TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO BÀI TẬP LỚN

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

CHƯƠNG TRÌNH QUẢN LÝ BỆNH NHÂN MẮC COVID

Giảng viên hướng dẫn:	Trần Thị Dung
Sinh viên:	**MSSV:**
Trần Đức Anh	6151071034
Nguyễn Hoàng Phát	6151071082
Trịnh Ngọc Minh	6151071073
Mã học phần:	CPM04.3

BẢNG PHÂN CHIA CÔNG VIỆC

Thành viên	Khối lượng công việc	Các phần được phân công
Trần Đức Anh	30%	- Class Person - Duyệt trước - Duyệt giữa - Duyệt sau - Thống kê theo nơi cách ly - Sửa
Nguyễn Hoàng Phát	40%	- Class Doubly - Sửa - Sắp xếp theo tên - Thống kê F - Thống kê sức khỏe bệnh nhân F0 - Thống kê sức khỏe bệnh nhân - Thông kê theo ngày cách ly dài
Trịnh Ngọc Minh	30%	- Program, display - Xóa - Giải phóng - Tìm kiếm - In file ra màn hình

(*) Trên đây là bảng phân chia công việc, trong quá trình làm bài, mọi thành viên đều hỗ trợ cho nhau để bài tập lớn được hoàn thiện nhất có thể nên số liệu chỉ minh họa không chính xác hoàn toàn.

(*) phân chia làm báo cáo được dựa theo các phần các thành viên làm trên 1 file google doc chung, mọi thành viên đều góp ý sửa chữa cho nhau để bài báo cáo hoàn thiện tốt nhất sau đó nhóm trưởng chuyển sang bản word rồi nộp cho giảng viên.

MỤC LỤC

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 "Quản lý bệnh nhân mắc covid". 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: s∗sizeof(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:

+	-	*	/	%	^	&
|	~	!	=	<	>	+=
-=	*=	/=	%=	^=	&=	|=
<<	>>	>>=	<<=	==	!=	<=
>=	&&	||	++	--	->*	,
->	[]	()	new	new[]	delete	delete[]

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

[] () -> ++ -- new delete

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:

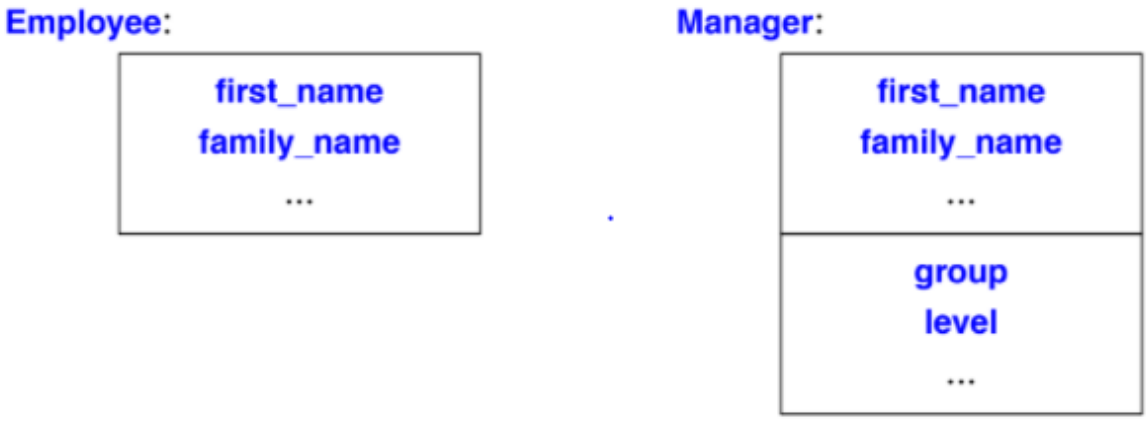
```
struct Manager : public Employee{
    list<Employee*>group;
}
```

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.

Một cách triển khai phổ biến và hiệu quả của khái niệm lớp dẫn xuất có **một đối tượng của lớp dẫn xuất được biểu diễn như một đối tượng của lớp cơ sở**, với thông tin thuộc về lớp dẫn xuất cụ thể được thêm vào cuối. Ví dụ:



PHÂN CẤP LỚP

1. Giới thiệu:

Trọng tâm chính của chương này là các kỹ thuật thiết kế, hơn là các tính năng ngôn ngữ. Các ví dụ được lấy từ thiết kế giao diện người dùng, nhưng tôi tránh chủ đề về lập trình hướng sự kiện như thường được sử dụng cho các hệ thống giao diện người dùng đồ họa (GUI).

2. Triển khai kế thừa:

Một cấu trúc phân cấp lớp sử dụng kế thừa thực thi (như thường thấy trong các chương trình cũ hơn). Lớp Ival_box xác định giao diện cơ bản cho tất cả các Ival_box và chỉ định triển khai mặc định mà các loại Ival_box cụ thể hơn có thể ghi đè bằng các phiên bản của riêng chúng. Ngoài ra, khai báo dữ liệu cần thiết để triển khai khái niệm cơ bản:

```
Class iVal_box {
Protected:
    Int val, low, high;
    Bool changed {false};
Public:
    Ival_box(int ll, int hh) : val {ll}, low {ll}, high {ll} { }
    Virtual int get_value() {changed = false, return val; }
    Virtual void set_value(int i) {changed = true, val = i; }
    Virtual void reset_value(int i) {changed = false, val = i; }
    Virtual void prompt() { }
    Virtual bool was_changed () const {return changed;}
};
```

