# **GIT**

1. **Khái niệm:** GIT là 1 hệ thống quản lý phiên bản phân tán, cho phép lưu trữ các phiên bản của dự án, có thể quay lại các phiên bản đã lưu trữ, cho phép mọi người làm việc cùng nhau. Phân tán là làm việc trên máy tính, mạng khác nhau
2. **GIT Repo: kho lưu trữ**
3. **GIT config – global user.name/user.email**
4. **GIT init: khởi tạo repo ở local**
5. **GIT status: trạng thái các file**
6. **GIT commit: ghi ở kho những thay đổi**
7. **Staging area: khu vực theo dõi những file đc thực hiện ở lần commit tiếp theo**
8. **GIT add <file> <file>** : đưa file vào staging area
9. **GIT commit -m**: đưa file ở staging area vào repo với các thay đổi
10. **GIT log**: lịch sử commit
11. branch: làm việc của mình không ảnh hưởng ngưởi khác
12. **GIT branch**: liệt kê các nhánh đang có
13. **GIT branch -d:** xóa branch
14. **GIT push origin -d <branch>:**  xóa branch trên remote
15. **GIT checkout -b <branch>:** tạo branch kết thừa các commit
16. **GIT checkout <hash log>:** trờ về phiên bản trước
17. **GIT restore –staged <filed>:** đưa từ staging về thư mục đang làm việc
18. **GIT merge <branch>**: đưa commit từ branch vào branch đang đứng
19. **GIT fetch:** lấy dữ liệu mới nhất từ repository từ xa (remote) về máy cục bộ (local), nhưng không tự động hợp nhất (merge) vào nhánh bạn đang làm việc.
20. **GIT stash:** tạm thời lưu lại các thay đổi hiện tại trong thư mục làm việc (working directory), để bạn có thể chuyển sang một nhánh khác hoặc làm việc khác mà không cần commit các thay đổi đó.
21. **GIT push**: đẩy commit từ local lên remote
22. **GIT push –force: ghi đè commit**
23. **GIT pull**: kéo về những thay đổi mới nhất từ remote
24. **GIT reset –hard <hash log>:** quay ngược thời gian (xóa luôn commit ở giữa)
25. **GIT tag <name>:** tạo tag
26. **GIT tag:** liệt kê tag
27. **.gitignore**: bỏ qua file khỏi staging

Th1: Sửa Commit: git commit –amend <-m> hoặc git reset –soft HEAD~<version> và commit lại

Th2: Xóa commit ở local: git reset –hard HEAD~<version>

Th3: quay về commit cũ: git reset –hard HEAD~<version>

Th4: chỉnh sửa commit cuối cùng trên remote: git commit –amend <-m> và git push -f origin <branch>

Th5: Quay trở lại commit history commit để phát triển tính năng riêng:

git checkout <hash log> -b <new branch> (cần phải checkout branch mới để tránh mất code)

# **MQTT : Message Queuing Telemetry Transport**

Đặc điểm: băng thông thấp, nhẹ

Publish/Subcribe: phát/ đăng kí chủ đề thông qua topic

Message: bản tin trao đổi

Topic: nhận message thông qua topic

Broker: là 1 server

# **C**

1. Biến trong C phân biệt chữ hoa, thường, phải được khịa áo trước khi sử dụng
2. Compiler: (gcc) 4 bước: tiền xử lý build các file .h sang .i : loại bỏ chú thích, copy code từ các file include sang, thay thế các macro

#include <>: chỉ thị bao hàm tệp (chèn file vào): dùng cho thư viện chuẩn, khi trình biên dịch thấy<> thì sẽ tìm kiếm trong thư mục cài đặt

#include “”: tìm kiếm trong thư mục hiện tại, các file do mình tự định nghĩa

#define: thay thế

##: nối chuỗi

#: chuẩn hóa đoạn văn bản lên chuỗi

…: (variadic)

