ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC & THUẬT MÁY TÍNH


# HỆ ĐIỀU HÀNH (CO2017)

---

## THỰC HÀNH 1

# GIỚI THIỆU TỔNG QUAN VỀ LẬP TRÌNH TRÊN HỆ ĐIỀU HÀNH LINUX

---

**Lớp: L02 – Học Kỳ HK251**
**Ngày nộp: 14/10/2025**

**GVHD: Thầy Nguyễn Minh Tâm**



| Sinh Viên | MSSV |
|---|---|
| Phạm Công Võ | 2313946 |



Thành Phố Hồ Chí Minh, Tháng 10/2025

# Mục lục

# List of Figures

# 1 Cơ sở lý thuyết

## 1.1 Giới thiệu chung về Linux và giao diện dòng lệnh (CLI)

Linux là một hệ điều hành mã nguồn mở được phát triển dựa trên nhân (kernel) Linux. Nó được sử dụng rộng rãi trong các máy chủ, hệ thống nhúng, thiết bị IoT và cả trong lĩnh vực học thuật – nghiên cứu. Một trong những điểm mạnh nổi bật của Linux là khả năng làm việc hiệu quả thông qua **giao diện dòng lệnh (Command Line Interface – CLI)**.

CLI cho phép người dùng nhập lệnh trực tiếp để ra lệnh cho hệ thống thực thi, thay vì sử dụng giao diện đồ họa (GUI). Mặc dù giao diện đồ họa trực quan và thân thiện, nhưng CLI lại mạnh mẽ, tiết kiệm tài nguyên, dễ tự động hóa và tương thích cao với các công cụ lập trình.

Khi làm việc trên Linux, người dùng thường tương tác thông qua **Terminal** – chương trình cho phép nhập và thực thi các lệnh. Một số cách mở Terminal trong Ubuntu:

- Nhấn tổ hợp phím `Ctrl + Alt + T`
- Hoặc vào menu: `Applications → Terminal`

Ví dụ giao diện dòng lệnh:

```
student@ubuntu:$
```

## 1.2 Khái niệm về Shell trong Linux

**Shell** là chương trình trung gian giữa người dùng và hệ điều hành, có nhiệm vụ nhận lệnh, phân tích cú pháp và gửi yêu cầu đến kernel để thực thi. Shell vừa là **trình thông dịch lệnh (command interpreter)** vừa là **ngôn ngữ lập trình kịch bản (scripting language)** cho phép tự động hóa các công việc.

Một số loại Shell phổ biến:

| **Tên Shell** | **Đặc điểm nổi bật** |
|---|---|
| Bash (Bourne Again Shell) | Phổ biến nhất, hỗ trợ scripting mạnh, có lịch sử lệnh, tự động hoàn thành. |
| Zsh (Z Shell) | Tùy biến cao, hỗ trợ plugin, giao diện đẹp hơn Bash. |
| Fish (Friendly Interactive Shell) | Dễ dùng, gợi ý lệnh thông minh, thân thiện cho người mới học. |
| Tcsh / Csh | Cú pháp tương tự ngôn ngữ C, thường dùng trong môi trường lập trình đặc thù. |
| Ksh (Korn Shell) | Hiệu suất cao, hỗ trợ lập trình script nâng cao. |

Trong hầu hết các hệ thống hiện nay, **Bash** là Shell mặc định, do đó các bài thực hành trong môn học này tập trung vào Bash.

## 1.3 Cấu trúc và hệ thống thư mục trong Linux

Linux sử dụng **hệ thống tệp phân cấp (hierarchical file system)** với thư mục gốc là "/".

| Thư mục | Chức năng |
|---|---|
| /home | Lưu dữ liệu người dùng. |
| /bin, /usr/bin | Chứa các chương trình, lệnh hệ thống. |
| /etc | Chứa các tệp cấu hình hệ thống. |
| /var | Lưu log và dữ liệu tạm thời. |
| /root | Thư mục riêng của người quản trị (root). |

## 1.4 Một số lệnh cơ bản trong Linux

Các lệnh trong Linux được sử dụng thông qua giao diện dòng lệnh (CLI).

Bảng dưới đây liệt kê một số lệnh cơ bản thường dùng cùng với chức năng và mô tả chi tiết:

| Chức năng | Lệnh | Mô tả |
|---|---|---|
| Hiển thị thư mục hiện tại | `pwd` | In ra đường dẫn tuyệt đối của thư mục hiện tại. |
| Di chuyển thư mục | `cd <path>` | Chuyển đến thư mục khác trong hệ thống. |
| Liệt kê nội dung | `ls` | Hiển thị danh sách các tệp và thư mục trong thư mục hiện tại. |
| Tạo thư mục mới | `mkdir <folder>` | Tạo một thư mục rỗng mới. |
| Xóa tệp/thư mục | `rm <name>` | Xóa tệp hoặc thư mục; dùng tùy chọn `-r` để xóa thư mục, `-f` để bỏ qua xác nhận. |
| Tạo tệp rỗng | `touch <filename>` | Tạo nhanh một tệp mới hoặc cập nhật thời gian truy cập tệp. |
| Sao chép tệp | `cp <source> <destination>` | Sao chép tệp hoặc thư mục; dùng tùy chọn `-i` để xác nhận khi ghi đè. |
| Di chuyển/đổi tên tệp | `mv <source> <destination>` | Di chuyển tệp sang vị trí khác hoặc đổi tên nếu cùng thư mục. |
| Hiển thị nội dung tệp | `cat <filename>` | Xem nội dung của tệp văn bản. |
| Chỉnh sửa tệp | `nano <filename>` | Mở trình soạn thảo văn bản `nano` ngay trong Terminal. |
| Xóa màn hình | `clear` | Dọn sạch nội dung hiển thị trong cửa sổ Terminal. |
| Xem lịch sử lệnh | `history` | Hiển thị danh sách các lệnh đã sử dụng trước đó. |
| Tìm kiếm tệp | `locate <filename>` | Tìm kiếm tệp nhanh chóng trong toàn hệ thống. |
| So sánh tệp | `diff <file1> <file2>` | So sánh sự khác biệt giữa hai tệp văn bản. |
| Kiểm tra dung lượng ổ đĩa | `df` | Hiển thị thông tin về dung lượng sử dụng và còn trống của ổ đĩa. |
| Kiểm tra dung lượng tệp/thư mục | `du` | Hiển thị kích thước của tệp hoặc thư mục, có thể dùng tùy chọn `-h` để hiển thị dạng dễ đọc (KB, MB, GB). |

## 1.5 Chuyển hướng dữ liệu và kỹ thuật Piping

Trong hệ điều hành Linux, mỗi lệnh khi thực thi đều tương tác với ba luồng dữ liệu cơ bản:

- **stdin (0)**: là luồng dữ liệu nhập vào từ bàn phím hoặc từ tệp khác.
- **stdout (1)**: là luồng dữ liệu xuất ra màn hình (Terminal) sau khi lệnh được thực thi thành công.
- **stderr (2)**: là nơi hiển thị các thông báo lỗi hoặc cảnh báo khi lệnh thực thi thất bại.

**Chuyển hướng dữ liệu (Redirection):** Chuyển hướng là kỹ thuật dùng để thay đổi điểm đến của đầu ra hoặc nguồn đầu vào của lệnh. Thay vì in ra màn hình, kết quả có thể được ghi vào tệp, hoặc lấy dữ liệu từ một tệp thay vì bàn phím.

- Dấu > dùng để ghi đè (overwrite) nội dung tệp.
- Dấu » dùng để ghi nối thêm (append) nội dung vào cuối tệp mà không xóa dữ liệu cũ.
- Dấu < dùng để lấy dữ liệu đầu vào từ một tệp.

**Kỹ thuật Piping (|):** Piping là kỹ thuật giúp kết nối nhiều lệnh lại với nhau sao cho đầu ra của lệnh thứ nhất (`stdout`) trở thành đầu vào (`stdin`) của lệnh tiếp theo. Cách này giúp xử lý dữ liệu một cách hiệu quả mà không cần tạo tệp tạm.

## 1.6 Lệnh `sudo` và `su`

Linux được thiết kế theo mô hình đa người dùng, trong đó quyền truy cập tài nguyên được phân tách chặt chẽ để đảm bảo an toàn hệ thống. Hai lệnh phổ biến giúp quản lý quyền truy cập là `sudo` và `su`.

| Lệnh | Mô tả |
|---|---|
| `sudo` | Thực thi lệnh với quyền quản trị (superuser) tạm thời, mà không cần đăng nhập vào tài khoản root. |
| `su` | Chuyển hẳn sang một tài khoản khác (thường là root), yêu cầu nhập mật khẩu của tài khoản đó. |

## 1.7 Quyền truy cập và mức độ bảo mật trong Linux

Mỗi tệp hoặc thư mục trong Linux đều có ba nhóm quyền cơ bản nhằm kiểm soát hành vi truy cập:

- **Chủ sở hữu (Owner)** – Người tạo ra tệp hoặc được gán quyền sở hữu.
- **Nhóm (Group)** – Một nhóm người dùng được gán quyền nhất định với tệp.
- **Người khác (Others)** – Tất cả người dùng còn lại trong hệ thống.

**Ba loại quyền cơ bản:**

| Ký hiệu | Quyền | Giá trị (số học) |
|---|---|---|
| r | Đọc (read) | 4 |
| w | Ghi (write) | 2 |
| x | Thực thi (execute) | 1 |

# 2 Báo cáo và Trả lời câu hỏi Lab 1

## 2.1 Giới thiệu chung

Trong quá trình học tập và tìm hiểu về hệ điều hành Linux, việc nắm vững các thao tác cơ bản trong môi trường dòng lệnh (Command Line Interface – CLI) đóng vai trò hết sức quan trọng. Đây không chỉ là nền tảng giúp người học hiểu rõ hơn về cơ chế hoạt động của hệ thống, mà còn là cơ sở để thực hiện các tác vụ quản trị, lập trình và tự động hóa sau này.