Việc triển khai mặc định của các chức năng là khá cẩu thả và được cung cấp ở đây chủ yếu để minh họa ngữ nghĩa dự định. Ví dụ, một lớp thực tế sẽ cung cấp một số kiểm tra phạm vi. Một lập trình viên nên sử dụng “ival classes” như thế này:

```
void interact(Ival_box pb){
    pb->prompt(); // alert user
    //...
    int i = pb->get_value();
    if(pb->was_changed()){
        //...new value, do something...
    } else {
        //...do something else
    }
}
void some_fct(){
    //Ival_slider derived from Ival_box
    unique_ptr<Ival_box>p1 {new Ival_slider{0,5}};
    interact(p1.get());

    unique_ptr<Ival_box>p2 {new Ival_dial{1,12}};
    interact(p2.get());
}

class Ival_box : public BBwidget{/*....*/}; // rewritten to use BBwidget
```

Hầu hết mã ứng dụng được viết dưới dạng (con trỏ tới) Ival_boxes thuần túy như cách tương tác (). Bằng cách đó, ứng dụng không muốn biết về số lượng lớn các biến thể tiềm ẩn của khái niệm Ival_box. Kiến thức của các lớp chuyên biệt như vậy bị cô lập trong tương đối ít chức năng tạo ra các đối tượng như vậy. Điều này cách ly người dùng khỏi những thay đổi trong việc triển khai các lớp dẫn xuất. Hầu hết các mã có thể bị lãng quên bởi thực tế là có nhiều loại Ival_box khác nhau.

Các loại Ival_box khác nhau được định nghĩa là các lớp bắt nguồn từ Ival_box. Ví dụ:

```
Class Ival_slider : public Ival_box {
Private:
    // nội dung đồ họa để xác định thanh trượt trông như thế nào, v.v.
Public:
    Ival_slider (int, int);
    Int get_value () override;
    Void promt () override;
}
```

Các thành viên dữ liệu của Ival_box đã được khai báo được bảo vệ để cho phép truy cập từ các lớp dẫn xuất. Do đó, Ival_slider :: g et_value () có thể gửi một giá trị vào Ival_box :: val. Thành viên được bảo vệ có thể truy cập được từ các thành viên của chính lớp và từ các thành viên của các lớp dẫn xuất, nhưng không phải đối với người dùng thông thường

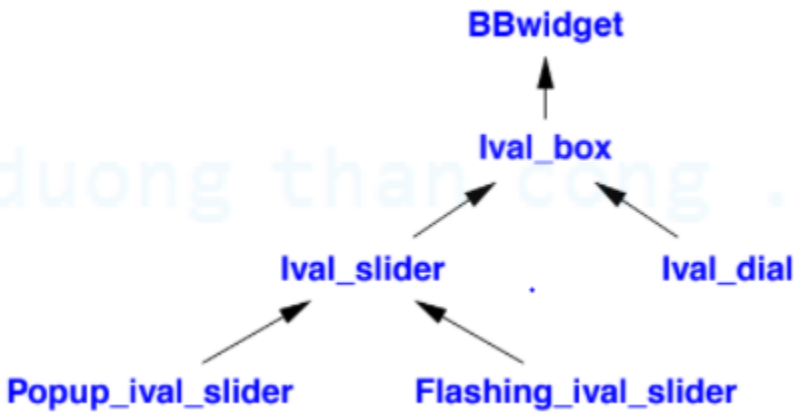
Ngoài Ival_slider, chúng tôi sẽ xác định các biến thể khác của khái niệm Ival_box. Chúng có thể bao gồm Ival_dial, cho phép bạn chọn một giá trị bằng cách xoay một núm; Flashing_ival_slider, nhấp nháy khi bạn yêu cầu promt (); và Popup_ival_slider, phản hồi với prompt () bằng cách xuất hiện ở một số nơi nổi bật, do đó khiến người dùng khó có thể bỏ qua.

Chúng ta sẽ lấy đồ họa từ đâu? Hầu hết các hệ thống giao diện người dùng cung cấp một lớp xác định các thuộc tính cơ bản của một thực thể trên màn hình. Vì vậy, nếu chúng tôi sử dụng hệ thống từ ‘‘ Big Bucks Inc. ’’, chúng tôi sẽ phải làm cho mỗi lớp Ival_slider, Ival_dial, v.v., của chúng tôi trở thành một loại BBwidget. Điều này đơn giản nhất sẽ đạt được bằng cách viết lại Ival_box của chúng tôi để nó bắt nguồn từ BBwidget.

