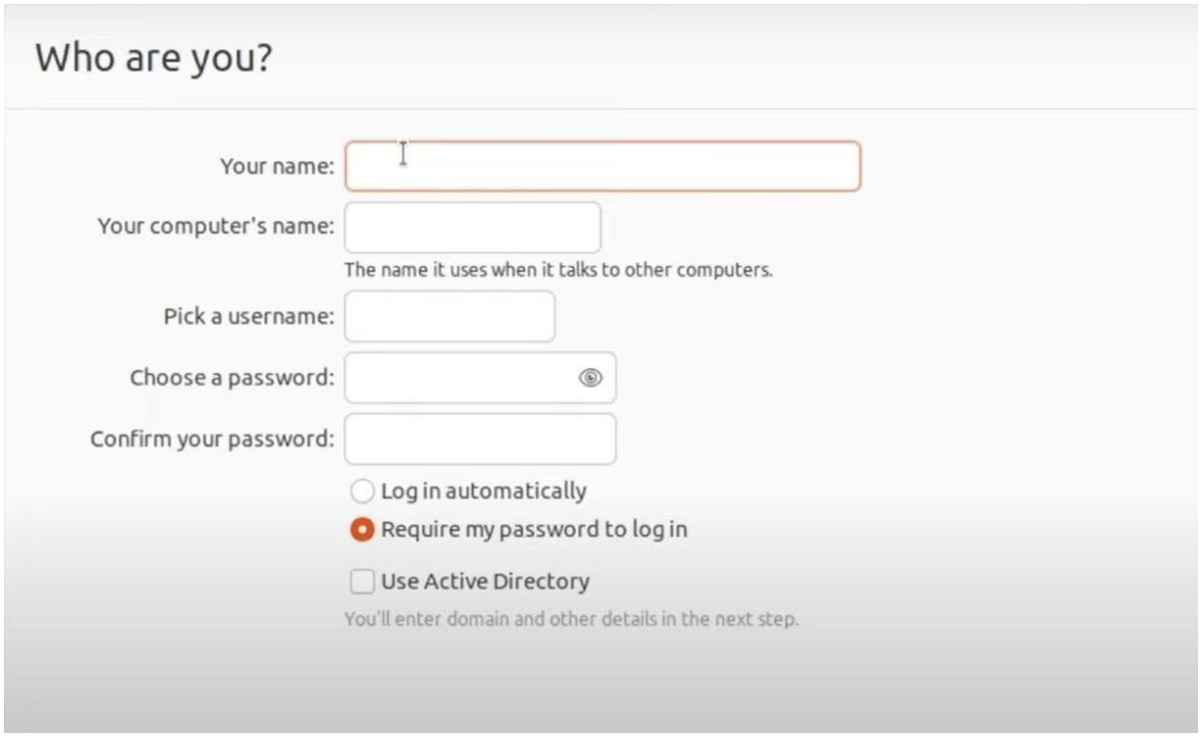
VII. Begin Installation – Select your language and click "Install Ubuntu." Choose Normal Installation and enable optional updates.



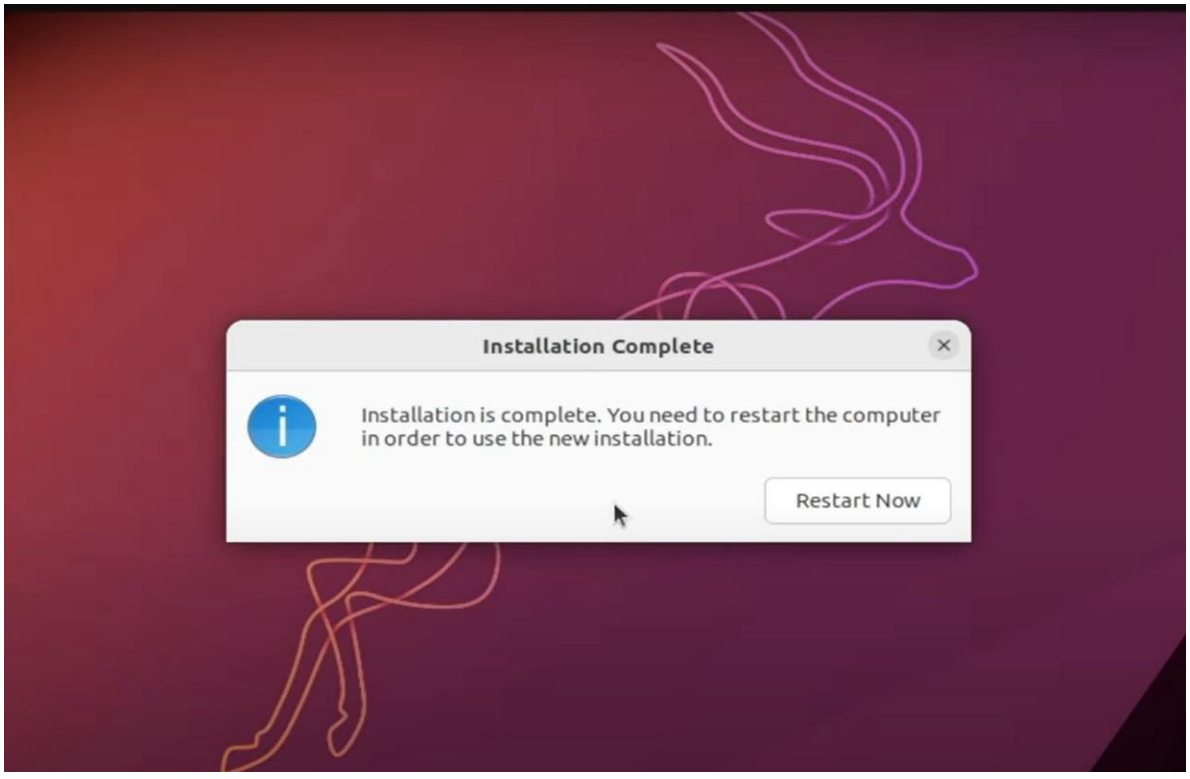
VIII. Disk Setup – Choose "Erase disk and install Ubuntu" (applies only to the virtual disk). Click "Install Now" to confirm.



IX. Now set your username and password.



X. Wait for Installation – The installer will copy files and configure the system. Once complete, click "Restart Now."



XI. Login – After reboot, enter the credentials you configured. Ubuntu is now ready for use inside VirtualBox.

PROGRAM 3 : Basic Linux Commands

I. ls

The ls command is used to list files. "ls" on its own lists all files in the current directory except for hidden files.

```
~/Workspace
❯ ls
anaconda3  Codebase  Experiments  Learning  OS  Projects

~/Workspace
❯ ls -l
total 0
drwxr-xr-x 1 aditya aditya 436 Dec 24 19:23 anaconda3
drwxr-xr-x 1 aditya aditya  14 Dec 29 19:58 Codebase
drwxr-xr-x 1 aditya aditya  84 Dec 28 18:00 Experiments
drwxr-xr-x 1 aditya aditya  60 Dec 23 22:43 Learning
drwxr-xr-x 1 aditya aditya 214 Dec 28 18:00 Projects
```

```
~
❯ ls -a
.          .claude.json                      .conda             Downloads   .mozilla    .python_history  Workspace
..         .claude.json.backup               .config            .flutter    Music       .Rhistory        .zcompdump
.anaconda  .claude.json.backup.1770128544988  .dart-tool         fvm         .npm        .rustup          .zsh_history
Backup     .claude.json.backup.1770128599020  devtools-dotfiles  .gitconfig  Pictures    .ssh             .zshrc
.cache     .claude.json.backup.1770128599698  Documents          .gnupg      .pki        Templates
.cargo     .claude.json.backup.1770128634035  Dotfiles           .ipython    .pub-cache  Videos
.claude    .claude.json.backup.1770128634268  .dotnet            .local      Public      .vscode

~
❯ ls -la ~/Workspace/Projects
total 0
drwxr-xr-x 1 aditya aditya 214 Dec 28 18:00 .
drwxr-xr-x 1 aditya aditya  88 Feb 11 21:42 ..
drwxr-xr-x 1 aditya aditya 102 Dec 28 15:33 Multiagent-Customer-Support
drwxr-xr-x 1 aditya aditya 260 Dec 28 15:33 Multiagent-Customer-Support-Basic
drwxr-xr-x 1 aditya aditya 116 Dec 28 15:34 PowershellAD-Query-Translator
drwxr-xr-x 1 aditya aditya  54 Dec 28 15:32 Python-Binance-Bot
```