Phần này của báo cáo sẽ dựa trên cơ sở lý thuyết đã trình bày ở chương trước để trả lời chi tiết các câu hỏi trong **Lab 1 – Introduction to Linux**. Cụ thể, nội dung tập trung vào việc:

- Phân tích đặc điểm của các loại `shell` trong Linux và vai trò của chúng.
- Thực hành các lệnh cơ bản như `ls` với các tùy chọn khác nhau, nhằm hiểu cách hệ thống tổ chức và hiển thị tệp tin.
- So sánh và áp dụng kỹ thuật **chuyển hướng dữ liệu (redirection)** và **piping** trong thực tế.
- Tìm hiểu sự khác biệt giữa hai lệnh `sudo` và `su` trong việc quản lý quyền truy cập của người dùng.
- Đánh giá mức độ an toàn của việc phân quyền, đặc biệt là quyền truy cập `777`, đối với các tệp và dịch vụ quan trọng trong hệ thống.

## 2.2 Các loại Shell phổ biến trong hệ điều hành Linux

**Câu hỏi:**
**Hãy liệt kê một số loại `shell` phổ biến được sử dụng trong hệ điều hành Linux và trình bày các đặc điểm nổi bật, ưu điểm của từng loại.**

***Trả lời:***

Trong hệ điều hành **Linux/UNIX**, *shell* đóng vai trò là giao diện dòng lệnh (**CLI – Command Line Interface**) giữa người dùng và nhân hệ điều hành (**kernel**). Shell tiếp nhận lệnh từ người dùng, phân tích cú pháp và chuyển lệnh cho kernel để thực thi. Từ khi UNIX ra đời, nhiều loại shell đã được phát triển nhằm tối ưu hóa khả năng tương tác, lập trình và tự động hóa. Dưới đây là phân tích chi tiết các loại shell tiêu biểu:

### Bourne Shell (sh)

**Nguồn gốc và phát triển:** Bourne Shell được phát triển bởi **Stephen Bourne** tại *AT&T Bell Labs* vào năm 1977. Đây là shell chuẩn đầu tiên trong UNIX, đặt nền móng cho các shell sau này.

**Đặc điểm kỹ thuật:**

- Cú pháp đơn giản, ít phức tạp, tập trung vào thực thi lệnh hệ thống.
- Không hỗ trợ các tính năng hiện đại như tự động hoàn thành (tab completion) hoặc lịch sử lệnh (command history).
- Hỗ trợ thực hiện các lệnh hệ thống, biến môi trường và lập trình shell cơ bản.

**Ưu điểm:**

- Tốc độ thực thi nhanh, phù hợp cho các tác vụ hệ thống nền tảng, tiêu tốn ít tài nguyên.

- Tính tương thích cao, hầu hết các shell khác đều được phát triển dựa trên chuẩn của Bourne Shell như Bash, Korn, Zsh.
- Thích hợp cho các script hệ thống chạy trong môi trường hạn chế (minimal system).

**Hạn chế:**

- Khó sử dụng trong môi trường tương tác, thiếu tính năng tiện ích.
- Không có cơ chế command editing và job control.

**Ứng dụng thực tế:** Bourne Shell vẫn được sử dụng trong các script hệ thống UNIX truyền thống và hệ thống nhúng (embedded systems) do tính nhẹ và ổn định cao.

## Bash (Bourne Again Shell)

**Nguồn gốc và phát triển:** Bash do **Brian Fox** phát triển cho dự án *GNU* năm 1989, là phiên bản mở rộng mạnh mẽ của Bourne Shell.

**Đặc điểm kỹ thuật:**

- Có khả năng xử lý mảng, biểu thức điều kiện, vòng lặp, và hàm – hỗ trợ tốt cho lập trình shell script phức tạp.
- Hỗ trợ lịch sử lệnh (command history), hoàn thành tự động (auto-completion) và biên tập dòng lệnh nâng cao.
- Hỗ trợ toán tử số học, logic, và chuỗi ký tự và cung cấp các biến đặc biệt (như ?,,!, ...) giúp theo dõi và kiểm soát tiến trình.

**Ưu điểm:**

- Là shell mặc định trong hầu hết các bản phân phối Linux (Ubuntu, Fedora, CentOS, v.v.).
- Tính tương thích cao với các shell Bourne cũ, phù hợp lập trình shell script, DevOps, CI/CD.
- Giao diện thân thiện, dễ học, dễ sử dụng, cộng đồng hỗ trợ rộng rãi.

**Hạn chế:**

- Dung lượng và mức tiêu thụ tài nguyên cao hơn Bourne Shell.

**Ứng dụng thực tế:** Là shell mặc định của hầu hết các bản Linux hiện nay; thường được sử dụng trong lập trình shell script, quản trị hệ thống và DevOps automation (CI/CD).

## C Shell (csh) và Tcsh

**Nguồn gốc và phát triển:** Được phát triển bởi Bill Joy (người sáng lập Sun Microsystems) phát triển vào cuối thập niên 1970. Tên gọi "C Shell" xuất phát từ cú pháp tương tự ngôn ngữ lập trình C. Phiên bản mở rộng là tcsh (tenex C shell).

**Đặc điểm kỹ thuật:**

- Hỗ trợ cú pháp lập trình dạng C, giúp dễ hiểu với lập trình viên ngôn ngữ C.

- Cung cấp tính năng alias và history sớm nhất trong các shell.
- Tcsh bổ sung thêm khả năng hoàn thành lệnh tự động và chỉnh sửa dòng lệnh.

**Ưu điểm:**

- Giao diện thân thiện với lập trình viên C.
- Tốc độ thực thi cao, dễ dàng quản lý nhiều tiến trình cùng lúc.

**Hạn chế:**

- Khó viết shell script phức tạp, do cú pháp ít linh hoạt hơn bash.
- Không tương thích hoàn toàn với Bourne Shell scripts.

**Ứng dụng thực tế:** Được sử dụng trong môi trường học thuật và nghiên cứu khoa học, hoặc khi cần tương tác nhanh với hệ thống qua dòng lệnh.

## Korn Shell (ksh)

**Nguồn gốc và phát triển:** Korn Shell do **David Korn** tại *AT&T Bell Labs* phát triển đầu thập niên 1980, kết hợp ưu điểm của Bourne Shell (`sh`) và C Shell (`csh`).

**Đặc điểm kỹ thuật:**

- Hỗ trợ lập trình shell mạnh mẽ với cấu trúc điều khiển phong phú như hàm, mảng, alias, history và xử lý tiến trình nền.
- Có cơ chế command history và alias tương tự Bash.

**Ưu điểm:**

- Hiệu năng cao, phù hợp cho hệ thống máy chủ cần xử lý nhiều tiến trình.
- Thường được sử dụng trong môi trường doanh nghiệp UNIX (như IBM AIX, HP-UX).

**Hạn chế:**

- Phổ biến kém hơn Bash trong các bản phân phối Linux.
- Một số tính năng độc quyền, không tương thích với các shell khác.

## Z Shell (zsh)

**Nguồn gốc và phát triển:** Z Shell do **Paul Falstad** phát triển năm 1990, kết hợp các ưu điểm của Bash, Tcsh và Ksh.

**Đặc điểm kỹ thuật:**

- Cung cấp các tính năng nâng cao về tự động hoàn thành lệnh, sửa lỗi cú pháp tự động, và gợi ý lệnh thông minh.
- Giao diện thân thiện, hỗ trợ tùy chỉnh prompt, plugin, và theme (qua công cụ Oh My Zsh).
- Có khả năng quản lý phiên làm việc và xử lý lệnh mạnh mẽ hơn Bash.

**Ưu điểm:**

- Cấu hình linh hoạt, thẩm mỹ và hiệu quả cao trong môi trường
- Tối ưu cho người dùng cá nhân và nhà phát triển phần mềm.
- Phù hợp với lập trình viên hiện đại, nhất là trong môi trường phát triển đa nền tảng.

**Hạn chế:**

- Cấu hình ban đầu phức tạp, cần người dùng có kinh nghiệm.
- Dễ bị "quá tải" tính năng nếu cài nhiều plugin không cần thiết.

**Ứng dụng thực tế:** Zsh đang trở thành shell mặc định của macOS (từ phiên bản Catalina) và được ưa chuộng trong giới lập trình viên, DevOps, Data Engineer nhờ sự linh hoạt và tính thẩm mỹ cao.

## 2.3 Thực hành và phân tích lệnh `ls`

**Bài tập:**
**Hãy thử nghiệm các tùy chọn khác nhau của lệnh `ls` (ví dụ: `-l`, `-a`, `-la`, ...), quan sát và phân tích ý nghĩa của kết quả đầu ra. Ngoài ra, hãy thử kết hợp nhiều tùy chọn cùng lúc và đánh giá sự khác biệt.**

***Trả lời:***

**Cú pháp tổng quát của lệnh `ls`:**

```
ls [options] [parameters]
```

- **options (tùy chọn):** Thay đổi hành vi hiển thị của lệnh.
- **parameters (tham số):** Thư mục hoặc tệp muốn thao tác; nếu bỏ trống, `ls` sử dụng thư mục hiện tại.

