VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Kỹ Thuật Lập Trình (OOP trong c++)

KTLT2 - HK242

TASK 8 Kiến trúc sư OOP - Kiểm soát và tổ chức

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 3/2025

1 Trắc Nghiệm

1. Đóng gói (Encapsulation) trong lập trình hướng đối tượng là gì?
   a) Quá trình ẩn giấu thông tin chi tiết của một lớp và chỉ cung cấp giao diện công khai.
   b) Kỹ thuật sử dụng con trỏ để trỏ đến dữ liệu trong lớp.
   c) Quá trình chia chương trình thành nhiều phần nhỏ hơn.
   d) Cách sử dụng lại mã nguồn bằng cách kế thừa từ lớp khác.

2. Mục đích chính của đóng gói là gì?
   a) Ngăn chặn truy cập trực tiếp vào dữ liệu nội bộ của lớp.
   b) Tăng tốc độ thực thi chương trình.
   c) Giảm dung lượng bộ nhớ sử dụng.
   d) Loại bỏ lỗi cú pháp trong mã nguồn.

3. Trong C++, có bao nhiêu mức độ truy cập được sử dụng trong đóng gói?
   a) 2 b) 3 c) 4 d) 5

4. Thành viên nào có thể truy cập vào các thuộc tính private của một lớp?
   a) Chỉ các phương thức của lớp đó.
   b) Bất kỳ lớp nào trong chương trình.
   c) Chỉ các lớp kế thừa từ lớp đó.
   d) Bất kỳ phương thức nào trong chương trình.

5. Getter và Setter trong đóng gói có tác dụng gì?
   a) Kiểm soát cách truy cập và thay đổi giá trị của các thuộc tính private.
   b) Tăng tốc độ truy cập dữ liệu trong lớp.
   c) Giảm số lượng dòng lệnh trong chương trình.
   d) Thay thế toàn bộ các thuộc tính của lớp.

6. Trong trường hợp nào dưới đây nguyên tắc đóng gói bị vi phạm?
   a) Một lớp cung cấp toàn bộ dữ liệu của nó ở mức public.
   b) Một lớp sử dụng các phương thức public để truy cập dữ liệu.
   c) Một lớp có các thuộc tính private nhưng có các phương thức getter.
   d) Một lớp có cả phương thức public và private.

7. Đâu là lợi ích của đóng gói trong lập trình hướng đối tượng?
   a) Bảo vệ dữ liệu khỏi bị thay đổi không mong muốn.
   b) Tăng kích thước mã nguồn.
   c) Giảm hiệu suất chương trình.
   d) Làm cho chương trình khó đọc hơn.

8. Đâu là cách tổ chức đúng của một lớp trong C++ sử dụng đóng gói?
   a) Đặt các thuộc tính ở mức private, sử dụng các phương thức public để truy cập dữ liệu.
   b) Đặt tất cả thuộc tính và phương thức ở mức public.
   c) Không sử dụng bất kỳ mức độ truy cập nào.
   d) Sử dụng toàn bộ thuộc tính ở mức protected.

9. Trong C++, từ khóa nào được sử dụng để khai báo thành viên private của một lớp?
   a) private b) protected c) public d) static

10. Chọn phát biểu đúng về đóng gói trong C++.
    a) Đóng gói giúp kiểm soát truy cập dữ liệu và giảm nguy cơ lỗi.
    b) Đóng gói làm cho mã nguồn phức tạp hơn và khó bảo trì.
    c) Đóng gói chỉ có tác dụng khi sử dụng với kế thừa.
    d) Đóng gói không ảnh hưởng đến độ an toàn của dữ liệu.

11. Hàm bạn (friend function) trong C++ là gì?

a) Một hàm có thể truy cập vào các thành viên private và protected của một lớp.
b) Một hàm chỉ có thể gọi bởi các đối tượng của lớp.
c) Một hàm được kế thừa từ lớp cha.
d) Một hàm chỉ có thể truy cập các thành viên public của lớp.

12. Từ khóa nào được sử dụng để khai báo một hàm bạn trong C++?

a) friend b) private c) protected d) public

13. Đâu là đặc điểm của một hàm bạn?

a) Không phải là thành viên của lớp nhưng có thể truy cập private và protected.
b) Là một phương thức thành viên của lớp.
c) Luôn phải được khai báo trong phạm vi public của lớp.
d) Không thể truy cập các thành viên private của lớp.

14. Khi nào cần sử dụng hàm bạn?

a) Khi một hàm bên ngoài cần truy cập các thành viên private của lớp.
b) Khi muốn kế thừa các phương thức từ lớp cha.
c) Khi cần bảo vệ hoàn toàn dữ liệu khỏi mọi truy cập bên ngoài.
d) Khi muốn một hàm có thể gọi được từ bất kỳ đâu trong chương trình.

