**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI TP. HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

**BÁO CÁO BÀI TẬP LỚN**

**ĐỀ TÀI : DINO RUN**

Giảng viên hướng dẫn: TRẦN THỊ DUNG

Sinh viên thực hiện: VÕ MINH KHA

Môn học: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lớp: CQ.62.CNTT

Khóa: 62

Tp. Hồ Chí Minh, năm 2022

# MỤC LỤC

**Lời giới thiệu** .........................................................................................................2

**Phần I: Dịch sách** .................................................................................................2

1. Class ........................................................................................................2
   1. Cơ bản về class............................................................................2
   2. Lớp và cấu trúc............................................................................3
   3. Hàm thành viên............................................................................4
      1. Hàm khởi tạo...................................................................4
      2. Các hàm thành viên không đổi........................................4
   4. [Static] thành viên.........................................................................5
      1. Các loại thành viên..........................................................5
   5. Nạp chồng toán tử.........................................................................6
2. Hàm khởi tạo, xóa, sao chép và di chuyển................................................6
   1. Giới thiệu......................................................................................7
   2. Hàm tạo, hàm hủy.........................................................................7

2.1.1 Hàm hủy và tài nguyên....................................................7

* 1. Sao chép và di chuyển...................................................................8
  2. Các chức năng xóa........................................................................8

1. Nạp chồng .................................................................................................10
   1. Giới thiệu......................................................................................10
   2. Nạp chồng hàm toán tử.................................................................10
      1. Toán tử 2 ngôi và 1 ngôi..................................................10
      2. Ý nghĩa được xác định trước cho các toán tử..................12
      3. Toán tử và các loại do người dùng xác định...................12
   3. Một loại số phức..........................................................................13
2. Toán tử đặt biệt........................................................................................13
   1. Giới thiệu.....................................................................................13
   2. Các toán tử đặt biệt......................................................................13
      1. Toán tử gián tiếp............................................................13
      2. Tăng và giảm.................................................................13
      3. Cấp phát và giải phóng..................................................14
   3. Các hàm hỗ trợ............................................................................14
   4. Hàm bạn......................................................................................14
3. Lớp dẫn xuất............................................................................................15
   1. Giới thiệu.....................................................................................15
   2. Lớp dẫn xuất................................................................................16
4. Phân cấp lớp.............................................................................................16
   1. Giới thiệu................................................................................17
   2. Triển khai kế thừa.....................................................................17**Phần II: Chương trình**..............................................................................18
5. Lý do chọn đề tài......................................................................................18
6. Mô tả bài toán...........................................................................................19
7. Giao diện chương trình.............................................................................19
8. Tính chất hướng đối tượng.......................................................................22
   1. Tính đóng gói...............................................................................22
   2. Tính trừu tượng............................................................................23
   3. Tính kế thừa.................................................................................24
   4. Tính đa hình.................................................................................26
9. Nội dung lý thuyết....................................................................................28
10. Kết luận.....................................................................................................39
    1. Kết quả đạt được...........................................................................39
    2. Nhược điểm...................................................................................39
    3. Hướng phát triển...........................................................................39
    4. Tài liệu tham khảo.........................................................................39

**Phụ Lục:** .................................................................................................................. 39

Hướng dẫn sử dụng………………………………………...………………50

# LỜI GIỚI THIỆU

Sau quá trình học tập và rèn luyện tại ngành Công nghệ thông tin trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh em đã được trang bị các kiến thức cơ bản, các kỹ năng thực tế để có thể hoàn thành đề tài “Dino run”. Nhóm em xin gửi lời cảm ơn chân thành đến quý thầy, cô bộ môn Công nghệ thông tin trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh đã quan tâm hướng dẫn truyền đạt học những kiến thức và kinh nghiệm cho em trong suốt thời gian học tập, và thực hiện bài tập lớn một cách tận tình và tâm huyết.

Nhóm em xin chúc quý thầy cô thật nhiều sức khỏe và luôn đạt được thành công trong cuộc sống. Đặc biệt nhóm em xin cảm ơn cô Trần Thị Dung người đã hướng dẫn và chỉ bảo trong quá trình thực hiện bài tập này. Sau một thời gian nỗ lực thực hiện thì đề tài cũng đã hoàn thành, nhưng không sao tránh khỏi những sai sót do nhóm em còn chưa có nhiều kinh nghiệm.

Nhóm em kính mong nhận được sự góp ý và nhận xét từ cô để có thể hoàn thiện và hoàn thành tốt hơn cho đề tài của mình.

# PHẦN I: DỊCH SÁCH

# CLASS

## 1. Cơ bản về Class

Các class C ++ là một công cụ để tạo các kiểu dữ liệu mới có thể được sử dụng thuận tiện như các kiểu dữ liệu tích hợp sẵn.

Dưới đây là một bản tóm tắt ngắn gọn về class:

* Một lớp là một kiểu dữ liệu do người dùng định nghĩa.
* Một lớp bao gồm một tập hợp các thành viên. Các loại thành viên phổ biến nhất là thành phần dữ liệu và hàm thành viên.
* Các hàm thành viên có thể xác định ý nghĩa của việc khởi tạo (tạo), sao chép, di chuyển và dọn dẹp ( tiêu hủy)
* Các thành viên được truy cập bằng cách sử dụng (dấu chấm) cho các đối tượng và > (mũi tên) cho con trỏ.
* Các toán tử chẳng hạn như +,!, Và [], có thể được định nghĩa cho một lớp.
* Một lớp là một không gian chứa các thành viên của nó.
* Các thành phần công khai cung cấp giao diện của lớp và các thành phần riêng cung cấp chi tiết triển khai.
* Một cấu trúc là một lớp mà các thành viên được mặc định là công khai.

## 2. Lớp và cấu trúc

Khởi tạo class X {...}; được gọi là định nghĩa lớp; nó xác định 1 kiểu gọi là X. Ngày xưa, định nghĩa lớp thường được gọi là khai báo lớp. Khai báo thì cũng không có định nghĩa, định nghĩa lớp có thể được sao chép trong các tệp nguồn khác nhau sử dụng #include mà không vi phạm quy tắc 1 định nghĩa. Theo định nghĩa, một cấu trúc là một lớp trong đó các thành viên được mặc định là public. Ví dụ: struct S {/ \* ... \* /}; chỉ đơn giản là viết tắt của class S {public: / \* ... \* /};

Hai định nghĩa này của S có thể thay thế cho nhau, mặc dù thông thường bạn nên theo một kiểu nhất định. Phong cách bạn sử dụng tùy thuộc vào hoàn cảnh và sở thích. Tôi có xu hướng sử dụng cấu trúc cho các lớp mà tôi nghĩ là "" chỉ là cấu trúc dữ liệu đơn giản. "" Nếu tôi nghĩ về một lớp là "" một loại thích hợp với một bất biến, "tôi sử dụng lớp. Các hàm tạo và hàm truy cập có thể khá hữu ích ngay cả đối với các cấu trúc, nhưng như một cách viết tắt chứ không phải là bảo đảm bất biến. Theo mặc định, các thành viên của một lớp là riêng tư. Không bắt buộc phải khai báo dữ liệu đầu tiên trong một lớp. Trên thực tế, việc đặt các thành phần dữ liệu cuối cùng thường nhấn mạnh các chức năng cung cấp giao diện người dùng công khai. Các chỉ định truy cập có thể được sử dụng nhiều lần trong một khai báo lớp duy nhất.

## 3. Hàm thành viên

Xem xét việc triển khai khái niệm ngày bằng cách sử dụng struct để xác định biểu diễn của Date và một tập hợp các hàm để thao tác với các biến kiểu này:

|  |
| --- |
| struct Date{ |
| int d,m,y;  }; void init\_date(Date& d,int,int,int); //hàm khởi tạo void add\_year(Date& d,int n); // hàm thêm n năm vào d |

Không có kết nối rõ ràng nào giữa kiểu dữ liệu, Date và các hàm này. Một kết nối kiểu này có thể được thiết lập bằng cách khai báo các chức năng là thành viên:

|  |
| --- |
| struct Date{ |
| int d,m,y;  void init\_date(Date& d,int,int,int); //hàm khởi tạo void add\_year(Date& d,int n); // hàm thêm n năm vào d |
| }; |

Các hàm được khai báo trong định nghĩa lớp được gọi là các hàm thành viên và chỉ có thể được gọi cho một biến cụ thể thuộc loại thích hợp bằng cách sử dụng tiêu chuẩn cú pháp để truy cập thành viên cấu trúc.