\_\_VAR\_ARGS\_\_: macro có số lượng tham số tùy biến

#undef: hủy thay thế

#if: kiểm tra macro

#ifdef, #ifndef: kiểm tra maccro đã được định nghĩa chưa

#endif: kết thúc định nghĩa

1. Include .c: nếu bị include trùng thì báo lỗi, còn include .h nếu có ifndef thì sẽ không báo lỗi khi bị trùng
2. Compiler: từ .i sang .s (assembly)
3. Assembler: từ .s sang .o (mã máy)
4. Linker: liên kết tất cả file .o sang .exe
5. Bit wise: tìm bit cuối: &1
6. Tìm bit thứ i: & (n >> i) & 1
7. Tắt bit i: n & ~(1<<i)
8. Bật bit i : n|(1<<i)
9. Đảo bit i: n^ (1<<i)
10. Độ ưu tiên: () -> \*/ -> +- -> bitwise
11. Big endian & little endian
    1. Little endian: LSB ở địa chỉ thấp, MSB ở địa chỉ cao -> thuận tiện ép kiểu và tính toán
    2. Big endian: LSB ở dịa chỉ cao, MSB ở đại chỉ thấp -> dễ cho người đọc
12. MSB & LSB:
    1. MSB: bên trái
    2. LSB: bên phải
13. Struct: một kiểu dữ liệu do người dùng định nghĩa, cho phép nhóm nhiều biến (có thể khác kiểu dữ liệu) lại với nhau dưới một tên duy nhất.
14. Union:
15. Padding alignment: lấy kiểu dữ liệu nguyên thủy lớn nhất làm size cơ sở, các kiểu nhỏ hơn phải padding (processor đọc từng word: 32bit: 4 bytes, 64bit: 8bytes)
16. Bit fields: tiết kiệm ô nhớ, nhưng chỉ dùng được cho số nguyên, không dùng được con trỏ
17. Tại sao nên sử dụng C thay vì Java?

C thường có hiệu suất tốt hơn so với Java. Thứ hai,

C cho phép truy cập và kiểm soát trực tiếp phần cứng của máy tính

C cho phép kiểm soát tài nguyên một cách cụ thể và hiệu quả hơn

1. Global là gì? (lưu trong data segment (DS) ở RAM nếu đc khởi tạo hoặc BSS nếu chưa khởi tạo hoặc =0)

là biến được khai báo bên ngoài bất kỳ hàm nào trong chương trình, và nó có thể được truy cập từ bất kỳ hàm nào trong cùng phạm vi đó. Nó tồn tại và giữ giá trị trong suốt quá trình thực thi của chương trình, không bị hủy khi thoát khỏi hàm nào đó.

1. Local (stack trong RAM) bị thu hồi vùng nhớ sau khi thoát khỏi hàm
2. Extern (không tạo biến mới, biến thật ở nơi khác)

Truy cập giá trị một biến toàn cục, mảng, hàm ở file khác

1. Static: lưu trong data-segment hoặc BSS nếu chưa khởi tạo

Static trong biến cục bộ:, không bị thu hồi vùng nhớ khi thoát khỏi hàm nhưng chỉ tồn tại trong phạm vi hàm

Static trong biến toàn cục: chỉ tồn tại trong file khai báo static, có thể khai báo lại cùng tên biến trong file khác mà không báo lỗi

Static trong hàm: chỉ được gọi hàm trong file có static

20, const (đặt trong RO data)  
 global const: RO Data segment

Local const: RO stack

1. Volatile (Chậm hơn vì luôn đọc từ RAM)

báo cho trình biên dịch rằng giá trị của biến có thể bị thay đổi bất cứ lúc nào, ngoài sự kiểm soát của chương trình — ví dụ:

* Bị thay đổi bởi phần cứng (thanh ghi)
* Bị thay đổi bởi một luồng khác (đa luồng)
* Bị thay đổi bởi ngắt (interrupt handler)

1. Register (chỉ sử dụng cho biến local)

Gắn vào trong thanh ghi CPU thay vì stack trong RAM, không cấp phát vùng nhớ nữa, sử dụng để bỏ qua quá trình load dữ liệu từ bộ nhớ để tăng tốc độ. Nếu không đủ thành ghi, compiler vẫn có thể lưu vào RAM như bình thường

-Không có địa chỉ bộ nhớ cụ thể nên không dùng được con trỏ để trỏ tới

23. Auto

Trong C: chỉ ra rằng biến là biến cục bộ (local) và được tự động cấp phát trên stack khi hàm được gọi. Tuy nhiên, mặc định tất cả biến local đều là auto, nên không cần viết ra.