**Các tùy chọn thường dùng:**

| Lệnh | Giải thích |
|---|---|
| ls | Liệt kê tên các file và thư mục trong thư mục hiện tại |
| ls -l | Liệt kê chi tiết (quyền, chủ sở hữu, kích thước, thời gian sửa đổi) |
| ls -a | Hiển thị tất cả file, bao gồm file ẩn (bắt đầu bằng '.') |
| ls -la | Kết hợp chi tiết + hiển thị file ẩn |
| ls -h | Hiển thị kích thước file dễ đọc (KB, MB, GB) |
| ls -t | Sắp xếp theo thời gian sửa đổi, mới → cũ |
| ls -r | Đảo ngược thứ tự hiển thị |
| ls -R | Liệt kê đệ quy tất cả thư mục con |
| ls -S | Sắp xếp theo kích thước file, lớn → nhỏ |
| ls –color=auto | Tô màu output theo loại file |

**Bảng 1:** *Các lệnh `ls` phổ biến trong Linux*

**Lưu ý:** Các tùy chọn có thể kết hợp, ví dụ: `ls -lah` kết hợp `-l`, `-a`, `-h`.

**Thí nghiệm thực tế:**

- **Lệnh cơ bản:**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls
```

**Kết quả:** Liệt kê tệp và thư mục trong thư mục hiện tại, không hiển thị chi tiết và không tệp ẩn.

Danh sách file

```
Report_Lab1.pdf  calc_pro.sh        calc_simple_v2.sh   document.txt
calc.sh          calc_simple_v1.sh  compare_2num.sh     compare_loop.sh
```

- **Liệt kê chi tiết: ls -l**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -l
```

**Ví dụ kết quả:**

Danh sách file

```
 total 44
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
-rwxr-xr-x 1 vopham05 vopham05 6206 Oct 12 00:13 calc.sh
-rwxr-xr-x 1 vopham05 vopham05 9078 Oct 10 17:32 calc_pro.sh
-rwxr-xr-x 1 vopham05 vopham05 2031 Oct 10 18:34 calc_simple_v1.sh
-rwxr-xr-x 1 vopham05 vopham05 1904 Oct 10 18:31 calc_simple_v2.sh
-rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
-rwxr-xr-x 1 vopham05 vopham05 1113 Oct 10 18:45 compare_for.sh
-rwxr-xr-x 1 vopham05 vopham05 1260 Oct 10 18:43 compare_loop.sh
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
-rwxr-xr-x 1 vopham05 vopham05 3903 Oct 10 18:52 student_average.sh
```

**Phân tích:**

- `d` = thư mục, `-` = tệp: Tất cả file trong output đều là tệp `-`.
- Quyền truy cập: `r` = đọc, `w` = ghi, `x` = thực thi.
- Chủ sở hữu và nhóm: `vopham05 vopham05`.
- Kích thước file: 0 bytes (Report_Lab1.pdf), 6206 bytes (calc.sh), 9078 bytes (calc_pro.sh).
- Ngày giờ chỉnh sửa cuối như Oct 12 15:17 cho Report_Lab1.pdf.

- **Hiển thị tệp ẩn: ls -a**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -a
```

**Kết quả:**

Danh sách file

```
.   .calc_ans   Report_Lab1.pdf  calc_pro.sh   document.txt
..  .calc_hist  calc.sh  calc_simple_v1.sh   student_average.sh
```

- `.` = Thư mục hiện tại, tham chiếu đến thư mục đang làm việc.
- `..` = Thư mục cha, tham chiếu đến thư mục chứa thư mục hiện tại.
- `.calc_ans`, `.calc_hist` = Các thư mục ẩn, thường dùng để lưu trữ kết quả, lịch sử tính toán hoặc dữ liệu tạm của Lab.
- `.pdf`, `.sh .txt` = Các file dữ liệu và script.

- **Kết hợp nhiều tùy chọn: ls -la**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -la
```

**Ví dụ kết quả:**

Danh sách file

```
 total 60
drwxr-xr-x 2 vopham05 vopham05 4096 Oct 12 15:20 .
drwxr-x--- 5 vopham05 vopham05 4096 Oct 10 18:27 ..
-rw-r--r-- 1 vopham05 vopham05    2 Oct  8 15:11 .calc_ans
-rw-r--r-- 1 vopham05 vopham05   50 Oct  8 15:11 .calc_hist
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
-rwxr-xr-x 1 vopham05 vopham05 6206 Oct 12 00:13 calc.sh
-rwxr-xr-x 1 vopham05 vopham05 9078 Oct 10 17:32 calc_pro.sh
-rwxr-xr-x 1 vopham05 vopham05 2031 Oct 10 18:34 calc_simple_v1.sh
-rwxr-xr-x 1 vopham05 vopham05 1904 Oct 10 18:31 calc_simple_v2.sh
-rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
-rwxr-xr-x 1 vopham05 vopham05 1113 Oct 10 18:45 compare_for.sh
-rwxr-xr-x 1 vopham05 vopham05 1260 Oct 10 18:43 compare_loop.sh
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
-rwxr-xr-x 1 vopham05 vopham05 3903 Oct 10 18:52 student_average.sh
```

- Liệt kê chi tiết các tệp kèm thêm tệp ẩn.

- **Hiển thị kích thước dễ đọc: ls -lh**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -lh
```

**Ví dụ kết quả:**

Danh sách file

```
total 44K
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
```

```
-rwxr-xr-x 1 vopham05 vopham05 6.1K Oct 12 00:13 calc.sh
-rwxr-xr-x 1 vopham05 vopham05 8.9K Oct 10 17:32 calc_pro.sh
-rwxr-xr-x 1 vopham05 vopham05 2.0K Oct 10 18:34 calc_simple_v1.sh
-rwxr-xr-x 1 vopham05 vopham05 1.9K Oct 10 18:31 calc_simple_v2.sh
-rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
-rwxr-xr-x 1 vopham05 vopham05 1.1K Oct 10 18:45 compare_for.sh
-rwxr-xr-x 1 vopham05 vopham05 1.3K Oct 10 18:43 compare_loop.sh
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
-rwxr-xr-x 1 vopham05 vopham05 3.9K Oct 10 18:52 student_average.sh
```

- Kích thước tệp được hiển thị theo đơn vị KB, MB.

- **Đảo ngược thứ tự: ls -ltr**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltr
```

**Ví dụ kết quả:**

Danh sách file

```
total 44
-rwxr-xr-x 1 vopham05 vopham05 9078 Oct 10 17:32 calc_pro.sh
-rwxr-xr-x 1 vopham05 vopham05 1904 Oct 10 18:31 calc_simple_v2.sh
-rwxr-xr-x 1 vopham05 vopham05 2031 Oct 10 18:34 calc_simple_v1.sh
-rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
-rwxr-xr-x 1 vopham05 vopham05 1260 Oct 10 18:43 compare_loop.sh
-rwxr-xr-x 1 vopham05 vopham05 1113 Oct 10 18:45 compare_for.sh
-rwxr-xr-x 1 vopham05 vopham05 3903 Oct 10 18:52 student_average.sh
-rwxr-xr-x 1 vopham05 vopham05 6206 Oct 12 00:13 calc.sh
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
```

- Liệt kê chi tiết theo thời gian từ cũ đến mới.

- **Lệnh kết hợp:** `ls -lth`

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -lth
```

**Ví dụ kết quả:**

Danh sách file

```
total 44K
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
-rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
-rwxr-xr-x 1 vopham05 vopham05 6.1K Oct 12 00:13 calc.sh
-rwxr-xr-x 1 vopham05 vopham05 3.9K Oct 10 18:52 student_average.sh
```

```
-rwxr-xr-x 1 vopham05 vopham05 1.1K Oct 10 18:45 compare_for.sh
-rwxr-xr-x 1 vopham05 vopham05 1.3K Oct 10 18:43 compare_loop.sh
-rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
-rwxr-xr-x 1 vopham05 vopham05 2.0K Oct 10 18:34 calc_simple_v1.sh
-rwxr-xr-x 1 vopham05 vopham05 1.9K Oct 10 18:31 calc_simple_v2.sh
-rwxr-xr-x 1 vopham05 vopham05 8.9K Oct 10 17:32 calc_pro.sh
```

- Lệnh `ls -lth` hiển thị chi tiết các file, dung lượng dễ đọc, sắp xếp theo thời gian sửa đổi gần nhất

- **Lệnh kết hợp: ls -ltsh**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltsh
```

**Ví dụ kết quả:**

Danh sách file

```
total 44K
  0 -rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:20 document.txt
  0 -rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
8.0K -rwxr-xr-x 1 vopham05 vopham05 6.1K Oct 12 00:13 calc.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 3.9K Oct 10 18:52 student_average.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 1.1K Oct 10 18:45 compare_for.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 1.3K Oct 10 18:43 compare_loop.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 2.0K Oct 10 18:34 calc_simple_v1.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 1.9K Oct 10 18:31 calc_simple_v2.sh
 12K -rwxr-xr-x 1 vopham05 vopham05 8.9K Oct 10 17:32 calc_pro.sh
```

**Các tùy chọn:**

- `-l` : Liệt kê chi tiết (quyền, owner, group, kích thước, thời gian sửa).
- `-t` : Sắp xếp theo thời gian sửa đổi, mới nhất lên đầu.
- `-h` : Hiển thị kích thước dễ đọc (KB, MB...).
- `-s` : Hiển thị kích thước file theo block (thường 1 block $\approx 1KB$).

- Lệnh `ls -ltsh` dùng để xem danh sách file chi tiết, vừa hiển thị dung lượng dễ đọc, vừa sắp xếp file mới nhất lên đầu, tiện kiểm tra các file lớn hoặc vừa thay đổi.

