**1. Cac buoc bien dich cua chuong trinh C/C++.**

**B1: Preprocessing:**

**Xử lý mã nguồn C++ và tạo ra một mã nguồn mới được gọi là mã nguồn tiền xử lý (preprocessed source):**

- Xử lý các hằng số: Trình biên dịch sẽ tìm và thay thế các hằng số được định nghĩa bằng #define.

- Xử lý các macro: Trình biên dịch sẽ tìm và thay thế các macro bằng giá trị tương ứng của chúng.

- Thêm các tệp header: Trình biên dịch sẽ thêm các tệp header được định nghĩa bằng #include vào mã nguồn.

🡪 Trình tiền xử lý sẽ tạo ra một tệp tin mới có đuôi ".i" (linux)

**B2: Compilation:**

**Chuyển đổi mã nguồn C++ thành mã nguồn Assembly**

- Phân tích cú pháp (Parsing): xác định cấu trúc

- Tạo ra cây cú pháp trừu tượng (Abstract Syntax Tree): biểu diễn cấu trúc mã nguồn

- Tạo ra mã nguồn Assembly: 0101

- Kiểm tra lỗi (Error Checking): lỗi cú pháp and logic

🡪Kết quả sẽ là một tệp tin có đuôi ".s" (linux)

**B3: Assembly:**

**Dụng trình dịch mã nguồn Assembly để chuyển đổi mã nguồn Assembly thành mã máy**

- Tạo ra tệp tin object (object file): Trình biên dịch sẽ tạo ra tệp tin object chứa mã máy tương ứng với mã nguồn Assembly.

🡪 Kết quả sẽ là một tệp tin có đuôi ".o" (linux)

**B4: Linking:**

- Trình biên dịch sẽ sử dụng trình liên kết (linker) để liên kết tất cả các tệp tin object và thư viện thành một tệp tin thực thi (executable file)

🡪 Kết quả sẽ là một tệp tin có đuôi ".dll" (linux)

**2. OOP la gi, 4 tinh chat, phan biet chung:**

**3. So sanh interface va abstract class, phan biet chung:**

**- Abstract class: là một lớp mà không thể tạo đối tượng trực tiếp từ lớp này, mà chỉ có thể tạo đối tượng từ các lớp con kế thừa nó**

+ có thể có các phương thức ảo (pure virtual method) và các phương thức thực thi, có thể chưa thuộc tính, biến

+ có ít nhất 1 pure virtual

+ có thể chứa cả phương thức thuần ảo và phương thức cụ thể, trong khi Interface chỉ chứa phương thức thuần ảo.

**- Interface: tập hợp các phương thức thuần ảo, tương tự như abstract class, nhưng không chứa bất kỳ phương thức cụ thể (implementation) nào, cũng không thể tạo đối tượng trực tiếp từ nó**

+ Trong C++, Interface được triển khai bằng cách sử dụng abstract class với tất cả các phương thức đều là phương thức thuần ảo.

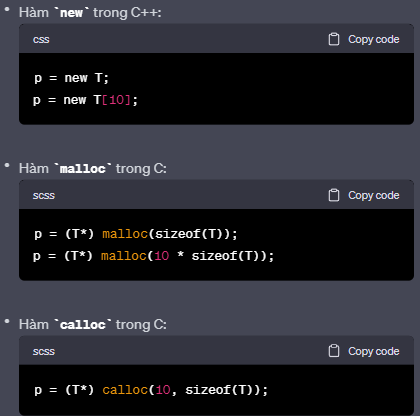
+ C++ 11 trở đi, có thể khai báo tường mình bằng từ khóa interface mà không cần thông qua abstract class

🡪 Một lớp có thể kế thừa nhiều Interface, nhưng chỉ có thể kế thừa một Abstract class.

**4. So sanh C struct, C++ struct, C++ class:**

**5. Phan biet new(C++) va malloc, calloc(C), delete(C++) va free(C++), viet syntas cua chung, Co the dung new - free hoac malloc - delete khong ?**

- **new** trong C++ và các hàm **malloc**, **calloc** trong C đều được sử dụng để cấp phát bộ nhớ động.



+ new:

* trả về địa chỉ của kiểu dữ liệu được cấp phát.
* cho phép cấp phát bộ nhớ động cho một đối tượng hoặc một mảng đối tượng
* có thể xử lý ngoại lệ và ném ra một ngoại lệ (exception) khi không thể cấp phát bộ nhớ.

+ malloc, calloc:

* trả về con trỏ void, phải ép kiểu để trở thành kiểu dữ liệu cần sử dụng.
* chỉ cho phép cấp phát bộ nhớ động cho một mảng các phần tử.
* chỉ trả về một con trỏ null khi không thể cấp phát bộ nhớ, không thể xử lý ngoại lệ.

+ delete:

* giải phóng bộ nhớ của một đối tượng được cấp phát bằng toán tử new hoặc new[]
* Khi sử dụng **delete**, hệ thống sẽ tự động gọi hàm hủy (destructor) của đối tượng đó trước khi giải phóng bộ nhớ,
* Delete ptr, delete ptr[]

+ free()

* để giải phóng bộ nhớ được cấp phát bằng hàm malloc hoặc calloc.
* với free thì không có hàm hủy được gọi.
* free(ptr), ptr địa chỉ vùng nhớ cần giải phóng

- Trong C++ và C, không nên sử dụng new và free hoặc malloc và delete kết hợp với nhau:

+ new và delete được thiết kế để hoạt động cùng nhau và có những tính năng đặc biệt như gọi hàm khởi tạo và hủy của đối tượng

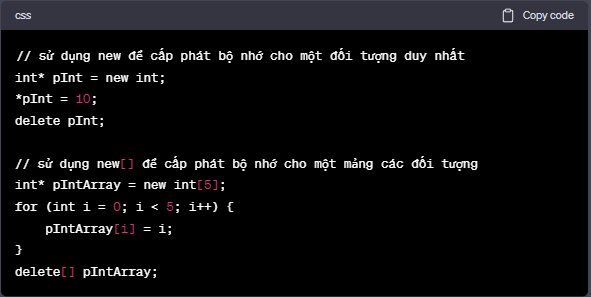
+ malloc và free chỉ đơn giản là cấp phát và giải phóng bộ nhớ.