15. Hàm bạn thường được sử dụng để làm gì trong C++?

a) Định nghĩa toán tử cho các lớp (operator overloading).
b) Tăng tốc độ thực thi của chương trình.
c) Giảm kích thước bộ nhớ sử dụng.
d) Giúp kế thừa nhiều lớp dễ dàng hơn.

16. Lớp bạn (friend class) trong C++ là gì?

a) Một lớp có thể truy cập trực tiếp vào các thành viên private và protected của một lớp khác.
b) Một lớp có thể kế thừa từ nhiều lớp khác nhau.
c) Một lớp không thể có bất kỳ phương thức nào.
d) Một lớp chỉ có thể chứa các phương thức public.

17. Đâu là cú pháp đúng để khai báo một lớp là bạn của một lớp khác?

a) `friend class ClassName;`
b) `private class ClassName;`
c) `protected class ClassName;`
d) `public class ClassName;`

18. Quan hệ giữa lớp bạn và lớp mà nó truy cập là gì?

a) Quan hệ một chiều, lớp bạn có thể truy cập lớp khai báo nó, nhưng ngược lại không đúng.
b) Quan hệ hai chiều, cả hai lớp đều có thể truy cập lẫn nhau.
c) Lớp bạn phải kế thừa từ lớp được khai báo.
d) Lớp bạn chỉ có thể truy cập các thành viên public của lớp khác.

19. Lớp bạn thường được sử dụng để làm gì trong lập trình hướng đối tượng?

a) Cho phép hai lớp có thể truy cập dữ liệu của nhau mà không cần getter/setter.
b) Giúp chương trình chạy nhanh hơn.
c) Giảm số lượng lớp cần thiết trong chương trình.
d) Bảo vệ dữ liệu khỏi truy cập bên ngoài.

20. Nhược điểm chính của hàm bạn và lớp bạn là gì?

a) Làm giảm tính đóng gói vì cho phép truy cập dữ liệu private.
b) Tăng cường bảo mật dữ liệu trong lớp.

c) Giúp chương trình dễ bảo trì hơn.
d) Giảm số lượng dòng lệnh trong chương trình.

21. Kế thừa (Inheritance) trong lập trình hướng đối tượng là gì?
    a) Một lớp mới có thể sử dụng lại các thuộc tính và phương thức của một lớp khác.
    b) Một kỹ thuật để giấu thông tin của một lớp.
    c) Một cách để tạo nhiều đối tượng từ một lớp.
    d) Một phương pháp để tối ưu hóa bộ nhớ trong C++.

22. Mục đích chính của kế thừa trong OOP là gì?
    a) Tái sử dụng mã và giảm trùng lặp.
    b) Tăng tốc độ thực thi chương trình.
    c) Giảm kích thước của chương trình.
    d) Giúp lớp con truy cập trực tiếp vào dữ liệu private của lớp cha.

23. Kế thừa thể hiện mối quan hệ nào giữa các lớp?
    a) Quan hệ "is-a" (là một).
    b) Quan hệ "has-a" (có một).
    c) Quan hệ một-nhiều.
    d) Quan hệ kết hợp (association).

24. Cú pháp khai báo một lớp kế thừa từ lớp khác trong C++ là gì?
    a) `class Derived : public Base`
    b) `class Base : public Derived`
    c) `class Derived extends Base`
    d) `class Base inherits Derived`

25. Một lớp có thể kế thừa từ bao nhiêu lớp trong C++?
    a) Không giới hạn.
    b) Chỉ một lớp.
    c) Tối đa hai lớp.
    d) Không thể kế thừa từ lớp khác.

26. Kế thừa đơn (Single Inheritance) là gì?
    a) Một lớp kế thừa từ một lớp duy nhất.
    b) Một lớp kế thừa từ nhiều lớp.
    c) Một lớp có thể có nhiều đối tượng.
    d) Một lớp có thể truy cập dữ liệu của lớp khác mà không cần kế thừa.

27. Trong kế thừa đa cấp (Multilevel Inheritance), điều gì xảy ra?
    a) Một lớp kế thừa từ lớp khác, và lớp này lại kế thừa từ một lớp khác nữa.
    b) Một lớp kế thừa từ nhiều lớp cha trực tiếp.
    c) Một lớp có thể tạo nhiều đối tượng từ lớp cha.
    d) Một lớp có thể gọi trực tiếp các phương thức của lớp cha mà không cần kế thừa.

28. Đặc điểm của kế thừa đa thừa (Multiple Inheritance) là gì?
    a) Một lớp có thể kế thừa từ nhiều lớp cha cùng lúc.
    b) Một lớp chỉ có thể kế thừa từ một lớp cha.
    c) Kế thừa từ một lớp trung gian.
    d) Một lớp có thể truy cập tất cả thuộc tính của các lớp khác mà không cần kế thừa.