## 2.4 So sánh kỹ thuật chuyển hướng đầu ra và Piping

**Câu hỏi:**
**Hãy so sánh sự khác biệt giữa kỹ thuật chuyển hướng đầu ra (> và >>) với kỹ thuật Piping (|) trong Linux. Phân tích trường hợp sử dụng phù hợp của từng kỹ thuật và minh họa bằng ví dụ cụ thể.**

***Trả lời:***

- **CHUYỂN HƯỚNG ĐẦU RA (REDIRECTION)**
  - Đưa kết quả xuất ra từ lệnh vào file thay vì hiển thị trên terminal. Giúp lưu dữ liệu để tham khảo hoặc sử dụng lại sau này.
  - **Các dạng phổ biến:**
    - > : Ghi đè file (nếu file tồn tại, nội dung cũ sẽ bị xóa).
    - >> : Ghi nối vào cuối file, giữ nguyên dữ liệu cũ.
  - **Ví dụ minh họa:**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltsh > file_list.txt
echo "Hello World!">> file_list.txt
```

**Kết quả:**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ cat file_list.txt
```

Danh sách file

```
total 48
 0 -rw-r--r-- 1 vopham05 vopham05    0 Oct 12 17:02 file_list.txt
 0 -rw-r--r-- 1 vopham05 vopham05    0 Oct 12 17:00 document.txt
 4 -rw-r--r-- 1 vopham05 vopham05  767 Oct 12 17:00 documet.txt
 0 -rw-r--r-- 1 vopham05 vopham05    0 Oct 12 15:17 Report_Lab1.pdf
 8 -rwxr-xr-x 1 vopham05 vopham05 6206 Oct 12 00:13 calc.sh
 4 -rwxr-xr-x 1 vopham05 vopham05 3903 Oct 10 18:52 student_average.sh
 4 -rwxr-xr-x 1 vopham05 vopham05 1113 Oct 10 18:45 compare_for.sh
 4 -rwxr-xr-x 1 vopham05 vopham05 1260 Oct 10 18:43 compare_loop.sh
 4 -rwxr-xr-x 1 vopham05 vopham05  780 Oct 10 18:38 compare_2num.sh
 4 -rwxr-xr-x 1 vopham05 vopham05 2031 Oct 10 18:34 calc_simple_v1.sh
 4 -rwxr-xr-x 1 vopham05 vopham05 1904 Oct 10 18:31 calc_simple_v2.sh
12 -rwxr-xr-x 1 vopham05 vopham05 9078 Oct 10 17:32 calc_pro.sh
Hello World!
```

  - **Trường hợp sử dụng:**
    - Lưu kết quả chạy lệnh để tham khảo sau
    - Tạo báo cáo hoặc log file

  - Khi muốn kết quả tồn tại lâu dài

- **Ưu điểm:** Kết quả lưu lại lâu dài, dễ dàng chia sẻ
- **Nhược điểm:** Không xử lý dữ liệu theo luồng; dùng > có thể mất dữ liệu cũ

- **PIPING (|)**
  - Truyền trực tiếp đầu ra của lệnh này làm đầu vào cho lệnh khác, tạo chuỗi xử lý dữ liệu liên tục mà không cần file trung gian
  - **Ví dụ minh họa 1:**
    - Lọc file có tên chứa "calc")

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltsh ls -ltsh | grep calc
```

**Kết quả:**

Danh sách file

```
 8.0K -rwxr-xr-x 1 vopham05 vopham05 6.1K Oct 12 00:13 calc.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 2.0K Oct 10 18:34 calc_simple_v1.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 1.9K Oct 10 18:31 calc_simple_v2.sh
 12K -rwxr-xr-x 1 vopham05 vopham05 8.9K Oct 10 17:32 calc_pro.sh
```

  - **Ví dụ minh họa 2:**
    - Lọc 3 file có tên chứa "calc" và sắp xếp theo kích thước từ lớn đến nhỏ

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltsh | grep calc | sort -h -r | head -3
```

**Kết quả:**

Danh sách file

```
 12K -rwxr-xr-x 1 vopham05 vopham05 8.9K Oct 10 17:32 calc_pro.sh
8.0K -rwxr-xr-x 1 vopham05 vopham05 6.1K Oct 12 00:13 calc.sh
4.0K -rwxr-xr-x 1 vopham05 vopham05 2.0K Oct 10 18:34 calc_simple_v1.sh
```

  - **Trường hợp sử dụng:**
    - Xử lý dữ liệu ngay lập tức, không cần file trung gian
    - Lọc, sắp xếp, đếm, hoặc trích xuất thông tin từ lệnh khác
    - Khi muốn nối nhiều lệnh thành chuỗi (pipeline)
  - **Ưu điểm:** Tiết kiệm bộ nhớ và thời gian, xử lý dữ liệu nhanh, dễ dàng kết hợp nhiều lệnh
  - **Nhược điểm:** Kết quả không lưu lâu dài, khó kiểm tra lại nếu pipeline phức tạp

- **Kết hợp cả hai kỹ thuật**

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ ls -ltsh | grep calc | sort -h -r > file_list.txt
```

  - Vừa xử lý dữ liệu trực tiếp (chuỗi lệnh), vừa lưu kết quả vào file để sử dụng sau
  - Thường dùng trong báo cáo, phân tích log, thống kê file

## 2.5 So sánh giữa lệnh `sudo` và `su`

**Câu hỏi:**
**Hãy trình bày sự khác biệt giữa hai lệnh `sudo` và `su`, làm rõ cơ chế hoạt động, phạm vi quyền hạn và tình huống sử dụng của mỗi lệnh.**

***Trả lời:***

- **Khái niệm:**
  - `su` **(switch user):** Chuyển sang một tài khoản khác, mặc định là root. Toàn bộ phiên làm việc sẽ chạy với quyền của user đích. Yêu cầu mật khẩu của user đích.
  - `sudo` **(superuser do):** Thực thi một lệnh với quyền root hoặc quyền user khác mà không cần đăng nhập vào tài khoản đó. Quyền được quản lý qua `/etc/sudoers`. Yêu cầu mật khẩu của user hiện tại.

- **Cơ chế hoạt động:**
  - `su`**:** Khi chạy `su`, một shell mới được mở dưới user đích, mọi lệnh trong shell đó chạy với quyền user mới.

```
vopham05@DESKTOP-2HD73M6:~/LAB1$ su          # chuyển sang root
vopham05@DESKTOP-2HD73M6:~/LAB1$ su vopham05 # chuyển sang user vopham05
```

  - `sudo`**:** Khi chạy `sudo`, chỉ lệnh đó được thực thi với quyền cao hơn. Sau khi lệnh kết thúc, user trở lại quyền ban đầu. Ví dụ:

```
sudo apt update
sudo cp file1 /root/
sudo -u vopham05 whoami
```

- **Phạm vi quyền hạn:**

| Tiêu chí | `su` | `sudo` |
|---|---|---|
| Quyền hạn | Toàn quyền của user đích | Quyền hạn hạn chế, theo sudoers |
| Phạm vi | Toàn bộ phiên shell | Chỉ lệnh được thực thi |
| Mật khẩu yêu cầu | Mật khẩu user đích | Mật khẩu user hiện tại |
| Theo dõi / log | Không log chi tiết | Log chi tiết từng lệnh |
| Bảo mật | Nguy cơ nếu mật khẩu root bị lộ | An toàn hơn, phân quyền chi tiết |
| Thao tác | Phải thoát shell để quay lại quyền cũ | Tự động trở về quyền user ban đầu sau lệnh |

- **Ưu nhược điểm:**
  - `su`:
    - Ưu điểm: Toàn quyền root, tiện thao tác nhiều lệnh liên tiếp.
    - Nhược điểm: Cần mật khẩu user đích, không log lệnh, rủi ro bảo mật cao.
  - `sudo`:
    - Ưu điểm: Thực thi lệnh riêng lẻ với quyền cao, log chi tiết, phân quyền linh hoạt.
    - Nhược điểm: Phải thêm `sudo` trước mỗi lệnh, cấu hình sudoers phức tạp.
- **Tình huống sử dụng:**
  - `su`: Khi cần toàn quyền root để chạy nhiều lệnh liên tiếp; quản trị hệ thống dài hạn.
  - `sudo`: Khi cần thực thi lệnh cụ thể với quyền root; bảo mật, kiểm soát quyền user; môi trường nhiều user.

## 2.6 Phân tích quyền truy cập 777 đối với các dịch vụ quan trọng

**Câu hỏi:**
**Hãy thảo luận về tác động và rủi ro bảo mật khi thiết lập quyền truy cập 777 cho các dịch vụ quan trọng như web hosting, cơ sở dữ liệu, hoặc các tệp hệ thống. Đưa ra quan điểm và đề xuất cách thiết lập quyền an toàn hơn trong thực tế.**

***Trả lời:***

- **Quyền 777 là gì và cơ chế hoạt động:**
  - Quyền 777 cho phép **Owner, Group và Others** đều có quyền đọc (r), ghi (w) và thực thi (x) trên file hoặc thư mục.
  - Khi áp dụng, **mọi user trên hệ thống** đều có thể đọc, chỉnh sửa, xóa hoặc thực thi file/thư mục mà không cần phân quyền đặc biệt.
  - Ký hiệu hiển thị: `-rwxrwxrwx`.
- **Rủi ro bảo mật đối với các dịch vụ quan trọng:**
  - **Web Hosting:**
    - File và thư mục web nếu đặt 777 có thể cho phép **mọi user, kể cả attacker** upload script độc hại hoặc chỉnh sửa mã nguồn.
    - Rủi ro thực tế: website bị *deface*, cài *backdoor*, hoặc đánh cắp dữ liệu người dùng.
  - **Cơ sở dữ liệu:**
    - File dữ liệu hoặc file cấu hình đặt 777 sẽ cho phép **mọi user** đọc, ghi hoặc xóa dữ liệu quan trọng.
    - Hậu quả có thể làm thông tin nhạy cảm bị lộ, dữ liệu bị thay đổi hoặc phá hủy, dẫn đến gián đoạn dịch vụ và mất uy tín.
  - **Tệp hệ thống:**
    - File hệ thống quan trọng đặt 777 cho phép **mọi user chỉnh sửa hoặc xóa file**, tạo nguy cơ chiếm quyền root, tạo tài khoản giả hoặc phá hủy hệ thống.

    * Đây là **lỗ hổng nghiêm trọng**, đặc biệt trên hệ thống multi-user hoặc server truy cập Internet.

- **Cách thiết lập quyền an toàn hơn:**
  - Nguyên tắc chung là chỉ nên cấp **quyền tối thiểu cần thiết** (Principle of Least Privilege), tránh mở rộng quyền cho Others nếu không cần thiết.
  - Thiết lập cụ thể:
    * **Web server:** Thư mục mã nguồn nên để 755 (Owner full quyền, Others chỉ đọc/execute); file script nên để 644 (Owner đọc/ghi, Others chỉ đọc).
    * **Cơ sở dữ liệu:** File cấu hình chứa mật khẩu nên để 600 (chỉ Owner đọc/ghi); thư mục dữ liệu chỉ cấp quyền cho user chạy DB, Others không truy cập.
    * **Tệp hệ thống:** Chỉ root có quyền ghi, các user khác chỉ được phép đọc khi cần thiết.
  - Ví dụ lệnh an toàn:

