:1. Kiến thức về lập trình C/C++: Phân biệt C/C++ (giống – khác nhau); Lập trình hướng đối tượng trong C++ - các tính chất của OOP và ví dụ; Phân biệt for, while, do while; Phân biệt Các lệnh Break, Switch, Continue, Exit, Return; kiến thức về Hàm, tham chiếu, tham trị; Con trỏ, Cấp phát bộ nhớ, Biến static, volotite; Linked list…Kiến thức về lập trình C/C++: Phân biệt C/C++ (giống – khác nhau); Lập trình hướng đối tượng trong C++ - các tính chất của OOP và ví dụ; Phân biệt for, while, do while; Phân biệt Các lệnh Break, Switch, Continue, Exit, Return; kiến thức về Hàm, tham chiếu, tham trị; Con trỏ, Cấp phát bộ nhớ, Biến static, volotite; Linked list…

***Phân biệt C/C++ (giống – khác nhau):***

Giống nhau: Cả hai đều có cú pháp tương tự và hỗ trợ lập trình hướng thủ tục.

Khác biệt: C++ hỗ trợ lập trình hướng đối tượng và có thêm nhiều tính năng mới như namespace, exception handling...

Lập trình hướng đối tượng (OOP - Object-Oriented Programming) là một phương pháp lập trình mà trong đó các đối tượng (objects) được tạo ra để đại diện cho các thực thể trong thế giới thực. Mỗi đối tượng có các thuộc tính (attributes) và phương thức (methods) để thực hiện các hoạt động cụ thể.

Lập trình hướng đối tượng trong C++ - các tính chất của OOP và ví dụ:

Tính đóng gói (Encapsulation): Ẩn thông tin và xử lý dữ liệu trong một lớp.

A computer screen with white text

Description automatically generated

Tính kế thừa (Inheritance): Lớp con có thể kế thừa các thuộc tính và phương thức của lớp cha.

A screenshot of a computer

Description automatically generated

Tính đa hình (Polymorphism): Đối tượng của các lớp con có thể được sử dụng như đối tượng của lớp cha.

A black and white screen

Description automatically generated

***Phân biệt for, while, do while:***

A black screen with white text

Description automatically generatedFor loop: Dùng khi biết trước số lần lặp cụ thể.

A black screen with white text

Description automatically generatedWhile loop: Sử dụng khi số lần lặp không xác định trước.

Do-while loop: Tương tự như while loop nhưng thực hiện ít nhất một lần.

A black screen with white text

Description automatically generated

***Phân biệt Các lệnh Break, Switch, Continue, Exit, Return:***

Break: Dừng vòng lặp hoặc switch và chuyển quyền điều khiển đến phía sau vòng lặp hoặc switch (cần có điều kiện if hoặc switch).

A screenshot of a computer

Description automatically generated

Switch: Dùng để kiểm tra giá trị của biểu thức và thực hiện các hành động tương ứng với từng giá trị.

A screenshot of a computer

Description automatically generated

Continue: Bỏ qua phần còn lại của vòng lặp và chuyển đến vòng lặp tiếp theo.(không cần đi theo điều kiện if, switch)

A screenshot of a computer

Description automatically generated

Return: Trả về kết quả từ một hàm.

A black rectangular object with a black border

Description automatically generated

***Hàm***: Trong lập trình, hàm là một khối mã được đặt tên, thực hiện một tác vụ cụ thể và có thể được gọi từ nhiều điểm trong chương trình.

A screenshot of a computer

Description automatically generated

***Tham số*** (Parameter): *Tham số là các biến được định nghĩa trong khai báo của một hàm hoặc phương thức.* Trong hàm calculate\_area(width, height), width và height là các tham số.

***Đối số*** (Argument): *Đối số là các giá trị được truyền vào một hàm khi gọi nó.* Ví dụ:Trong lời gọi hàm calculate\_area(5, 10), 5 và 10 là các đối số.

***Truyền tham số (Passing Parameters):***

* Truyền tham số là quá trình truyền dữ liệu vào hàm.
* Trong truyền tham số, bạn truyền giá trị của biến từ nơi gọi hàm (đối số) vào hàm (tham số).
* Hàm nhận giá trị của biến truyền vào như một bản sao và thực hiện các thao tác trên bản sao đó, không ảnh hưởng đến biến gốc ngoài phạm vi của hàm.
* Cách truyền này được sử dụng khi bạn không muốn thay đổi giá trị của biến gốc.