Theo cách đó, tất cả các lớp của chúng ta kế thừa tất cả các thuộc tính của một BBwidget. Ví dụ: mọi Ival_box đều có thể được đặt trên màn hình, tuân theo các quy tắc về kiểu đồ họa, được thay đổi kích thước, được kéo xung quanh, v.v., theo tiêu chuẩn do hệ thống BBwidget đặt ra. Hệ thống phân cấp lớp của chúng ta sẽ trông như thế này:

```
class Ival_slider : public Ival_box{/*....*/};
class Ival_dial : public Ival_box{/*....*/};
class Flashing_ival_slider : public Ival_slider {/*....*/};
class Popup_ival_slider : public Ival_slider {/*....*/};
```

Hoặc có dạng đồ thị:



PHẦN II: CHƯƠNG TRÌNH

1. Lý do chọn đề tài

Đại dịch covid-19, đã mang tới nhiều áp lực cho nền ý tế của nhiều nước trên thế giới. Kể cả những nước có nền y tế tiên tiến nhất trên thế giới như Mỹ, Ý, Anh...Cũng phải chịu những áp lực rất lớn về việc giải quyết các ca bệnh, quản lý bệnh nhân, truy vết các đối tượng F0....Ở Việt Nam ta còn là một nước đang phát triển, dân số đông, nền y tế còn một số hạn chế nhất định, thế nên những khó khăn đó còn lớn hơn bội phần. Để giảm tải một phần áp lực cho bộ y tế, Chính phủ và các bộ ngành liên quan ở nước ta đã chỉ thị việc áp dụng công nghệ cao vào việc chống dịch.

Công nghệ được ứng dụng nhằm ngăn chặn sự lây lan của virus, chăm sóc các bệnh nhân và giảm áp lực cho hệ thống y tế. Công nghệ hỗ trợ Chính phủ, các bộ, ngành, địa phương đánh giá chính xác được tình hình và kịp thời đưa ra quyết định phù hợp trong quá trình dập dịch. Như vậy thì công nghệ phải dễ dùng, thuận tiện và bảo vệ dữ liệu của người dùng.

Trong cuộc chiến chống COVID-19 thời gian qua cho thấy, việc ứng dụng công nghệ hiệu quả đã trở thành một trong những yếu tố giúp nhiều quốc gia có thể khống chế được dịch bệnh. Việc người dân, các tổ chức, chính quyền khai thác và sử dụng hiệu quả giải pháp công nghệ trong phòng, chống dịch bệnh sẽ giúp cho phạm vi khoanh vùng chính xác hơn, giảm bớt việc cách ly nhầm, cách ly trên diện rộng. Như vậy xã hội vẫn có thể duy trì hoạt động được bình thường, nhà máy, khu công nghiệp, các địa điểm kinh doanh không bị đứt gãy hoạt động sản xuất, giao thương, buôn bán.

Bản thân nhóm 1 cũng có một bạn đã từng bị nhiễm covid-19 nên nhận thức rõ được những khó khăn ấy thực tế hơn. 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 "quản lý bệnh nhân bị mắc covid 19". Với mong muốn ứng dụng này, đáp ứng được nhu cầu cấp thiết của việc quản lý bệnh nhân lúc này, góp một phần sức lực nhỏ hòa vào công cuộc chống dịch của cả nước. 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 quản lý bệnh nhân mắc covid là một chương trình dành để quản lý các bệnh nhân mắc covid mục đích để tránh sự thống kê không chính xác và đưa

ra các số liệu không có căn cứ, muốn làm được chương trình này trước tiên chúng ta phải sử dụng cấu trúc class để lưu trữ đối tượng nhiều thuộc tính, sau đó chúng ta sử dụng các cấu trúc dữ liệu, các thuật toán, cấu trúc lặp,... để quản lý chương trình.

Chương trình bọn em gồm có 12 chức năng chính, trong đó có 1 số chức năng nổi bật như là: sửa, tìm kiếm, xóa, thêm, thống kê bệnh nhân,... các cấu trúc dữ liệu được sử dụng là cây nhị phân tìm kiếm và danh sách liên kết đôi cùng với 2 thuật toán nổi bật sắp xếp nhanh và sắp xếp trộn và các kiến thức về lập trình hướng đối tượng như tính trừu tượng, đa hình, kế thừa, đóng gói. Lý do bọn em sử dụng cấu trúc dữ liệu cây nhị phân tìm kiếm và danh sách liên kết đôi, các thuật toán sắp xếp nhanh, sắp xếp trộn là thứ nhất là bọn em muốn ứng dụng từ những kiến thức đã học được ở môn cấu trúc dữ liệu và giải thuật, lập trình nâng cao, thứ hai là nó thích hợp với chương trình quản lý bọn em hướng tới, độ phức tạp thấp, dễ quản lý, dễ sử dụng.

3. Giao diện chương trình

1. Hình ảnh đăng nhập + hình ảnh sau khi bạn nhập sai:

```
##############################   LOGIN   ##############################
#============================================================#
Please enter password (5 characters): *****
------------------------------------------------------------
                       YOU STILL HAVE 3 ACCESS
##############################   LOGIN   ##############################
#============================================================#
Please enter password (5 characters): *****
------------------------------------------------------------
                       YOU STILL HAVE 2 ACCESS
##############################   LOGIN   ##############################
#============================================================#
Please enter password (5 characters): *****
------------------------------------------------------------
                       YOU STILL HAVE 1 ACCESS
##############################   LOGIN   ##############################
#============================================================#
Please enter password (5 characters):
```

2. Hình ảnh menu (sau khi đăng nhập thành công):

```
######################## MANAGEMENT PATIENTS ########################
#===================================================================#
######    Class: CNTT K61  BM CNTT - DH GTVT PH.TPHCM        ######
#===================================================================#
#===================================================================#
# 1-Edit patient                                  7-Check healthy   #
# 2-Statistics F by id                            8-Print file      #
# 3-Status statistic F0                           9-Print Patient   #
# 4-Sort by name                                  10-Remove Patient #
# 5-Patients have Q-Day > 21                      11-Add Patinet    #
# 6-Patients at Q-Place                           12-Find Patinet   #
######                      0-Exit program                      ######
#===================================================================#
#####################################################################
What is your choice:
```