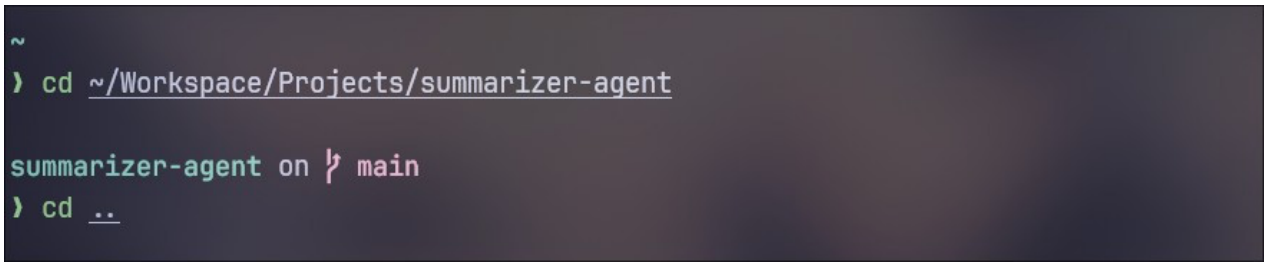
II. pwd

Use the pwd command to write to standard output the full path name of your current directory (from the /(root) directory).

```
~
❯ pwd
/home/aditya

~
❯ cd ~/Workspace

~/Workspace
❯ pwd
/home/aditya/Workspace
```

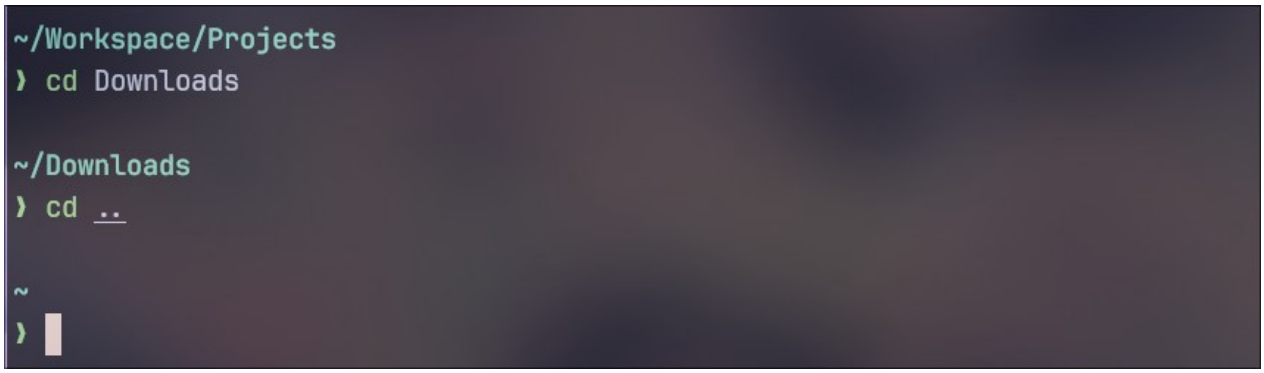
III. cd

The 'cd' command allows users to change their current working directory within the file system.



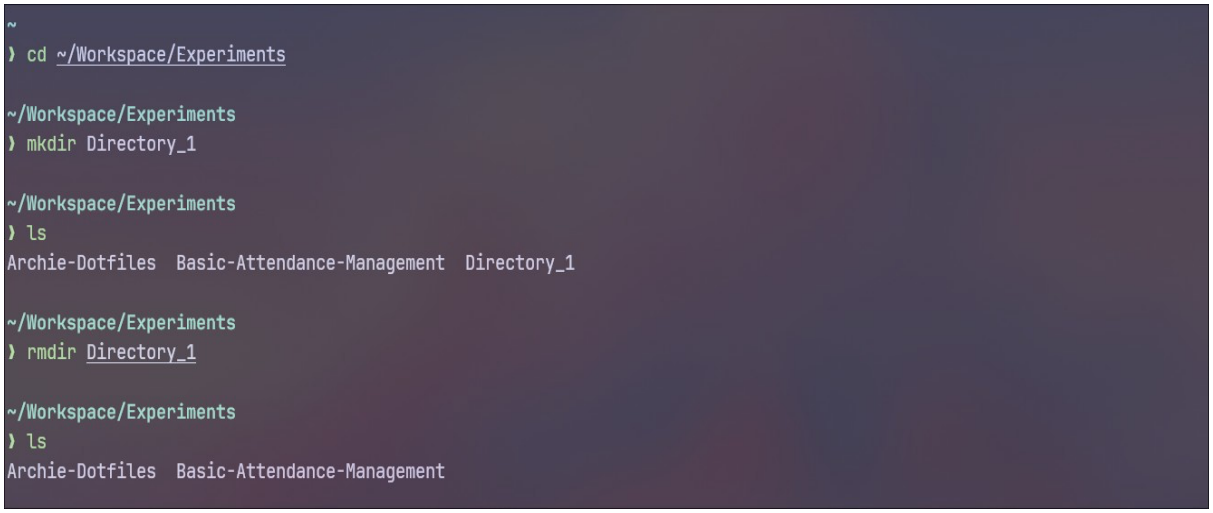
IV. cd ..

This command ise used to move to the previous directory.



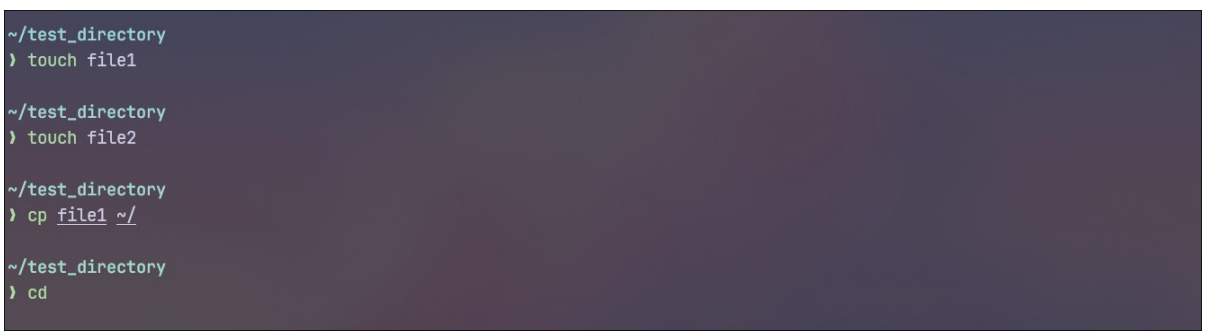
V. mkdir

The mkdir (make directory) command in the Unix, DOS, DR FlexOS, IBM OS/2, MicrosoftWindows, and ReactOS operating systems



VI. touch

It is used to create a file without any content. The file created using the touch command is empty.



VII. rm

Use the rm command to remove files you no longer need. The rm command removes the entries for a specified file,

```
~/test_directory
❯ mv file1 ~/Workspace

~/test_directory
❯ ls
file2

~/test_directory
❯ rm file2

~/test_directory
❯ ls
```

VIII. cat

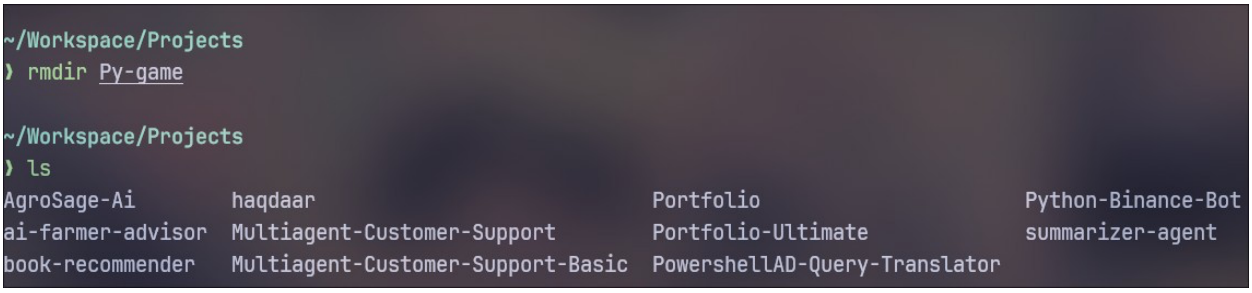
The cat (concatenate) command in Linux is used to view, create, and combine file contents directly.

```
~/test_directory
❯ cat -n file1
     1  This is the first file used to test basic shell commands.
     2  This is Operating System practical.
     3  Hello!!!!%

~/test_directory
❯ cat -b file1
     1  This is the first file used to test basic shell commands.
     2  This is Operating System practical.
     3  Hello!!!!%

~/test_directory
❯ cat -A file2
We are writing in the file now.%
```

```
~/test_directory took 18s
❯ cat file1
This is the first file used to test basic shell commands.
This is Operating System practical.

~/test_directory
❯ cat file1 file2
This is the first file used to test basic shell commands.
This is Operating System practical.
This is the second file.

~/test_directory
❯ cat > file2
We are writing in the file now.%
~/test_directory took 19s
❯ cat file2
We are writing in the file now.%
~/test_directory
❯ cat >> file1
Hello!!!!%
~/test_directory took 13s
❯ cat file1
This is the first file used to test basic shell commands.
This is Operating System practical.
```