***Truyền tham chiếu (Passing by Reference):***

* Truyền tham chiếu là quá trình truyền một tham chiếu đến một biến vào hàm.
* Tham chiếu là biệt danh khác của biến gốc và thực sự chỉ là một tên mới cho biến đó.
* Khi thay đổi giá trị của tham chiếu trong hàm, thay đổi cũng được áp dụng cho biến gốc.
* Tham chiếu giúp truy cập và thay đổi dữ liệu của biến gốc một cách hiệu quả hơn so với truyền tham trị.
* Cách truyền này được sử dụng khi bạn muốn thực hiện các thay đổi trực tiếp vào giá trị của biến gốc.

***Truyền con trỏ (Passing by Pointer):***

* Truyền con trỏ là quá trình truyền địa chỉ của một biến vào hàm thông qua một con trỏ.
* Con trỏ là một biến chứa địa chỉ bộ nhớ của biến gốc.
* Bằng cách truyền con trỏ, bạn có thể thay đổi giá trị của biến gốc từ bên trong hàm.
* Con trỏ cung cấp khả năng truy cập trực tiếp đến vùng nhớ của biến gốc.
* ***A computer screen shot of a program

  Description automatically generated***Cách truyền này được sử dụng khi bạn muốn truyền và thay đổi giá trị của biến từ bêntrong hàm.

***Overloading*** là một khái niệm trong lập trình cho phép bạn định nghĩa nhiều phương thức hoặc toán tử cùng tên nhưng khác nhau về số lượng hoặc kiểu dữ liệu của các tham số đầu vào. Khi gọi một phương thức hoặc toán tử đã được nạp chồng (overloaded), trình biên dịch sẽ tự động chọn phương thức hoặc toán tử tương ứng dựa trên kiểu dữ liệu và số lượng tham số truyền vào.

A computer screen shot of a program code

Description automatically generated

***typedef và #define*** đều được sử dụng để định nghĩa các bí danh (aliases) trong C++. Tuy nhiên, chúng có những đặc điểm và cách sử dụng khác nhau:

Typedef:

* typedef được sử dụng để tạo ra bí danh cho một kiểu dữ liệu đã tồn tại.
* Cú pháp của typedef là: typedef existing\_type new\_type;
* Ví dụ: typedef int integer; sẽ tạo ra một bí danh mới có tên là integer cho kiểu dữ liệu int.
* typedef chỉ có thể định nghĩa các bí danh cho kiểu dữ liệu, không thể định nghĩa hằng số hoặc macro.

Define:

* #define được sử dụng để định nghĩa hằng số hoặc macro.
* Cú pháp của #define là: #define identifier value
* Ví dụ: #define PI 3.14159 sẽ định nghĩa một hằng số có tên là PI với giá trị là 3.14159.
* #define có thể định nghĩa hằng số hoặc macro, không giới hạn trong việc định nghĩa các kiểu dữ liệu.

***Độ Phức Tạp Tính Toán Của Thuật Toán :*** [#7 [C++]. Phân Tích Độ Phức Tạp Của Thuật Toán | Độ Phức Tạp Tính Toán Của Thuật Toán (youtube.com)](https://www.youtube.com/watch?v=0W060mNbi40&list=PLux-_phi0Rz0Hq9fDP4TlOulBl8APKp79&index=9)

***Con trỏ***: Con trỏ là biến chứa địa chỉ của một biến khác.

A black screen with white text

Description automatically generated

***Cấp phát bộ nhớ***: Cấp phát bộ nhớ là quá trình đặt dữ liệu trong bộ nhớ, thường được sử dụng để tạo ra các biến động.

A screenshot of a computer

Description automatically generated

* **Cấp phát bộ nhớ trên ngăn xếp (Stack Memory Allocation):** Biến cục bộ và tham số hàm được cấp phát trên ngăn xếp. Bộ nhớ được giải phóng tự động khi biến hoặc hàm kết thúc.
* **Cấp phát bộ nhớ trên heap (Heap Memory Allocation):** Bộ nhớ được cấp phát trên heap bằng các hàm như malloc(), calloc(), realloc(). Bộ nhớ cần được giải phóng thủ công bằng hàm free().