Trong C+: auto nghĩa là suy luận kiểu tự động, cần phải khởi tạo giá trị ngay khi khai báo biến thì nó mới biết được kiểu dữ liệu

1. Storage-class specifiers: 1 biến không được đặt nhiều storage-class specifiers
   1. Typedef
   2. Extern
   3. Static
   4. Auto
   5. Register
2. Memory layout:
   1. Text-segment: lưu code , readonly
   2. DS (data-segment): global/static được khởi tạo khác 0
   3. BSS: global/static chưa được khởi tạo hoặc = 0
   4. Heap: cấp phát động (malloc/calloc/free/new/delete) phình dần lên và cần được giải phóng (memory leak)
   5. Stack (cấp phát tự động LIFO): parameter – return address – saved previous frame pointer (EBP: chỉ về vị trí đầu tiên của function)– local variable

EBP (đầu stack của hàm) – ESP (cuối cùng stack của hàm)– EIP (instruction stack)

1. Array: là 1 nhóm các ô nhớ liền kề nhau dùng để lưu trữ các phần tử cùng kiểu
   1. Tên mảng thực chất là 1 con trỏ, luôn trỏ đến địa chỉ đầu tiên của mảng
   2. Nếu không gán giá trị trước cho mảng thì phải khai báo số lượng phần tử trong mảng
   3. Có thể gán giá trị riêng lẻ cho từng phần tử
   4. Truyền mảng vào hàm trong C là truyền tham chiếu (truyền địa chỉ phần tử đầu tiên của mảng vào hàm) arr = &arr = &arr[0]
   5. Khi truyền mảng vào hàm thì truyền như biến, không cần &
   6. Để trả về mảng từ hàm: cần khai báo kiểu hàm là con trỏ, khai báo mảng static và biến con trỏ để gán mảng trả về
2. Con trỏ
   1. Con trỏ thông thường
   2. Là biến mà giá trị của biến là địa chỉ của biến khác
   3. Size con trỏ là cố định tùy vào kiến trúc máy tính, không phụ thuộc kiểu dữ liệu
   4. Truyền con trỏ vào hàm khi muốn hàm đó thay đổi giá trị biến bên ngoài hoặc muốn truyền mảng vào
   5. Void pointer: (con trỏ tổng quát) là con trỏ tới một địa chỉ ô nhớ không xác định kiểu dữ liệu.
      1. có thể trỏ tới bất kỳ kiểu dữ liệu nào:
      2. Phải ép kiểu (cast) khi truy cập giá trị mà void\* trỏ tới
   6. Function pointer: Gọi một hàm thông qua con trỏ. , mỗi function khi khởi tạo đều có 1 địa chỉ
      1. Typedef void (\*function)(uint8\_ t)
      2. Mảng con trỏ hàm: (\*function[2]) (uint8\_t, uint8\_t) ={}
3. Hàm main() thực thi thành công sẽ tự động return 0, có thể return 0 hoặc không cần.
4. Tham số là biến nhận dữ liệu đầu vào cho hàm xử lý, Đối số là biến truyền vào lời gọi hàm
5. typedef: tạo tên mới cho 1 kiểu dữ liệu
6. enum: định nghĩa tập hợp các hằng số có tên dễ đọc, dễ dùng thay cho các con số
7. Union:
   1. Là kiểu dữ liệu đặc biệt giống struct, nhưng các thành viên dùng chung cùng một vùng nhớ.
   2. Tại một thời điểm, chỉ một thành viên của union có giá trị hợp lệ.
   3. Kích thước của union Là kích thước lớn nhất của bất kỳ thành viên nào.
8. Malloc/Calloc/Free
   1. Malloc: 1 tham số, Bộ nhớ không được khởi tạo
   2. Calloc: 2 tham số, Bộ nhớ được khởi tạo về 0.
   3. Free: giải phóng bộ nhớ, gán lại cho con trỏ = NULL
9. Memory leak: xảy ra khi bạn cấp phát bộ nhớ động bằng malloc, calloc, realloc... nhưng không free nó
10. Stack overflow là lỗi xảy ra khi chương trình dùng quá nhiều bộ nhớ ngăn xếp (stack) — vùng bộ nhớ dùng để lưu trữ: do đệ quy vô tận, mảng cục bộ lớn
    1. Biến cục bộ (local variables)
    2. Con trỏ trả về hàm
    3. Thông tin ngữ cảnh của lời gọi hàm
11. Inline Function: (trình biên dịch) (hàm nội tuyến) là một hàm mà khi biên dịch, thân hàm sẽ được chèn trực tiếp vào nơi gọi hàm, thay vì thực hiện việc nhảy đến địa chỉ hàm như thông thường. Trình biên dịch có thể bỏ qua từ khóa inline nếu thấy không hiệu quả (tùy vào tối ưu của compiler), không thể sử dụng trong đệ quyoo