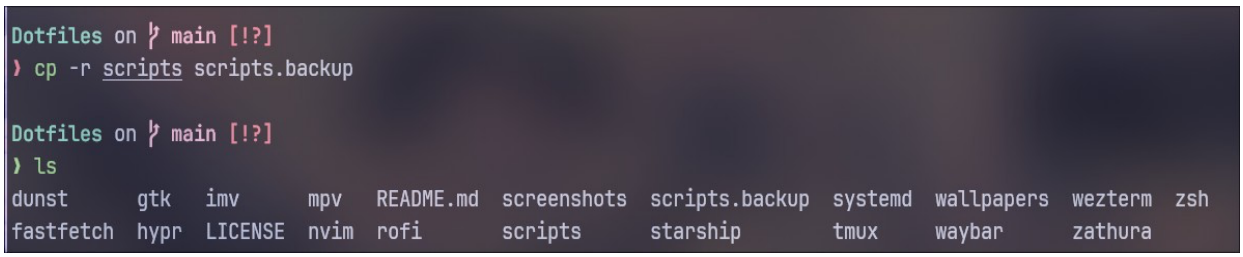
IX. rmdir

The rmdir command removes the directory, specified by the Directory parameter, from the system. The directory must be empty before you can remove it



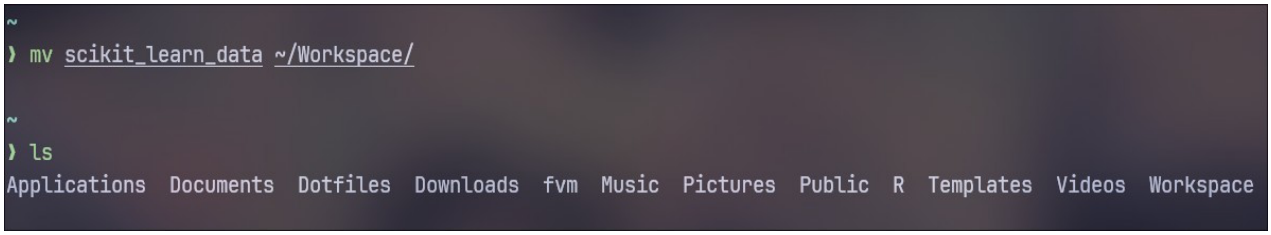
X. cp

Copies a file or directory to a destination path.



XI. mv

Moves a file to a different location or renames it.



XII. chmod

Modifies read, write, and execute permissions for a file or directory.

PROGRAM 4 : Study of vi editor

Introduction to VI Editor

The **vi editor** (Visual Editor) is a standard text editor available on virtually all Unix and Linux systems. Developed by **Bill Joy** in the late 1970s as part of the BSD Unix project, vi is known for its speed, efficiency, and keyboard-centric operation. It is the default editor in minimal and headless environments where graphical tools are unavailable.

Vi operates in a **modal design** — different modes serve different purposes, allowing powerful text manipulation without a mouse.

Operating Modes

Mode	Purpose	How to Enter
Command Mode	Navigate, delete, copy, paste, search	Default; press Esc from other modes
Insert Mode	Type and edit text	Press i, a, o from Command Mode
Last Line Mode	Save, quit, or run commands	Press : from Command Mode

Common VI Commands

- **Navigation (Command Mode)**

Command	Action
h, j, k, l	Move left, down, up, right
w	Move to next word
gg	Go to beginning of file
G	Go to end of file

- **Editing**

Command	Action
i	Insert before cursor
a	Append after cursor
o	Open new line below
dd	Delete current line
yy	Yank (copy) current line
p	Paste after cursor
u	Undo last action

- **Search and Replace**

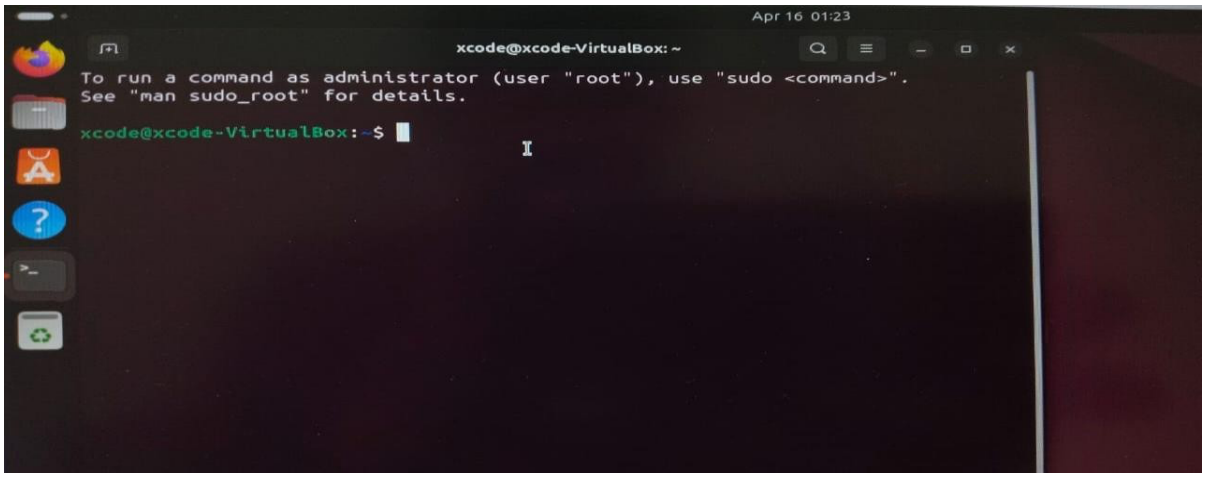
Command	Action
/pattern	Search forward for pattern
?pattern	Search backward
n	Move to next match
:%s/old/new/g	Replace all occurrences in file

- **Save and Exit**

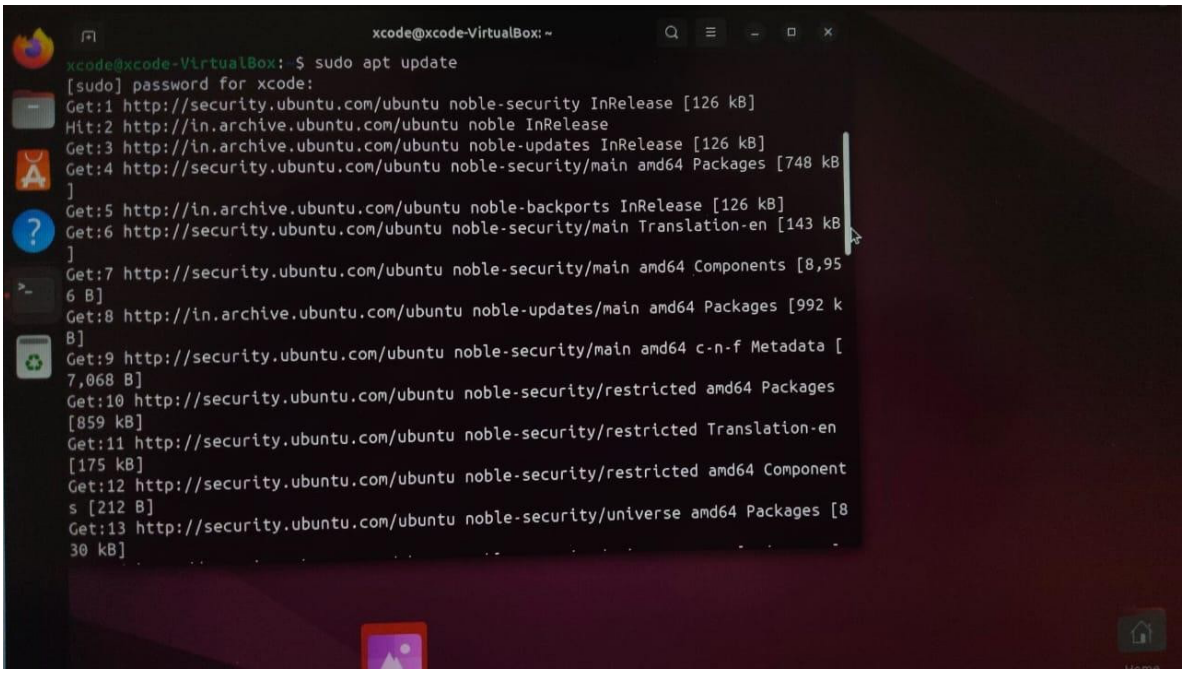
Command	Action
:w	Save file
:wq	Save and quit
:q!	Quit without saving