***Biến static***: Biến static tồn tại trong suốt thời gian chương trình chạy và chỉ được khởi tạo một lần.

A computer screen with white text

Description automatically generated

***Biến volatile***: Biến volatile được sử dụng khi biến có thể thay đổi bất kỳ lúc nào từ bên ngoài chương trình.

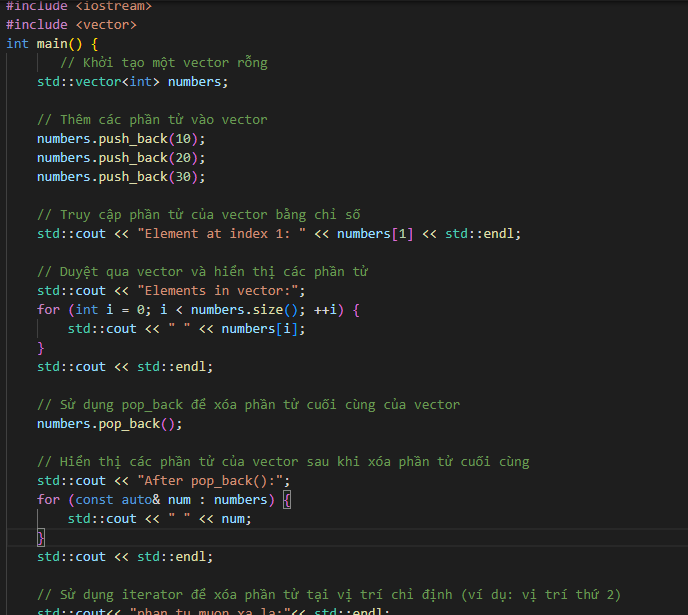
A black and white screen with white text

Description automatically generated

***Linked list:*** Linked list là một cấu trúc dữ liệu mà mỗi phần tử được liên kết với phần tử tiếp theo thông qua các con trỏ.~ danh sách

A screenshot of a computer program

Description automatically generated

***vector*** là một cấu trúc dữ liệu dạng mảng linh hoạt (dynamic array) trong thư viện chuẩn STL (Standard Template Library): 

A computer screen with colorful code

Description automatically generated

***std::pair*** là một cấu trúc dữ liệu được sử dụng để lưu trữ một cặp giá trị : A computer screen with white text

Description automatically generated

***Inline***: hàm inline và macro có một số điểm tương đồng trong việc thay thế mã nguồn, nhưng hàm inline thường được ưu tiên hơn trong các ứng dụng C++

A computer screen shot of text

Description automatically generated

***Template*** cho phép bạn viết mã chung cho nhiều kiểu dữ liệu khác nhau mà không cần phải viết lại mã cho mỗi kiểu dữ liệu riêng biệt.

A computer screen shot of text

Description automatically generated

// - ***Biến cục bộ (`localVar`)***: Được khai báo và sử dụng trong phạm vi hàm `main()`.

// - ***Biến static (`staticVar`)***: Là biến cục bộ được khai báo bên trong hàm `main()` với từ khóa `static`. Biến này vẫn giữ giá trị sau khi ra khỏi phạm vi hàm.

// ***- Biến toàn cục (`globalVar`)***: Được khai báo ở bên ngoài hàm `main()` với từ khóa `extern`, và được sử dụng trong hàm `main()`. Biến này có thể được truy cập từ bất kỳ hàm nào trong chương trình.

***vector và linked list*** trong C++ đều là cấu trúc dữ liệu để lưu trữ và quản lý các phần tử. Tuy nhiên, chúng có những đặc điểm khác nhau:

Vector:

* Vector là một mảng động, tức là kích thước của nó có thể thay đổi trong quá trình chạy.
* Dữ liệu được lưu trữ liên tiếp trong bộ nhớ, giúp truy cập nhanh chóng đến các phần tử bằng cách sử dụng chỉ số.
* Việc chèn và xóa phần tử từ cuối vector hoặc sử dụng phương thức push\_back() và pop\_back() là hiệu quả với độ phức tạp thời gian O(1).
* Tuy nhiên, chèn hoặc xóa phần tử ở giữa vector có độ phức tạp thời gian O(n) vì cần phải di chuyển các phần tử khác.