# **C++**

1. Type promotion(thăng cấp kiểu) : compiler sẽ tự động convert đến kiểu của toán hạng có kiểu dữ liệu lớn nhất
2. Namespace: (không gian tên) trong C++ là để tránh xung đột tên giữa các biến, hàm, class, v.v... đặc biệt khi bạn dùng thư viện lớn hoặc nhiều file có thể trùng tên với nhau.
3. [Có thể biên dịch một chương trình mà không có hàm main hay không ?](https://techacademy.edu.vn/cau-hoi-phong-van-c-tong-hop-1000-cau-hoi/#10_Co_the_bien_dich_mot_chuong_trinh_ma_khong_co_ham_main_hay_khong)

Có thể biên dịch, nhưng không thể liên kết (link) thành file thực thi hoàn chỉnh nếu là một chương trình thông thường.

1. Truyền tham chiếu bằng &: làm việc với biến gốc mà tham chiếu đến khác với lấy địa chỉ
2. Parameter mặc định trong hàm: được gán giá trị trước, nếu không truyền đối số thì hàm sẽ sử dụng giá trị mặc định này
3. Goto: nhảy đến 1 label, phạm vi trong hàm, không nên sử dụng Goto vì có thể phá vỡ cấu trúc chương trình
4. Continue: nhảy tới cuối thân vòng lặp và thực hiện vòng lặp tiếp theo
5. Con trỏ NULL: báo hiệu không có vùng nhớ nào được cấp phát cho con trỏ này