Installing VIM (VI improved)

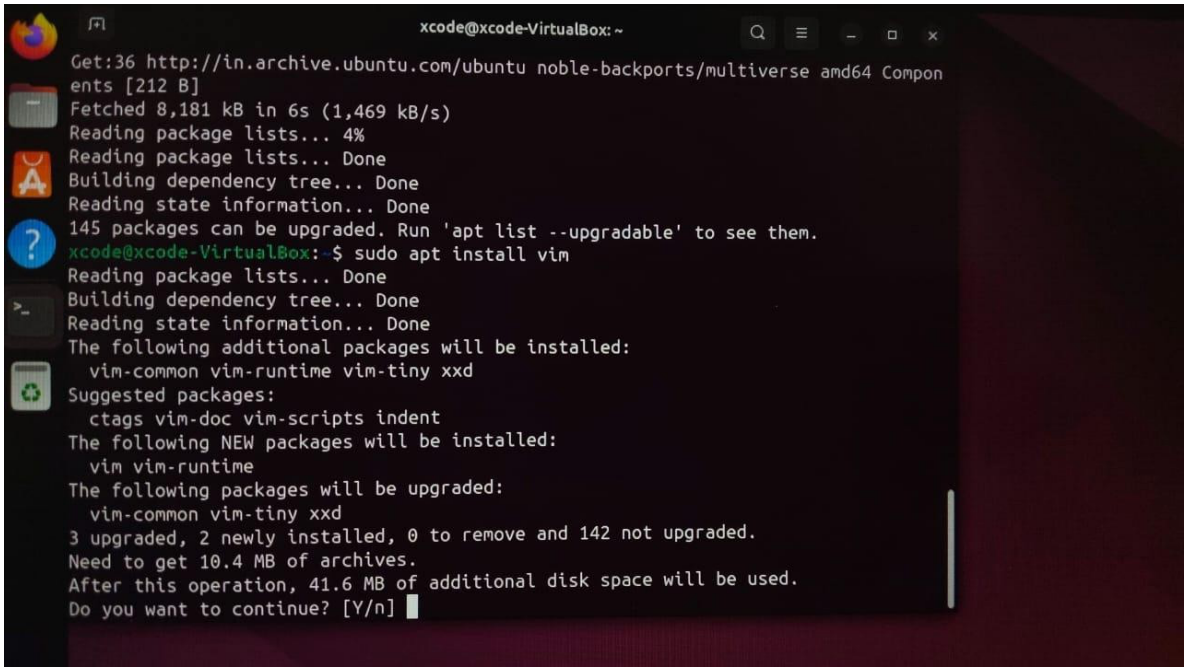
- Open the terminal , you can open the terminal by pressing Ctrl + Alt + T



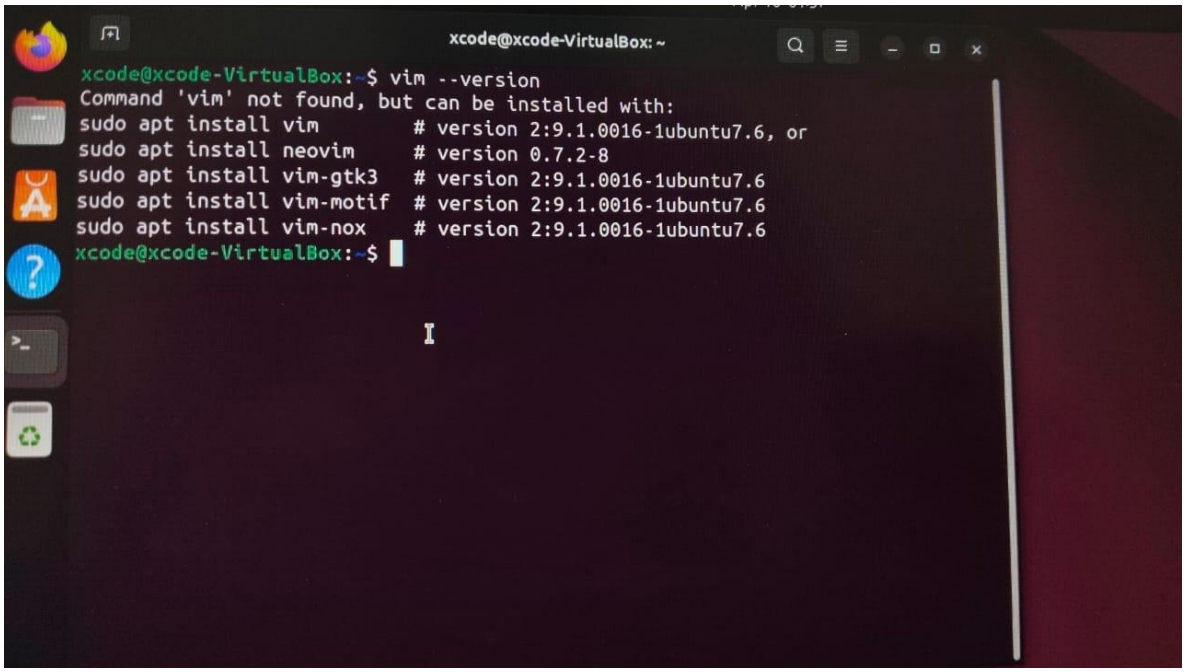
- Update the Package List using the command : sudo apt update



- Install vim (Improved vi editor) using the command sudo apt install vim. This will install the full feature version of vi.



- Verify the Installation , After installation check the version to confirm it's installed.



PROGRAM 5 : Linux pipe and filter commands

Introduction

In Linux, pipes and filters are core concepts of the shell that enable efficient text processing. A pipe (|) connects the output of one command as input to the next, while filter commands transform, search, or extract data from text streams.

Command Syntax:

command_name [options] [arguments]

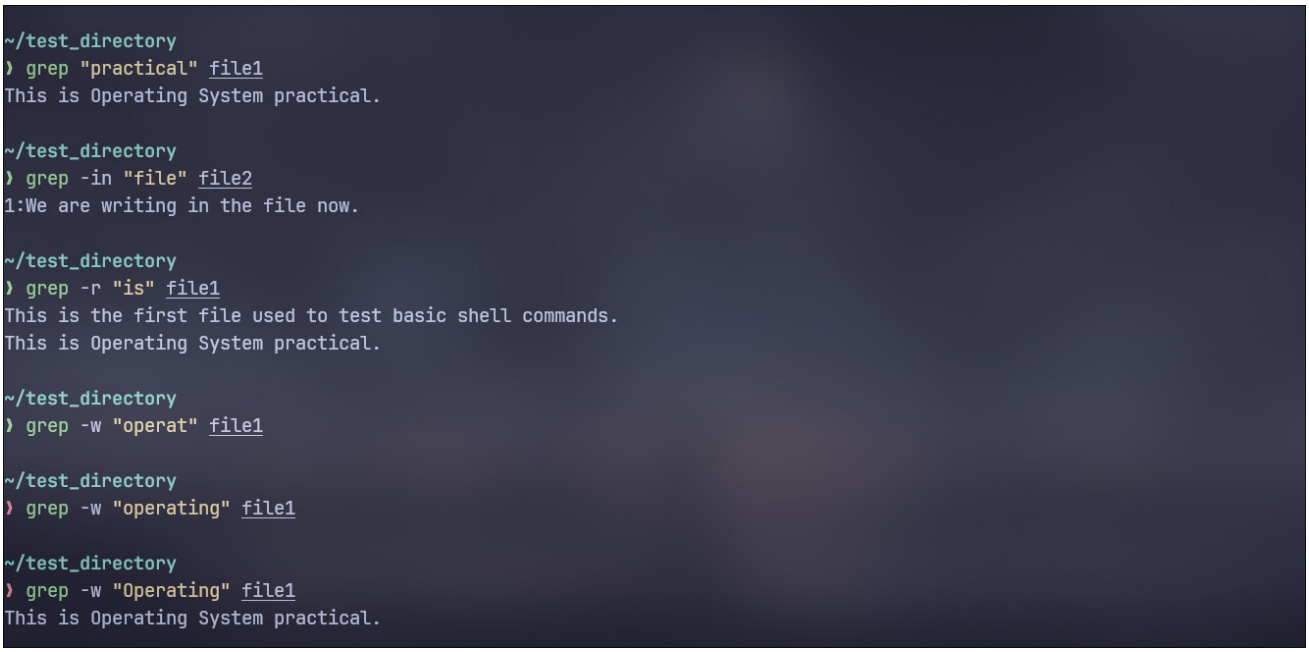
Options are case-sensitive and can be combined.

Short form: cat -n file.txt,

Long form: cat --number file.txt.