Các cấu trúc khác nhau có thể có nhiều hàm thành viên cùng tên. chúng ta phải phân biệt tên cấu trúc khi xác định hàm thành viên

Trong 1 hàm thành viên, tên thành viên có thể được sử dụng mà không cần có sự tham khảo rõ ràng đến đối tượng. Trong trường hợp đó, tên có thể đề cập đến thành viên đó của đối tượng mà hàm được xác định. Ví dụ Date::init() xác định cho ngày.

### 3.1 Hàm khởi tạo

Việc sử dụng các hàm như init () để cung cấp khởi tạo cho các lớp là không phù hợp và dễ xảy ra lỗi. Bởi vì không có chỗ nào khẳng định rằng một đối tượng phải được khởi tạo, một lập trình viên có thể quên làm như vậy - hoặc làm như vậy hai lần (thường cho kết quả tệ như nhau). Một cách tiếp cận tốt hơn là cho phép lập trình viên khai báo một hàm với mục đích rõ ràng là khởi tạo các đối tượng. Bởi vì như vậy một hàm xây dựng các giá trị của một kiểu nhất định, nó được gọi là một constructor. Một phương thức khởi tạo được công nhận khi có cùng tên với chính lớp đó.

|  |
| --- |
| class Date { |
| int d, m, y; public:  Date(int dd, int mm, int yy); // constructor // ... }; |

Khi 1 lớp có phương thức khởi tạo, tất cả các đối tượng của lớp sẽ được khởi tạo bởi lệnh của phương thức. nếu constructor yêu cầu các đối số, các đối số này phải được cung cấp. Bằng cách cung cấp một số hàm tạo, chúng tôi có thể cung cấp nhiều cách khác nhau để khởi tạo các đối tượng của một kiểu.

Ví dụ:

|  |
| --- |
| class Date { |
| int d, m, y; public:  // ...  Date(int, int, int); // day, month, year  Date(int, int); // day, month, today’s year  Date(int); // day, today’s month and year  Date(); // default Date: today  Date(const char ); // date in string representation }; |

Các hàm tạo tuân theo các quy tắc chồng giống như các hàm thông thường. Miễn là các hàm tạo có đủ khác biệt về kiểu đối số của chúng, trình biên dịch có thể chọn đúng để sử dụng:’

Sự gia tăng của các hàm tạo trong ví dụ Date là điển hình. Khi thiết kế một lớp, một lập trình viên luôn bị kích thích để thêm các tính năng chỉ vì ai đó có thể muốn chúng. Nó khiến cho việc suy nghĩ nhiều hơn để quyết định cẩn thận những tính năng nào thực sự cần thiết và chỉ bao gồm những tính năng đó. Tuy vậy, suy nghĩ thêm thường dẫn đến các chương trình nhỏ hơn và dễ hiểu hơn. Một cách để giảm số lượng các hàm liên quan là sử dụng các đối số mặc định . Đối với Date, mỗi đối số có thể được cung cấp một giá trị mặc định được hiểu là '' chọn giá trị mặc định:

hôm nay. ''

Lưu ý rằng bằng cách đảm bảo khởi tạo các đối tượng đúng cách, các hàm tạo sẽ đơn giản hóa rất nhiều thực hiện các chức năng thành viên. Đã cho các hàm tạo, các hàm thành viên khác không còn để đối phó với khả năng dữ liệu chưa được khởi tạo.

**3.2 Các hàm thành viên không đổi**

Date xác định cung cấp các hàm thành viên để gán một giá trị cho Date. Rất tiếc, chúng tôi không cung cấp cách kiểm tra giá trị của Ngày. Sự cố này có thể dễ dàng được khắc phục bằng cách thêm các chức năng đọc ngày, tháng và năm

Hằng số sau danh sách đối số (trống) trong khai báo hàm chỉ ra rằng các hàm này không sửa đổi trạng thái của Date. Đương nhiên, trình biên dịch sẽ bắt gặp những lỗi. Ví dụ:

|  |
| --- |
| int Date::year() const{ |
| return ++y; // error : cố gắng thay đổi giá trị thành viên trong hàm const  } |

Khi một hàm thành viên const được định nghĩa bên ngoài lớp của nó, thì hậu tố const là bắt buộc. Nói cách khác, const là một phần của kiểu Date :: day (), Date :: month () và Date :: year ().

Một hàm thành viên const có thể được gọi cho cả các đối tượng const và không phải const, trong khi một hàm thành viên không phải const chỉ có thể được gọi cho các đối tượng không phải const.

**4. [static]Thành viên**

Sự tiện lợi của giá trị mặc định cho Date được mua bằng cái giá của một vấn đề ẩn. Lớp Date trở nên phụ thuộc vào biến toàn cục Today. Lớp Date này chỉ có thể được sử dụng Today được định nghĩa và sử dụng chính xác bởi mọi đoạn mã. Đây là loại ràng buộc khiến một lớp trở nên vô nghĩa. Người dùng không hài lòng khi cố gắng sử dụng các lớp phụ thuộc ngữ cảnh như vậy và việc bảo trì trở nên lộn xộn. Có thể ‘‘ chỉ một biến toàn cục nhỏ ’’ không quá khó quản lý, nhưng kiểu đó dẫn đến mã không thể sử dụng ngoại trừ lập trình ban đầu của nó. nên tránh trường hợp này. chúng ta có thể thuận lợi khi không có sự cản trở của một biến toàn cầu có thể truy cập công khai. Một biến là một phần của một lớp, không phải là một phần của đối tượng lớp đó, được gọi là một thành viên STATIC. Có bản sao chính xác của thành viên STATIC thay vì một bản sao cho mỗi đối tượng, như đối với các thành viên NON STATIC thông thường. Tương tự, một hàm cần quyền truy cập vào các thành viên của một lớp, nhưng không cần được gọi cho một đối tượng cụ thể, được gọi là hàm thành viên STATIC. Đây là một thiết kế bảo toàn nghĩa của các giá trị phương thức khởi tạo mặc định cho Date mà không gặp các vấn đề bắt nguồn từ việc phụ thuộc khác

|  |
| --- |
| class Date { |
| int d, m, y; static Date default\_date; public:  Date(int dd =0, int mm =0, int yy =0); // ... static void set\_default(int dd, int mm, int yy); // set default\_date to Date(dd,m m,yy) }; |

Bây giờ chúng ta có thể xác định hàm tạo DATE để sử dụng default\_date **4.1 Các loại thành viên**

Các kiểu và các kiểu bí danh kiểu có thể là thành viên của một lớp. Ví dụ: template<typename T>

Một lớp thành viên (thường được gọi là lớp lồng nhau) có thể tham chiếu đến các kiểu và các thành viên static của lớp bao quanh nó. Nó chỉ có thể tham chiếu đến các thành viên non static khi nó được cung cấp một đối tượng của lớp bao quanh để tham chiếu đến.

Các lớp thành viên là một sự tiện lợi về mặt ký hiệu hơn là một tính năng có tầm quan trọng cơ bản. Mặt khác, bí danh thành viên rất quan trọng như là cơ sở của kỹ thuật lập trình chung dựa trên các kiểu liên kết. Enum thành viên thường là một sự thay thế cho các lớp enum.

## 5. Nạp chồng toán tử

|  |
| --- |
| bool operator!=(Date, Date); // toán tử khác |
| bool operator<(Date, Date); // toán tử bé hơn |
| bool operator>(Date, Date); // toán tử lớn hơn |
| // ...  Date& operator++(Date& d) { return d.add\_day(1); } // toán tử tăng Date lên 1 ngà y |
| Date& operator−−(Date& d) { return d.add\_day(−1); } // toán tử giảm Date xuống 1 |
| ngày  Date& operator+=(Date& d, int n) { return d.add\_day(n); } // cộng thêm n ngày  Date& operator−=(Date& d, int n) { return d.add\_day(−n); } // trừ n ngày  Date operator+(Date d, int n) { return d+=n; } // cộng n ngày |
| Date operator−(Date d, int n) { return d+=n; } // trừ n ngày |
| ostream& operator<<(ostream&, Date d); // output istream& operator>>(istream&, Date& d); // input |

Các toán tử này được định nghĩa để tránh quá tải và hưởng lợi từ tra cứu phụ thuộc vào đối số. Đối với Date, những toán tử này có thể được coi là những tiện ích đơn thuần. Tuy nhiên, đối với nhiều loại - chẳng hạn như số phức, vectơ, và các đối tượng giống hàm - việc sử dụng các toán tử thông thường khiến mọi người cho rằng định nghĩa của họ gần như là bắt buộc. Đối với Date, tôi đã muốn cung cấp + = và - =

dưới dạng các hàm thành viên thay vì add\_day ().Lưu ý rằng việc gán và khởi tạo sao chép được cung cấp theo mặc định.

