**About myself:**

My name is Tiến Long. I have recently finished my course at Bach Khoa University. After learining for 4 years, I found out that my interest is Programming language and Linux OS. That’s why I chose to follow this career path. I am always willing to learn new knowledge to improve my skills like Programing Langue C, Python, Bash Shell and Makefile, Linux Kernel, Device Tree, Tmux and GitLab CI/CD.And finally I am confident that my responsibility and learning spirit will help me achieve my goal of becoming a professional embedded linux engineer.

**What are your strengths?**

One of my strengths is having knowledge in embedded C programming. Additionally, I have applied my knowledge by completing some projects, for example like a traffic signal system, or investigating unit test. These experiences have allowed me to apply my programming skills in practical scenarios and enhance my problem-solving abilities to work as an embedded software engineer.

**What are your weaknesses?**

My biggest weakness is I find it difficult to say no to people. Whenever somebody asks me for help or advice, I always say yes, which sometimes results in me taking on too many responsibilities. However, I am aware of this weakness and taking steps to improve and be more mindful to check my capacity for additional work before committing

**What are your short term goals?**

My short-term goals is to gain more understanding in current project also the knowledge required for that project. Additionally, by learning and actively seeking feedback, I aim to continuously improve my skills to finish my work better.

**Long-term goals:** My long-term goal is to advance into a role as an Embedded Linux Engineer, where I can contribute to the development of many software solutions, particularly in embedded systems and software architecture.

**Dự án đã làm ở công ty thực tập:**

Mô phỏng hệ thống lái điện EPS, mục tiêu chung là xây dựng lại được 1 hệ thống lái trợ lực điện, mục tiêu cá nhân là tìm hiểu về thuật toán của trợ lực (thu thập dữ liệu từ moment xoắn, speed và góc lái gửi tới ECU sử dụng feedback control, speed-based control, proportional control), tìm hiểu động lực học của hệ thống lái và áp dụng simulink + solidwork để đưa ra mô hình mô phỏng

Thông qua thời gian thực tập ở công ty em biết được thêm về các thuật toán cũng như các loại ECU được sử dụng trên xe

Dự án game-text-base:

Ở dự án này em đã sử dụng 2 thread để write() và read() dữ liệu từ 2 người chơi và sử dụng pipe để giao tiếp dữ liệu, struct để chứa thông tin của 2 nhân vật và các function để tương tác giữa 2 nhân vật

Dự án đèn giao thông:

Sử dụng STM32 với IDE để tạo đèn giao thông với nhiều mode, chống nhiễu bằng software cho button, hiển thị với UART và sử dụng Finite State Machine để giao tiếp

**Nguyên nhân gây ra Memory Leak**

1. **Quên free(ptr); bộ nhớ**: Bộ nhớ được phân bổ nhưng không được giải phóng.
2. **Mất tham chiếu đến bộ nhớ**: Con trỏ trỏ đến bộ nhớ được phân bổ bị gán lại hoặc biến chứa con trỏ đó bị huỷ trước khi bộ nhớ được giải phóng. (vd xảy ra trong struct có chứa ptr tới allocatted memory, khi allocate trong struct rồi free struct đó nhưng chưa free memory kia)

Xử lý: free pointer và gán biến pointer về NULL, sd tool có sẵn trong IDE để phát hiện memory leak

**Con trỏ NULL**

**Định nghĩa:** Con trỏ NULL là con trỏ không trỏ đến bất kỳ địa chỉ hợp lệ nào trong bộ nhớ. Nó thường được gán giá trị **NULL** (hoặc **0** trong C) để chỉ rõ rằng con trỏ này hiện tại không trỏ đến đối tượng nào.

**Hành vi:**

* Khi con trỏ NULL được dereference (truy cập nội dung mà nó trỏ đến), chương trình sẽ gây ra lỗi và thường bị crash (sập) vì nó trỏ đến một địa chỉ bộ nhớ không hợp lệ.
* Kiểm tra con trỏ NULL trước khi dereference là một thực hành tốt để tránh lỗi chương trình.