Int \*ptr {0} hoặc Int \*ptr = nullptr

1. Con trỏ lơ lửng: con trỏ trỏ tới vùng nhớ chưa được cấp phát hoặc trỏ tới 1 con trỏ NULL
2. Con trỏ hằng:**const int\*ptr** trỏ tới 1 biến là const hoặc biến thông thường, có thể đổi được địa chỉ mà nó trỏ đến nhưng không đổi được giá trị
3. Hằng con trỏ: **int \*const ptr** là con trỏ không thể thay đổi địa chỉ vùng nhớ mà nó trỏ tới nhưng thay đổi được giá trị, phải được khởi tạo khi khịa báo và địa chỉ không thay đổi về sau
4. Hằng con trỏ hằng: **const int \*const prt**: vừa không thay đổi được địa chỉ mà nó trỏ đến vừa không thay đổi được giá trị
5. Dùng tham chiếu nếu bạn cần truy cập hoặc thay đổi đối tượng một cách đơn giản, an toàn, không cần trỏ linh hoạt.
6. Dùng con trỏ khi bạn cần quản lý bộ nhớ linh hoạt, có thể không có đối tượng, hoặc thao tác với cấu trúc động phức tạp.
7. Cấp phát bộ nhớ:
   1. Tĩnh: tồn tại suốt chương trình
   2. Tự động: thu hồi khi kết thúc hàm
   3. Động: thời điểm chương trình đang chạy. Gán địa chỉ được cấp phát vào 1 con trỏ (new)
   4. Giải phóng: delete;
8. Cấp phát mảng động:
   1. Int \*arr = new int[length];
   2. Int \*arr = new int[] {1,2,3} => lỗi
   3. Delete[] arr;
9. Sự khác biệt giữa new và malloc() là gì?
   1. New: gọi constructor, có kiểu trả về
   2. Malloc: không gọi constructor, trả về void , cần ép kiểu
10. Biến tham chiếu: không được cấp phát bộ nhớ mà chỉ làm bí danh cho đối tượng được tham chiếu, phải được khởi tạo khi khai báo, không thể thay đổi tham chiếu đến biến khác khi đã khởi tạo
11. Tham chiếu hằng: tham chiếu đến 1 biến hằng
12. Function Overloading: sử dụng cùng 1 tên gọi cho nhiều hàm cùng mục đích nhưng khác nhau về kiểu dữ liệu tham số hoặc số lượng tham số.
    1. hàm chỉ khác nhau về kiểu trả về hoặc tham số hàm là typedef, con trỏ với mảng thì không nạp chồng được.
    2. Const trong tham số chỉ nạp chồng được khi tham số const là tham chiếu hoặc con trỏ
13. Operator overloading: định nghĩa lại toán tử
14. Class: 
    1. là một khuôn mẫu (template/blueprint) để tạo ra các đối tượng.
    2. Nó định nghĩa các thuộc tính (biến – data members) và hành vi (hàm – member functions) mà đối tượng của lớp đó có.
15. Objects:
    1. Là một thể hiện (instance) cụ thể của một lớp.
    2. Lưu ở stack (nếu kích thước lớn thì nên cấp phát động)
    3. Khi bạn tạo một biến từ lớp, bạn đang tạo một đối tượng.
    4. Mỗi đối tượng có vùng nhớ riêng để chứa dữ liệu của nó, nhưng dùng cùng một cấu trúc hành vi từ lớp.
16. Contructor: hàm khởi tạo luôn ở public, tự động chạy khi khởi tạo đối tượng. Không có kiểu trả về nào mà được chỉ định cho hàm tạo. Hàm tạo ở lớp cơ sở thì không được lớp dẫn xuất kế thừa
17. Định nghĩa hàm: nếu định nghĩa bên ngoài class thì phải sử dụng dấu ::
18. Con trỏ this: là một con trỏ ngầm định đặc biệt có sẵn trong tất cả các hàm thành viên của lớp. Nó trỏ đến đối tượng hiện tại đang gọi hàm. Khi bạn gọi một hàm thành viên trên một đối tượng, con trỏ this sẽ chứa địa chỉ của đối tượng đó.
19. Friend function: là hàm được định nghĩa ngoài vùng của một class nhưng có quyền truy cập tất cả các thành phần private và protected của class. Hàm ạn không thuộc class nên không được thừa kế
20. Friend class: khi class B là bạn class A, tất cả hàm thành phần của B sẽ là hàm bạn của A. Ngược lại không đúng. Một class có thể có nhiều class bạn và một class có thể là class bạn của nhiều class
21. Biến static trong OOP:
    1. chỉ có 1 bản sao của thành phần static. Thành phần static của class được dùng chung bởi mọi đối tường của class chứa nó.
    2. Dữ liệu kiểu static được khởi tọa mặc định là 0 khi đối tượng đầu tiên được tạo nếu ta không chủ định tạo cho nó 1 giá trị khác
    3. Ta không thể khởi tạo giá trị cho biến static trong class nhưng có thể làm ở ngoài class bằng dấu ::