# HÀM KHỞI TẠO, XÓA, SAO CHÉP VÀ DI CHUYỂN

## 1. Giới thiệu

Chương này tập trung vào các khía cạnh kỹ thuật ‘‘ vòng đời ’’ của đối tượng: Làm cách nào để chúng ta tạo một đối tượng, cách chúng ta sao chép nó, cách chúng ta di chuyển nó và làm cách nào để dọn dẹp nó sau khi nó biến mất? Định nghĩa thích hợp của '' copy '' và '' move '' là gì?

## 2. Hàm tạo và hàm hủy

Chúng ta có thể chỉ định cách khởi tạo một đối tượng của một lớp bằng cách định nghĩa một phương thức khởi tạo. Để bổ sung cho các hàm tạo, chúng ta có thể xác định một hàm hủy để đảm bảo '' dọn dẹp '' tại điểm phá hủyđối tượng (ví dụ: khi nó vượt ra khỏi phạm vi). Một số thao tác hiệu quả nhất để quản lý tài nguyên trong C ++ dựa trên các cặp hàm tạo / hủy. Vì vậy, các kỹ thuật khác dựa trên một cặp hành động, chẳng hạn như thực hiện / hoàn tác, bắt đầu / dừng, trước / sau, v.v. Ví dụ:

|  |
| --- |
| struct Tracer { |
| string mess;  Tracer(const string& s) :mess{s} { cout << mess; }  ~Tracer() {cout << "˜" << mess; }  };  void f(const vector<int>& v){ Tracer tr {"in f()\n"}; for (auto x : v) {  Tracer tr {string{"v loop "}+to<string>(x)+'\n'}; // ...  }  } |

Chúng ta có thể thực hiện lệnh f({2,3,5});

Điều này sẽ hiện ra: in\_f() v loop 2 ˜v loop 2 v loop 3 ˜v loop 3 v loop 5 ˜v loop 5 ˜in\_f()

### 2.1 Hàm hủy và tài nguyên

Một hàm tạo khởi tạo một đối tượng. Nói cách khác, nó tạo ra môi trường mà các chức năng thành viên hoạt động. Đôi khi, việc tạo ra môi trường đó liên quan đến việc có được một tài nguyên - chẳng hạn như tệp, khóa hoặc một số bộ nhớ - phải được giải phóng sau khi sử dụng. Do đó, một số lớp cần một hàm để đảm bảo sẽ được khai báo khi một đối tượng bị hủy theo cách tương tự như cách một phương thức khởi tạo đảm bảo sẽ được gọi khi một đối tượng được tạo. Không thể tránh khỏi, một hàm như vậy được gọi là hàm hủy. Tên của hàm hủy là ˜ theo sau là tên lớp, ví dụ ~Vector (). Một nghĩa của ~ là ‘‘ bổ sung ’’ và hàm hủy cho một lớp bổ sung cho các hàm tạo của nó. Một hàm hủy không nhận đối số và một lớp chỉ có thể có một hàm hủy. Bộ hủy được gọi ngầm khi một biến tự động vượt ra khỏi phạm vi, một đối tượng trên cửa hàng miễn phí bị xóa, v.v. Chỉ trong những trường hợp rất hiếm, người dùng mới cần gọi hàm hủy một cách rõ ràng. Hàm hủy thường dọn dẹp và giải phóng tài nguyên.

Ví dụ:

|  |
| --- |
| class Vector { |
| public:  Vector(int s) :elem{new double[s]}, sz{s} { }; // constructor: acquire memory  ˜Vector() { delete[] elem; } // destructor: release memory |
| // ... |
| private:  double elem; // elem points to an array of sz doubles int sz; // sz is non-negative }; |

Tại đây, Vectơ v1 bị hủy khi thoát khỏi f (). Ngoài ra, Vector tạo trên class bởi f () sử dụng new sẽ bị hủy bởi lệnh xóa. Trong cả hai trường hợp, hàm hủy của Vector được gọi để giải phóng (phân bổ) bộ nhớ được cấp phát bởi hàm tạo. Điều gì sẽ xảy ra nếu hàm tạo không có đủ bộ nhớ?

Ví dụ: ssizeof(double) or (s+s)sizeof(double) có thể lớn hơn dung lượng bộ nhớ khả dụng (tính bằng byte). Trong trường hợp đó, một ngoại lệ std :: bad\_alloc bị vứt bởi new và cơ chế xử lý ngoại lệ mở ra các hàm hủy thích hợp để lấy và giải phóng tất cả bộ nhớ . Phương thức quản lý tài nguyên dựa trên phương thức khởi tạo / hủy này được gọi là Khởi tạo Tài nguyên hay đơn giản là RAII. Một cặp hàm tạo / hàm hủy phù hợp là cơ chế thông thường để thực hiện khái niệm về một đối tượng có kích thước thay đổi trong C ++. Các vùng chứa thư viện tiêu chuẩn, chẳng hạn như vectơ và map không có thứ tự, sử dụng các biến thể của kỹ thuật này để cung cấp lưu trữ cho các phần tử của chúng. không có trình hủy được khai báo, chẳng hạn như kiểu tích hợp, được coi là có trình hủy không làm gì cả. Lập trình viên khai báo hàm hủy cho một lớp cũng phải quyết định xem đối tượng của lớp đó có thể được sao chép hoặc di chuyển hay không.

## 3. Sao chép và Di chuyển

Khi chúng ta cần chuyển một giá trị từ a sang b, chúng ta thường có hai lựa chọn khác nhau về mặt logic:

* Sao chép là ý nghĩa quy ước của x = y; nghĩa là, giá trị của x và y đều bằng giá trị của y trước khi gán.

+ Sao chép cho một lớp X được xác định bằng hai phép toán:

* Sao chép hàm tạo: X (const X &)
* Sao chép phép gán: X & operator = (const X &)
* Di chuyển các x với giá trị cũ của y và y với một số trạng thái đã chuyển. Đối với các trường hợp, vùng chứa, trạng thái được chuyển từ đó là ‘‘ trống ’’. Sự phân biệt này bị nhầm lẫn và thực tế là chúng ta sử dụng cùng một ký hiệu cho cả di chuyển và sao chép. Thông thường, không thể xóa đi bước di chuyển, trong khi một bản sao có thể (vì nó có thể cần lấy tài nguyên), và di chuyển thường hiệu quả hơn một bản sao. Khi bạn thực hiện một thao tác di chuyển, bạn nên để đối tượng nguồn ở trạng thái hợp lệ nhưng không xác định vì cuối cùng nó sẽ bị hủy và trình hủy không thể hủy đối tượng còn lại ở trạng thái không hợp lệ. Ngoài ra, các thuật toán thư viện tiêu chuẩn dựa vào việc có thể gán cho (sử dụng di chuyển hoặc sao chép) một đối tượng được chuyển đến. Vì vậy, hãy thiết kế các bước di chuyển của bạn để các đối tượng nguồn cho phép hủy và chuyển nhượng. Để tránh lặp lại, hãy sao chép và di chuyển các định nghĩa mặc định . Ví dụ:

|  |
| --- |
| template<class T> |
| void swap(T& a, T& b)  { |
| const T tmp = a; // put a copy of a into tmp |
| a = b; // put a copy of b into a b = tmp; // put a copy of tmp into b |
| }; |

## 4. Các chức năng deleted

Chúng ta có thể ‘‘ xóa ’’ một hàm; nghĩa là, chúng ta có thể nói rằng một hàm không tồn tại và không thể sử dụng nó. chức năng rõ ràng nhất là loại bỏ các chức năng mặc định khác. Ví dụ, người ta thường muốn ngăn chặn việc sao chép các lớp được sử dụng làm cơ sở vì việc sao chép như vậy dễ dẫn đến việc cắt lớp.

|  |
| --- |
| class Base { |
| // ...  Base& operator=(const Base&) = delete;// disallow copying  Base(const Base&) = delete;  Base& operator=(Base&&) = delete; // disallow moving  Base(Base&&) = delete;  }; |
| Base x1; |
| Base x2 {x1}; // error : no copy constructor |

Một chức năng nữa là kiểm soát nơi có thể cấp phát một lớp

Lưu ý sự khác biệt giữa một hàm = deleted và một hàm chỉ đơn giản là chưa được khai báo.