Common Pipe and Filter Commands

I. grep



II. egrep

```
~/test_directory
❯ grep -c "This" file1
2

~/test_directory
❯ grep "operat^" file1

~/test_directory
❯ grep "[0,9]" file3
1,2,3,46,58,67,76,84,

~/test_directory
❯ grep -E "Operating|shell" file1
This is the first file used to test basic shell commands.
This is Operating System practical.

~/test_directory
❯

~/test_directory
❯ grep -E ".*(ing|ed)\b" file1
This is the first file used to test basic shell commands.
This is Operating System practical.
```

III. fgrep

```
~/test_directory
❯ grep -F ".*" file2

~/test_directory
❯ grep -F "." file2
We are writing in the file now.

~/test_directory
❯ grep -F "[INFO] file" file1

~/test_directory
❯ grep -F "[first] file" file1

~/test_directory
❯ grep -F -e "commands" -e "Operating" file1
This is the first file used to test basic shell commands.
This is Operating System practical.
```

IV. cut

```
~
❯ cut -d' ' -f2 data.txt
Bharati
Poonam
Seema
```

V. sort

```
~
❯ sort -t' ' -k1 -n data.txt
1 Bharati CS
2 Poonam ECE
3 Seema CE
```

VI. awk

```
~/test_directory
❯ awk '{ print }' file2
We are writing in the file now.

~/test_directory
❯ awk '{ print $1 }' file1
This
This
Hello!!!!

~/test_directory
❯ awk '{ print NR ":" $0 }' file1
1:This is the first file used to test basic shell commands.
2:This is Operating System practical.
3:Hello!!!!

~/test_directory
❯ awk 'NF > 3' file2
We are writing in the file now.

~/test_directory
❯ awk '$3 == "is" { print $2 }' file1
```

VII. tr

```
~
❯ echo "hello world" | tr 'a-z' 'A-Z'
HELLO WORLD
```

VIII. uniq

```
~
❯ sort fruits.txt | uniq -c
      1 apple
      1 banana
      2 mango
```

IX. head

```
~ took 21s
❯ head file1.txt
Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the late
st stable versions of most software by following a rolling release model.
The default installation is a minimal base system, configured by the user to only add what is purposely required.
```

X. tac

```
~
❯ tac file1.txt
The default installation is a minimal base system, configured by the user to only add what is purposely required.
Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the late
st stable versions of most software by following a rolling release model.
```

XI. wc arfile

```
~
❯ wc file1.txt
  2  46 308 file1.txt
```

XII. nl arfile

```
~
❯ nl file1.txt
     1  Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide
the latest stable versions of most software by following a rolling release model.
     2  The default installation is a minimal base system, configured by the user to only add what is purposely required
.
```

XIII. cal/ncal

```
~
❯ cal
     April 2026
Su Mo Tu We Th Fr Sa
          1  2  3  4
 5  6  7  8  9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30
```

XIV. pipe

```
~
❯ cat file1.txt | grep "Linux" | wc -l
1
```

PROGRAM 6 : Study of Shell Scripts

Introduction to Shell Scripting
A shell script is a plain text file containing a sequence of shell commands executed in order. It enables automation of repetitive tasks, system administration, and batch processing. Shell scripts are interpreted by the shell (e.g., bash) rather than compiled.

Writing and Executing a Shell Script

Step 1 – Create the script using any text editor:

vi myscript.sh

Step 2 – Add the shebang line at the top to specify the interpreter:

#!/bin/bash

Step 3 – Write your commands within the file.

Step 4 – Grant execute permission:

chmod +x myscript.sh
or
chmod 755 myscript.sh

Step 5 – Execute the script:

./myscript.sh
bash myscript.sh

Variables in Shell:
To process our data/information, data must be kept in computers RAM memory. RAM memory is divided into small locations, and each location had unique number called memory location/address, which is used to hold our data.
Programmer can give a unique name to this memory location/address called memory variable or variable (It's a named storage location that may take different values, but only one at a time).
In Linux (Shell), there are two types of variable:

- System variables - Created and maintained by Linux itself. This type of variable defined in CAPITAL LETTERS.
- User defined variables (UDV) - Created and maintained by user. This type of variable defined in lower letters. You can see system variables by giving command like $set, some of the important System variables are:

System Variable	Meaning
BASH=/bin/bash	Our shell name
BASH_VERSION=1.14.7(1)	Our shell version name
COLUMNS=80	No. of columns for our screen
HOME=/home/vivek	Our home directory
LINES=25	No. of columns for our screen
LOGNAME=students	students Our logging name
OSTYPE=Linux	Our Os type
PATH=/usr/bin:/sbin:/bin:/usr/sbin	Our path settings
PS1=[\u@\h \W]\$	Our prompt settings
PWD=/home/students/Common	Our current working directory
SHELL=/bin/bash	Our shell name
USERNAME=vivek	User name who is currently login to this PC

The read Statement:
Use to get input (data from user) from keyboard and store (data) to variable.
Syntax:
read variable1, variable2,...variableN
Following script first ask user, name and then waits to enter name from the user via keyboard. Then user enters name from keyboard (after giving name you have to press ENTER key) and entered name through keyboard is stored (assigned) to variable fname

Why Command Line arguments required: Telling the command/utility which option to use.Informing the utility/command which file or group of files to process (reading/writing of files).
Let's take rm command, which is used to remove file, but which file you want to remove and how you will tail this to rm command (even rm command don't ask you name of file that you would like to remove). So, what we do is we write command as follows:
$ rm {file-name} Here rm is command and filename are file which you would like to remove. This way you tail rm command which file you would like to remove. So, we are doing one way communication with our command by specifying filename Also you can pass command line arguments to your script to make it more users friendly. But how we access command line argument in our script.

Command Line Arguments
Shell scripts can accept arguments at the time of execution:

./myscript.sh arg1 arg2

Inside the script, $1 refers to arg1, $2 to arg2, and $@ refers to all arguments. This allows scripts to be made flexible and reusable.

I/O Redirection

Operator	Function
>	Redirect output to file (overwrites)
>>	Append output to file
<	Read input from file

Example:

```
ls > filelist.txt       # Save ls output to file
date >> filelist.txt    # Append date output
cat < filelist.txt      # Read input from file
```

Pipes

A pipe (|) connects the stdout of one command to the stdin of another:

ls -l | grep ".sh"

This lists all .sh files by piping ls output through grep.

PROGRAM 7 : Shell Programming

I. **Print Prime Numbers**

```
#!/bin/bash

echo "enter a

number" read num

flag=1

for((i=2;i<num;i++))

do

  if((num%i==0))

    then

      flag=

      0

      break

    fi

done

if((flag==1))

then

   echo

"prime" else

   echo "not prime"

     fi
```

OUTPUT

```
~
❯ ./prime.sh
enter a number
97
prime
```

II. Find factorial of a number

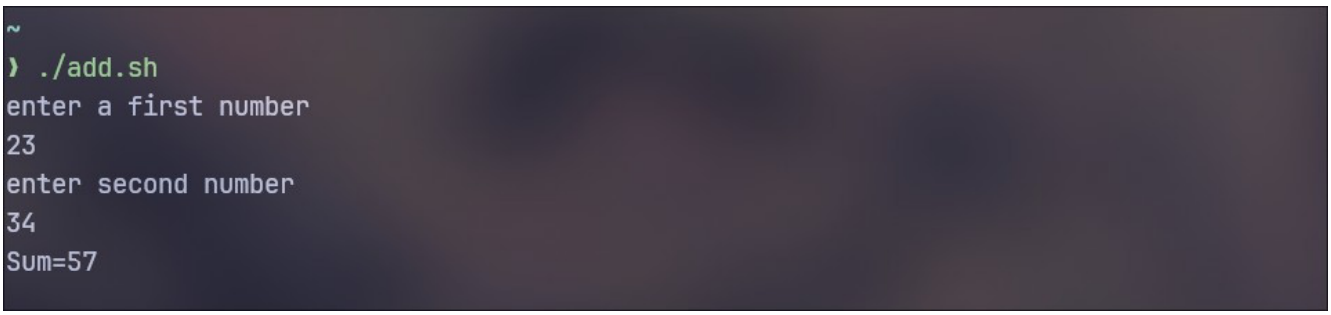
```
#!/bin/bash

echo "Enter a number"

read num

fact=1

for((i=1;i<=num;i++)

) do

fact=$((fact*i))

done

    echo "factorial of $num is $fact"
```

OUTPUT

III. Addition of two numbers

```
#!/bin/bash

echo "enter a first number"

read a

echo "enter second number"

read b

sum=$((a+b))

    echo "Sum=$sum"
```