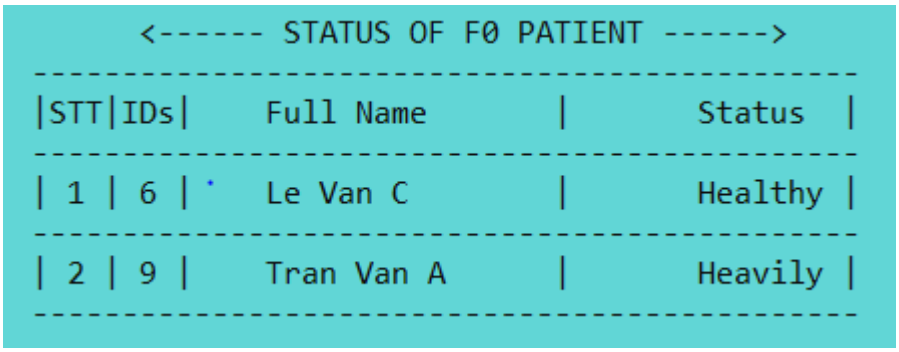
3. Hình ảnh chức năng 1 (find patient):

```
Enter id of patient need to find: 1
|STT|  ID  |      Full Name      |      Birth          |      Address        |      Status  |      Infect  |      Inject  |      Q_Place |      Q_Day  |
| 1 |  1   |      Nguyen Van D   |      21/12/1895     |      Quan 3         |      1       |      2       |      1       |      Quan 9  |      21     |
```

4. Hình ảnh chức năng 2(statistics F by id):

```
<-------- THE LIST OF F-PATIENTS BY ID -------->
        ---------------------------------
        |STT|IDs|   Full Name       | F |
        ---------------------------------
        | 1 | 4 |   Nguyen Van A    | 0 |
        ---------------------------------
        | 2 | 6 |   Le Van C        | 0 |
        ---------------------------------
        | 3 | 2 |   Tran Thi B      | 1 |
        ---------------------------------
        | 4 | 7 |   Tran Thi G      | 1 |
        ---------------------------------
        | 5 | 1 |   Nguyen Van D    | 2 |
        ---------------------------------
        | 6 | 3 |   Ngo Van E       | 3 |
        ---------------------------------
        | 7 | 5 |   Ly Thi F        | 4 |
        ---------------------------------
```

5. Hình ảnh chức năng 3 (status statistic F0):



```
        <------ STATUS OF F0 PATIENT ------>
-------------------------------------------------
|STT|IDs|     Full Name        |       Status  |
-------------------------------------------------
| 1 | 6 |    Le Van C          |       Healthy |
-------------------------------------------------
| 2 | 9 |    Tran Van A        |       Heavily |
-------------------------------------------------
```

6. Hình ảnh chức năng 4 (sort by name):

```
|STT|  ID   |      Full Name      |      Birth        |      Address      |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  9    |      Tran Van A     |      21/5/2002    |      Quan 9       |      2       |      NO      |      2       |      Quan 9   |      30     |
| 2 |  2    |      Tran Thi B     |      23/5/1994    |      Quan 9       |      2       |      4       |      0       |      Quan 9   |      42     |
| 3 |  6    |      Le Van C       |      2/1/1990     |      Quan 7       |      0       |      NO      |      1       |      Quan 4   |      35     |
| 4 |  1    |      Nguyen Van D   |      21/12/1895   |      Quan 3       |      1       |      2       |      1       |      Quan 9   |      21     |
| 5 |  3    |      Ngo Van E      |      16/2/2000    |      Quan 2       |      1       |      1       |      2       |      Quan 12  |      14     |
| 6 |  5    |      Ly Thi F       |      23/5/2002    |      Quan 12      |      2       |      3       |      2       |      Quan 12  |      14     |
| 7 |  7    |      Tran Thi G     |      30/6/1996    |      Quan 10      |      0       |      6       |      1       |      Quan 5   |      28     |
Press any key to continue . . .
```

7. Hình ảnh chức năng 5 (Patients have Q-Day > 21):

```
|STT|  ID   |      Full Name      |      Birth        |      Addrress     |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  2    |      Tran Thi B     |      23/5/1994    |      Quan 9       |      2       |      4       |      0       |      Quan 9   |      42     |
| 2 |  6    |      Le Van C       |      2/1/1990     |      Quan 7       |      0       |      NO      |      1       |      Quan 4   |      35     |
| 3 |  7    |      Tran Thi G     |      30/6/1996    |      Quan 10      |      0       |      6       |      1       |      Quan 5   |      28     |
| 4 |  9    |      Tran Van A     |      21/5/2002    |      Quan 9       |      2       |      NO      |      2       |      Quan 9   |      30     |
-->The number of people have quarantine day > 21: 4
Press any key to continue . . .
```