# NẠP CHỒNG

## 1. Giới thiệu

Để thuận tiện cho việc trình bày và thảo luận liên quan đến các khái niệm được sử dụng thường xuyên, hầu hết các lĩnh vực kĩ thuật và phi kĩ thuật phát triển các kí hiệu viết tắt thông thường, ví dụ (x+y\*z) thay vì (nhân y với z rồi cộng kết quả cho x).

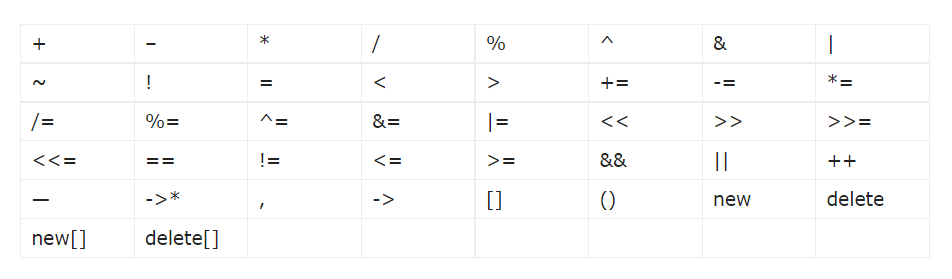
Giống như hầu hết các ngôn ngữ khác, C ++ hỗ trợ một tập hợp các toán tử cho các kiểu tích hợp của nó. Tuy nhiên, hầu hết các khái niệm mà các toán tử được sử dụng thông thường không phải là các kiểu tích hợp sẵn trong C ++, vì vậy chúng phải được biểu diễn dưới dạng các kiểu do người dùng định nghĩa. Ví dụ: nếu bạn cần số học phức tạp, đại số ma trận, tín hiệu logic hoặc chuỗi ký tự trong C ++, bạn sử dụng các class để biểu diễn các khái niệm này.

Việc xác định các toán tử cho các class như vậy đôi khi cho phép lập trình viên cung cấp một ký hiệu thông thường và thuận tiện hơn để thao tác các đối tượng hơn là có thể đạt được chỉ bằng cách sử dụng ký hiệu chức năng cơ bản.

|  |
| --- |
| class complex { // số phức đơn giản |
| double re, im; public:  complex(double r, double i) :re{r}, im{i} { } complex operator+(complex); complex operator (complex); }; |

Đây là một cách triển khai đơn giản của khái niệm số phức. **2. Nạp chồng hàm toán tử**

Các hàm xác định nghĩa cho các toán tử có thể được khai báo:



Tên của một hàm toán tử là toán tử từ khóa được theo sau bởi chính toán tử đó, ví dụ, toán tử <<. Một hàm toán tử được khai báo và có thể được gọi như bất kỳ hàm nào khác. Việc sử dụng toán tử chỉ là cách viết tắt cho một lệnh gọi rõ ràng của hàm toán tử. Ví dụ:

|  |
| --- |
| void f(complex a, complex b) |
| { complex c = a + b; // rút gọn  complex d = a.operator+(b); // rõ ràng } |

Với định nghĩa trên đây về số phức, hai bộ khởi tạo như nhau.

### 2.1. Toán tử 2 ngôi và 1 ngôi

Một toán tử 2 ngôi có thể được định nghĩa bởi một hàm thành viên không tĩnh mang một đối số hoặc một hàm không phải thành viên mang hai đối số.

|  |
| --- |
| class X { public: |
| void operator+(int);  X(int);  }; void operator+(X,X); void operator+(X,double); void f(X a){ a+1; // a.operator+(1) 1+a; // ::operator+(X(1),a) a+1.0; // ::operator+(a,1.0) } |

Toán tử một ngôi, dù là tiền tố hay hậu tố, nó có thể được xác định bởi một hàm thành viên không tĩnh không có đối số hoặc một hàm không phải là một đối số.

|  |
| --- |
| class X { |
| public: // hàm thành viên  X operator&(); // tiền tố 1 ngôi (&)  X operator&(X); // 2 ngôi & (And) |
| X operator++(int); // tăng hậu tố |
| X operator&(X,X); // Lỗi: có 3 thứ  X operator/(); // Lỗi: có 1 ngôi |
| }; |
| // hàm không là thành viên  X operator−(X); // tiền tố 1 ngôi (-)  X operator−(X,X); // 2 ngôi (-)  X operator−−(X&,int); // giảm hậu tố  X operator−(); // Lỗi: không có toán hạng  X operator−(X,X,X); // Lỗi: có 3 thứ  X operator%(X); // Lỗi: có 1 ngôi (%) |

### 2.2. Ý nghĩa được xác định trước cho các toán tử

Ý nghĩa của một số toán tử dựng sẵn được định nghĩa tương đương với một số kết hợp của các toán tử khác trên cùng các đối số.

|  |
| --- |
| class X { |
| public:  void operator=(const X&) = delete; void operator&() = delete;  void operator,(const X&) = delete;  }; void f(X a, X b){ a = b; // Lỗi : không có toán tử =() |
| &a; // Lỗi : không có toán tử &() |
| a,b; // Lỗi : không có toán tử,()  } |

Ngoài ra, chúng có thể được đưa ra các nghĩa mới bằng các định nghĩa phù hợp.

### 2.3. Toán tử và các loại do người dùng xác định

Một hàm toán tử phải là một thành viên hoặc có ít nhất một đối số của kiểu do người dùng xác định (các hàm xác định lại toán tử **new** và **delete** không cần). Một hàm toán tử nhằm chấp nhận một kiểu dựng sẵn vì toán hạng đầu tiên của nó không thể là một hàm thành viên.

Liệt kê là kiểu do người dùng xác định để chúng ta có thể xác định các toán tử cho chúng

|  |
| --- |
| enum Day { sun, mon, tue, wed, thu, fri, sat }; |
| Day& operator++(Day& d){ return d = (sat==d) ? sun : static\_cast<Day>(d+1); } |

## 3. Một loại số phức

|  |
| --- |
| void f(complex x, complex y, complex z){ |
| complex r1 {x+y+z}; // r1 = operator+(operator+(x,y),z) |
| complex r2 {x}; // r2 = x r2 += y; // r2.operator+=(y) |
| r2 += z; // r2.operator+=(z)  } |

Ngoại trừ sự khác biệt về tính hiệu quả có thể xảy ra, các tính toán của r1 và r2 là tương đương.

# TOÁN TỬ ĐẶC BIỆT

## 1. Giới thiệu

Nạp chồng không chỉ dành cho các phép toán số học và logic. Trên thực tế, các toán tử đóng vai trò quan trọng trong việc thiết kế vùng chứa, '' con trỏ thông minh '', iterator và các class khác liên quan đến quản lý tài nguyên.

## 2. Các toán tử đặc biệt



Các toán tử [] (chỉ số dưới) và () (gọi) là một trong những toán tử hữu ích nhất do người dùng xác định.

### 2.1. Toán tử gián tiếp

Toán tử gián tiếp, (còn được gọi là toán tử mũi tên), có thể được định nghĩa là toán tử hậu tố một ngôi.