```
# Thiết lập quyền an toàn cho web server
chown www-data:www-data /var/www/html -R
chmod 755 /var/www/html
chmod 644 /var/www/html/index.php

# Thiết lập quyền cho file cấu hình database
chown mysql:mysql /etc/mysql/my.cnf
chmod 600 /etc/mysql/my.cnf
```

- **Kết luận:**
  - Quyền 777 cực kỳ nguy hiểm cho các dịch vụ quan trọng như web, database và hệ thống.
  - Thực tế cần cân nhắc kỹ quyền theo từng user và dịch vụ, vừa đảm bảo hoạt động vừa tăng cường bảo mật.
  - Áp dụng nguyên tắc quyền tối thiểu cần thiết, kết hợp kiểm tra và audit định kỳ, là cách bảo vệ hệ thống hiệu quả nhất.

## 2.7 Tìm hiểu về Makefile trong quá trình biên dịch chương trình

**Câu 1:**
**Lợi ích của Makefile? Hãy đưa ra ví dụ minh họa**

***Trả lời:***

- **Makefile là gì?**
  - Makefile là một tập tin hướng dẫn công cụ `make` cách biên dịch và liên kết chương trình từ các file nguồn.
  - Thay vì gõ nhiều lệnh biên dịch thủ công, Makefile giúp **tự động hóa toàn bộ quá trình**, đặc biệt hữu ích với các dự án có nhiều file nguồn.
  - Makefile cũng xác định phụ thuộc giữa các file (.c, .o), nên chỉ biên dịch những file thay đổi, giúp tiết kiệm thời gian và công sức.

- **Lợi ích của Makefile**
  - **Tự động hóa quá trình biên dịch:**
    - Khi một file nguồn thay đổi, Makefile chỉ biên dịch **file đó và các file phụ thuộc**, không cần biên dịch lại toàn bộ dự án.
    - Giúp tiết kiệm thời gian, đặc biệt với dự án lớn có nhiều file.
  - **Quản lý phụ thuộc giữa các file:**
    - Makefile xác địnhnmối quan hệ giữa file nguồn (.c) và file đích (.o hoặc executable).
    - Tránh lỗi do thiếu file hoặc biên dịch nhầm thứ tự, đảm bảo chương trình luôn được build đúng.
  - **Dễ bảo trì và mở rộng:**
    - Trong các dự án nhiều file, Makefile giúp tập trung các lệnh build, liên kết và dọn dẹp (clean) trong một tập tin duy nhất.
    - Khi thêm file mới hoặc thay đổi cấu trúc dự án, chỉ cần sửa Makefile, không phải nhớ từng lệnh biên dịch thủ công.
  - **Tiết kiệm thời gian và giảm lỗi:**
    - Chỉ biên dịch những file thay đổi → **tiết kiệm thời gian build**.
    - Không cần gõ nhiều lệnh gcc/clang thủ công → **giảm nguy cơ sai sót**.
- **Ví dụ minh họa Makefile**

  **Cấu trúc dự án game nhỏ:**

Game

```
game/
|-- main.c        # file chính
|-- player.c      # các hàm liên quan đến nhân vật
  | player.h
|-- enemy.c       # các hàm liên quan đến kẻ thù
  | enemy.h
|-- Makefile
```

  **Nội dung Makefile:**

Makefile

```
# Compiler và flags
CC = gcc
CFLAGS = -Wall -g

# Object files và executable
OBJ = main.o player.o enemy.o
TARGET = game

# Rule tạo executable từ các object
$(TARGET): $(OBJ)
```

```
 $(CC) $(CFLAGS) -o $(TARGET) $(OBJ)

# Rule tạo file .o từ file .c
%.o: %.c
 $(CC) $(CFLAGS) -c $< -o $@

# Rule dọn dẹp file tạm
clean:
 rm -f $(OBJ) $(TARGET)
```

**Lợi ích của Makefile dựa trên ví dụ:**

- **Tự động hóa quá trình biên dịch:**
  - Khi thay đổi `player.c`, Makefile chỉ biên dịch `player.o` và liên kết lại chương trình `game`.
  - Không cần biên dịch lại `main.o` hay `enemy.o`, giúp tiết kiệm thời gian.
- **Quản lý phụ thuộc giữa các file:**
  - Makefile xác định `game` phụ thuộc vào `main.o`, `player.o`, `enemy.o`.
  - Make tự động kiểm tra file nào thay đổi và build đúng thứ tự, tránh lỗi build nhầm hoặc thiếu file.
- **Dễ bảo trì và mở rộng:**
  - Khi thêm file mới, ví dụ `level.c`, chỉ cần thêm vào biến `OBJ` trong Makefile.
  - Rules `%.o: %.c` và `clean` vẫn hoạt động bình thường, không cần viết lại từng lệnh biên dịch.
- **Tiết kiệm thời gian và giảm lỗi:**
  - Chỉ biên dịch những file thay đổi → giảm thời gian build.
  - Không cần gõ thủ công nhiều lệnh gcc → giảm nguy cơ sai sót.
- **Dọn dẹp dễ dàng:**
  - Rule `clean` xóa các file tạm (`.o`) và executable (`game`).
  - Giúp build lại từ đầu khi cần, đảm bảo môi trường biên dịch luôn sạch sẽ.

**Câu 2:**
**Tại sao lần biên dịch đầu tiên thường mất nhiều thời gian hơn so với các lần biên dịch tiếp theo?**

***Trả lời:***

- **Lần biên dịch đầu tiên:**
  - Khi chạy `makefile` lần đầu, hầu hết các file object (.o) và executable chưa tồn tại trên hệ thống. Makefile phải biên dịch tất cả các file nguồn (.c) thành các file object (.o).
  - Sau khi biên dịch xong các file object, Makefile liên kết tất cả các object để tạo ra chương trình thực thi (executable).
  - Toàn bộ quá trình này tốn nhiều thời gian vì mỗi file phải build từ đầu và không có file object nào để tái sử dụng.

- **Các lần biên dịch tiếp theo:**
  - Makefile kiểm tra thời gian sửa đổi (timestamp) của từng file nguồn so với file object tương ứng.
  - Nếu file nguồn không thay đổi, file object vẫn còn mới hơn → Make bỏ qua việc biên dịch file đó.
  - Nếu file nguồn thay đổi, Make chỉ biên dịch file đó và sau đó liên kết lại toàn bộ chương trình.
  - Những file không thay đổi không phải biên dịch lại, giúp tiết kiệm đáng kể thời gian và tài nguyên.
- **Tóm Lại:**
  - Lần biên dịch đầu tiên lâu vì phải xây dựng toàn bộ từ file nguồn, không có dữ liệu trung gian.
  - Các lần biên dịch tiếp theo nhanh hơn nhờ cơ chế incremental build, chỉ biên dịch những file thay đổi và liên kết lại chương trình.
  - Cơ chế này tiết kiệm thời gian, giảm công sức, hạn chế sai sót so với việc biên dịch thủ công từng file.

**Câu 3:**
**Makefile có cơ chế hỗ trợ cho các ngôn ngữ lập trình khác không? Nếu có, hãy cho ví dụ**

***Trả lời:***

Makefile không giới hạn chỉ sử dụng với ngôn ngữ C/C++ mà còn có thể áp dụng với nhiều ngôn ngữ khác như Python, Java, Rust, Go,... Bản chất của Makefile là xác định **dependencies** giữa các file và lệnh thực thi, do đó có thể dùng để tự động hóa việc build, test hoặc chạy chương trình trong bất kỳ ngôn ngữ nào có lệnh dòng lệnh.

## Ví dụ 1: C++

Giả sử dự án C++ gồm các file:

Game

```
game/
  |-- main.cpp
  |-- player.cpp
  |-- player.h
  |-- enemy.cpp
  |-- enemy.h
```

Một Makefile đơn giản có thể như sau:

Danh sách file

```
# Compiler và flags
CXX = g++
CXXFLAGS = -Wall -g
```

```
# Object files và executable
OBJ = main.o player.o enemy.o
TARGET = game

# Rule tạo executable từ các object
$(TARGET): $(OBJ)
    $(CXX) $(CXXFLAGS) -o $(TARGET) $(OBJ)

# Rule tạo file .o từ file .cpp
%.o: %.cpp
    $(CXX) $(CXXFLAGS) -c $< -o $@

# Rule dọn dẹp
clean:
    rm -f $(OBJ) $(TARGET)
```

**Giải thích:**

- Lần biên dịch đầu tiên sẽ build tất cả file `.cpp` thành object và liên kết thành `game`.
- Các lần biên dịch tiếp theo chỉ biên dịch những file thay đổi, tiết kiệm thời gian nhờ cơ chế incremental build.

## Ví dụ 2: Python

Python không cần biên dịch, nhưng Makefile vẫn hữu ích để tự động hóa chạy script, kiểm tra style code và dọn dẹp môi trường.

Makefile