**Ví dụ:**

c

Sao chép mã

int \*ptr = NULL; if (ptr != NULL) { printf("%d\n", \*ptr); // Không xảy ra vì ptr là NULL }

**Con trỏ void**

**Định nghĩa:** Con trỏ void là một con trỏ mà không xác định kiểu dữ liệu của đối tượng mà nó trỏ đến. Nó được định nghĩa với kiểu **void \***.

**Hành vi:**

* Con trỏ void có thể trỏ đến bất kỳ loại dữ liệu nào, nhưng nó không thể dereference trực tiếp vì kiểu dữ liệu không xác định.
* Để sử dụng con trỏ void, cần phải cast nó về kiểu con trỏ cụ thể trước khi dereference.

**Ví dụ:**

c

Sao chép mã

void \*ptr; int a = 10; ptr = &a; // ptr trỏ đến một số nguyên printf("%d\n", \*(int \*)ptr); // Cần cast ptr về int\* trước khi dereference

**Con trỏ treo (Dangling Pointer)**

**Định nghĩa:** Con trỏ treo là con trỏ mà vẫn trỏ đến một vùng bộ nhớ mà đã bị giải phóng hoặc không còn hợp lệ.

**Hành vi:**

* Dereference con trỏ treo sẽ dẫn đến hành vi không xác định (undefined behavior), có thể gây ra lỗi chương trình hoặc dữ liệu không hợp lệ.
* Tránh con trỏ treo bằng cách gán giá trị NULL cho con trỏ sau khi giải phóng bộ nhớ.

**Ví dụ:**

c

Sao chép mã

int \*ptr = (int \*)malloc(sizeof(int)); \*ptr = 10; free(ptr); // ptr trở thành con trỏ treo ptr = NULL; // Tránh treo bằng cách gán NULL

**Con trỏ wild (Wild Pointer)**

**Định nghĩa:** Con trỏ wild là con trỏ mà chưa được khởi tạo và có giá trị không xác định. Nó trỏ đến địa chỉ bộ nhớ ngẫu nhiên, có thể không hợp lệ hoặc không an toàn để sử dụng.

**Hành vi:**

* Dereference con trỏ wild cũng dẫn đến hành vi không xác định và có thể gây ra lỗi chương trình hoặc crash.
* Luôn khởi tạo con trỏ trước khi sử dụng để tránh con trỏ wild.

**Ví dụ:**

c

Sao chép mã

int \*ptr; // ptr là con trỏ wild vì chưa được khởi tạo ptr = NULL; // Khởi tạo ptr để tránh trở thành con trỏ wild

**Tổng kết**

* **Con trỏ NULL:** Trỏ đến không có địa chỉ hợp lệ, tránh dereference con trỏ NULL.
* **Con trỏ void:** Con trỏ chung cho mọi kiểu dữ liệu, cần cast về kiểu cụ thể trước khi dereference.
* **Con trỏ treo:** Trỏ đến vùng bộ nhớ đã giải phóng, tránh bằng cách gán NULL sau khi giải phóng.
* **Con trỏ wild:** Chưa được khởi tạo, có giá trị ngẫu nhiên và không xác định, cần khởi tạo trước khi sử dụng.

trình biên dịch, thông dịch

**Trình biên dịch và thông dịch khác nhau như thế nào?**

**Trình biên dịch (Compiler)**

Trình biên dịch là một chương trình chuyển đổi mã nguồn của một ngôn ngữ lập trình (như C, C++, Java) thành mã máy hoặc mã trung gian (như bytecode) mà máy tính có thể hiểu và thực thi trực tiếp. Quá trình này diễn ra trước khi chương trình được chạy.