Ví dụ:

|  |  |
| --- | --- |
| class Ptr {  X operator−>(); | }; |

Việc biến đổi đối tượng p thành con trỏ p.operator () không phụ thuộc vào thành viên m được trỏ tới. Đó là nghĩa mà toán tử () là một toán tử hậu tố một ngôi. Tuy nhiên, không có cú pháp mới nào được giới thiệu, vì vậy tên thành viên vẫn được yêu cầu sau dấu

|  |
| --- |
| void g(Ptr p){ |
| X q1 = p−>; // Lỗi  X q2 = p.operator−>(); // OK } |

### 2.2. Tăng và Giảm

Khi con người phát minh ra ''con trỏ thông minh'', họ thường quyết định cung cấp toán tử tăng ++ và toán tử giảm −− để phản ánh việc sử dụng các toán tử này cho các kiểu tích hợp. Điều này đặc biệt rõ ràng và cần thiết khi mục đích là thay thế một loại con trỏ thông thường bằng một loại '' con trỏ thông minh '' có cùng ngữ nghĩa, ngoại trừ việc nó thêm một chút kiểm tra lỗi thực thi. Ví dụ: hãy xem xét một chương trình truyền thống.

|  |
| --- |
| void f1(X a){// cách truyền thống |
| X v[200]; X p = &v[0]; p−−; p = a; // oops: p ngoài phạm vi  ++p;  p = a; // OK } |

Ở đây, chúng ta có thể muốn thay thế X  bằng một đối tượng của một lớp Ptr <X> chỉ có thể được tham chiếu nếu nó thực sự trỏ đến X.

|  |
| --- |
| void f2(Ptr<X> a){ // đã kiểm tra  X v[200]; |
| Ptr<X> p(&v[0],v); p−−; p = a; // Lỗi thực thi: p ngoài phạm vi  ++p; p = a; // OK } |

### 2.3. Cấp phát và giải phóng

Toán tử new có được bộ nhớ của nó bằng cách gọi một toán tử new (). Tương tự, toán tử xóa giải phóng bộ nhớ của nó bằng cách gọi một toán tử delete ().

|  |
| --- |
| void operator new(size\_t); // sử dụng cho từng đối tượng |
| void operator new[](size\_t); // sử dụng cho mảng void operator delete(void , size\_t); // sử dụng cho từng đối tượng void operator delete[](void , size\_t); // sử dụng cho mảng |

## 3. Các hàm hỗ trợ

Một tập hợp các hàm hữu ích, luồng I/O, hỗ trợ các vòng lặp phạm vi cho, so sánh và nối. Tất cả những điều này đều phản ánh các lựa chọn thiết kế được sử dụng cho std::string. Cụ thể, << chỉ in các ký tự mà không cần thêm định dạng và >> bỏ qua khoảng trắng đầu tiên trước khi đọc cho đến khi tìm thấy khoảng trắng kết thúc.

|  |
| --- |
| ostream& operator<<(ostream& os, const String& s){ |
| return os << s.c\_str(); } |

|  |
| --- |
| istream& operator>>(istream& is, String& s){ |
| s = ""; // xóa xâu mục tiêu is>>ws; // bỏ qua khoảng trắng char ch = ' '; while(is.get(ch) && !isspace(ch)) s += ch; return is; } |

|  |
| --- |
| bool operator!=(const String& a, const String& b){ |
| return !(a==b); } |

## 4. Hàm bạn

Một hàm được khai báo friend được cấp quyền truy cập vào việc triển khai một lớp giống như một hàm thành viên nhưng độc lập với lớp đó.

Ví dụ, chúng ta có thể xác định một toán tử nhân Ma trận với một Vectơ. Tuy nhiên, thói quen nhân của chúng ta không thể là thành viên của cả hai. Ngoài ra, chúng ta không thực sự muốn cung cấp các chức năng truy cập cấp thấp để cho phép mọi người dùng có thể đọc và ghi toàn bộ biểu diễn của cả Ma trận và Vectơ. Để tránh điều này, chúng ta khai báo toán tử là bạn của cả hai

|  |
| --- |
| constexpr rc\_max {4}; // size của hàng và cột |
| class Matrix { Vector v[rc\_max]; friend Vector operator(const Matrix&, const Vector&); }; class Vector { float v[rc\_max]; friend Vector operator(const Matrix&, const Vector&); }; |

Bây giờ toán tử () có thể tiếp cận việc triển khai cả Vectơ và Ma trận. Điều đó sẽ cho phép các kỹ thuật triển khai phức tạp, nhưng thực hiện đơn giản:

|  |
| --- |
| Vector operator (const Matrix& m, const Vector& v){  Vector r; |
| for (int i = 0; i!=rc\_max; i++) { // r[i] = m[i] \* v; r.v[i] = 0; for (int j = 0; j!=rc\_max; j++) |
| r.v[i] += m.v[i].v[j]  v.v[j]; |
| } return r; } |

# LỚP DẪN XUẤT

**1. Giới thiệu:**

Một khái niệm không tồn tại một cách cô lập. Nó tồn tại cùng với các khái niệm liên quan và phần lớn sức mạnh của nó có được từ các mối quan hệ với các khái niệm khác nhau. Ví dụ, cố gắng giải thích những gì một chiếc xe hơi. Bạn sẽ sớm giới thiệu các khái niệm về bánh xe, động cơ, người lái xe, người đi bộ, xe tải, xe cứu thương, đường sá, dầu nhớt, vé chạy quá tốc độ, nhà nghỉ, v.v. Vì chúng tôi sử dụng các lớp để giải thích các khái niệm, vấn đề trở thành cách thể hiện mối quan hệ giữa các khái niệm. Tuy nhiên, chúng tôi không thể diễn đạt mối quan hệ tùy ý trực tiếp bằng ngôn ngữ lập trình. Ngay cả khi chúng tôi có thể, chúng tôi sẽ không muốn. Để trở nên hữu ích, các lớp của chúng ta nên được định nghĩa hẹp hơn so với các khái niệm hàng ngày của chúng và chính xác hơn.

Khái niệm về một lớp dẫn xuất và các cơ chế ngôn ngữ liên quan của nó được cung cấp để thể hiện các mối quan hệ thứ bậc, nghĩa là, để thể hiện tính chung giữa các lớp. Ví dụ, các khái niệm về hình tròn và hình tam giác có liên quan với nhau ở chỗ chúng đều là hình dạng; nghĩa là họ có chung khái niệm về một hình dạng. Do đó, chúng ta xác định rõ ràng lớp Circle và lớp Triangle để có chung lớp Shape. Trong trường hợp đó, lớp chung, ở đây Shape, được gọi là lớp cơ sở hoặc siêu lớp và các lớp dẫn xuất từ lớp đó, ở đây Circle và Triangle, được gọi là lớp dẫn xuất hoặc các lớp con. Biểu diễn hình tròn và hình tam giác trong một chương trình mà không liên quan đến khái niệm về hình dạng sẽ là thiếu một thứ thiết yếu. Chương này là sự khám phá ý nghĩa của ý tưởng đơn giản này, nó là cơ sở cho cái thường được gọi là lập trình hướng đối tượng. Các tính năng ngôn ngữ hỗ trợ xây dựng các lớp mới từ những lớp hiện có:

* Triển khai tính kế thừa : để tiết kiệm việc triển khai bằng cách chia sẻ các phương thức, thuộc tính được cung cấp bởi một lớp cơ sở
* Kế thừa giao diện: cho phép các lớp dẫn xuất khác nhau được sử dụng thay thế cho nhau thông qua giao diện được cung cấp bởi một lớp cơ sở chung.

Kế thừa giao diện thường được gọi là đa hình thời gian chạy (hoặc đa hình động). Ngược lại, việc sử dụng thống nhất các lớp không liên quan đến tính kế thừa được cung cấp bởi các mẫu, thường được gọi là đa hình thời gian biên dịch (hoặc đa hình tĩnh).

## 2. Lớp dẫn xuất

Xây dựng một chương trình giao dịch với những người được một công ty tuyển dụng. Một chương trình có thể có cấu trúc dữ liệu như thế này

|  |
| --- |
| struct Employee{ |
| string name, |
| char middle\_initial; |
| Date hiring\_date;  } |

Tiếp theo, chúng ta định nghĩa cấu trúc của người quản lý

|  |
| --- |
| struct Manager { |
| Employee emp;  list<Employee\*> group; } |

Một nhà quản lý cũng là một nhân viên; dữ liệu Employee được lưu trữ trong thành viên “emp” của đối tượng Manager. Điều này có thể hiển nhiên đối với người đọc là con người - đặc biệt là người đọc cẩn thận - nhưng không có gì nói với trình biên dịch và các công cụ khác rằng Manager cũng là Employee. Manager  không phải là Employee , vì vậy người ta không thể đơn giản sử dụng cái này khi người kia được yêu cầu. Đặc biệt, người ta không thể đặt một Manager vào danh sách Employee mà không cần viết mã đặc biệt. Chúng ta có thể sử dụng chuyển đổi loại rõ ràng trên Manager  hoặc đưa địa chỉ của thành viên “emp” vào danh sách

Employee. Tuy nhiên, cả hai giải pháp đều không phù hợp và có thể khá mù mờ. Cách tiếp cận đúng là tuyên bố rõ ràng rằng Manager là Employee, với một vài thông tin được thêm vào:

|  |
| --- |
| struct Manager : public Employee{ |
| list<Employee\*>group;  } |

Manager có nguồn gốc từ Employee, và ngược lại, Employee là một lớp cơ sở cho Manager. Lớp Manager có các thuộc tính của lớp Employee (tên người, phòng ban, v.v.) ngoài các thành viên của chính nó (nhóm, cấp, v.v.).