29. Kế thừa phân cấp (Hierarchical Inheritance) là gì?
    a) Nhiều lớp con kế thừa từ một lớp cha chung.
    b) Một lớp kế thừa từ nhiều lớp cha.
    c) Một lớp kế thừa từ một lớp cha duy nhất.
    d) Một lớp không thể kế thừa từ lớp khác.

30. Kế thừa lai (Hybrid Inheritance) là gì?
    a) Kết hợp nhiều kiểu kế thừa khác nhau trong một chương trình.
    b) Chỉ kế thừa từ một lớp duy nhất.
    c) Một kỹ thuật để tạo đối tượng từ một lớp.
    d) Một cách để giới hạn quyền truy cập vào dữ liệu.

31. Khi một lớp kế thừa từ lớp khác với từ khóa public, điều gì xảy ra?

a) Thành viên public và protected của lớp cha giữ nguyên trong lớp con.
b) Thành viên private của lớp cha có thể truy cập trực tiếp từ lớp con.
c) Tất cả thành viên của lớp cha trở thành private trong lớp con.
d) Lớp con không thể truy cập bất kỳ thành viên nào của lớp cha.

32. Khi kế thừa protected, điều gì xảy ra?

a) Thành viên public và protected của lớp cha trở thành protected trong lớp con.
b) Thành viên private của lớp cha trở thành protected trong lớp con.
c) Lớp con không thể truy cập thành viên protected của lớp cha.
d) Thành viên public của lớp cha trở thành public trong lớp con.

33. Khi kế thừa private, điều gì xảy ra?

a) Thành viên public và protected của lớp cha trở thành private trong lớp con.
b) Thành viên private của lớp cha trở thành public trong lớp con.
c) Lớp con có thể truy cập trực tiếp vào tất cả các thành viên của lớp cha.
d) Không có sự thay đổi nào.

34. Khi tạo một đối tượng của lớp con, trình biên dịch sẽ gọi hàm tạo theo thứ tự nào?

a) Từ lớp cha đến lớp con.
b) Từ lớp con đến lớp cha.
c) Chỉ gọi hàm tạo của lớp con.
d) Chỉ gọi hàm tạo của lớp cha.

35. Khi một đối tượng bị hủy, trình biên dịch gọi hàm hủy theo thứ tự nào?

a) Từ lớp con đến lớp cha.
b) Từ lớp cha đến lớp con.
c) Chỉ gọi hàm hủy của lớp con.
d) Chỉ gọi hàm hủy của lớp cha.

36. Khi nào nên sử dụng kế thừa trong thiết kế phần mềm?

a) Khi có quan hệ "is-a" giữa hai lớp.
b) Khi một lớp cần truy cập dữ liệu private của lớp khác.
c) Khi muốn chia sẻ một biến toàn cục giữa nhiều lớp.
d) Khi muốn giảm kích thước của chương trình.

37. Một lớp cơ sở (base class) tốt nên có đặc điểm gì?

a) Chỉ chứa các thuộc tính và phương thức chung giữa các lớp con.
b) Chứa toàn bộ logic của chương trình.
c) Chỉ chứa các phương thức private.
d) Không có bất kỳ phương thức ảo nào.

38. Điều gì có thể xảy ra nếu một lớp con kế thừa từ một lớp cha không phù hợp?

a) Lớp con có thể kế thừa các phương thức không liên quan.
b) Lớp con không thể truy cập bất kỳ phương thức nào của lớp cha.
c) Lớp cha sẽ bị thay đổi mỗi khi lớp con cần thêm chức năng.
d) Mã nguồn của lớp cha sẽ bị xóa.

39. Một lợi ích chính của kế thừa là gì?

a) Cho phép tái sử dụng mã bằng cách chia sẻ logic chung giữa các lớp.
b) Cho phép một lớp truy cập trực tiếp dữ liệu private của lớp khác.
c) Giảm kích thước tệp nguồn trong chương trình.
d) Tăng tốc độ thực thi chương trình.

40. Trong trường hợp nào không nên sử dụng kế thừa?

a) Khi hai lớp không có quan hệ "is-a" rõ ràng.
b) Khi cần chia sẻ mã nguồn giữa nhiều lớp.
c) Khi muốn mở rộng chức năng của lớp cha.
d) Khi một lớp cần truy cập một phương thức từ lớp khác.

41. Nếu một lớp A chứa một lớp B thay vì kế thừa từ B, điều này thể hiện nguyên tắc thiết kế nào?
    a) "Has-a" (có một) thay vì "is-a" (là một).
    b) "Is-a" (là một).
    c) "Uses-a" (sử dụng một).
    d) "Belongs-to" (thuộc về).