**Đặc điểm:**

1. **Biên dịch toàn bộ chương trình:** Trình biên dịch dịch toàn bộ mã nguồn thành mã máy hoặc mã trung gian trước khi chạy.
2. **Tạo file thực thi:** Sau khi biên dịch, nó tạo ra một file thực thi (.exe, .out) có thể chạy độc lập mà không cần trình biên dịch.
3. **Hiệu suất:** Chương trình biên dịch thường chạy nhanh hơn vì mã máy đã được tối ưu hóa trước.
4. **Phát hiện lỗi:** Các lỗi cú pháp và ngữ pháp được phát hiện trong quá trình biên dịch, trước khi chương trình chạy.
5. **Ví dụ:** GCC cho C/C++, javac cho Java.

**Ví dụ:**

c

Sao chép mã

#include <stdio.h> int main() { printf("Hello, World!\n"); return 0; }

Biên dịch với GCC:

sh

Sao chép mã

gcc -o hello hello.c ./hello

**Thông dịch (Interpreter)**

Trình thông dịch là một chương trình thực thi mã nguồn của một ngôn ngữ lập trình trực tiếp mà không cần chuyển đổi thành mã máy trước. Nó thực thi mã nguồn từng dòng một.

**Đặc điểm:**

1. **Thực thi từng dòng:** Thông dịch mã nguồn từng dòng một trong thời gian chạy.
2. **Không tạo file thực thi:** Không tạo ra file thực thi độc lập; mã nguồn cần trình thông dịch để chạy.
3. **Hiệu suất:** Chương trình thông dịch thường chạy chậm hơn do cần thực thi từng dòng mã nguồn và không được tối ưu hóa trước.
4. **Phát hiện lỗi:** Lỗi được phát hiện trong thời gian chạy, nên có thể phát hiện lỗi trong một số dòng mã trước khi gặp lỗi.
5. **Ví dụ:** Python, Ruby, JavaScript.

**Ví dụ:**

python

Sao chép mã

print("Hello, World!")

Chạy với Python:

sh

Sao chép mã

python hello.py

**So sánh chi tiết**

A screenshot of a computer

Description automatically generated

**Kết luận**

* **Trình biên dịch** phù hợp cho các ứng dụng yêu cầu hiệu suất cao và phát hiện lỗi sớm, như phần mềm hệ thống, ứng dụng máy tính để bàn.
* **Trình thông dịch** thích hợp cho phát triển nhanh, dễ dàng chỉnh sửa và thử nghiệm mã nguồn, như kịch bản (scripting), ứng dụng web.

Big-endian và little-endian là hai cách khác nhau để lưu trữ và sắp xếp các byte của dữ liệu đa byte (như số nguyên 32-bit hoặc 64-bit) trong bộ nhớ máy tính. Sự khác biệt chính giữa chúng nằm ở thứ tự các byte.

**Little-endian**

Trong cách sắp xếp little-endian, byte có trọng số thấp nhất (least significant byte - LSB) được lưu trữ ở địa chỉ bộ nhớ thấp nhất, và byte có trọng số cao nhất (most significant byte - MSB) được lưu trữ ở địa chỉ bộ nhớ cao nhất.

**Ví dụ:** Giả sử bạn có số nguyên 32-bit 0x12345678. Cách lưu trữ trong bộ nhớ little-endian sẽ như sau:

A black rectangular object with a white border

Description automatically generated

**Big-endian**

Trong cách sắp xếp big-endian, byte có trọng số cao nhất (MSB) được lưu trữ ở địa chỉ bộ nhớ thấp nhất, và byte có trọng số thấp nhất (LSB) được lưu trữ ở địa chỉ bộ nhớ cao nhất.