Khởi tạo thường được biểu diễn bằng đồ thị bởi một con trỏ từ lớp dẫn xuất đến lớp cơ sở của nó cho biết rằng lớp dẫn xuất tham chiếu đến cơ sở của nó (thay vì ngược lại):

Một lớp dẫn xuất thường được cho là kế thừa các thuộc tính từ cơ sở của nó, vì vậy mối quan hệ còn được gọi là kế thừa. Một lớp cơ sở đôi khi được gọi là lớp cha và lớp dẫn xuất là lớp con. Tuy nhiên, thuật ngữ này gây nhầm lẫn cho những người quan sát rằng dữ liệu trong một đối tượng lớp dẫn xuất là một tập siêu dữ liệu của một đối tượng thuộc lớp cơ sở của nó. Một lớp dẫn xuất thường lớn hơn (và không bao giờ nhỏ hơn) so với lớp cơ sở của nó theo nghĩa là nó chứa nhiều dữ liệu hơn và cung cấp nhiều chức năng hơn.

# PHẦN II: PHÂN TÍCH ĐỀ TÀI

# DINO RUN

# 1. Lý do chọn đề tài

Sự bùng nổ của cộng nghệ thông tin toàn cầu hiện nay, đã có những bước phát triển mạnh mẽ của các công nghệ số. Chắc hẳn ai sử dụng trình duyệt Google Chrome cũng từng chơi trò khủng long né chướng ngại vật mỗi khi bị mất mạng. Vì nhóm khá thích game này và nhằm mục đích giải trí sau những giờ học tập căng thẳng cũng như để luyện khả năng code môn Lập trình Hướng đối tượng nên nhóm đã lấy “DINO RUN” làm đề tài bài tập lớn của mình.

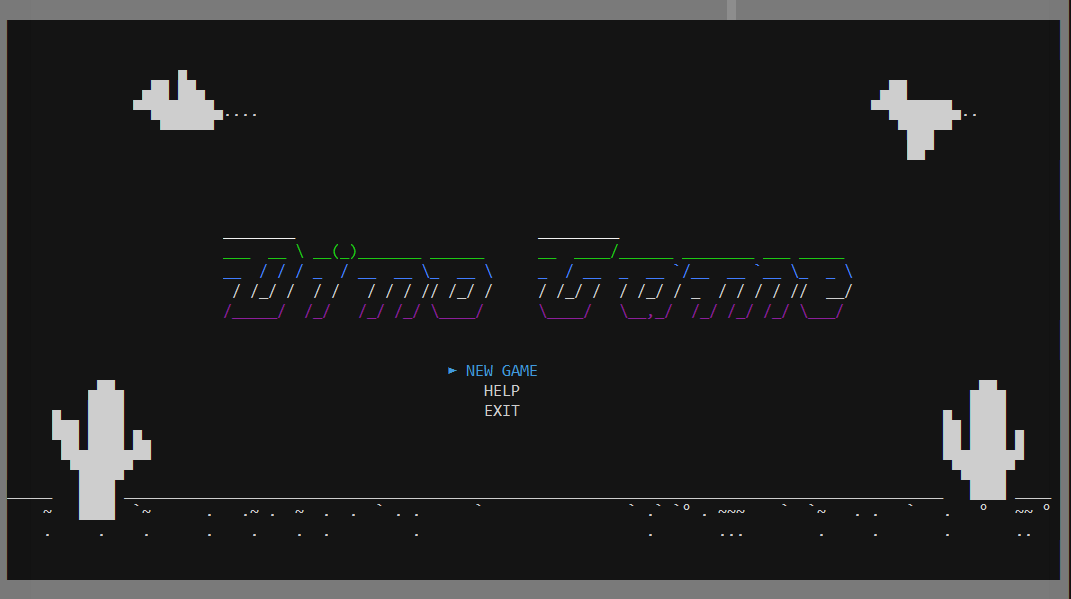
Sau một quá trình dài học tập môn Lập trình Hướng đối tượng Chúng em đã được trang bị những kiến thức, kỹ năng cơ bản để hoàn thành đề tài “DINO RUN”. Với mong muốn ứng dụng này đáp ứng được nhu cầu giải trí cho người dùng. Do thiếu kiến thức và đang trong quá trình trao dồi thêm, việc sai sót là điều không thể tránh khỏi. Kính mong nhận được sự góp ý và nhận xét từ quý thầy cô để giúp chúng em có thể hoàn thiện hơn trong môn học, học tập và rút ra những kinh nghiệm quý báu trong tương lai.

# 2. Mô tả bài toán

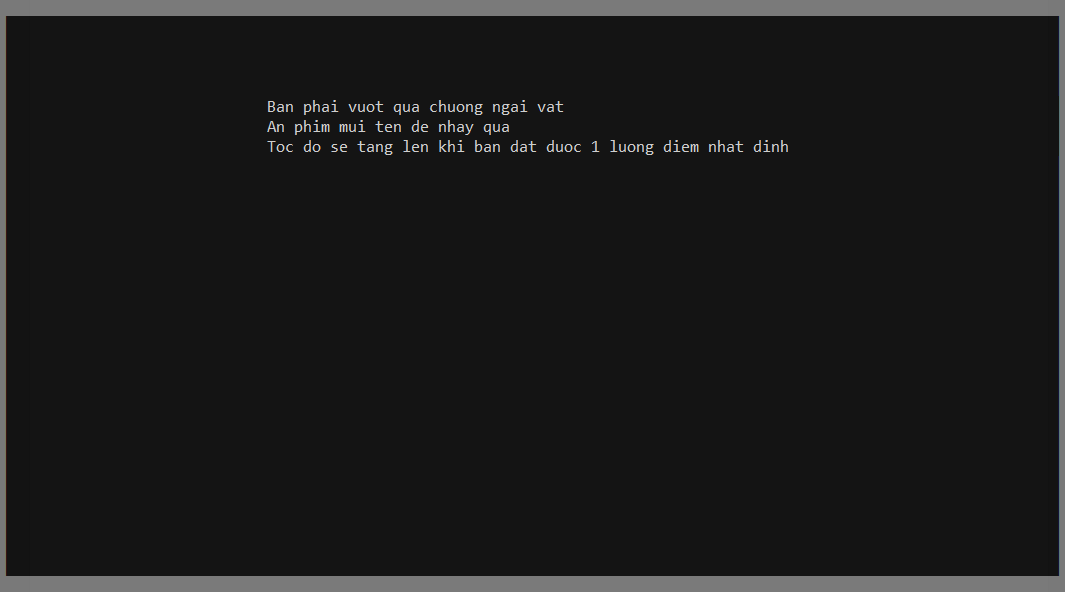
Chương trình game dino run là một trò chơi được xây dựng từ nhiều cấu trúc class để lưu trữ đối tượng có nhiều thuộc tính, sau đó sử dụng các thuật toán, vòng lặp… để game được hoàn thành. Game có 4 chức năng chính gồm điều khiển, kiểm tra chạm, vòng lặp giao diện và ghi điểm. đồ hoạ game làm những hàm đồ hoạ cơ bản trên màn hình console để tạo ra giao diện game đơn giản nhất.