42. Trong một ứng dụng mô phỏng động vật, bạn có lớp Bird với phương thức fly(). Tuy nhiên, bạn cần tạo lớp Penguin (chim cánh cụt), nhưng nó không bay được. Giải pháp nào hợp lý nhất?
    a) Di chuyển phương thức fly() ra khỏi lớp Bird và đặt vào một lớp riêng biệt.
    b) Tạo một lớp Penguin kế thừa từ Bird và ghi đè phương thức fly() để trống.
    c) Không tạo lớp Penguin vì nó không thể bay.
    d) Xóa phương thức fly() khỏi lớp Bird.

43. Khi kế thừa public, điều gì xảy ra với các thành viên protected của lớp cha?
    a) Chúng trở thành protected trong lớp con.
    b) Chúng trở thành private trong lớp con.
    c) Chúng trở thành public trong lớp con.
    d) Chúng không thể truy cập từ lớp con.

44. Lợi ích chính của kế thừa nhiều lớp là gì?
    a) Giúp một lớp kết hợp chức năng từ nhiều lớp cha.
    b) Giảm số lượng dòng lệnh trong chương trình.
    c) Tránh được xung đột giữa các lớp cha.
    d) Giảm thời gian thực thi của chương trình.

45. Một trong những vấn đề phổ biến nhất của kế thừa nhiều lớp là gì?
    a) Vấn đề kim cương (Diamond Problem), khi hai lớp cha kế thừa từ cùng một lớp gốc.
    b) Không thể sử dụng con trỏ đến lớp cha.
    c) Lớp con không thể truy cập các thành viên của lớp cha.
    d) Không thể sử dụng từ khóa virtual trong kế thừa.

46. Trong kế thừa nhiều lớp, làm thế nào để tránh vấn đề kim cương?
    a) Sử dụng kế thừa ảo (virtual inheritance).
    b) Tránh sử dụng hàm tạo trong lớp cha.
    c) Không bao giờ sử dụng kế thừa trong C++.
    d) Chỉ kế thừa từ một lớp duy nhất.

47. Đâu là cú pháp đúng để khai báo một lớp kế thừa từ hai lớp cha trong C++?
    a) `class Derived : public Base1, public Base2`
    b) `class Base1, Base2 : public Derived`
    c) `class Derived inherits Base1, Base2`
    d) `class Derived : Base1, Base2`

48. Kế thừa nhiều lớp (Multiple Inheritance) trong C++ là gì?
    a) Một lớp kế thừa từ nhiều lớp cha cùng lúc.
    b) Một lớp có nhiều đối tượng cùng loại.
    c) Một lớp chỉ có thể kế thừa từ một lớp duy nhất.
    d) Một lớp kế thừa từ một lớp cha nhưng có thể có nhiều phương thức.

49. Lợi ích chính của kế thừa nhiều lớp là gì?
    a) Giúp một lớp kết hợp chức năng từ nhiều lớp cha.
    b) Giảm số lượng dòng lệnh trong chương trình.
    c) Tránh được xung đột giữa các lớp cha.
    d) Giảm thời gian thực thi của chương trình.

50. Một trong những vấn đề phổ biến nhất của kế thừa nhiều lớp là gì?
    a) Vấn đề kim cương (Diamond Problem), khi hai lớp cha kế thừa từ cùng một lớp gốc.
    b) Không thể sử dụng con trỏ đến lớp cha.
    c) Lớp con không thể truy cập các thành viên của lớp cha.
    d) Không thể sử dụng từ khóa virtual trong kế thừa.

51. Trong kế thừa nhiều lớp, làm thế nào để tránh vấn đề kim cương?
    a) Sử dụng kế thừa ảo (virtual inheritance).
    b) Tránh sử dụng hàm tạo trong lớp cha.
    c) Không bao giờ sử dụng kế thừa trong C++.
    d) Chỉ kế thừa từ một lớp duy nhất.

52. Đâu là cú pháp đúng để khai báo một lớp kế thừa từ hai lớp cha trong C++?

a) `class Derived : public Base1, public Base2`
b) `class Base1, Base2 : public Derived`
c) `class Derived inherits Base1, Base2`
d) `class Derived : Base1, Base2`

53. Quá tải hàm (Function Overloading) trong C++ là gì?

a) Việc định nghĩa nhiều hàm cùng tên nhưng khác danh sách tham số.
b) Việc ghi đè (override) một hàm trong lớp con.
c) Việc gọi một hàm từ một lớp cha bằng con trỏ.
d) Việc sử dụng nhiều toán tử khác nhau trên cùng một biến.

54. Hai hàm có thể được quá tải nếu chúng khác nhau về điều gì?

a) Số lượng tham số hoặc kiểu dữ liệu của tham số.
b) Kiểu trả về của hàm.
c) Tên của hàm.
d) Thứ tự khai báo trong chương trình.

55. Lợi ích chính của quá tải hàm là gì?

a) Giúp sử dụng cùng một tên hàm cho nhiều kiểu dữ liệu khác nhau.
b) Giảm số lượng lớp trong chương trình.
c) Tăng tốc độ thực thi của chương trình.
d) Giúp lớp con ghi đè phương thức của lớp cha.