8. Hình ảnh chức năng 6 (Patients at Q-Place):

```
Place: Quan 9
|STT|  ID   |      Full Name      |      Birth        |      Addrress     |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  1    |      Nguyen Van D   |      21/12/1895   |      Quan 3       |      1       |      2       |      1       |      Quan 9   |      21     |
Place Quan 9 has 1 patients
Place: Quan 5
|STT|  ID   |      Full Name      |      Birth        |      Addrress     |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  7    |      Tran Thi G     |      30/6/1996    |      Quan 10      |      0       |      6       |      1       |      Quan 5   |      28     |
Place Quan 5 has 1 patients
Place: Quan 4
|STT|  ID   |      Full Name      |      Birth        |      Addrress     |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  6    |      Le Van C       |      2/1/1990     |      Quan 7       |      0       |      NO      |      1       |      Quan 4   |      35     |
Place Quan 4 has 1 patients
Place: Quan 12
|STT|  ID   |      Full Name      |      Birth        |      Addrress     |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day  |
| 1 |  5    |      Ly Thi F       |      23/5/2002    |      Quan 12      |      2       |      3       |      2       |      Quan 12  |      14     |
| 2 |  3    |      Ngo Van E      |      16/2/2000    |      Quan 2       |      1       |      1       |      2       |      Quan 12  |      14     |
Place Quan 12 has 2 patients
Press any key to continue . . .
```

9. Hình ảnh chức năng 7 (check healthy):

```
  <------ PATIENT HEALTHY LIST NOW ------>
-----------------------------------------------
|STT|IDs|    Full Name       |       Health  |
-----------------------------------------------
| 1 | 1 |    Nguyen Van D    |       Well    |
-----------------------------------------------
| 2 | 2 |    Tran Thi B      |       Bad     |
-----------------------------------------------
| 3 | 3 |    Ngo Van E       |       Well    |
-----------------------------------------------
| 4 | 5 |    Ly Thi F        |       Bad     |
-----------------------------------------------
| 5 | 6 |    Le Van C        |       Well    |
-----------------------------------------------
| 6 | 7 |    Tran Thi G      |       Well    |
-----------------------------------------------
| 7 | 9 |    Tran Van A      |       Bad     |
-----------------------------------------------
```

10. Hình ảnh chức năng 8 (print file):

```
Enter id of patient need to edit: 4
id: 9
Name: Tran Van A
-Birthday:
Day: 21
Month: 5
Year: 2002
Address: Quan 9
NOTE - input status: healthy=0, lightly=1, heavily =2
Status: 2
NOTE - input infection: 'NO' or id of before patients
Infection: NO
Injection: 2
Quarantine place: Quan 9
Day need to quarantine: 30
```

data.txt - Notepad

File Edit Format View Help

STT	ID	Full Name	Birth	Address	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	2	Tran Thi B	23/5/1994	Quan 9	2	4	0	Quan 9	42
3	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
4	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
5	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

11. Hình ảnh chức năng 9 (print patient):

STT	ID	Full Name	Birth	Address	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	2	Tran Thi B	23/5/1994	Quan 9	2	4	0	Quan 9	42
3	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
4	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
5	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

```
Press any key to continue . . .
```

12. Hình ảnh chức năng 10 (remove patient):

```
Enter id of patient need to remove: 2
Remove success!
```

STT	ID	Full Name	Birth	Address	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
3	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
4	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
5	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
6	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

```
Press any key to continue . . .
```

13. Hình ảnh chức năng 11 (add patient):

```
Enter information of patient need to add:
id: 6
Name: Tran Van C
-Birthday:
Day: 21
Month: 5
Year: 2002
Address: Quan 9
NOTE - input status: healthy=0, lightly=1, heavily =2
Status: 2
NOTE - input infection: 'NO' or id of before patients
Infection: NO
Injection: 2
Quarantine place: Quan 9
Day need to quarantine: 23
Add success!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
3	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
4	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
5	6	Tran Van C	21/5/2002	Quan 9	2	NO	2	Quan 9	23
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

```
Press any key to continue . . .
```

14. Hình ảnh chức năng 12 (edit patient):

```
Edit success!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	2	Tran Thi B	23/5/1994	Quan 9	2	4	0	Quan 9	42
3	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
4	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
5	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

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**.

Tính đóng gói - Data encapsulation là một kỹ thuật đóng gói dữ liệu, và các hàm mà sử dụng chúng và trừu tượng hóa dữ liệu là một kỹ thuật chỉ trưng bày tới các Interface và ẩn Implementation Detail (chi tiết trình triển khai) tới người sử dụng.

C++ hỗ trợ các thuộc tính của đóng gói và ẩn dữ liệu thông qua việc tạo các kiểu tự định nghĩa (user-defined), gọi là classes. 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.

Ví dụ:

```
class DList{
    private: //thuộc tính được che dấu

        DNode* head;
        DNode* tail;
        int size;
    public: //chỉ có những phương thức này được truy cập vào thuộc tính

        DList(){
            head = tail = NULL;
            size = 0;
        }
        ~DList(){}
        DNode* getHead();
        DNode* getTail();
        int getSize();
        void setSize(int size);
        DNode* CreateNode(Patient val);
        void print();
};
```

Các biến head, tail, và size là **private**. Nghĩa là chúng chỉ có thể được truy cập bởi các thành viên khác của lớp DList, và không thể bởi bất kỳ phần khác trong chương trình. Đây là một cách thực hiện tính đóng gói trong C++.

Để hàm ngoài truy cập vào hoặc thay đổi các thuộc tính private hay protected, ta có thể truy cập thông qua các hàm Setter hay Getter được xây dựng ở public.

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.