**Ví dụ:** Giả sử bạn có số nguyên 32-bit 0x12345678. Cách lưu trữ trong bộ nhớ big-endian sẽ như sau:

A black rectangular object with a white border

Description automatically generated

**Tầm quan trọng của Endianness**

* **Giao tiếp giữa các hệ thống:** Khi truyền dữ liệu giữa các hệ thống có kiến trúc khác nhau (một hệ thống sử dụng little-endian, hệ thống khác sử dụng big-endian), việc hiểu và chuyển đổi endianness là cần thiết để đảm bảo dữ liệu được giải mã chính xác.
* **Đọc file nhị phân:** Khi đọc file nhị phân tạo bởi hệ thống sử dụng endianness khác, cần phải xử lý chuyển đổi endianness để đọc dữ liệu đúng cách.
* **Lập trình hệ thống:** Trong lập trình hệ thống hoặc phát triển hệ điều hành, việc xử lý đúng endianness là rất quan trọng.

<https://viblo.asia/p/little-endian-vs-big-endian-E375z0pWZGW>

c khác c++

1. **Cú pháp:**
   * C++ bao gồm cú pháp của C, nhưng có thêm nhiều tính năng bổ sung như lớp (class), kế thừa, đa hình, quản lý bộ nhớ tự động (RAII), và nhiều tính năng hướng đối tượng khác.
   * C++ cũng hỗ trợ kiểu dữ liệu tự định nghĩa thông qua lớp và cấu trúc, trong khi C chỉ có cấu trúc.
2. **Tính đa hình:**
   * C++ hỗ trợ tính đa hình (polymorphism) thông qua kế thừa và đa hình động (runtime polymorphism) bằng cách sử dụng các lớp cơ sở và lớp dẫn xuất, cùng với việc sử dụng con trỏ tới lớp cơ sở và hàm ảo (virtual functions).
   * Trong khi đó, C không có tính đa hình như vậy và sử dụng con trỏ hàm để đạt được một số mức độ linh hoạt tương tự.
3. **Quản lý bộ nhớ:**
   * Trong C++, có thể sử dụng các cơ chế như tham chiếu thông minh (smart pointers) và lớp tự giải phóng bộ nhớ (RAII) để quản lý bộ nhớ tự động.
   * Trong C, bạn phải quản lý bộ nhớ thủ công bằng cách sử dụng các hàm như **malloc()** và **free()**.
4. **Thư viện chuẩn:**
   * Có một số khác biệt trong thư viện chuẩn của hai ngôn ngữ. Ví dụ, C++ có thư viện chuẩn có tên **iostream** để nhập xuất, trong khi C sử dụng **stdio.h** hoặc **stdlib.h** cho mục đích tương tự.
5. **Hiệu suất:**
   * Trong một số trường hợp, mã C có thể hiệu suất hơn mã C++ vì không có overhead của một số tính năng của C++. Tuy nhiên, trong nhiều trường hợp, sự khác biệt về hiệu suất không đáng kể và sự lựa chọn giữa C và C++ thường dựa vào yêu cầu cụ thể của dự án.