# 3. Giao diện chương trình

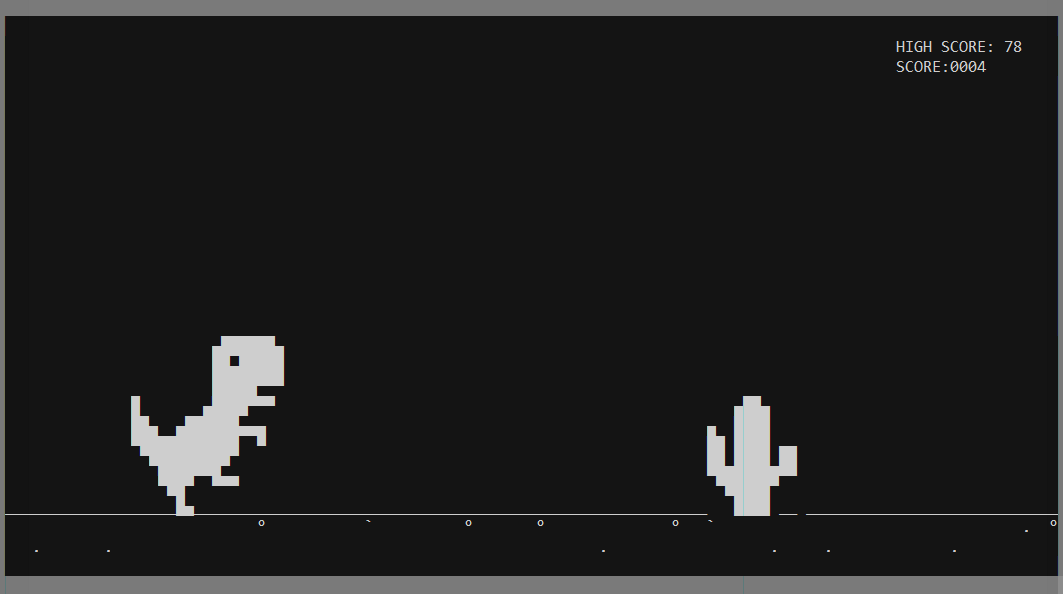
1. Hình ảnh giao diện đầu của game(MENU):



1. Hình ảnh giao diện phần trợ giúp(HELP):



1. Hình ảnh giao diện khi chơi game (NEW GAME):



1. Hình ảnh khi dino nhảy vượt chướng ngại vật(chức năng điều khiển):



1. Hình ảnh khi chạm vào vật cản (kiểm tra chạm):



# 4. Tính chất hướng đối tượng

Chương trình được sử dụng đầy đủ 4 tính chất của hướng đối tượng: tính đóng gói, tính trừu tượng, tính kế thừa và tính đa hình.

## 4.1 Tính đóng gói

Tính đóng gói (Encapsulation) là một khái niệm của lập trình hướng đối tượng mà ràng buộc dữ liệu và các hàm mà thao tác dữ liệu đó, và giữ chúng an toàn bởi ngăn cản sự gây trở ngại và sự lạm dụng từ bên ngoài. Tính đóng gói dẫn đến khái niệm OOP quan trọng là **Data Hiding**.

Một lớp có thể chứa các thành viên **private, protected và public**. Theo mặc định, tất cả thuộc tính được định nghĩa trong một lớp là **private** hoặc **protected**.

Tính đóng gói trong bài tập:

|  |
| --- |
|  |

Các biến xdino, ydino, h và check\_die là **protected**. Nghĩa là chúng chỉ có thể được truy cập bởi các thành viên khác của lớp DINO, và không thể bởi bất kỳ phần khác trong chương trình.

## 4.2 Tính trừu tượng

Trừu tượng hóa dữ liệu (Data abstraction) liên quan tới việc chỉ cung cấp thông tin cần thiết tới bên ngoài và ẩn chi tiết cơ sở của chúng, ví dụ: để biểu diễn thông tin cần thiết trong chương trình mà không hiển thị chi tiết về chúng.

Tính trừu tượng trong bài tập:

|  |
| --- |
|  |
|  |

Lớp DINO hay lớp VC(vật cản) chỉ cần quan tâm đến toạ độ x, y của con khủng long hay toạ độ x, y của vật cản nhưng chúng đều ở trong protected và muốn truy cập vào hoặc thay đổi các thuộc tính **private** hay **protected** từ bên ngoài thì phải thông qua các cơ chế được cung cấp sẵn,ta có thể truy cập thông qua các hàm get\_x, get\_y, get\_h, get\_xVc,.. được chúng em xây dựng ở **public** để có thể lấy toạ độ ra và sử dụng**.**

## 4.3 Tính kế thừa

Một trong những khái niệm quan trọng nhất trong lập trình hướng đối tượng là Tính kế thừa (Inheritance). Là sự liên quan giữa hai class với nhau, trong đó có class cơ sở (Base Class) và class con (Derived Class). Khi kế thừa class con được hưởng tất cả các phương thức và thuộc tính của class cơ sở. Tuy nhiên, nó chỉ được truy cập các thành viên public và protected của class cơ sở. Nó không được phép truy cập đến thành viên private của class cơ sở.

Khi kế thừa từ một lớp đang tồn tại ta có sử dụng lại các phương thức và thuộc tính của lớp cơ sở, đồng thời có thể khai báo thêm các phương thức và thuộc tính khác. Khi kế thừa từ một lớp cơ sở, lớp cơ sở đó có thể được kế thừa thông qua kiểu kế thừa là **public**, **protected** hoặc **private**.

Chúng ta hiếm khi sử dụng kiểu kế thừa **protected** hoặc **private**, nhưng kiểu kế thừa **public** thì được sử dụng phổ biến hơn. Trong khi sử dụng các kiểu kế thừa khác sau, bạn nên ghi nhớ các quy tắc sau:

* **Kiểu kế thừa Public:** Khi kế thừa từ một lớp cơ sở là **public**, thì các thành viên **public** của lớp cơ sở trở thành các thành viên **public** của lớp kế thừa; và các thành viên **protected** của lớp có sở trở thành các thành viên **protected** của lớp kế thừa. Một thành viên là **private** của lớp cơ sở là không bao giờ có thể được truy cập trực tiếp từ một lớp kế thừa, nhưng có thể truy cập thông qua các lời gọi tới các thành viên **public** và **protected** của lớp cơ sở đó.
* **Kiểu kế thừa protected:** Khi kế thừa từ một lớp cơ sở là **protected**, thì các thành viên **public** và **protected** của lớp cơ sở trở thành các thành viên **protected** của lớp kế thừa.
* **Kiểu kế thừa private:** Khi kế thừa từ một lớp cơ sở là **private**, thì các thành viên **public** và **protected** của lớp cơ sở trở thành các thành viên **private** của lớp kế thừa.

Cụ thể trong bài nhóm em sử dụng kiểu kế thừa public.

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

Lớp Ground được kế thừa public từ lớp cơ sở là DINO, lớp VC được kế thừa public từ lớp Ground. Như vậy, Ground có thể sử dụng tất cả phương thức của DINO và VC có thể sử dụng tất cả các phương thức từ Ground lẫn DINO, trừ thuộc tính private không thể truy xuất. đây gọi là tính kế thừa nhiều mức,

## 4.4 Tính đa hình

**Đa hình (polymorphism)** nghĩa là có nhiều hình thái khác nhau. Tiêu biểu là, đa hình xuất hiện khi có một cấu trúc cấp bậc của các lớp và chúng là liên quan với nhau bởi tính kế thừa.

Đa hình trong C++ nghĩa là một lời gọi tới một hàm thành viên sẽ làm cho một hàm khác để được thực thi phụ thuộc vào kiểu của đối tượng mà triệu hồi hàm đó.

Một hàm **virtual** là một hàm trong một lớp cơ sở mà được khai báo bởi sử dụng từ khóa virtual trong C++. Việc định nghĩa trong một lớp cơ sở một hàm virtual, với phiên bản khác trong một lớp kế thừa, báo cho compiler rằng: chúng ta không muốn Static Linkage cho hàm này. Những gì chúng ta làm là muốn việc lựa chọn hàm để được gọi tại bất kỳ điểm nào đã cung cấp trong chương trình là dựa trên kiểu đối tượng, mà với đó nó được gọi

|  |
| --- |
|  |
|  |
|  |

Cụ thể trong bài tập, nhóm em cho đa hình hàm draw ở lớp cơ sở DINO, và hàm draw sẽ được định nghĩa lại ở lớp Ground, khi gọi hàm draw ở kiểu Ground thì nó sẽ chạy và tìm xem ngoài draw ở lớp DINO ra còn có hàm draw ở lớp nào mà nó được định nghịa lại không, nếu có thì nó sẽ gọi hàm đã được định nghĩa lại ở lớp Ground để thực hiện. và nếu gọi theo kiểu DINO thì nó sẽ lấy draw được định nghĩa ở lớp DINO để thực hiện.