56. Trong trường hợp nào sau đây, hai hàm không thể quá tải?

a) Chỉ khác nhau về kiểu trả về.
b) Chỉ khác nhau về số lượng tham số.
c) Chỉ khác nhau về kiểu dữ liệu của tham số.
d) Chỉ khác nhau về thứ tự tham số.

57. Quá tải toán tử (Operator Overloading) trong C++ là gì?

a) Định nghĩa lại cách một toán tử hoạt động trên kiểu dữ liệu người dùng định nghĩa.
b) Ghi đè một toán tử của C++ để thay đổi thứ tự ưu tiên.
c) Sử dụng toán tử trong các hàm bạn (friend functions).
d) Tạo toán tử mới trong C++.

58. Một toán tử có thể được quá tải bằng cách nào?

a) Định nghĩa lại toán tử bằng một hàm thành viên hoặc hàm bạn.
b) Ghi đè toán tử trong lớp con.
c) Sử dụng từ khóa ‘override‘ trong C++.
d) Thay đổi thứ tự ưu tiên của toán tử.

59. Đâu là một toán tử hợp lệ để quá tải trong C++?

a) +
b) ::
c) .
d) sizeof

60. Khi quá tải toán tử ‘«‘ để xuất dữ liệu của một lớp, kiểu nào nên được sử dụng?

a) Hàm bạn (friend function).
b) Hàm thành viên.
c) Biến toàn cục.
d) Macro.

61. Khi nào nên sử dụng quá tải toán tử?

a) Khi cần sử dụng toán tử với kiểu dữ liệu tự định nghĩa.
b) Khi muốn thay đổi ưu tiên của toán tử.
c) Khi cần tối ưu hóa hiệu suất chương trình.
d) Khi muốn giảm số lượng dòng lệnh trong chương trình.

62. Khi quá tải toán tử gán ‘=‘, điều gì quan trọng cần lưu ý?

a) Trả về một tham chiếu đến đối tượng hiện tại (‘*this‘).

b) Không thể quá tải toán tử ‘=‘.

c) Trả về một con trỏ mới đến đối tượng.

d) Chỉ có thể quá tải toán tử này bằng hàm bạn.

2 Đọc Code

Câu 1. Kết quả của chương trình

```cpp
class BankAccount {
private:
    double balance;

public:
    BankAccount(double initialBalance) {
        if (initialBalance >= 0) {
            balance = initialBalance;
        } else {
            balance = 0;
            cout << "Invalid balance, setting to
             ↪ 0." << endl;
        }
    }

    void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        } else {
            cout << "Insufficient funds!" <<
             ↪ endl;
        }
    }

    double getBalance() const {
        return balance;
    }
};

BankAccount acc(100);
acc.withdraw(50);
cout << "Balance: " << acc.getBalance() << endl;

acc.withdraw(60);
cout << "Balance: " << acc.getBalance() << endl;
```

Kết quả và giải thích: ...

- ...
- ...

Câu 2. Tìm lỗi quyền truy cập trong lớp và hàm

```cpp
#include <iostream>
using namespace std;

class Base {
private:
    int privateVar = 10;

protected:
    int protectedVar = 20;

public:
    int publicVar = 30;

    void showValues() {
        cout << "Private: " << privateVar <<
        ↪ endl;
        cout << "Protected: " << protectedVar <<
        ↪ endl;
        cout << "Public: " << publicVar << endl;
    }
};

class Derived : public Base {
public:
    void accessVariables() {
        cout << privateVar;
        cout << "Protected: " << protectedVar <<
        ↪ endl;
        cout << "Public: " << publicVar << endl;
    }
};

Base obj;
cout << obj.privateVar;
cout << obj.protectedVar;
cout << "Public: " << obj.publicVar << endl;

Derived d;
d.accessVariables();
```

Kết quả và giải thích: ...

- ...
- ...

Câu 3. Sử dụng hàm bạn trong C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Rectangle {
private:
    int width, height;

public:
    Rectangle(int w, int h) : width(w), height(h) {}

    friend void printArea(const Rectangle& rect);
};


void printArea(const Rectangle& rect) {
    cout << "Area: " << rect.width * rect.height << endl;
}

Rectangle r(4, 5);
printArea(r);
```

Câu 4. Sử dụng lớp bạn trong C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Engine; // Khai báo trước để sử dụng trong Car

class Car {
private:
    string model;
    int horsepower;

public:
    Car(string m, int hp) : model(m), horsepower(hp) {}

    // Khai báo Engine là lớp bạn
    friend class Engine;
};

class Engine {
public:
    void showCarInfo(const Car& car) {
        cout << "Car Model: " << car.model << endl;
        cout << "Horsepower: " << car.horsepower << endl;
    }
};

Car myCar("Tesla Model S", 670);
Engine e;
e.showCarInfo(myCar);
```