|  |  |  |
| --- | --- | --- |
| **Tham số** | **C** | **C++** |
| **Mô hình lập trình** | Ngôn ngữ lập trình theo cấu trúc hoặc thủ tục. | Ngôn ngữ lập trình theo cấu trúc và hướng đối tượng. |
| **Lịch sử** | Do nhà khoa học Dennis Ritchie phát triển vào năm 1972 tại phòng thí nghiệm Bell. | Do Bjarne Stroustup phát triển vào năm 1979. |
| **Phương pháp** | Theo phương pháp từ trên xuống dưới. | Theo phương pháp từ dưới lên trên. |
| **Từ khóa** | Chứa 32 từ khóa | Chứa 63 từ khóa |
| **Kiểu dữ liệu** | Hỗ trợ các kiểu dữ liệu tích hợp. | Hỗ trợ cả kiểu dữ liệu tích hợp và do người dùng xác định. |
| **Phần mở rộng file** | .c | .cpp |
| **File header** | <stdio.h> | <iostream.h> |
| **Phân bổ và giải phóng bộ nhớ** | Dùng calloc() và malloc() cho phân bổ bộ nhớ động và free() cho bỏ phân bổ bộ nhớ. | Dùng toán tử mới để phân bổ bộ nhớ và xóa toán tử để giải phóng bộ nhớ. |
| **Chỉ định truy cập** | Không hỗ trợ chỉ định truy cập | Hỗ trợ chỉ định truy cập |
| **Bảo mật** | C không có bất kỳ tính năng bảo mật nào nên có thể bị người ngoài thao túng | C ++ là một ngôn ngữ an toàn vì nó cung cấp các tính năng bảo mật như ẩn và đóng gói dữ liệu |
| **Biến tham chiếu** | Không hỗ trợ | Hỗ trợ |
| **Overloading và Overriding hàm** | Không hỗ trợ | Hỗ trợ |
| **Xử lý ngoại lệ** | C không hỗ trợ xử lý ngoại lệ trực tiếp, nó sử dụng chức năng hỗ trợ xử lý ngoại lệ | C++ hỗ trợ trực tiếp xử lý ngoại lệ với sự trợ giúp của khối try – catch |
| **Phân chia chương trình** | C là ngôn ngữ thủ tục, vì vậy mã viết bằng C được chia thành các khối riêng biệt được gọi là hàm | C++ là ngôn ngữ hướng đối tượng, do đó viết code và được chia thành lớp, đối tượng |
| **Hàm nội tuyến** | Không hỗ trợ | Hỗ trợ |
| **Kiểu hướng** | Ngôn ngữ hướng hàm | Ngôn ngữ hướng đối tượng |
| **Khả năng tương thích** | Code được viết bằng C có thể chạy trên trình biên dịch C++ vì C là ngôn ngữ nền tảng | Code viết bằng ngôn ngữ C++ có thể chạy trên trình biên dịch C vì ngôn ngữ C++ bao gồm khái niệm OOP |
| **Dữ liệu và hàm** | Tách biệt | Được đóng gói chung với nhau |
| **Hàm input và output** | Các hàm scanf() và printf() được sử dụng để lấy đầu vào và đầu ra tương ứng | Các hàm cin và cout được sử dụng để lấy đầu vào và đầu ra tương ứng |
| **Lập trình ứng dụng** | Ngôn ngữ C phù hợp hơn để triển khai ở mức độ thấp như trình điều khiển mạng, trình soạn thảo văn bản, trình biên dịch chương trình, v.v. | Ngôn ngữ C ++ phù hợp hơn để triển khai cấp cao như phát triển trò chơi, đồng hồ thông minh, hệ thống nhúng, v.v. |
| **Namespace** | Để tránh xung đột và sắp xếp mã, cần có namespace nhưng C không hỗ trợ | Hỗ trợ namespace |
| **Được sử dụng bởi** | [MySQL](https://quantrimang.com/cong-nghe/huong-dan-cai-mysql-tren-windows-va-truy-cap-tu-xa-145550), Windows Kerne, Oracle Database, Telegram... | Google Chrome, Torque 3-D game, Microsoft Office, ... |

reuse c++ -> tính kế thừa

Trong lập trình hướng đối tượng, tính kế thừa (inheritance) là khả năng tái sử dụng mã nguồn bằng cách cho phép một lớp (class) mới được tạo ra từ một lớp hiện có, được gọi là lớp cơ sở (base class). Lớp mới này, gọi là lớp dẫn xuất (derived class), kế thừa tất cả các thành phần của lớp cơ sở, bao gồm các thành phần dữ liệu (data members) và các phương thức (methods). Nó cũng có thể mở rộng hoặc thay đổi các tính năng của lớp cơ sở.

Ví dụ, giả sử chúng ta có một lớp cơ sở **Shape** đại diện cho các hình học, và chúng ta muốn tạo ra một lớp dẫn xuất **Rectangle** để đại diện cho các hình chữ nhật. Ta có thể sử dụng tính kế thừa để tái sử dụng mã nguồn từ lớp **Shape**:

A computer screen shot of a program code

Description automatically generated  
2 câu code  
deadlock