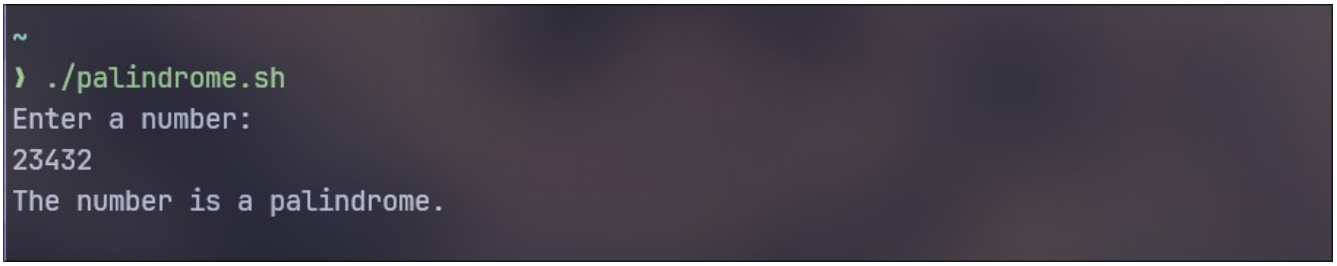
OUTPUT



IV. Check Palindrome

```
#!/bin/bash

echo "Enter a

number:" read num

original=$num

reverse=0

while [ $num -gt 0

] do

   digit=$((num % 10))

   reverse=$((reverse * 10 + digit))

   num=$((num / 10))

done


if [ $original -eq $reverse ]

then

   echo "The number is a palindrome."

else

   echo "The number is not a

      palindrome." fi
```

OUTPUT

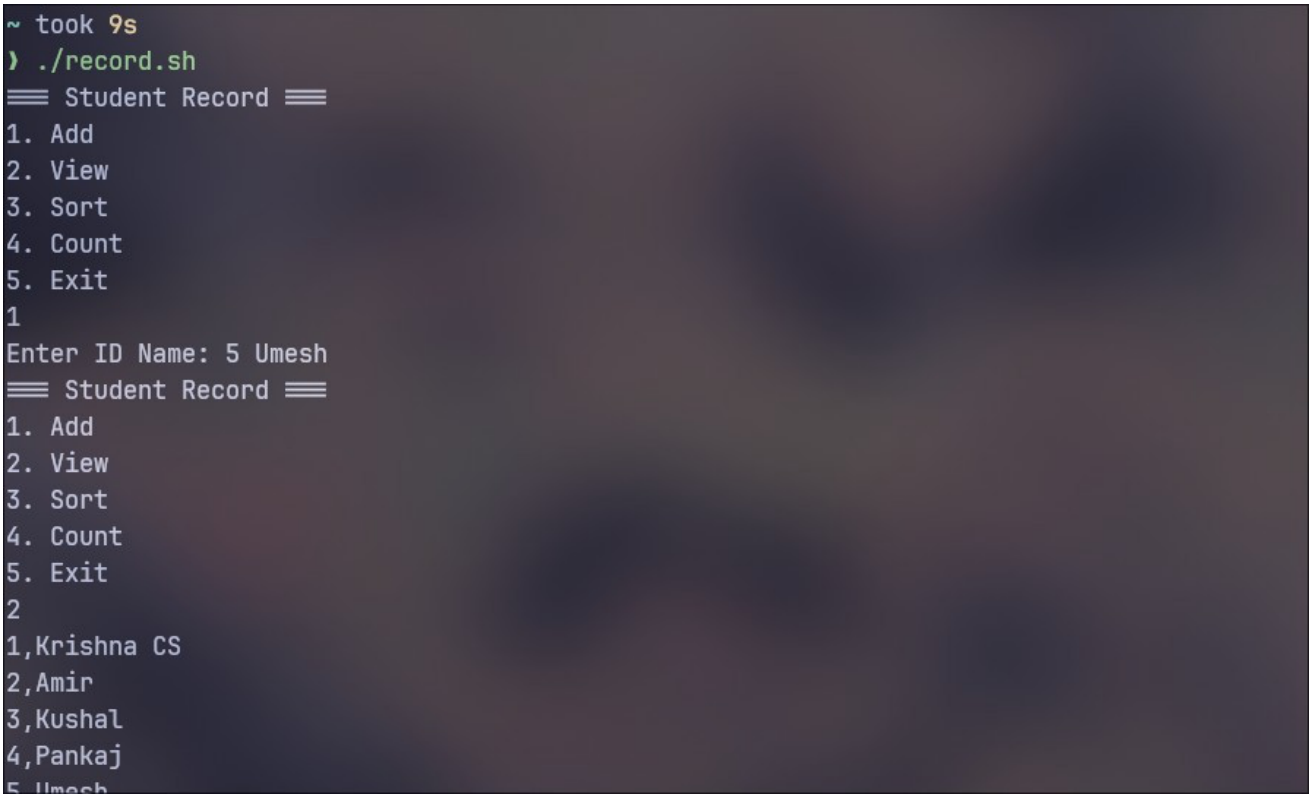


V. Menu Driven shell program

```
#!/bin/bash

file="students.txt"

while true; do

  echo "=== Student Record

  ===" echo "1. Add"

  echo "2. View"

  echo "3. Sort"

  echo "4. Count"

  echo "5. Exit"

  read ch


  case $ch in

  1)

    read -p "Enter ID Name: " id name

    echo "$id,$name" >>$file

    ;;

  2)

    cat $file

    ;;

  3)

    sort -t',' -k2 $file

    ;;
```

```
  4)
    wc -l $file
    ;;
  5)
    break
    ;;
  *)
    echo "Wrong choice"
    ;;
  esac
done
```

OUTPUT



PROGRAM 8 : AWK Commands

Basic AWK Text Processing

I. Print all lines

a. awk '{ print }' file.txt

```
~
❯ awk '{ print }' file.txt
Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the late
st stable versions of most software by following a rolling release model.
The default installation is a minimal base system, configured by the user to only add what is purposely required.

~ _
```

II. Print first column

a. awk '{ print $1 }' file.txt

```
~
❯ awk '{ print $1 }' file.txt
Arch
The
```

III. Print multiple columns

a. awk '{ print $1, $3 }' file.txt

```
~
❯ awk '{ print $1,$3 }' file.txt
Arch is
The installation
```

IV. Print line numbers

a. awk '{ print NR, $0 }' file.txt

```
~
❯ awk '{ print NR, $0 }' file.txt
1 Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the la
test stable versions of most software by following a rolling release model.
2 The default installation is a minimal base system, configured by the user to only add what is purposely required.
```

V. Count total lines

a. awk 'END { print NR }' file.txt

```
~
❯ awk 'END { print NR }' file.txt
2
```

AWK Commands for pattern matching

I. Print lines containing a word

awk '/operating/ { print }' file.txt

```
~
❯ awk '/Linux/ { print }' file.txt
Arch Linux is an independently developed, x86-64 general-purpose GNU/Linux distribution that strives to provide the late
st stable versions of most software by following a rolling release model.
```

II. Print lines NOT matching a pattern

awk '!/operating/ { print }' file.txt

```
~
❯ awk '!/Linux/ { print }' file.txt
The default installation is a minimal base system, configured by the user to only add what is purposely required.
```

III. Match start of line

awk '/^An/' file.txt

```
~
❯ awk '/^T/' file.txt
The default installation is a minimal base system, configured by the user to only add what is purposely required.
```

IV. Print rows where column > value

awk '$2 > 50 { print }' file.txt

```
~
❯ awk '$2 > 50 { print }' students.txt
1,Krishna CS
```

V. Print specific column condition

awk '$1 == "mango" { print $2 }' file.txt

```
~
❯ awk '$1 == "mango" { print $2 }' fruits.txt
yellow
orange
```

VI. Count matching rows

awk '/error/ { count++ } END { print count }' file.txt

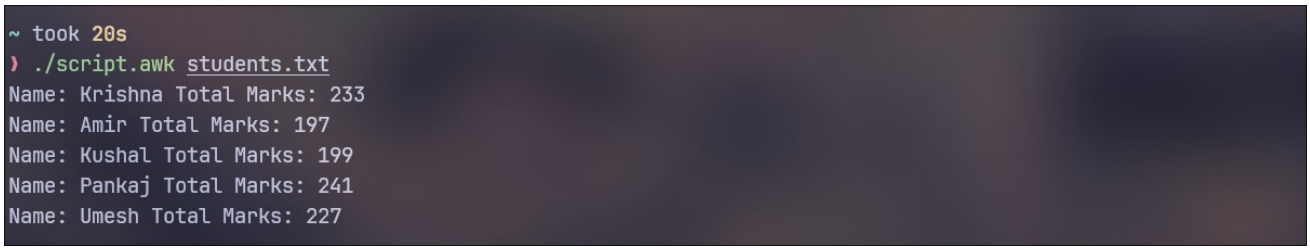
```
~
❯ awk '/installation/ { count++ } END { print count }' file.txt
1
```