1. Hàm static trong OOP: cho phép hàm đó có khả năng độc lập với tất cả các đối tượng của class.
   1. Có thể được gọi ngay cả khi không có bất kì đối tượng nào tồn tại
   2. Chỉ có thể truy cập được các biến và hàm static trong class và các hàm khác ngoài class
   3. Thuộc class nhưng không dùng được con trỏ this
2. Đối tượng hằng: chỉ có thể gọi tới hàm hằng ngoại trừ hàm tạo và hủy
   1. Hàm hằng chỉ có thể gọi tới và sử dụng các hàm hằng. Từ khóa const đặt sau tên hàm
3. **Tính đóng gói:** là cơ chế che giấu các chi tiết bên trong của một đối tượng và chỉ cho phép truy cập thông qua các **phương thức công khai (getter, setter)**
4. **Tính kế thừa:** cho phép một lớp con (subclass) kế thừa các thuộc tính (biến) và phương thức (hàm) từ một lớp cha (superclass).
5. Tính trừu tượng: là khả năng ẩn đi chi tiết thực thi, chỉ hiển thị những gì cần thiết với người dùng.  
   Nói cách khác, bạn chỉ cung cấp giao diện, còn chi tiết bên trong được ẩn đi.
6. Tính đa hình: cho phép một hành động có thể được thực hiện theo nhiều cách khác nhau, tùy vào ngữ cảnh hoặc đối tượng cụ thể.
7. Phương thức ảo: thêm từ khóa virtual ở phương thức lớp cha, lớp con sẽ định nghĩa lại phương thức đó, dùng con trỏ lớp cha trỏ đến lớp con thì sẽ sử dụng phương thức lớp con. Hàm ảo cho phép gọi đúng hàm của lớp con qua con trỏ lớp cha.
8. Phương thức thuần ảo: virtual + tên = 0; phải định nghĩa ở lớp con
9. Sự khác biệt giữa ghi đè và nạp chồng phương thức
   1. Nạp chồng là khả năng các hàm có cùng tên được xác định miễn là các phương thức này có các chữ ký khác nhau.
   2. Ghi đè phương thức là khả năng của lớp kế thừa viết lại những phương thức ảo của lớp cơ sở.
10. Có 2 loại đa hình:
    1. *Đa hình thời gian biên dịch*
    * Nạp chồng hàm
    * Nạp chồng toán tử
    1. *Đa hình thời gian chạy*
    * Ghi đè hàm
    * Hàm ảo
11. Có thể overloaf function hàm huỷ không?
    1. Không vì không có đối số để phân biệt
    2. Có thể dùng hàm hủy ảo
12. Thứ tự hàm tạo: lớp cha -> lớp con
13. Tại sao chúng ta cần destructor trong một lớp (class) trong C++?