Câu 5. Kế thừa đơn trong C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Animal {
public:
    void makeSound() {
        cout << "Animal makes a sound" << endl;
    }
};

class Dog : public Animal {
public:
    void bark() {
        cout << "Dog barks" << endl;
    }
};

Dog d;
d.makeSound();  // Gọi hàm từ lớp cha
d.bark();       // Gọi hàm từ lớp con
```

Câu 6. Kế thừa đa cấp trong C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Grandparent {
public:
    void show() {
        cout << "Grandparent class" << endl;
    }
};

class Parent : public Grandparent {
public:
    void display() {
        cout << "Parent class" << endl;
    }
};

class Child : public Parent {
public:
    void print() {
        cout << "Child class" << endl;
    }
};
Child c;
c.show();    // Từ Grandparent
c.display(); // Từ Parent
c.print();   // Từ Child
```

Câu 7. Kế thừa nhiều lớp trong C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Engine {
public:
    void start() {
        cout << "Engine started" << endl;
    }
};

class Wheels {
public:
    void roll() {
        cout << "Wheels rolling" << endl;
    }
};

class Car : public Engine, public Wheels {
public:
    void drive() {
        cout << "Car is driving" << endl;
    }
};
Car c;
c.start();  // Từ Engine
c.roll();   // Từ Wheels
c.drive();  // Từ Car
```

Câu 10. Mức độ truy cập trong kế thừa

Kết quả, tìm lỗi và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    int publicVar = 1;

protected:
    int protectedVar = 2;

private:
    int privateVar = 3;
};

class DerivedPublic : public Base {
public:
    void show() {
        cout << "Public: " << publicVar << endl;
        cout << "Protected: " << protectedVar << endl;
        cout << "Private: " << privateVar << endl;
    }
};

DerivedPublic d;
d.show();
cout << "Access publicVar: " << d.publicVar << endl;
cout << "Access protectedVar: " << d.protectedVar << endl;
```

Câu 11. Kế thừa protected

Kết quả, tìm lỗi và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    int publicVar = 1;

protected:
    int protectedVar = 2;
};

class DerivedProtected : protected Base {
public:
    void show() {
        cout << "Public as Protected: " <<
        ↪ publicVar << endl;
        cout << "Protected: " << protectedVar
        ↪ << endl;
    }
};

DerivedProtected d;
d.show();
cout << d.publicVar;
```

Câu 12. Kế thừa private

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    int publicVar = 1;

protected:
    int protectedVar = 2;
};

class DerivedPrivate : private Base {
public:
    void show() {
        cout << "Public as Private: " <<
        ↪ publicVar << endl;
        cout << "Protected as Private: " <<
        ↪ protectedVar << endl;
    }
};

DerivedPrivate d;
d.show();
cout << d.publicVar;
```

Câu 13. Truy cập private bị chặn trong kế thừa

Kết quả, tìm lỗi và giải thích: ...

- ...
- ...

```cpp
class Base {
private:
    int privateVar = 5;

public:
    void showPrivate() {
        cout << "Private: " << privateVar <<
        ↪ endl;
    }
};

class Derived : public Base {
public:
    void accessPrivate() {
        cout << privateVar;
    }
};

Derived d;
d.showPrivate();
d.accessPrivate();
```

Câu 14. So sánh mức độ kế thừa

Kết quả, tìm lỗi và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    int publicVar = 1;
protected:
    int protectedVar = 2;
};

class PublicDerived : public Base {};
class ProtectedDerived : protected Base {};
class PrivateDerived : private Base {};

PublicDerived pub;
cout << pub.publicVar << endl;
cout << pub.protectedVar << endl;

ProtectedDerived prot;
cout << prot.publicVar << endl;

PrivateDerived priv;
cout << priv.publicVar << endl;
```

Câu 15. Thứ tự gọi hàm tạo trong kế thừa

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    Base() {
        cout << "Base constructor called!" <<
        ↪ endl;
    }
};

class Derived : public Base {
public:
    Derived() {
        cout << "Derived constructor called!"
        ↪ << endl;
    }
};

Derived* d = new Derived();
delete d;
```

Câu 16. Thứ tự gọi hàm hủy trong kế thừa

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Base {
public:
    Base() {
        cout << "Base constructor called!" <<
        ↪ endl;
    }
    ~Base() {
        cout << "Base destructor called!" <<
        ↪ endl;
    }
};

class Derived : public Base {
public:
    Derived() {
        cout << "Derived constructor called!"
        ↪ << endl;
    }
    ~Derived() {
        cout << "Derived destructor called!"
        ↪ << endl;
    }
};

Derived* d = new Derived();
delete d;
```

Câu 17. Truyền tham số từ hàm tạo lớp con sang lớp cha

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Base {
protected:
    int value;
public:
    Base(int v) {
        value = v;
        cout << "Base constructor called with value: " << value << endl;
    }
};

class Derived : public Base {
public:
    Derived(int v) : Base(v) {
        cout << "Derived constructor called with value: " << value << endl;
    }
};

Derived d(10);
```