```
# Rule chạy chương trình Python
run:
    python3 main.py

# Rule kiểm tra style code
lint:
    flake8 src/

# Rule dọn dẹp file cache
clean:
    rm -rf __pycache__ *.pyc
```

**Giải thích:**

- Makefile giúp tự động hóa các tác vụ thường dùng, như chạy chương trình (`make run`), kiểm tra style (`make lint`) và dọn dẹp cache (`make clean`).
- Cơ chế dependencies vẫn có thể áp dụng nếu cần build các file trung gian hoặc tạo output từ nhiều script.

## 2.8 Kết quả thực hành 2 Bài Tập

### 2.8.1 Thực thi Bài tập 3.6

Đoạn code dưới đây triển khai một máy tính dòng lệnh nâng cao theo yêu cầu của bài Lab cho phép thực hiện các phép toán nhị phân và lưu trữ kết quả trước đó để tái sử dụng.

Máy tính này cho phép người dùng nhập trực tiếp các phép toán nhị phân theo định dạng `<số> <toán_tử> <số>`, hỗ trợ năm phép toán cơ bản: Cộng, Trừ, Nhân, Chia và Chia lấy dư (Modulo). Kết quả của phép tính trước đó được lưu trong biến đặc biệt `ANS` và có thể sử dụng lại trong các phép tính tiếp theo.

Mã nguồn của chương trình máy tính dòng lệnh nâng cao có thể tham khảo tại: calc.sh

```
#!/bin/bash
#============================================================
# CALCULATOR - Simple Command Line Calculator
# Supports: +, -, x, /, % and keeps calculation history
#============================================================
#------------------------------------------------------------
# BLOCK 1: INITIALIZATION
#------------------------------------------------------------
# File path to store the result of the previous calculation
ANS_FILE=~/.calc_ans
# File path to store the history of the last 5 calculations
HISTORY_FILE=~/.calc_history
# Create ANS file if it doesn't exist, initialize with 0
if [ ! -f "$ANS_FILE" ]; then
    echo "0" > "$ANS_FILE"
fi
# Create history file if it doesn't exist
if [ ! -f "$HISTORY_FILE" ]; then
    touch "$HISTORY_FILE"
fi
# Clear the screen to start the program
clear
#------------------------------------------------------------
# BLOCK 2: MAIN LOOP
#------------------------------------------------------------
# Infinite loop to continuously receive user input
while true; do
    # Display the prompt ">>" and wait for user input
```

```
        printf ">> "
        read -r input
        #-------------------------------------------------------
        # BLOCK 3: SPECIAL COMMAND HANDLING
        #-------------------------------------------------------
        # EXIT command: exit the program
        if [ "$input" = "EXIT" ]; then
            break
        fi
        # HIST command: display the last 5 calculations
        if [ "$input" = "HIST" ]; then
            cat "$HISTORY_FILE"
            echo ""
            echo "Press any key to continue..."
            read -n 1 -s -r
            clear
            continue
        fi
        # ANS command: display the previous result
        if [ "$input" = "ANS" ]; then
            cat "$ANS_FILE"
            echo ""
            echo "Press any key to continue..."
            read -n 1 -s -r
            clear
            continue
        fi
        #-------------------------------------------------------
        # BLOCK 4: INPUT PARSING AND VALIDATION
        #-------------------------------------------------------
        # Split the input into 3 parts: num1 operator num2
        read -r num1 operator num2 <<< "$input"
        # Replace keyword 'ANS' with the previous calculation result
        if [ "$num1" = "ANS" ]; then
            num1=$(cat "$ANS_FILE")
        fi
```

```
65          if [ "$num2" = "ANS" ]; then
66              num2=$(cat "$ANS_FILE")
67          fi
68          # Check if all 3 parts are entered, otherwise  syntax error
69          if [ -z "$num1" ] || [ -z "$operator" ] || [ -z "$num2" ];
             then
70              echo "SYNTAX ERROR"
71              echo "Press any key to continue..."
72              read -n 1 -s -r
73              clear
74              continue
75          fi
76          # Check if operator is valid (+, -, x, /, %)
77          case "$operator" in
78              "+"|"-"|"/"|"%"|"x")
79                  # Valid operator, continue
80                  ;;
81              *)
82                  # Invalid operator
83                  echo "SYNTAX ERROR"
84                  echo "Press any key to continue..."
85                  read -n 1 -s -r
86                  clear
87                  continue
88                  ;;
89          esac
90
91          #-------------------------------------------------------
92          # BLOCK 5: CALCULATION
93          #-------------------------------------------------------
94          # Check division by zero (for / and %)
95          if ([ "$operator" = "/" ] || [ "$operator" = "%" ]) && [ "
    $num2" = "0" ]; then
96              echo "MATH ERROR"
97              echo "Press any key to continue..."
98              read -n 1 -s -r
```

```
99              clear
100             continue
101         fi
102         # Convert 'x' to '*' so bc can recognize it
103         calc_operator="$operator"
104         if [ "$operator" = "x" ]; then
105             calc_operator="*"
106         fi
107         # Perform the calculation using bc
108         if [ "$operator" = "%" ]; then
109         # Modulus operation, integer result only
110         result=$(echo "scale=0; $num1 $calc_operator $num2" | bc -l)
111         else
112         raw_result=$(echo "scale=10;
113                 $num1 $calc_operator $num2" | bc -l)
114         # Round to 2 decimal places
115         result=$(printf "%.2f" "$raw_result")
116         fi
117
118         #--------------------------------------------------------
119         # BLOCK 6: DISPLAY AND STORE RESULT
120         #--------------------------------------------------------
121         # Print the result to the screen
122         echo "$result"
123         # Save the result to the ANS file for later use
124         echo "$result" > "$ANS_FILE"
125         # Create a history entry in the format
126         history_entry="$input = $result"
127         # Update history file (keep only last 5 entries)
128         echo "$history_entry" > history.tmp
129         cat "$HISTORY_FILE" >> history.tmp
130         head -n 5 history.tmp > "$HISTORY_FILE"
131         rm history.tmp
132
133
134
```

```
        #------------------------------------------------------
        # BLOCK 7: WAIT FOR USER AND CLEAR SCREEN
        #------------------------------------------------------
        # Notify and wait for user to press any key
        echo ""
        echo "Press any key to continue..."
        read -n 1 -s -r