Để giải phóng tài nguyên và tránh rò rỉ bộ nhớ

1. Hàm hủy ảo: Để đảm bảo hàm hủy của lớp con cũng được gọi, bạn cần khai báo hàm hủy trong lớp cha là hàm hủy ảo (virtual) để tránh rò rỉ bộ nhớ khi lớp con không được hủy
2. Toán tử phân giải phạm vi là gì?
   1. Toán tử phân giải phạm vi trong C++ là ::, được gọi là **scope resolution operator**. Nó dùng để truy cập thành phần nằm ngoài phạm vi hiện tại, chẳng hạn như
   2. **Truy cập biến/toán tử toàn cục (global)**
   3. Định nghĩa hàm thành viên bên ngoài lớp
   4. Truy cập thành phần của namespace
3. mutable trong C++ :
   1. được dùng để cho phép thay đổi một thành viên dữ liệu trong một object const, ngay cả khi các hàm thành viên là const.
   2. mutable chỉ dùng cho biến thành viên, không áp dụng cho biến cục bộ trong hàm.
4. Early Binding (hay còn gọi là Static Binding) là quá trình liên kết lời gọi hàm với định nghĩa hàm được thực hiện tại thời điểm biên dịch (compile-time). Điều kiện xảy ra:
   1. Trình biên dịch biết rõ kiểu của đối tượng và hàm sẽ được gọi.
   2. Không có tính đa hình động (dynamic polymorphism).
   3. Hàm không được khai báo là virtual.
5. Namespace : là một cách tổ chức mã nguồn để tránh xung đột tên giữa các hàm, biến, lớp... khi chương trình có nhiều phần được viết bởi các lập trình viên khác nhau, hoặc dùng nhiều thư viện.
   1. Cú pháp: namespace ten\_namespace{}
   2. Sử dụng Using namespace ten\_namespace hoặc ten\_namespace::
   3. Bí danh: đặt 1 tên namespace ngắn gọn gán cho namespace có tên dài
6. STL:
   1. STL (Standard Template Library) là thư viện chuẩn trong C++, cung cấp một tập hợp mạnh mẽ các cấu trúc dữ liệu và thuật toán được xây dựng bằng template. STL giúp bạn lập trình nhanh hơn, hiệu quả hơn và dễ bảo trì hơn.
   2. Thành phần: Container (cấu trúc DL), Algorithm(thuật toán), Iterator(truy cập phần tử trong container)
   3. Giảm thời gian viết code, Giảm lỗi logic. Dễ bảo trì, dễ đọc.
7. Ngoại lệ (exception) trong C++ : là một cơ chế để xử lý các lỗi bất ngờ hoặc tình huống không mong muốn xảy ra trong chương trình. Thay vì để chương trình gặp sự cố và dừng đột ngột, bạn có thể ném một ngoại lệ và xử lý nó ở nơi khác trong chương trình, giúp chương trình tiếp tục chạy một cách an toàn.
8. Đa kế thừa là một cơ chế trong lập trình hướng đối tượng, cho phép một lớp con kế thừa từ nhiều lớp cơ sở (base classes). Điều này có nghĩa là một lớp con có thể thừa hưởng các đặc tính và hành vi (attributes và methods) từ hơn một lớp cơ sở.
   1. Tái sử dụng mã , Mở rộng chức năng:
   2. Khi có nhiều hàm tên giống nhau => lỗi không nhận biết được hàm nào được gọi, phải dùng :: để chỉ rõ hàm trong lớp nào
   3. Diamond problem: kế thừa của nhiều lớp cha cùng kế thừa lớp ông => chồng chéo chức năng => sử dụng Virtual (thừa kế ảo)
9. Kế thừa ảo: là một kỹ thuật đảm bảo chỉ một bản sao của các biến thành viên của lớp cơ sở được kế thừa bởi các lớp có nguồn gốc từ cháu.
10. Khái niệm về con trỏ thông minh (smart pointers) trong C++

Con trỏ thông minh là các đối tượng trong C++ giúp tự động quản lý bộ nhớ, tránh rò rỉ bộ nhớ và đảm bảo việc giải phóng bộ nhớ sau khi không còn sử dụng. C++ cung cấp các loại con trỏ thông minh như unique\_ptr, shared\_ptr, và weak\_ptr.

54. Sự khác biệt giữa con trỏ unique\_ptr, shared\_ptr và weak\_ptr trong C++

* unique\_ptr: Là con trỏ thông minh chỉ sở hữu bộ nhớ của một đối tượng duy nhất. Khi unique\_ptr ra khỏi phạm vi, bộ nhớ sẽ tự động được giải phóng.
* shared\_ptr: Cho phép chia sẻ quyền sở hữu một đối tượng giữa nhiều con trỏ. Bộ nhớ sẽ chỉ được giải phóng khi tất cả các shared\_ptr đến đối tượng đó đều bị hủy.
* weak\_ptr: Được sử dụng để theo dõi một đối tượng mà không ảnh hưởng đến việc giải phóng bộ nhớ của đối tượng đó.

55. Giải thích khái niệm của đa hình tĩnh (static polymorphism) và đa hình động (dynamic polymorphism) trong C++

* Đa hình tĩnh (Static Polymorphism): Là kỹ thuật trong C++ cho phép chọn hàm gọi tại thời gian biên dịch, thường sử dụng overloading (nạp chồng) hoặc templates.
* Đa hình động (Dynamic Polymorphism): Là kỹ thuật cho phép gọi hàm vào thời gian chạy thông qua con trỏ hoặc tham chiếu đến lớp cơ sở, với virtual functions.