AWK Scripts

I. Program to find total marks of Students

```
{
   total = $2 + $3 + $4
   print "Name:", $1, "Total Marks:", total
}
```

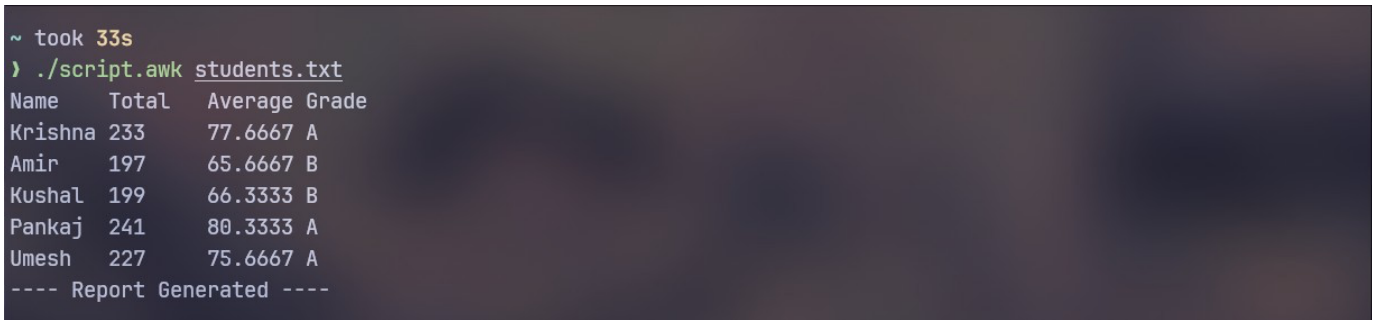
OUTPUT



II. Program to find and display the average marks and grades of students.

```
#!/usr/bin/awk -f

BEGIN {
   print "Name\tTotal\tAverage\tGrade"
}

{
   total = $2 + $3 + $4
   avg = total / 3

   if (avg >= 75)
      grade = "A"
   else if (avg >= 50)
      grade = "B"
   else
      grade = "C"

   print $1 "\t" total "\t" avg "\t" grade
}

END {
   print "---- Report Generated ---- "
}
```

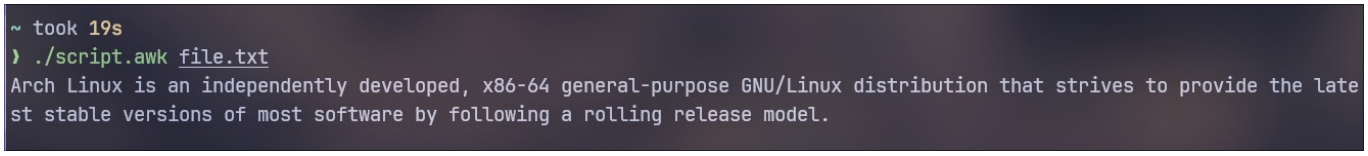
OUTPUT



III. Program to find duplicate lines in a file.

```
#!/usr/bin/awk -f

{
   count[$0]++
}

END {
   for (line in count) {
      if (count[line] > 1)
         print line
   }
}
```

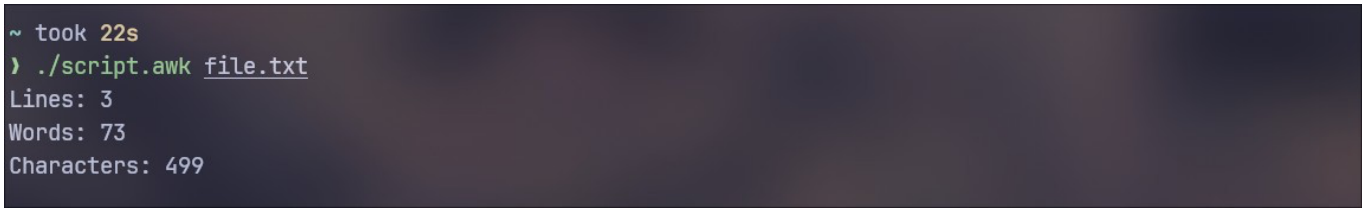
OUTPUT



IV. Program to count number of lines, words, and characters

```
#!/usr/bin/awk -f

{
    lines++ words
    += NF
    chars += length($0)
}

END {
    print "Lines:", lines print
    "Words:", words
    print "Characters:", chars
}
```

OUTPUT



PROGRAM 9 : Write a program for CPU Scheduling Algorithms in C Programming

First Come First Serve:

```
#include <stdio.h>
int main() {
   int pid[15], bt[15], n;
   printf("Enter the number of processes: ");
   scanf("%d", &n);
   printf("Enter process IDs: ");
   for (int i = 0; i < n; i++) scanf("%d", &pid[i]);
   printf("Enter burst times: ");
   for (int i = 0; i < n; i++) scanf("%d", &bt[i]);

   int wt[n];
   wt[0] = 0;
   for (int i = 1; i < n; i++)
      wt[i] = bt[i-1] + wt[i-1];

   float twt = 0, tat = 0;
   printf("PID\tBurst\tWait\tTAT\n");
   for (int i = 0; i < n; i++) {
      printf("%d\t%d\t%d\t%d\n", pid[i], bt[i], wt[i], bt[i]+wt[i]);
      twt += wt[i];
      tat += (wt[i] + bt[i]);
   }
   printf("Avg Waiting Time: %.2f\n", twt/n);
   printf("Avg Turnaround Time: %.2f\n", tat/n);
   return 0;
}
```

OUTPUT

```
Enter the number of processes: 5
Enter process IDs: 1 2 3 4 5
Enter burst times: 2 3 6 4 1
PID     Burst   Wait    TAT
1       2       0       2
2       3       2       5
3       6       5       11
4       4       11      15
5       1       15      16
Avg Waiting Time: 6.60
Avg Turnaround Time: 9.80
```

Shortest Job First:

```
include <stdio.h>
int main() {
    int bt[20], p[20], wt[20], tat[20], n, total=0, totalT=0, pos, temp;
    float avg_wt, avg_tat;
    printf("Enter number of processes: ");
    scanf("%d", &n);
    printf("Enter Burst Times:\n");
    for (int i = 0; i < n; i++) {
        printf("P%d: ", i+1);
        scanf("%d", &bt[i]);
        p[i] = i+1;
    }
    for (int i = 0; i < n; i++) {
        pos = i;
        for (int j = i+1; j < n; j++)
            if (bt[j] < bt[pos]) pos = j;
        temp = bt[i]; bt[i] = bt[pos]; bt[pos] = temp;
        temp = p[i];  p[i] = p[pos];  p[pos] = temp;
    }
    wt[0] = 0;
    for (int i = 1; i < n; i++) {
        wt[i] = 0;
        for (int j = 0; j < i; j++) wt[i] += bt[j];
        total += wt[i];
    }
    avg_wt = (float)total / n;
    printf("Process\tBurst\tWait\tTAT\n");
    for (int i = 0; i < n; i++) {
        tat[i] = bt[i] + wt[i];
        totalT += tat[i];
        printf("P%d\t%d\t%d\t%d\n", p[i], bt[i], wt[i], tat[i]);
    }
    avg_tat = (float)totalT / n;
    printf("Avg Waiting Time: %.2f\n", avg_wt);
    printf("Avg Turnaround Time: %.2f\n", avg_tat);
    return 0;
}
```

OUTPUT

```
Enter number of processes: 5
Enter Burst Times:
P1: 2
P2: 5
P3: 2
P4: 4
P5: 7
Process Burst   Wait    TAT
P1      2       0       2
P3      2       2       4
P4      4       4       8
P2      5       8       13
P5      7       13      20
Avg Waiting Time: 5.40
Avg Turnaround Time: 9.40
```

Round Robin:

```
#include <stdio.h>
int main() {
   int n;
   printf("Enter Total Number of Processes: ");
   scanf("%d", &n);
   int wait_time=0, ta_time=0, arr_time[n], burst_time[n], temp_burst_time[n];
   int x = n;
   for (int i = 0; i < n; i++) {
      printf("Process %d - Arrival Time: ", i+1); scanf("%d", &arr_time[i]);
      printf("Process %d - Burst Time: ", i+1);   scanf("%d", &burst_time[i]);
      temp_burst_time[i] = burst_time[i];
   }
   int time_slot;
   printf("Enter Time Quantum: ");
   scanf("%d", &time_slot);
   int total=0, counter=0, i;
   printf("PID\tBurst\tTAT\tWait\n");
   for (total=0, i=0; x!=0; ) {
      if (temp_burst_time[i] <= time_slot && temp_burst_time[i] > 0) {
         total += temp_burst_time[i];
         temp_burst_time[i] = 0;
         counter = 1;
      } else if (temp_burst_time[i] > 0) {
         temp_burst_time[i] -= time_slot;
         total += time_slot;
      }
      if (temp_burst_time[i] == 0 && counter == 1) {
         x--;
         printf("P%d\t%d\t%d\t%d\n", i+1, burst_time[i],
               total-arr_time[i], total-arr_time[i]-burst_time[i]);
         wait_time += total - arr_time[i] - burst_time[i];
         ta_time   += total - arr_time[i];
         counter = 0;
      }
      if (i == n-1) i = 0;
      else if (arr_time[i+1] <= total) i++;
      else i = 0;
   }
   printf("Avg Waiting Time: %.2f\n", (float)wait_time/n);
   printf("Avg Turnaround Time: %.2f\n", (float)ta_time/n);
   return 0;
}
```