        # Clear the screen to prepare for next calculation
        clear
    done
```

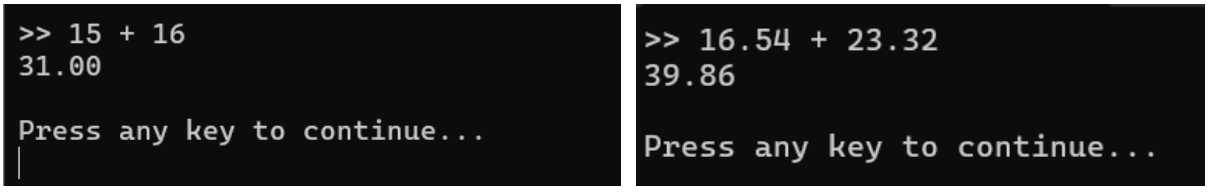
Listing 1: Code in the calc.sh file

**** Kết quả thực thi và minh họa đầu ra***

Trong phần này, chúng tôi trình bày kết quả đầu ra của chương trình `calc.sh` khi thực hiện các phép tính cơ bản bao gồm: phép cộng, phép trừ, phép nhân, phép chia và phép chia lấy dư (modulo).

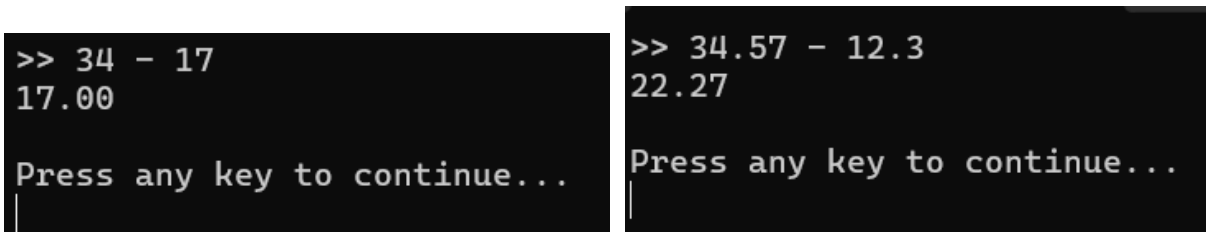
Mỗi ví dụ minh họa cho cách chương trình xử lý dữ liệu đầu vào và hiển thị kết quả tương ứng, qua đó thể hiện tính chính xác và khả năng vận hành ổn định của chương trình.

- **Phép cộng (+):**
  Kết quả thực hiện phép cộng hai số được minh họa như hình dưới đây.



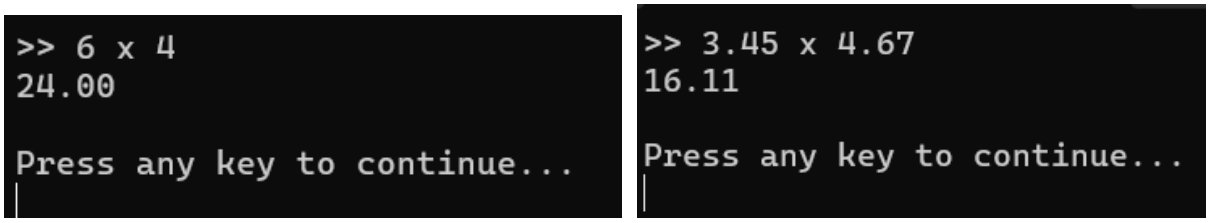

**Hình 1:** *Các kết quả kiểm thử cho Testcase 0*

- **Phép trừ (-):**
  Ví dụ về phép trừ giữa hai số được thể hiện trong hình sau.



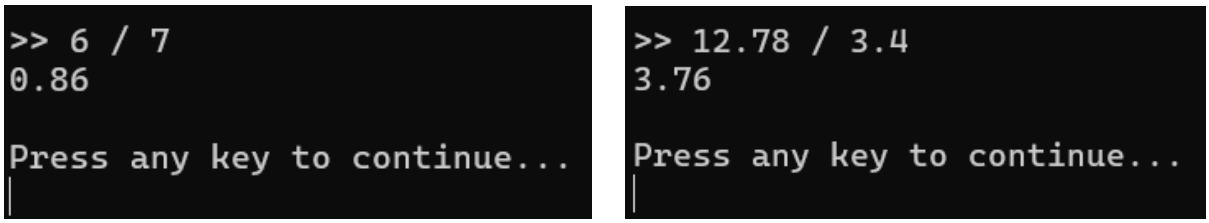

**Hình 2:** *Các kết quả kiểm thử cho Testcase 1*

- **Phép nhân (x):**
  Kết quả của phép nhân được thể hiện ở hình minh họa bên dưới.



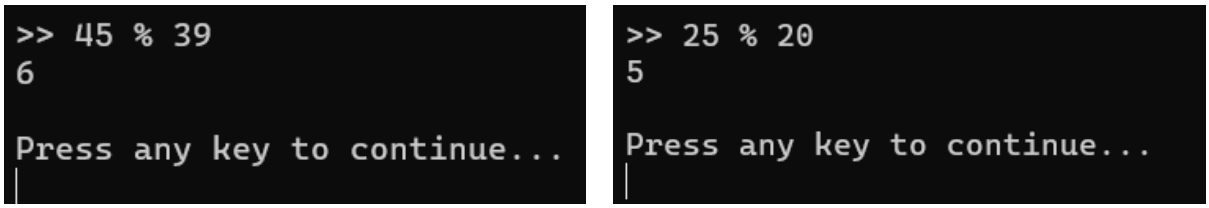

**Hình 3:** *Các kết quả kiểm thử cho Testcase 2*

- **Phép chia (/):**
  Kết quả của phép chia hai số với phần thập phân được làm tròn đến hai chữ số.



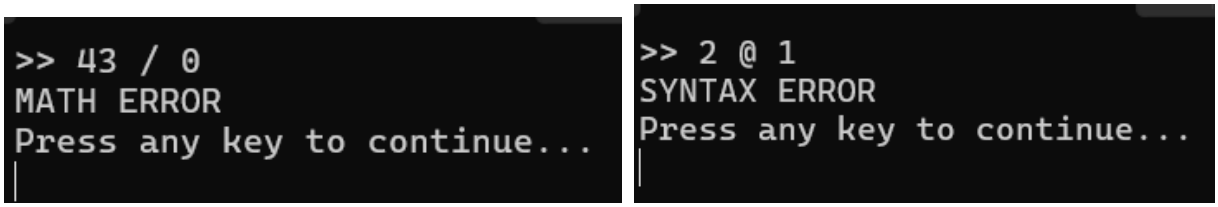

**Hình 4:** *Các kết quả kiểm thử cho Testcase 3*

- **Phép chia lấy dư (%):**
  Ví dụ về phép chia lấy dư được trình bày ở hình dưới đây.




**Hình 5:** *Các kết quả kiểm thử cho Testcase 4*

- **Lỗi chia cho 0 (Math Error) và cú pháp nhập sai (Syntax Error):**

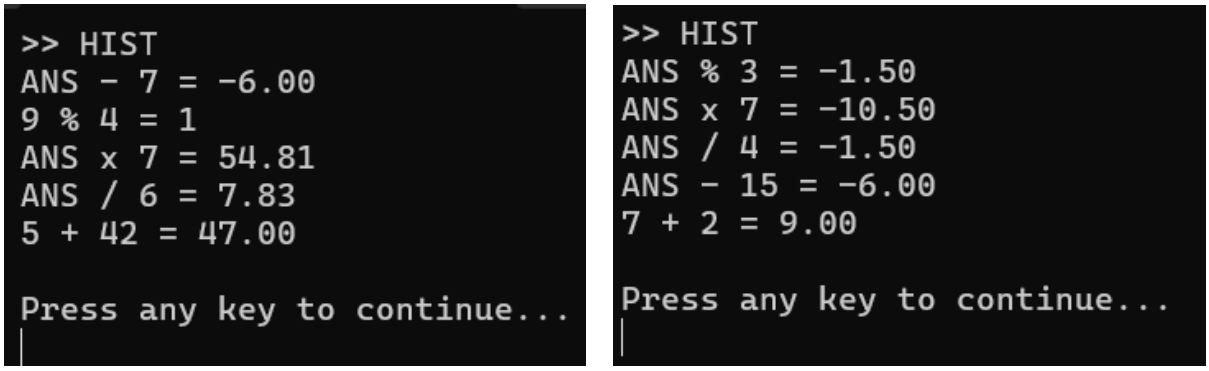



**Hình 6:** *Các kết quả kiểm thử cho Testcase 5*

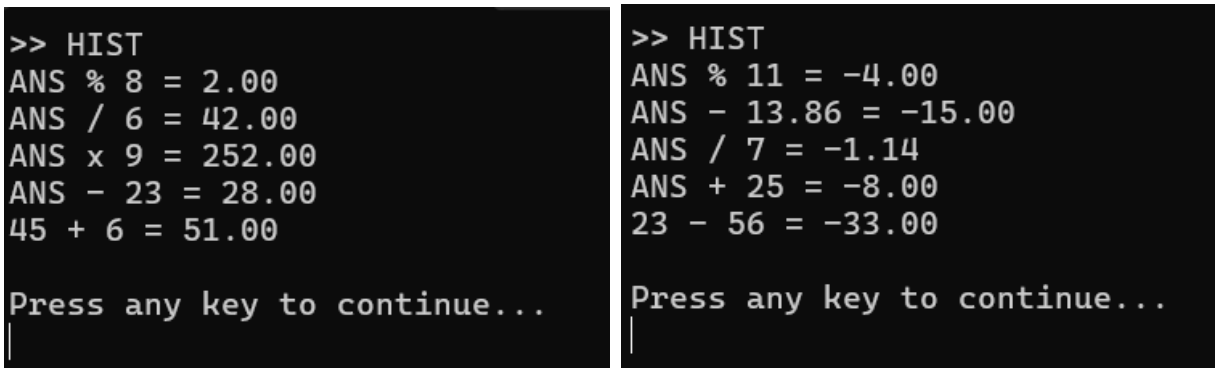
- **Thoát khỏi chương trình (Exit Command):**


**Hình 7:** *Các kết quả kiểm thử cho Testcase 6*

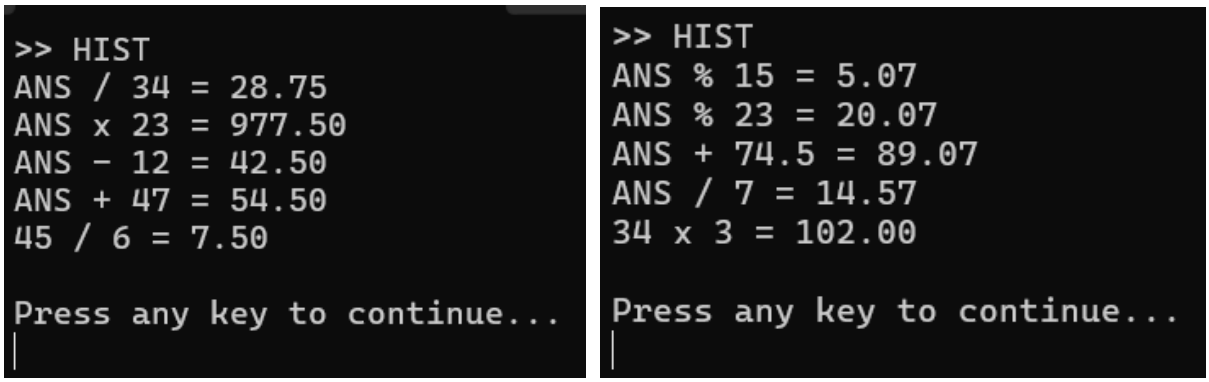
- **Hiển thị lịch sử 5 phép tính gần nhất (Lệnh HIST):**
  Trong chương trình `calc.sh`, người dùng có thể xem lại lịch sử của 5 phép tính gần nhất thông qua lệnh `HIST`. Đặc biệt, chương trình hỗ trợ biến `ANS`, cho phép lưu trữ kết quả của phép tính cuối cùng để sử dụng lại ở các phép tính tiếp theo. Nhờ đó, người dùng có thể thực hiện chuỗi phép toán liên tiếp mà không cần nhập lại toàn bộ giá trị trước đó, giúp thao tác nhanh và linh hoạt hơn.




**Hình 8:** *Các kết quả kiểm thử cho Testcase 7 - 8*




**Hình 9:** *Các kết quả kiểm thử cho Testcase 9 - 10*




**Hình 10:** *Các kết quả kiểm thử cho Testcase 11 - 12*

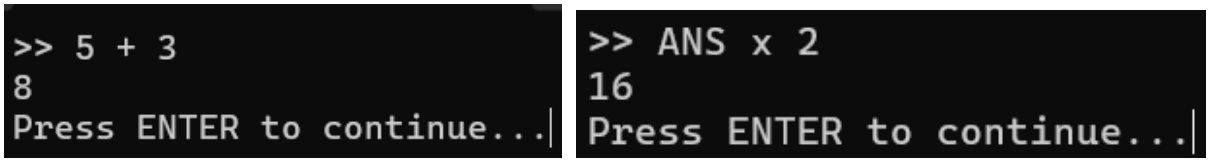
### 2.8.2 Thực thi Bài tập 5.3

Triển khai lại calculator từ Shell Script sang C, với các yêu cầu:

- **Makefile**: tạo `calc`, có hai target `all` và `clean`.
- **Chương trình C**:
  - Main: `calc.c` (nhập/xuất, gọi hàm tính toán)
  - Logic: các file C riêng (ví dụ `operations.c`) (+, -, *, /)
  - Không cần chức năng `HIST`
- **Input/Output**: giống Shell Script, có thể đọc từ bàn phím hoặc `input.txt`.
- **Executable**: `calc` (Windows: `calc.exe`).

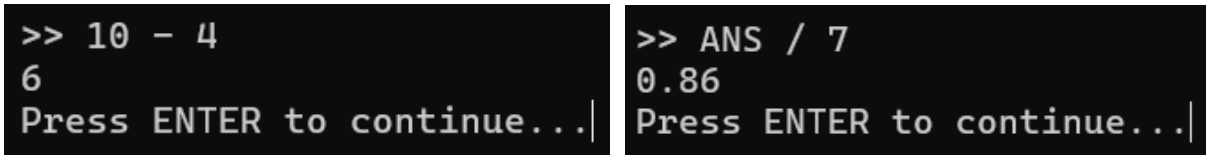
**** Kết quả thực thi và minh họa đầu ra***

- **Testcase 1:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.



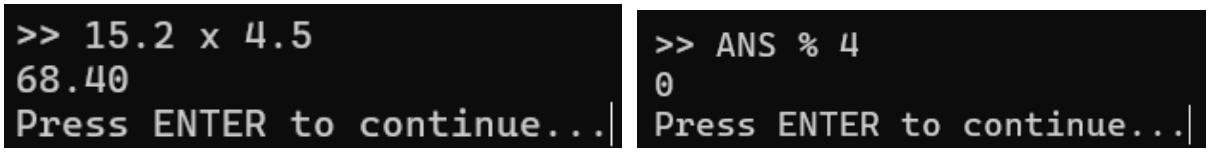

**Hình 11:** *Các kết quả kiểm thử cho Testcase 1*

- **Testcase 2:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.



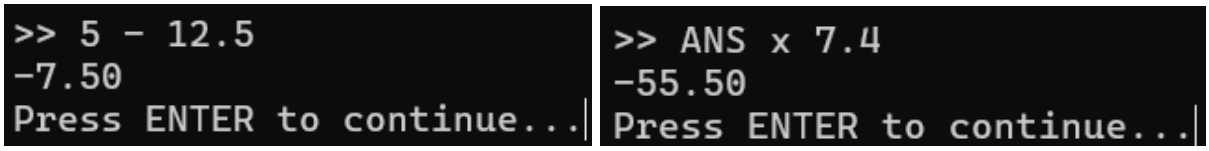

**Hình 12:** *Các kết quả kiểm thử cho Testcase 2*

- **Testcase 3:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.



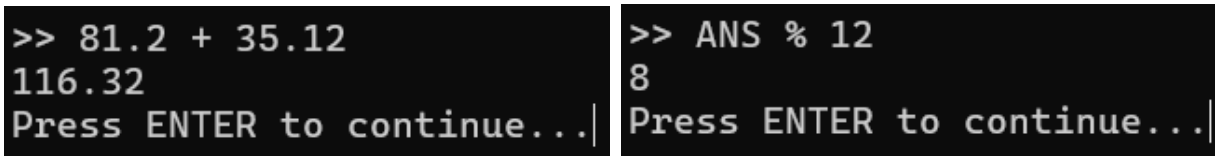

**Hình 13:** *Các kết quả kiểm thử cho Testcase 3*

- **Testcase 4:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.



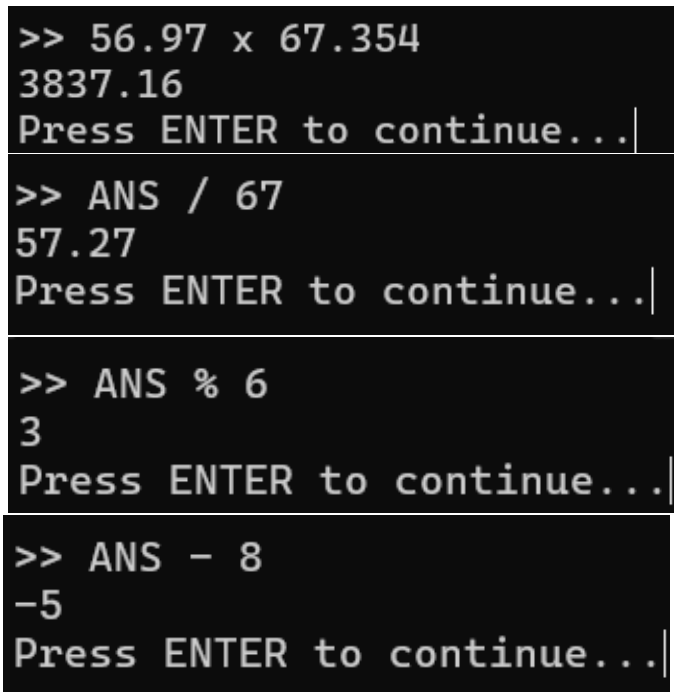

**Hình 14:** *Các kết quả kiểm thử cho Testcase 4*

- **Testcase 5:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.



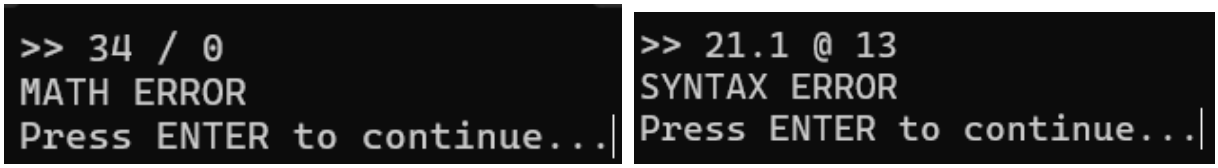

**Hình 15:** *Các kết quả kiểm thử cho Testcase 5*

- **Testcase 6:**
  Kết quả thực hiện phép tính hai số được minh họa như hình dưới đây.




**Hình 16:** *Các kết quả kiểm thử cho Testcase 6*

- **Testcase 6:**
  Lỗi chia cho 0 (Math Error) và cú pháp nhập sai (Syntax Error)




**Hình 17:** *Các kết quả kiểm thử cho Testcase 7*

- **Testcase 7:**
  Thoát khỏi chương trình (Exit Command)