56. Khái niệm về hàm lambda trong C++

Hàm lambda là một hàm vô danh (anonymous function) trong C++ có thể được định nghĩa ngay trong code và có thể sử dụng các biến bên ngoài phạm vi của nó

64. Thread

* Thread (hay luồng) là đơn vị thực thi độc lập trong chương trình. Mỗi thread có thể thực hiện một tác vụ riêng biệt trong khi các thread khác có thể chạy song song

58. Khái niệm về đa luồng (multithreading) trong C/C++

Đa luồng là khả năng của một chương trình để thực hiện nhiều tác vụ đồng thời, với mỗi luồng thực hiện một phần công việc độc lập.

59. Cách đồng bộ hóa các luồng (thread synchronization) trong C/C++

Để đồng bộ hóa các luồng trong C++, bạn có thể sử dụng các công cụ như mutex, lock\_guard, hoặc condition\_variable để tránh tình trạng tranh chấp tài nguyên.

60. Sự khác biệt giữa các loại biến đồng bộ (synchronized variables) như mutex, semaphore và critical section trong C/C++

* mutex: Là một công cụ dùng để đồng bộ hóa các luồng, cho phép chỉ một luồng có thể sở hữu mutex tại một thời điểm. Các luồng khác phải chờ cho đến khi mutex được giải phóng.
* semaphore: Là một công cụ đồng bộ hóa mạnh mẽ hơn, có thể kiểm soát số lượng luồng truy cập tài nguyên. Semaphore có thể có một giá trị đếm, và luồng sẽ bị chặn nếu giá trị đó là 0.
* critical section: Là một vùng mã trong đó chỉ có một luồng có thể thực thi tại một thời điểm.

61. Lock và Mutex

* Mutex: Là một đối tượng dùng để bảo vệ dữ liệu chia sẻ trong môi trường đa luồng, đảm bảo chỉ có một luồng có thể truy cập dữ liệu tại một thời điểm. mutex (viết tắt của mutual exclusion) có thể được khóa (lock) và mở khóa (unlock).
* Lock: Là một công cụ giúp dễ dàng quản lý việc khóa và mở khóa một mutex. std::lock\_guard và std::unique\_lock là các loại lock phổ biến giúp đảm bảo mutex được khóa và mở khóa đúng cách. Khi sử dụng lock, mutex sẽ tự động được khóa khi lock được tạo và tự động mở khóa khi lock bị hủy (scope hết hạn).
* Lock\_guard: Không thể mở khóa thủ công (mở khóa tự động khi ra khỏi phạm vi)
* Unique\_lock: Có thể mở khóa thủ công và khóa lại khi cần

65. Template

* Template là một tính năng trong C++ cho phép bạn viết các hàm hoặc lớp có thể làm việc với nhiều kiểu dữ liệu khác nhau mà không cần phải viết lại mã cho mỗi kiểu. C++ hỗ trợ function templates và class templates.

66. Variadic Template

* Variadic Template là một kỹ thuật cho phép bạn viết các hàm hoặc lớp có thể chấp nhận số lượng tham số không xác định. Điều này rất hữu ích khi làm việc với các hàm nhận một số lượng tham số thay đổi.

67. Sự khác nhau giữa list và vector

* std::list: Là một danh sách liên kết (doubly linked list) trong C++, nơi mỗi phần tử giữ một con trỏ đến phần tử trước và sau nó. Điều này cho phép thêm và xóa phần tử ở đầu và cuối danh sách nhanh chóng, nhưng truy cập các phần tử ngẫu nhiên chậm (O(n)).
  + Ưu điểm: Thêm và xóa phần tử nhanh chóng.
  + Nhược điểm: Truy cập phần tử ngẫu nhiên chậm.
* std::vector: Là một mảng động trong C++, nơi các phần tử được lưu trữ trong bộ nhớ liên tiếp. vector cho phép truy cập phần tử nhanh chóng, nhưng việc thêm hoặc xóa phần tử ở giữa mảng có thể tốn chi phí.
  + Lưu trong heap
  + Ưu điểm: Truy cập phần tử nhanh chóng và hiệu quả về bộ nhớ.
  + Nhược điểm: Thêm và xóa phần tử ở giữa mảng chậm.