OUTPUT

```
Enter Total Number of Processes: 4
Process 1 - Arrival Time: 3
Process 1 - Burst Time: 5
Process 2 - Arrival Time: 2
Process 2 - Burst Time: 6
Process 3 - Arrival Time: 1
Process 3 - Burst Time: 3
Process 4 - Arrival Time: 4
Process 4 - Burst Time: 7
Enter Time Quantum: 2
PID     Burst   TAT     Wait
P3      3       12      9
P1      5       13      8
P2      6       16      10
P4      7       17      10
Avg Waiting Time: 9.25
Avg Turnaround Time: 14.50
```

Priority Based Scheduling:

```
#include <stdio.h>
void swap(int *a, int *b) { int t = *a; *a = *b; *b = t; }
int main() {
   int n;
   printf("Enter Number of Processes: ");
   scanf("%d", &n);
   int b[n], p[n], index[n];
   for (int i = 0; i < n; i++) {
      printf("Burst Time and Priority for P%d: ", i+1);
      scanf("%d %d", &b[i], &p[i]);
      index[i] = i+1;
   }
   for (int i = 0; i < n; i++) {
      int a = p[i], m = i;
      for (int j = i; j < n; j++)
         if (p[j] > a) { a = p[j]; m = j; }
      swap(&p[i], &p[m]);
      swap(&b[i], &b[m]);
      swap(&index[i], &index[m]);
   }
   int t = 0;
   printf("Execution Order:\n");
   for (int i = 0; i < n; i++) {
      printf("P%d executed from %d to %d\n", index[i], t, t+b[i]);
      t += b[i];
   }
   printf("\nPID\tBurst\tWait\tTAT\n");
   int wt = 0;
   for (int i = 0; i < n; i++) {
      printf("P%d\t%d\t%d\t%d\n", index[i], b[i], wt, wt+b[i]);
      wt += b[i];
   }
   return 0;
}
```

OUTPUT

```
Enter Number of Processes: 4
Burst Time and Priority for P1: 3 1
Burst Time and Priority for P2: 5 3
Burst Time and Priority for P3: 2 4
Burst Time and Priority for P4: 1 7
Execution Order:
P4 executed from 0 to 1
P3 executed from 1 to 3
P2 executed from 3 to 8
P1 executed from 8 to 11

PID     Burst   Wait    TAT
P4      1       0       1
P3      2       1       3
P2      5       3       8
P1      3       8       11
```

PROGRAM 10 : Write a program for File Allocation Stratergies in C Programming

Sequential File Allocation:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50
int main() {
   int memory[SIZE], start, length, i, j, choice;
   for (i = 0; i < SIZE; i++) memory[i] = 0;
   while (1) {
      printf("\nSequential File Allocation\n1.Allocate\n2.Display\n3.Exit\nChoice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter start block and length: ");
            scanf("%d %d", &start, &length);
            if (start < 0 || start + length > SIZE) { printf("Invalid range.\n"); break; }
            int flag = 0;
            for (j = start; j < start+length; j++)
               if (memory[j] == 1) { flag = 1; break; }
            if (!flag) {
               for (j = start; j < start+length; j++) memory[j] = 1;
               printf("File allocated successfully.\n");
            } else printf("Some blocks already allocated.\n");
            break;
         case 2:
            printf("Block status:\n");
            for (i = 0; i < SIZE; i++) printf("%d ", memory[i]);
            printf("\n"); break;
         case 3: exit(0);
         default: printf("Invalid choice.\n");
      }
   }
}
```

OUTPUT

```
Sequential File Allocation
1.Allocate
2.Display
3.Exit
Choice: 1
Enter start block and length: 1 23
File allocated successfully.

Sequential File Allocation
1.Allocate
2.Display
3.Exit
Choice: 2
Block status:
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Sequential File Allocation
1.Allocate
2.Display
3.Exit
Choice: 3
```

Indexed File Allocation:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50
int main() {
   int memory[SIZE], indexBlock, blocks[SIZE], n, i, choice;
   for (i = 0; i < SIZE; i++) memory[i] = 0;
   while (1) {
      printf("\nIndexed File Allocation\n1.Allocate\n2.Display\n3.Exit\nChoice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter index block: ");
            scanf("%d", &indexBlock);
            if (indexBlock < 0 || indexBlock >= SIZE || memory[indexBlock]) {
               printf("Index block unavailable.\n"); break;
            }
            printf("Number of blocks needed: ");
            scanf("%d", &n);
            printf("Enter block numbers:\n");
            int success = 1;
            for (i = 0; i < n; i++) {
               scanf("%d", &blocks[i]);
               if (blocks[i] < 0 || blocks[i] >= SIZE || memory[blocks[i]] || blocks[i] == indexBlock)
               { success = 0; printf("Block %d unavailable.\n", blocks[i]); }
            }
            if (success) {
               memory[indexBlock] = 1;
               for (i = 0; i < n; i++) memory[blocks[i]] = 1;
               printf("File allocated successfully.\n");
            } else printf("Allocation failed.\n");
            break;
         case 2:
            printf("Block status:\n");
            for (i = 0; i < SIZE; i++) printf("%d ", memory[i]);
            printf("\n"); break;
         case 3: exit(0);
         default: printf("Invalid choice.\n");
      }
   }
}
```

OUTPUT

```
Indexed File Allocation
1.Allocate
2.Display
3.Exit
Choice: 1
Enter index block: 02
Number of blocks needed: 10
Enter block numbers:
1 3 5 7 9 10 11 13 15 17
File allocated successfully.

Indexed File Allocation
1.Allocate
2.Display
3.Exit
Choice: 2
Block status:
0 1 1 1 0 1 0 1 0 1 1 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Indexed File Allocation
1.Allocate
2.Display
3.Exit
Choice: 3
```

Linked File Allocation:

```
#include <stdio.h>
#include <stdlib.h>
#define SIZE 50
int main() {
   int memory[SIZE], startBlock, length, blocks[SIZE];
   int i, choice;
   for (i = 0; i < SIZE; i++) memory[i] = 0;
   while (1) {
      printf("\nLinked File Allocation\n1.Allocate\n2.Display\n3.Exit\nChoice: ");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter start block and file length: ");
            scanf("%d %d", &startBlock, &length);
            if (startBlock < 0 || startBlock >= SIZE || memory[startBlock]) {
               printf("Invalid start block.\n"); break;
            }
            blocks[0] = startBlock;
            int success = 1;
            printf("Enter block numbers (excluding start):\n");
            for (i = 1; i < length; i++) {
               scanf("%d", &blocks[i]);
               if (blocks[i] < 0 || blocks[i] >= SIZE || memory[blocks[i]])
               { success = 0; printf("Block %d unavailable.\n", blocks[i]); }
            }
            if (success) {
               for (i = 0; i < length; i++) memory[blocks[i]] = 1;
               printf("Linked allocation successful.\nChain: ");
               for (i = 0; i < length; i++) {
                  printf("%d", blocks[i]);
                  if (i != length-1) printf(" -> ");
               }
               printf("\n");
            } else printf("Allocation failed.\n");
            break;
         case 2:
            printf("Block status:\n");
            for (i = 0; i < SIZE; i++) printf("%d ", memory[i]);
            printf("\n"); break;
         case 3: exit(0);
         default: printf("Invalid choice.\n");
      }
   }
}
```

OUTPUT

```
Linked File Allocation
1.Allocate
2.Display
3.Exit
Choice: 1
Enter start block and file length: 02 12
Enter block numbers (excluding start):
1 2 3 4 5 6 7 8 9 10 11 12
Linked allocation successful.
Chain: 2 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11

Linked File Allocation
1.Allocate
2.Display
3.Exit
Choice: Invalid choice.

Linked File Allocation
1.Allocate
2.Display
3.Exit
Choice: 2
Block status:
0 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Linked File Allocation
1.Allocate
2.Display
3.Exit
Choice: 3
```