Câu 20. Quá tải hàm trong lớp C++

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Calculator {
public:
    int add(int a, int b) {
        return a + b;
    }

    float add(float a, float b) {
        return a + b;
    }

    int add(int a, int b, int c) {
        return a + b + c;
    }
};

Calculator calc;

cout << "2 integers: " << calc.add(2, 3) << endl;
cout << "2 floats: " << calc.add(2.5f, 3.5f) << endl;
cout << "3 integers: " << calc.add(1, 2, 3) << endl;
```

Câu 19. Quá tải toán tử so sánh, nhập (‘»‘), và xuất (‘«‘) cho lớp Vector (Mảng)

```cpp
class Vector {
private:
    int* arr;
    int size;
public:
    Vector(int s) : size(s) {
        arr = new int[size];
        for (int i = 0; i < size; i++) arr[i] = 0;
    }
    ~Vector() {delete[] arr;}

    bool operator==(const Vector& v) const {
        if (size != v.size) return false;
        for (int i = 0; i < size; i++) {
            if (arr[i] != v.arr[i]) return false;
        }
        return true;
    }

    bool operator!=(const Vector& v) const {
        return !(*this == v);
    }

    friend ostream& operator<<(ostream& os, const Vector& v) {
        os << "[";
        for (int i = 0; i < v.size; i++) {
            os << v.arr[i];
            if (i < v.size - 1) os << ", ";
        }
        os << "]";
        return os;
    }

    friend istream& operator>>(istream& is, Vector& v) {
        for (int i = 0; i < v.size; i++) {
            is >> v.arr[i];
        }
        return is;
    }
};

int n;
cout << "Enter size of vectors: ";
cin >> n;

Vector v1(n), v2(n);
cout << "Enter elements for Vector 1: ";
cin >> v1;
cout << "Enter elements for Vector 2: ";
cin >> v2;