- Nếu sử dụng thì sẽ có thể dẫn đến các lỗi khó xử lý, như gây ra memory leak hoặc xung đột khi giải phóng bộ nhớ.

**7. Khac nhau giua new() va new[], delete và delete[]**

- new được sử dụng để cấp phát bộ nhớ động cho một đối tượng duy nhất 🡪 dùng delete để giải phóng memory

- new[] cấp phát đủ bộ nhớ cho tất cả các phần tử trong mảng , trả về phần tử first của mảng 🡪 dùng delete[] để giải phòng bộ nhớ đã cấp phát



**8. So sánh truyền tham trị, tham chiếu, con trỏ, chúng là gì:**

|  |  |  |
| --- | --- | --- |
|  | **Truyền tham chiếu** | **Truyền con trỏ** |
| **Ưu điểm** | - truyền tham chiếu nhanh hơn so với truyền giá trị vì ta không cần sao chép giá trị vào một biến khác.  - tiết kiệm bộ nhớ  - Cho phép hàm ghi đè trực tiếp vào biến đầu vào | - truyền nhiều giá trị vào trong một hàm thông qua mảng con trỏ, danh sách liên kết con trỏ.  - truyền địa chỉ biến vào hàm🡪 thay đổi giá trị dễ dàng  - Cho phép ta truyền các đối tượng lớn hơn kích thước của thanh ghi đầu vào thông qua con trỏ. |
| **Nhược điểm** | Nếu đối tượng đầu vào đã bị hủy trong khi hàm đang được thực thi, thì việc truyền tham chiếu sẽ dẫn đến lỗi. | Nếu không sử dụng con trỏ đúng cách, ta có thể dẫn đến những lỗi logic khó khắc phục. |

**9. Khac nhau giua i++ va++i:**

++i return i had ++;

i++ return i begin and i hadn’t ++.

**10. Khac nhau giua nullptr va NULL.**

- C++11 trở đi, nullptr là con trỏ null

- NULL đã có từ xưa và nó là có giá trị =0

**12. Smart Pointer la gi.**

- Là một loại con trỏ thông minh được sử dụng trong các ngôn ngữ lập trình như C++ để quản lý bộ nhớ động một cách an toàn và hiệu quả.

- Tự động quản lý việc trỏ tới một đối tượng và tự động giải phóng bộ nhớ động khi đối tượng không còn được sử dụng.

- Sử dụng smart pointer giúp tránh những lỗi phổ biến liên quan đến quản lý bộ nhớ như trỏ tới một vùng nhớ đã được giải phóng hoặc quên giải phóng bộ nhớ.

- Các loại smart pointer phổ biến: unique\_ptr và shared\_ptr.

**11. Pointer la gi, phan biet \*ptr,&ptr,\*&ptr.**

- Pointer làmột biến đặc biệt lưu trữ địa chỉ bộ nhớ của một biến khác.

- Kích thước của con trỏ không phụ thuộc vào kiểu dữ liệu mà chỉ phụ thuộc và HDH của PC, 64 bit 8 byte, 32 bit 4 byte. kiểu dữ liệu của con trỏ: unsign\_int.

- \*&ptr :

**13. Uu nhuoc diem cua unique, shared.**

|  |  |  |
| --- | --- | --- |
|  | **Unique** | **Shared** |
| **Ưu điểm** | - Quản lý bộ nhớ tự động, tránh được việc quên giải phóng bộ nhớ dẫn đến memory leak.  - Tốc độ truy xuất của unique pointer nhanh hơn shared pointer vì không cần quản lý thêm một đếm count trỏ đến cùng 1 vùng nhớ  - Có thể sử dụng unique pointer để quản lý một đối tượng được tạo ra động (dynamic object).  - Unique pointer hỗ trợ việc chuyển giao quyền sở hữu (move ownership), cho phép tránh việc sao chép đối tượng (copy object). | - Giống như unique pointer, shared pointer cũng hỗ trợ quản lý bộ nhớ tự động.  - Cho phép chia sẻ quyền sở hữu giữa các smart pointer, giúp giảm độ phức tạp của mã nguồn.  - Có thể sử dụng shared pointer để quản lý một đối tượng được tạo ra động (dynamic object). |
| **Nhược điểm** | - Không thể copy, chỉ có thể chuyển quyền sở hữu (move) nên sử dụng unique pointer trong một số tình huống cần copy object sẽ không thực sự tiện lợi.  - Không thể chia sẻ quyền sở hữu (shared ownership) giữa các unique pointer, nên không phù hợp cho các tình huống cần sử dụng con trỏ chia sẻ. | - Tốc độ truy xuất chậm hơn unique pointer do cần quản lý thêm một đếm số lượng shared pointer trỏ đến cùng một vùng nhớ.  - Trong một số trường hợp, việc sử dụng shared pointer có thể dẫn đến vấn đề vòng lặp  - Shared pointer không hỗ trợ chuyển giao quyền sở hữu |

Tùy vào tình huống cụ thể mà lựa chọn sử dụng unique pointer hay shared pointer. Nếu chỉ cần quản lý một đối tượng tĩnh (static object) và không cần chia sẻ giữa các đối tượng khác thì có thể sử dụng unique pointer,

1. Viet ham swap ma k dung bien temp.