# 5. Nội dung lý thuyết :

## 5.1 Chức năng vòng lặp giao diện :

Dùng hàm GOTOXY để tìm tới toạ độ của đối tượng đã được khởi tạo trước để in đối tượng ra màn hình.

|  |
| --- |
|  |
|  |
|  |

Tiếp theo là mặt đất để in ra mặt đất trước tiên chúng em đã khai báo một mảng 2 chiều trong class Ground để định nghĩa ra hàng và cột của mặt đất.

|  |
| --- |
|  |

Dùng hàm rand để tạo ra những vị trí ngẫu nhiên trên mặt đất in ra những viên sỏi đá từ các ký tự bàn phím.

|  |
| --- |
|  |

Khởi tạo ra mặt đất và in mặt đất ra màn hình theo toạ độ đã được khởi tạo trước.

Dùng vòng lặp while để in ra màn hình liên lục các đối tượng trên.

## 5.2 Chức năng điều khiển :

Trong vòng lặp while đã tạo để in ra màn hình các đối tượng, ta tổng hợp các hàm xử lý chức năng in ra màn hình, sau khi đã in ra màn hình và làm cho các đối tượng chuyển động, ta cần một hàm để xử lý tình huống khi ấn nút mũi tên hướng lên thì khủng long sẽ nhảy vượt qua chướng ngại vật. Ở đây chúng em sử dụng hàm bắt sự kiện nhấn phím kbhit trong thư viện conio.h.

Để có thể thực hiện bắt sự kiện bàn phím với bất kỳ ngôn ngữ nào, ta cần biết mã ASCII của các phím đó. Sau đây là mã ASCII của các phím thông dụng trên bàn phím.

**Mã ASCII của các phím thông** **dụng**:

* Phím Enter : 13
* Phím ESC: 27
* Phím Tab: 9
* Mũi tên lên: 72
* Mũi tên xuống: 80
* Mũi tên sang trái: 75
* Mũi tên sang phải: 77
* Phím cách: 32

Sau đây là phần bắt phím trong bài tập:

|  |
| --- |
|  |

Dòng if(kbhit()) để kiểm tra có sự kiện nhập phím của người dùng hay không. Nếu có, dùng getch() để tóm phím đó. Nếu như phím ấn vào là phím mũi tên lên có giá trị ascii là 72 thì khủng long sẽ được in nhảy dần lên trên và đến mỗi khoảng thì lại in rơi xuống vị trí ban đầu

Tạo ra biến int check để tạo ra trường hợp in khủng long ở các trạng thái: ở mặt đất(check= 0), nhảy(check= 1), rơi xuống(check= 2).

|  |
| --- |
|  |
|  |
|  |

## 5.3 Chức năng xử lý thắng thua:

Để xử lý trường hợp thắng hay thua trong game thì chúng em tạo ra một hàm bool để kiểm tra trường hợp khủng long chạm vào vật cản.

Trong quá trình chơi toạ độ của vật cản sẽ liên tục thay đổi để đồi chạy cắt ngang qua khủng long. Chúng em sẽ tạo ra các biến và gán chúng bằng các toạ độ của khủng long và vật cản, lúc này thì chúng em sẽ kiểm tra toạ độ của khủng long đã trùng với toạ độ của vật cản hay chưa.

Sau đây là phần code kiểm tra chạm nằm trong class VC trong bài tập:

|  |
| --- |
|  |
|  |

Ở đây chúng em sẽ chia toạ độ của khủng long ra thành 3 trường hợp để kiểm tra chạm.

Trường hợp thứ nhất thì chúng em sẽ xét toạ độ ở phần bên trái(**xchan\_trai**) với toạ độ bên trái của vật cản(**x\_vc**) và toạ độ phần bên phải của khủng long(**xchan\_phai**) với toạ độ bên phải của vật cản(**x\_vc\_max**) liệu có trùng với nhau hay không.

Trường hợp thứ 2 cũng tương tự ta xét **xchan\_trai** với **x\_vc\_max** và **xchan\_phai** với **x\_vc** liệu có trùng với nhau hay không.

Và nếu một trong 2 trường hợp trên là đúng sẽ kiểm tra tiếp trường hợp thứ 3 xét toạ độ cuối cùng của khủng long theo chiều y(ychan) với toạ độ của y vật cản(y\_Vc) và toạ độ của y của mặt đất(ydat) xem nó có trùng với nhau không.

Nếu tất cả các trường hợp trên đều đúng thì hàm kiểm tra chạm sẽ trả về giá trị true và game sẽ tự động thông báo và thoát ra khỏi vòng lặp.



|  |
| --- |
|  |

Và ngược lại nếu không đúng thì game sẽ tiếp tục đến khi hàm trả về là true.

## 5.4 Chức năng ghi điểm:

Hàm tính điểm và in ra màn hình trong bài tập:

|  |
| --- |
|  |
|  |
|  |

Điểm sẽ được tăng lên 1 cứ mỗi lần biến **dem** tăng đến giá trị nhất định

Ở chức năng này chúng em sử dụng **vector** để có thể lấy điểm cao nhất( highscore) và in ra màn hình.

Không giống như [array (mảng)](https://topdev.vn/blog/4-ung-dung-cua-array-from/), chỉ một số giá trị nhất định có thể được lưu trữ dưới một tên biến duy nhất. Vector trong C++ giống dynamic array (mảng động) nhưng có khả năng tự động thay đổi kích thước khi một phần tử được chèn hoặc xóa tùy thuộc vào nhu cầu của tác vụ được thực thi, với việc lưu trữ của chúng sẽ được vùng chứa tự động xử lý.

Không giống như mảng động, cái mà không biết được độ dài của mảng mà nó đang trỏ tới là bao nhiêu, **vectors** tự theo dõi độ dài của chính nó. Chúng ta có thể lấy được độ dài của vector thông qua hàm **size()**.

Và dựa vào tính chất này chúng em đã dùng các hàm **begin()** và **end()** để có thể đặt vòng lặp vào phần tử đầu với cuối của file().

Code đọc ghi file trong bài tập:

|  |
| --- |
|  |

Dùng hàm **sort()** để sắp xếp các giá trị điểm theo chiều từ thấp tới cao, khi sắp xếp xong thì trả về giá trị cuối trong file(là điểm cao nhất sau khi sắp xếp).

Cơ chế ghi điểm vào file trong code:

|  |
| --- |
|  |
|  |
|  |

Cứ mỗi khi chết thì điểm chơi được sẽ được ghi vào file ở dạng TXT.

# 6. Kết luận :

## 6.1 Kết quả đạt được :

- Nhóm chúng em đã hoàn thành xong game Dino run với các chức năng cơ bản khác nhau như: điều khiển khủng long, ghi điểm, lưu file, xuất điểm lớn nhất trong file ,...

## 6.2 Nhược điểm

* Chưa hiểu sâu về đồ hoạ nên làm còn nhiều khó khăn.
* Không có nhiều ý tưởng hay về chương trình đang làm, không có nhiều chức năng hay, mới lạ.

## 6.3 Hướng phát triển

- Nhóm chúng em sẽ cố gắng học hỏi và phát triển game Dino run với nhiều chức năng hơn, ứng dụng vào thực tế nhiều hơn, thỏa mãn người dùng.

## 6.4 Tài liệu tham khảo

Giáo trình tham khảo: The C++ Programming Language.

[Học xong lập trình hướng đối tượng chỉ trong 1 giờ | Vũ Nguyễn Coder:](https://www.youtube.com/watch?v=b5yPlqMrraE)

[Hướng dẫn làm game khủng long - T-rex Run bằng C++](https://www.google.com/search?q=H%C6%B0%E1%BB%9Bng+d%E1%BA%ABn+l%C3%A0m+game+kh%E1%BB%A7ng+long+-+T-rex+Run+b%E1%BA%B1ng+C%2B%2B&rlz=1C1GCEA_enVN1030VN1030&oq=H%C6%B0%E1%BB%9Bng+d%E1%BA%ABn+l%C3%A0m+game+kh%E1%BB%A7ng+long+-+T-rex+Run+b%E1%BA%B1ng+C%2B%2B&aqs=chrome..69i57.601j0j7&sourceid=chrome&ie=UTF-8)

**Phụ Lục:**