Ví dụ minh họa:

```
class Person{
    private:
        string id;
        string name;
        string address;
        int day;
        int month;
        int year;
    public:
        Person();
        ~Person(){}

};
```

Lớp Person về con người nói chung chỉ cần quan tâm đến id, tên, địa chỉ, ngày sinh mà không cần phải quản lý thêm về chiều cao, cân nặng, sở thích, tôn giáo …

4.3 Tính kế thừa

Một trong những khái niệm quan trọng nhất trong lập trình hướng đối tượng là **Tính kế thừa (Inheritance)**. Kế thừa trong C++ 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.

Tư tưởng của kế thừa trong C++ là có thể tạo ra một class mới được xây dựng trên các lớp đang tồn tại. 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**. Kiểu kế thừa trong C++ được xác định bởi Access-specifier đã được giải thíc ở trên.

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.

```
class Person{ //base class

    private:
        string id;
        string name;
        string address;
        int day;
        int month;
        int year;
    public:
        Person();
        ~Person(){}
        Person(string id, string name, string address, int day, int month, int ye
ar);
        Person(const Person& other);
};
```

```
class Patient:public Person {  //derived class

    private:
        int status;
        string infection;
        int injection;
        string place;
        int q_day;
    public:
        Patient(): Person(){
            this->status = 0;
            this->infection = "NO";
            this->place = "";
            this->q_day = 0;
        }
        ~Patient(){}
```

```
        Patient(string id, string  name, int day, int month, int year, string add
ress, int status, string infection, string place, int q_day): Person(id, name, ad
dress, day, month, year){
            this->status = status;
            this->infection = infection;
            this->place = place;
            this->q_day = q_day;
        }
        Patient(const Patient &other){
            *this = other;
        }
};
```

Lớp Patient được kế thừa public từ lớp cơ sở là Person. Như vậy, Patient có thể sử dụng tất cả phương thức của Person, trừ thuộc tính private không thể truy xuất.

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.

Ví dụ:

```
class Person{ //base class
    private:
        string id;
        string name;
        string address;
        int day, month, year;
    public:
        Person();
        ~Person(){}
        virtual void toStream(istream& input){

            cout<<"id: ";
            input>>std::ws;//skip whitespace
            input>>id;
            cout<<"Name: ";
            input>>std::ws;
            getline(input,name);
            cout<<"-Birthday: "<<endl;
                cout<<"Day: ";
                input>>day;
                cout<<"Month: ";
                input>>month;
                cout<<"Year: ";
                input>>year;
            cout<<"Address: ";
            input>>std::ws;
            getline(input, address);
        }

        friend istream& operator>>(istream&, Person &){

            person.toStream(input);
            return input;
        }
};
```

```
class Patient:public Person { //derived class
    private:
        int status;
        string infection;
        int injection;
        string place;
        int q_day;
    public:
        Patient(): Person();
```

```
        ~Patient(){}
        virtual void toStream(istream& input){

            Person::toStream(input);
                cout<<"Status: ";
                input>>status;
            cout<<"Infection: ";
            input>>infection;
                cout<<"Injection: ";
                input>>injection;
            cout<<"Quarantine place: ";
            getline(input,place);
            cout<<"Day need to quarantine: ";
            input>>q_day;
        }
};
```

Cụ thể trong bài, nhóm em cho đa hình hàm nhập bằng nạp chồng toán tử ở lớp cơ sở Person. Vì hàm friend không thể đa hình nên ta sử dụng thêm một hàm khác làm hàm ảo, ở đây là hàm toStream(), hàm này sẽ được định nghĩa lại ở lớp Patient và hàm friend ở lớp Person sẽ gọi tới nó để thực thi.

5. Nội dung lý thuyết

5.1 Cây nhị phân tìm kiếm

5.1.1 Duyệt trước (preOrder)

Quy trình duyệt PreOrder sẽ thực hiện theo thứ tự Node -> Left -> Right, cụ thể như sau:

1. Ghé thăm Node root
2. Gọi đệ quy duyệt qua cây con bên trái
3. Gọi đệ quy duyệt qua cây con bên phải

Ví dụ:

```
DList BST::preOrder(Node* root){
 if(root != NULL){
change.Add(root->data);
preOrder(root->left);
preOrder(root->right);
}
return change;
}
```

5.1.2 Duyệt giữa (inOrder)

Quy trình duyệt inOrder sẽ thực hiện theo thứ tự Left ->Node -> Right, cụ thể như sau:

1. Gọi đệ quy duyệt qua cây con bên trái
2. Ghé thăm Node root
3. Gọi đệ quy duyệt qua cây con bên phải

Ví dụ:

```
DList BST::inOrder(Node* root){
 if(root != NULL){
inOrder(root->left);
change.push(root->data);
 inOrder(root->right);
}
return change;
 }
```

5.1.3 Duyệt sau (posOrder)

Quy trình duyệt posOrder sẽ thực hiện theo thứ tự Left -> Right -> Node, cụ thể như sau:

1. Gọi đệ quy duyệt qua cây con bên trái
2. Gọi đệ quy duyệt qua cây con bên phải
3. Ghé thăm Node root

Ví dụ:

```
DList BST::posOrder(Node* root){
 if(root != NULL){
posOrder(root->left);
posOrder(root->right);
change.push(root->data);
 }
return change;
}
```

5.1.4 Thêm một Node vào cây (add)

Việc thêm 1 phần tử vào cây nhị phân tìm kiếm vẫn phải đảm bảo được các ràng buộc của một BST đã trình bày. Vì vậy, bạn cần phải tìm kiếm vị trí thích hợp trong BST để lưu giữ nó. Nếu bạn để ý, bạn sẽ nhận ra vị trí của các root được thêm vào sẽ luôn Node lá(không có child nào hết). Như vậy, tại vị trí đó trước khi các Node mới tới ở thì nó là NULL. Ta có quy trình như sau:

1. Nếu Node hiện tại = NULL, đó là vị trí cần thêm. Thêm vào BST và kết thúc.

2. Nếu giá trị cần thêm < giá trị root hiện tại, gọi đệ quy add vào cây con bên trái.

3. Nếu giá trị cần thêm > giá trị root hiện tại, gọi đệ quy add vào cây con bên phải.

Ví dụ:

```
bool compare(string str1, string str2){ //hàm ngoài để so sánh id
int n=str1.length();
int m=str2.length();
if(n==m) return (str1<str2);
return n<m;
}
Node* BST::add(Node* root, Patient val){
 if(root == NULL) return new Node(val);
if(compare(val.getId(),root->data.getId()))
root->left = add(root->left, val);
else root->right = add(root->right, val);
return root;
}
```

5.1.5 Xóa một Node trong cây nhị phân (erase)

Để bảo toàn phép loại bỏ node trên cây nhị phân tìm kiếm ta cần xem xét đầy đủ các trường hợp sau:

- Trường hợp 1: nếu node loại bỏ là node lá thì sau khi loại bỏ node ta vẫn nhận được một cây nhị phân tìm kiếm. Trong trường hợp này ta chỉ việc liên kết node cha của p với nhánh cây con phải của p.
- Trường hợp 2: nếu node loại bỏ chỉ có cây con trái hoặc cây con phải. Trong trường hợp này ta chỉ việc liên kết node cha của p với nhánh cây con trái của p hoặc phải của p.
- Trường hợp 3: nếu node loại bỏ có cả hai cây con, ta cần dùng thêm node thế thân và xóa đi node đó.

Ví dụ:

```
// so sánh id để lấy giá trị bên trái
bool BST::leftOf(Patient val, Node* root ){
    return compare(val.getId(),root->data.getId());
}
// so sánh id để lấy giá trị bên phải
bool BST::rightOf(Patient val, Node* root ){
    return compare(root->data.getId(),val.getId());
}

//tìm node có giá trị nhỏ nhất
Patient BST::leftMostValue( const Node* root ){
    while(root->left != NULL)
        root = root->left;
    return root->data;
}
Node* BST::erase(Node* root, Patient val){
//nếu cây rỗng không có gì để loại bỏ.
    if(root == NULL) return root;
    if(leftOf(val,root))  //nếu node cần loại có giá trị nhỏ hơn node gốc
        root->left=erase(root->left,val);  //tìm sang cây con trái để loại
    else if(rightOf(val,root))  //nếu node cần loại có giá trị lớn hơn node gốc

       root->right=erase(root->right,val);  //tìm sang cây con phải để loại
    else {  //Chú ý chỗ này: nếu tìm thấy node có giá trị value cần loại bỏ
        if(root->left ==NULL){  //nếu node cần loại chỉ có cây con phải
            Node* newRoot=root->right;
            delete root;
            return newRoot;
        }
        if(root->right == NULL){  //nếu node cần loại chỉ có cây con trái
            Node* newRoot = root->left;
            delete root;
            return newRoot;
        }
// trường hợp node có cả hai cây con
```

```
//ta lấy node trái nhất của cây con phải
        root->data = leftMostValue(root-
>right); //thay thế nội dung node cần loại bỏ
        root->right = erase(root->right, root->data); //loại bỏ node thay thế
    }
    return root;
}
```

5.1.6 Tìm kiếm trong cây nhị phân (search)

Phép tìm node có giá trị value trên cây tìm kiếm được thực hiện như sau:

- Nếu node cần tìm có giá trị lớn hơn nội dung node gốc thì ta tìm sang cây con bên phải.
- Nếu node cần tìm có giá trị bé hơn nội dung node gốc thì ta tìm sang cây con bên trái.
- Đưa ra kết luận tìm thấy node hay không tìm thấy node.

```
Node* BST::search(Node* root, Patient val){
    if (root == NULL)
        return NULL;
//nếu tìm thấy node, thì trả về node
    if(root->data.getId() == val.getId()){
        return root;
    }
//nếu node có giá trị lớn hơn value
    else if (leftOf(val, root)){
        return search(root->left, val); //tìm ở cây con bên trái
    }
//nếu node có giá trị bé hơn value
    else if(rightOf(val, root)){
        return search(root->right, val); //tìm ở cây con bên phải
    }       }
```

5.1.7 Sửa dữ liệu trên cây nhị phân (edit)

Để không ảnh hưởng đến quy luật cây nhị phân, nhóm em đã sử dụng lại các chức năng cơ bản của cây như search, add và earse.

Đầu tiên cho tìm kiếm Node cần xóa, nếu Node có tồn tại trên cây sẽ sử dụng hàm xóa một Node của cây xóa đi Node đó. Sau đó cho nhập dữ liệu mới và thêm lại vào cây.