cout << "Vector 1: " << v1 << endl;
cout << "Vector 2: " << v2 << endl;

```

```
55 if (v1 == v2)
56     cout << "Vectors are equal!" << endl;
57 else
58     cout << "Vectors are not equal!" << endl;
```

Kết quả và giải thích: ...

- ...
- ...

3 Bài Tập thiết kế

Câu 1: Thiết kế hệ thống động vật với kế thừa nhiều cấp

Mô tả hệ thống:

Hệ thống bao gồm một lớp cha chung 'Animal' và các lớp con kế thừa theo cấp bậc:

1. 'Mammal' kế thừa từ 'Animal', đại diện cho động vật có vú.
2. 'Bird' kế thừa từ 'Animal', đại diện cho loài chim.
3. 'Dog' kế thừa từ 'Mammal', đại diện cho chó.
4. 'Duck' kế thừa từ 'Bird', đại diện cho vịt.

- **Lớp Animal (Lớp cha)**
  - **Thuộc tính:**
    - `string name` - Tên của động vật.
  - **Phương thức:**
    - `Animal(string n)` - Khởi tạo động vật với tên.
- **Lớp Mammal (Kế thừa từ Animal)**
  - **Thuộc tính:**
    - `bool hasFur` - Xác định động vật có lông hay không.
  - **Phương thức:**
    - `Mammal(string n, bool fur)` - Khởi tạo động vật có vú.
- **Lớp Bird (Kế thừa từ Animal)**
  - **Thuộc tính:**
    - `bool canFly` - Xác định chim có thể bay hay không.
  - **Phương thức:**
    - `Bird(string n, bool fly)` - Khởi tạo loài chim.
- **Lớp Dog (Kế thừa từ Mammal)**
  - **Thuộc tính:**
    - `string breed` - Giống chó.
  - **Phương thức:**
    - `Dog(string n, string b)` - Khởi tạo đối tượng chó.
- **Lớp Duck (Kế thừa từ Bird)**
  - **Thuộc tính:**
    - `bool isDomestic` - Xác định vịt là vịt nhà hay vịt hoang dã.
  - **Phương thức:**
    - `Duck(string n, bool fly, bool domestic)` - Khởi tạo vịt.

Quan hệ kế thừa:

$$\text{Animal} \rightarrow \begin{cases} \text{Mammal} \rightarrow \text{Dog} \\ \text{Bird} \rightarrow \text{Duck} \end{cases}$$

Câu 2: Thiết kế hệ thống AI với kế thừa nhiều cấp

Mô tả hệ thống:

Hệ thống bao gồm một lớp cha chung 'AIModel' và các lớp con kế thừa theo cấp bậc:

1. 'MachineLearningModel' kế thừa từ 'AIModel', đại diện cho các mô hình học máy.
2. 'DeepLearningModel' kế thừa từ 'MachineLearningModel', đại diện cho mô hình học sâu.
3. 'NeuralNetwork' kế thừa từ 'DeepLearningModel', đại diện cho mạng nơ-ron nhân tạo.
4. 'CNN' kế thừa từ 'NeuralNetwork', đại diện cho mạng tích chập (Convolutional Neural Network).
5. 'RNN' kế thừa từ 'NeuralNetwork', đại diện cho mạng nơ-ron hồi tiếp (Recurrent Neural Network).

- **Lớp AIModel (Lớp cha)**
  - **Thuộc tính:**
    - `string modelName` - Tên mô hình AI.
  - **Phương thức:**
    - `AIModel(string name)` - Khởi tạo mô hình AI.
- **Lớp MachineLearningModel (Kế thừa từ AIModel)**
  - **Thuộc tính:**
    - `string algorithm` - Thuật toán học máy sử dụng.
  - **Phương thức:**
    - `MachineLearningModel(string name, string algo)` - Khởi tạo mô hình học máy.
- **Lớp DeepLearningModel (Kế thừa từ MachineLearningModel)**
  - **Thuộc tính:**
    - `int numLayers` - Số lượng tầng trong mạng.
  - **Phương thức:**
    - `DeepLearningModel(string name, string algo, int layers)` - Khởi tạo mô hình học sâu.
- **Lớp NeuralNetwork (Kế thừa từ DeepLearningModel)**
  - **Thuộc tính:**
    - `int numNeurons` - Số lượng nơ-ron trong mạng.
  - **Phương thức:**
    - `NeuralNetwork(string name, string algo, int layers, int neurons)` - Khởi tạo mạng nơ-ron.
- **Lớp CNN (Kế thừa từ NeuralNetwork)**
  - **Thuộc tính:**
    - `int numFilters` - Số lượng bộ lọc (filters) trong mạng CNN.
  - **Phương thức:**
    - `CNN(string name, string algo, int layers, int neurons, int filters)` - Khởi tạo mạng CNN.
- **Lớp RNN (Kế thừa từ NeuralNetwork)**
  - **Thuộc tính:**
    - `int timeSteps` - Số bước thời gian trong mạng RNN.

- **Phương thức:**
  - `RNN(string name, string algo, int layers, int neurons, int steps)` - Khởi tạo mạng RNN.

Quan hệ kế thừa:

$$\text{AIModel} \rightarrow \left\{ \begin{array}{l} \text{MachineLearningModel} \rightarrow \text{DeepLearningModel} \rightarrow \text{NeuralNetwork} \rightarrow \left\{ \begin{array}{l} \text{CNN} \\ \text{RNN} \end{array} \right. \end{array} \right.$$

Câu 9: Thiết kế hệ thống nhân vật trong game với đa thừa kế

Mô tả hệ thống:

Hệ thống bao gồm nhiều lớp mô tả các thành phần chính của một nhân vật trong game, sử dụng đa thừa kế:

1. 'Entity' là lớp cha chung cho tất cả các đối tượng trong game.
2. 'Movable' mô tả khả năng di chuyển, kế thừa từ 'Entity'.
3. 'Attackable' mô tả khả năng tấn công, kế thừa từ 'Entity'.
4. 'Player' kế thừa đồng thời từ 'Movable' và 'Attackable', đại diện cho nhân vật người chơi.

- **Lớp Entity (Lớp cha chung)**
  - **Thuộc tính:**
    - `string name` - Tên của thực thể.
  - **Phương thức:**
    - `Entity(string n)` - Khởi tạo thực thể với tên.
- **Lớp Movable (Kế thừa từ Entity)**
  - **Thuộc tính:**
    - `float speed` - Tốc độ di chuyển.
  - **Phương thức:**
    - `Movable(string n, float spd)` - Khởi tạo đối tượng có thể di chuyển.
    - `void move()` - Di chuyển nhân vật.
- **Lớp Attackable (Kế thừa từ Entity)**
  - **Thuộc tính:**
    - `int attackPower` - Sức tấn công.
    - `int health` - Lượng máu của nhân vật.
  - **Phương thức:**
    - `Attackable(string n, int atk, int hp)` - Khởi tạo đối tượng có thể tấn công.
    - `void attack()` - Tấn công kẻ địch.
- **Lớp Player (Kế thừa từ Movable và Attackable)**
  - **Thuộc tính:**
    - `int level` - Cấp độ của nhân vật.
  - **Phương thức:**
    - `Player(string n, float spd, int atk, int hp, int lvl)` - Khởi tạo nhân vật người chơi.

* `void levelUp()` - Tăng cấp nhân vật.

Quan hệ kế thừa:

$$\text{Entity} \rightarrow \begin{cases} \text{Movable} \\ \text{Attackable} \end{cases} \qquad \text{Movable, Attackable} \rightarrow \text{Player}$$