Linked List:

* Linked list cũng là một cấu trúc dữ liệu động, nhưng dữ liệu không được lưu trữ liên tiếp trong bộ nhớ. Mỗi nút (node) trong danh sách chứa một phần dữ liệu và một con trỏ đến nút tiếp theo.
* *Truy cập các phần tử trong linked list có thể yêu cầu thời gian O(n), vì bạn cần phải đi qua từ đầu đến cuối danh sách.*
* Chèn và xóa phần tử ở bất kỳ vị trí nào trong linked list đều có độ phức tạp thời gian O(1), miễn là bạn biết trước vị trí cụ thể của nó.

// ***virtual*** được sử dụng để khai báo một hàm ảo trong một lớp cơ sở (base class) để cho phép các lớp con (derived class) cài đặt lại (override) hàm đó theo nhu cầu của chúng:

A computer screen shot of code

Description automatically generated

***Union***: Giống như cấu trúc, kiểu hợp cũng có nhiều thành phần nhưng các thành phần của chúng sử dụng chung nhau một vùng nhớ. Do vậy kích thước của một kiểu hợp là độ dài của trường lớn nhất và việc thay đổi một thành phần sẽ ảnh hưởng đến tất cả các thành phần còn lại.

A screen shot of a computer code

Description automatically generated

// ***con trỏ this*** trong C++ được sử dụng để trỏ đến đối tượng hiện tại mà phương thức được gọi trên đó. Việc sử dụng this giúp tránh nhầm lẫn giữa biến thành viên của đối tượng hiện tại và các biến cùng tên trong phạm vi khác, đặc biệt là trong trường hợp lớp có các phương thức hoặc toán tử được định nghĩa cho việc gán giá trị cho các biến thành viên.

A computer screen shot of text

Description automatically generated

***Constructor*** là một phương thức đặc biệt của một lớp trong ngôn ngữ lập trình hướng đối tượng, được sử dụng để khởi tạo đối tượng của lớp đó. *Constructor thường được gọi tự động khi một đối tượng được tạo ra và có cùng tên với tên của lớp.*

Một số điểm quan trọng về constructor:

* Tên constructor giống tên lớp: Constructor có tên giống với tên của lớp mà nó thuộc về.
* Không có kiểu trả về: Constructor không có kiểu trả về, thậm chí cũng không có kiểu void.
* Có thể có tham số hoặc không có tham số: Constructor có thể nhận tham số hoặc không nhận tham số. Những constructor mà không nhận tham số được gọi là constructor mặc định.
* Dùng để khởi tạo đối tượng: Constructor được sử dụng để khởi tạo giá trị ban đầu cho các thành viên của đối tượng và thực hiện các công việc khởi tạo khác nếu cần.
* Xóa contructor bằng cú pháp ~Name() // để bảo mật, khóa các phương thức trong khối ;

A screenshot of a computer

Description automatically generated

***Các loại kế thừa :***

Private : không cho con sử dụng, tiết lộ ra ngoài (trọng số 3)

Protected: Cho con sử dụng, không lộ ra ngoài (trọng số 2)

Public: cho con sử dụng, cho tiết lộ ra ngoài (trọng số 1)

Xét theo trọng số:

* 3+1, 3+2, 3+3: con không được thừa hưởng, sử dụng phần private của cha cả ở trong lẫn ngoài class.
* 2+1, 2+2, 2+3: được sử dụng ở trong, không sử dụng ở ngoài , có thể thay đổi, tuy cùng tên nhưng trở thành giao thức của con.
* 1+3, 1+2: được sử dụng ở trong, không sử dụng ở ngoài, có thể thay đổi, tuy cùng tên nhưng trở thành giao thức của con.
* 1+1, được sử dụng ở trong, được sử dụng ở ngoài, có thể thay đổi luôn, , tuy cùng tên nhưng trở thành giao thức của con.

Tính chất của ***hàm bạn (friend)*** bao gồm:

// Không phải là thành viên của lớp: Hàm bạn không thuộc về lớp mà nó được khai báo là bạn, nhưng nó có thể truy cập vào tất cả các thành viên private và protected của lớp đó.