```
>> EXIT
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$
```

- **Kết quả thực thi lệnh `make all` và `make clean`**

```
vopham05@DESKTOP-2HD73
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$ ls -lh
total 20K
-rw-r--r-- 1 vopham05 vopham05  348 Oct 12 19:29 Makefile
-rw-r--r-- 1 vopham05 vopham05 5.9K Oct 12 20:38 calc.c
-rw-r--r-- 1 vopham05 vopham05  253 Oct 12 20:27 operations.c
-rw-r--r-- 1 vopham05 vopham05  193 Oct 12 19:48 operations.h
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$ make all
gcc -Wall -Wextra -std=c99 -c calc.c
gcc -Wall -Wextra -std=c99 -c operations.c
gcc -Wall -Wextra -std=c99 -o calc calc.o operations.o
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$ la -lh
total 52K
-rw-r--r-- 1 vopham05 vopham05  348 Oct 12 19:29 Makefile
-rwxr-xr-x 1 vopham05 vopham05  17K Oct 12 22:42 calc
-rw-r--r-- 1 vopham05 vopham05 5.9K Oct 12 20:38 calc.c
-rw-r--r-- 1 vopham05 vopham05 6.4K Oct 12 22:42 calc.o
-rw-r--r-- 1 vopham05 vopham05  253 Oct 12 20:27 operations.c
-rw-r--r-- 1 vopham05 vopham05  193 Oct 12 19:48 operations.h
-rw-r--r-- 1 vopham05 vopham05 1.6K Oct 12 22:42 operations.o
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$ make clean
rm -f calc calc.o operations.o .calc_ans
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$ ls -lh
total 20K
-rw-r--r-- 1 vopham05 vopham05  348 Oct 12 19:29 Makefile
-rw-r--r-- 1 vopham05 vopham05 5.9K Oct 12 20:38 calc.c
-rw-r--r-- 1 vopham05 vopham05  253 Oct 12 20:27 operations.c
-rw-r--r-- 1 vopham05 vopham05  193 Oct 12 19:48 operations.h
vopham05@DESKTOP-2HD73M6:~/LAB1/EX_5.3_calc$
```

Hình minh họa quá trình biên dịch và làm sạch chương trình bằng Makefile. Ban đầu, trong thư mục chỉ có các tệp mã nguồn gồm `calc.c`, `operations.c`, `operations.h` và tệp `Makefile`.

Khi thực hiện lệnh `make all`, Makefile tiến hành biên dịch toàn bộ chương trình. Trên màn hình hiển thị các dòng lệnh cho thấy quá trình biên dịch từng tệp mã nguồn riêng lẻ, sau đó liên kết chúng lại để tạo ra tệp thực thi có tên `calc`. Sau khi hoàn tất, trong thư mục xuất hiện thêm các tệp trung gian `.o` và tệp thực thi `calc`, chứng tỏ chương trình đã được biên dịch thành công.

Tiếp đó, khi chạy lệnh `make clean`, Makefile thực hiện chức năng dọn dẹp - xóa các tệp được tạo ra trong quá trình biên dịch trước đó. Kết quả là thư mục trở lại trạng thái ban đầu, chỉ còn lại các tệp mã nguồn và tệp `Makefile`.

# Tài liệu