1. **Ưu điểm của Makefile là gì? Hãy nêu ví dụ?**

Lợi ích của MakeFile: giúp cho compile các dự án code phức tạp. Makefile liên kết các luật với nhau lại để có thể compile chương trình, từ đó thay vì cần phải rất nhiều câu lệnh để compile được code thì ta chỉ cần 1 lệnh. Makefiles giúp ta quản lý nhiều dự án phức tạp 1 cách tự động. Ví dụ ta có các chương trình như sau:

**Adder.c có hàm adder(int n, int n) trả về giá trị n + n**

**Square.c có hàm int square(int n) trả về giá trị n \* n;**

**Ta có hàm main.c như sau**

**main.c**

#include <stdio.h>

#include "functions.h"

int main(){

   printf ("the addition of 5 and 10 is %d \n",adder(5,10));

   printf( "The square of 25 is, %d \n", square(25));

   return 0;

}

Ta có thể compile bằng 1 lệnh như sau: **gcc  main.c adder.c square.c multiplier.c -o calculator.**

Lệnh này sẽ compile 4 file và sinh ra các mã lệnh theo hệ nhị phân. Và muốn sử dụng chương trình thì ta chỉ cần chạy mã lệnh này thôi.

Tóm lại lợi ích của việc sử dụng Makefiles là:

* Makefile dùng để compile code 1 cách đơn giản hơn chỉ dùng 1 lệnh đơn giản.
* Makefile sử dụng cơ chế trong Linux. Nó lưu mọi thứ đã được thay đổi, bằng cách này nó tiết kiệm thời gian compile. Trước makefile thì ta cần phải compile từng dòng code 1, giờ thì nó đã được lưu nếu nó đã từng được compile trước đó.

1. **Biên dịch – compiling lần đầu tiên thường tốn nhiều thời gian hơn so với những lần biên dịch kế tiếp – next recompiling. Hãy đưa ra lý do?**

Ở lần biên dịch đầu tiên có thể tốn nhiều chi phí về thời gian hơn các lần biên dịch sau, lí do chính là ở lần đầu tiên, hệ thống cần tạo cơ sở và nền tảng về thông tin và dữ liệu, đa số những nền tảng đó sẽ được tái sử dụng hoặc cập nhật ở các lần biên dịch sau thay vì phải tạo mới hoàn toàn, cụ thể:

* Phải tạo ra nhiều tập tin tạm thời: trong lần đầu tiên biên dịch, thường phải tạo mới các tập tin: tệp đối tượng, tệp nhị nhân, tệp liên kết, … Ở các lần biên dịch kế tiếp chỉ cần cập nhật dữ liệu và thông tin, không nhất thiết phải tạo lại các file tạm thời đó.
* Phải tiếp xúc và phân tích các cú pháp và tạo AST (abstract syntax tree – cây cú pháp trừu tượng)
* Phải tối ưu hóa quá trình tạo machine code: ở lần đầu tiên cần thực hiện cẩn thận và chi tiết quá trình tối ưu để tối đa hóa hiệu quả hoạt động, các lần biên dịch kế tiếp có thể kế thừa lại những giá trị đã có ở lần đầu tiên.

1. **Ở các ngôn ngữ lập trình khác có cơ chế Makefile nào hay không? Nếu có, hãy đưa ví dụ?**

* Có.
* Ví dụ cơ chế Makefile trong ngôn ngữ lập trình Python

*.PHONY: all clean test*

*PYTHON := python3*

*PIP := pip3*

*all: install test*

*install:*

*@echo "Installing dependencies..."*

*$(PIP) install -r requirements.txt*

*test:*

*@echo "Running tests..."*

*$(PYTHON) -m unittest discover -v tests*

*clean:*

*@echo "Cleaning up..."*

*rm -rf \_\_pycache\_\_ .pytest\_cache*

Trong Makefile này:

* all: Là một mục tiêu mặc định, chạy các mục tiêu install và test.
* install: Cài đặt các dependency từ file requirements.txt.
* test: Chạy các bài kiểm thử bằng cách sử dụng unittest.
* clean: Xóa các tệp và thư mục tạm thời được tạo ra trong quá trình xây dựng và kiểm thử.

Để sử dụng Makefile này, cần tạo một file requirements.txt chứa danh sách các dependency của dự án. Ngoài ra, cần tổ chức các bài test của mình trong thư mục tests và đặt tên các tệp kiểm thử bằng cách bắt đầu bằng test.

Đồng thời, có thể chạy các lệnh sau trong thư mục dự án của mình:

* make hoặc make all để cài đặt các dependencies và chạy các bài kiểm thử.
* make clean để dọn dẹp các tệp và thư mục tạm thời.
* make install để chỉ cài đặt dependencies.
* make test để chỉ chạy các bài kiểm thử.