// Có thể truy cập vào tất cả các thành viên của lớp: Hàm bạn có thể truy cập vào tất cả các thành viên private và protected của lớp đó mà không cần phải là một phần của lớp.

// Không có con trỏ this được truyền vào: Hàm bạn không có con trỏ this để tham chiếu đến các thành viên của đối tượng của lớp

// Truy cập vào các thành viên private và protected của nhiều lớp: Một hàm bạn của nhiều lớp

// có quyền truy cập vào các thành viên private và protected của tất cả các lớp mà nó được khai báo là bạn. ***Có thể thay đổi protected và private t cách gián tiếp bằng 1 hàm friend***

A screenshot of a computer program

Description automatically generated

// ***Phép toán cho lớp (còn được gọi là toán tử nạp chồng)*** là cách để định nghĩa các hành vi của các toán tử như +, -, \*, /, ==, <, >, v.v. cho các lớp đối tượng trong C++.

// Bằng cách nạp chồng các toán tử này, bạn có thể thực hiện các phép toán trên các đối tượng của lớp một cách tự nhiên giống như các kiểu dữ liệu cơ bản.

// Phép toán cho lớp được sử dụng khi bạn muốn thực hiện các phép toán toán học, so sánh hoặc gán giữa các đối tượng của lớp.

A screen shot of a computer program

Description automatically generated

Trong đó **const Complex& other** coi như 1 cấu trúc mặc định có ý nghĩacó nghĩa là một tham chiếu hằng đến một đối tượng của lớp **Complex**, và trong phương thức nạp chồng toán tử, giá trị của đối tượng được tham chiếu không được thay đổi.

***Sử dụng toán tử nạp chồng :***

A black background with white text

Description automatically generated

Thao tác với tệp :

A screen shot of a computer

Description automatically generated

A screen shot of a computer code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

CẤU TRÚC DỮ LIỆU:

***td::set*** là một cấu trúc dữ liệu mà mỗi phần tử trong set là duy nhất và được tự động sắp xếp theo thứ tự tăng dần. Set này được triển khai dưới dạng một cây tìm kiếm nhị phân cân bằng (balanced binary search tree), thường là cây đỏ-đen (red-black tree).

Dưới đây là một định nghĩa cụ thể của std::set trong C++:

* Set: Một cấu trúc dữ liệu trong C++ STL (Standard Template Library) để lưu trữ các phần tử duy nhất được sắp xếp theo thứ tự tăng dần. Mỗi phần tử trong set là duy nhất, có nghĩa là không có hai phần tử nào có giá trị bằng nhau.

Set cung cấp các tính năng sau:

* Duy nhất: Mỗi phần tử trong set là duy nhất, không thể có hai phần tử trùng nhau trong set.
* Sắp xếp: Các phần tử trong set được tự động sắp xếp theo thứ tự tăng dần.
* Tìm kiếm hiệu quả: Việc tìm kiếm một phần tử trong set được thực hiện với độ phức tạp logarit (O(log n)) với n là số lượng phần tử trong set.
* Thao tác chèn và xóa hiệu quả: Thao tác chèn và xóa một phần tử trong set cũng được thực hiện với độ phức tạp logarit.

A computer screen shot of a program code

Description automatically generated

A computer screen shot of a program code

Description automatically generated

Một ***std::multiset*** trong C++ là một cấu trúc dữ liệu tập hợp được sắp xếp không giảm, mà cho phép các phần tử có thể xuất hiện nhiều lần và được lưu trữ theo thứ tự tăng dần. (***các API tương tự như set***)

***std::unordered\_set*** trong C++ là một cấu trúc dữ liệu tập hợp không có thứ tự, nơi mà mỗi phần tử là duy nhất và được lưu trữ dựa trên giá trị hash của nó.

***myMultiset.begin()*** ***và myMultiset.rbegin()*** đều trả về một ***iterator*** đến phần tử đầu tiên của std::multiset, nhưng chúng khác nhau về ý nghĩa của "begin".

* myMultiset.begin() trả về một iterator đến phần tử đầu tiên của std::multiset, nghĩa là phần tử có giá trị nhỏ nhất trong tập hợp.
* myMultiset.rbegin() trả về một reverse iterator đến phần tử cuối cùng của std::multiset, nghĩa là phần tử có giá trị lớn nhất trong tập hợp.

A computer screen shot of a computer code

Description automatically generated

A black and white rectangle

Description automatically generated

***std::map*** trong C++ là một cấu trúc dữ liệu ánh xạ (associative container) được sử dụng để lưu trữ các cặp key-value, ***trong đó mỗi key là duy nhất và tương ứng với một value***. *std::map sắp xếp các cặp key-value theo thứ tự tăng dần của key* và cho phép truy cập nhanh chóng vào mỗi value thông qua key tương ứng. (tất cả các API đề tìm kiếm, truy xuất theo key chứ k truy xuất theo value)

Đây là một số đặc điểm chính của std::map:

* Cặp key-value: Mỗi phần tử trong std::map là một cặp key-value, trong đó mỗi key là duy nhất và tương ứng với một value.
* Sắp xếp: Các phần tử được sắp xếp theo thứ tự tăng dần của key.
* Tìm kiếm nhanh chóng: Thời gian truy cập, chèn và xóa các phần tử trong std::map là logaritmic.
* Key có thể là bất kỳ kiểu dữ liệu so sánh được: Key trong std::map có thể là bất kỳ kiểu dữ liệu nào mà có thể so sánh được (bao gồm các kiểu dữ liệu nguyên thủy và các kiểu dữ liệu do người dùng tự định nghĩa).

A computer screen shot of a program code

Description automatically generated

***Multimap:*** Multimap trong C++ là một cấu trúc dữ liệu tương tự như Map, nhưng cho phép lưu trữ nhiều cặp key-value với cùng một key. Điều này có nghĩa là mỗi key có thể được liên kết với nhiều hơn một giá trị. Multimap được triển khai dưới dạng cây tìm kiếm nhị phân cân bằng (balanced binary search tree) hoặc cây đỏ đen (red-black tree) để đảm bảo thứ tự tăng dần của key.

A triangle with numbers and letters on it

Description automatically generated***Unordered map:*** Unordered map là một cấu trúc dữ liệu trong C++ cung cấp các phần tử dưới dạng một cặp key-value không theo thứ tự cụ thể. Unordered map thường được triển khai dưới dạng bảng băm (hash table), cho phép truy cập ngẫu nhiên vào các phần tử trong thời gian hằng số. Mặc dù không duy trì thứ tự của các phần tử, nhưng nó cung cấp hiệu suất cao cho các thao tác chèn, xóa và truy vấn.

***Cây nhị phân*** là một cấu trúc dữ liệu dạng cây mà mỗi nút trong cây có không quá hai nút con, được gọi là nút con trái và nút con phải. Cấu trúc của cây nhị phân là một cấu trúc dữ liệu phổ biến được sử dụng trong nhiều ứng dụng, bao gồm cả cơ sở dữ liệu và các thuật toán tìm kiếm.

***Cây nhị phân hoàn chỉnh*** là một loại cây nhị phân mà tất cả các tầng đều được lấp đầy đầy đủ từ trái sang phải, trừ tầng cuối cùng, nơi có thể thiếu một số nút ở bên phải. Các nút trên tầng cuối cùng phải được lấp đầy từ trái sang phải nếu có thể.

***Heap*** là một cấu trúc dữ liệu cây nhị phân đặc biệt, thường được sử dụng để triển khai các thuật toán liên quan đến việc tìm kiếm, thêm, hoặc loại bỏ phần tử có giá trị cực đại hoặc cực tiểu.

***Heap*** thường được triển khai dưới dạng một mảng, trong đó mỗi nút có chỉ số nhất định. Cụ thể, nút có chỉ số i sẽ có hai con ở các chỉ số 2\*i + 1 và 2\*i + 2.

***Heap*** có hai loại chính:

* Max-heap: Trong đó giá trị của mỗi nút lớn hơn hoặc bằng giá trị của cả hai con của nó.
* Min-heap: Trong đó giá trị của mỗi nút nhỏ hơn hoặc bằng giá trị của cả hai con của nó.

Heap thường được sử dụng trong các thuật toán như Heap Sort, Priority Queues, và Dijkstra's Algorithm để thực hiện các thao tác tối ưu như tìm kiếm phần tử cực đại/cực tiểu trong thời gian độ phức tạp logarit.