Ví dụ: Sửa thông tin một bệnh nhân trong cây nhị phân

```
void BST::edit(Node* root){
    Patient find;
    string id_fix;
    cout<<"Enter id of patient need to edit: ";
    cin>>id_fix;
    find.setId(id_fix);
    if(search(root, find)==NULL){
        cout<<"No such patient has been found !!"<<endl;
        return;
    }else{
        erase(root,find);
        size--;
        do{
            cin>>find;
            if(existPatient(find.getId())){
                cout<<"Same id, enter again!!"<<endl;
            }
            if(!validInfection(find.getInfect())){
                cout<<"Invalid infection!!"<<endl;
            }
        }while((existPatient(find.getId())) || (!validInfection(find.getInfect())
));
        root = add(root,find);
    }
    cout<<"Edit success!"<<endl;
}
```

5.1.8 Giải phóng cây nhị phân (free)

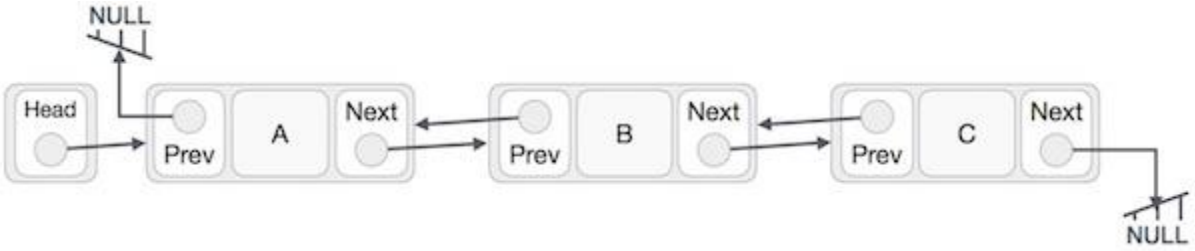
- Thao tác loại bỏ cây con gốc root: nguyên tắc loại bỏ cây con gốc root được thực hiện bằng cách giải phóng từng node lá trên cây cho đến khi giải phóng đến node gốc. Thao tác được tiến hành như sau:

```
void BST::Free( Node* root ){
    //nếu cây không còn cây con thì thoát chương trình.
    if(root == NULL) return;
    Free(root->left); //loại bỏ cây con bên trái
    Free(root->right);//loại bỏ cây con bên phải
    delete root; //giải phóng node
    size = 0;
}
```

5.2 Danh sách liên kết đôi

Danh sách liên kết đôi (**Doubly linked list**) là danh sách liên kết mà mỗi phần tử có hai liên kết đến phần tử liền trước và liền sau nó. Khi duyệt các nút sẽ thực hiện

theo hai chiều về trước và về sau thay vì thực hiện duyệt một chiều như danh sách liên kết đơn.



Danh sách liên kết đôi có các thông số cần quan tâm:

- Giá trị (data).
- Mối liên kết tới Node khác (pPrev và pNext).

Ở mối liên kết tới Node khác, thay vì chỉ có mỗi pNext trỏ tới phần tử sau nó như DSLK đơn, thì trong DSLK đôi cần có thêm pPrev để trỏ tới phần tử trước nó.

Vậy danh sách liên kết đôi có gì khác so với danh sách liên kết đơn

Mỗi phần tử trong danh sách liên kết đơn chứa một tham chiếu đến phần tử tiếp theo trong danh sách, trong khi mỗi phần tử trong danh sách liên kết đôi chứa các tham chiếu đến phần tử tiếp theo cũng như phần tử trước đó trong danh sách. Danh sách liên kết đôi yêu cầu nhiều không gian hơn cho mỗi phần tử trong danh sách và các thao tác cơ bản như chèn và xóa phức tạp hơn vì chúng phải xử lý hai tham chiếu. Nhưng danh sách liên kết đôi cho phép thao tác dễ dàng hơn vì nó cho phép duyệt qua danh sách theo hướng tiến và lùi.

Cụ thể bài nhóm em sử dụng danh sách liên kết đôi với 3 chức năng là thêm vào đầu, thêm vào cuối và xóa danh sách.

5.2.1 Thêm vào đầu danh sách liên kết đôi

Ví dụ:

```
class DNode {
    public:
        Patient data;
        DNode* prev;
        DNode* next;
        DNode(Patient data){
            this->data = data;
            prev = NULL;
            next = NULL;
```

```
        }
};
class DList{
    private:
        DNode* head;
        DNode* tail;
        int size;
    public:
        DList(){
            head = tail = NULL;
            size = 0;
        }
};
void DList::push(Patient val){//thêm đầu
    DNode* p = new DNode(val);
    if(size == 0){
        head = tail = p;
    }
    else{
        head->prev = p;
        p->next = head;
        head = p;
    }
    size++;
}
```

Đầu tiên khai báo Node p. Nếu size hiện tại của danh sách lên kết = 0 (chưa có phần tử nào) thì cho head và tail = p. Ngược lại nếu danh sách đã có phần tử, ta sẽ gán p cho previous của head, rồi gán head cho next của p, cuối cùng cho head = p và tăng size của danh sách lên.

5.2.2 Thêm vào cuối danh sách liên kết đôi

Ví dụ:

```
void DList::Add(Patient val){//thêm cuối
    DNode* p = new DNode(val);
    if(size == 0) {
        head = tail = p;
    }else{
        tail->next = p;
        p->prev = tail;
        tail = p;
    }
    size++;
}
```

Đầu tiên ta cũng khai báo Node p. Tương tự như thêm vào đầu, ta cũng kiểm tra xem nếu size = 0, ta gán head và tail = p. Ngược lại, ta gán p cho next của tail, rồi gán tail cho pevious của p, sau đó cho tail = p và tăng size của danh sách.

5.2.3 Xóa danh sách liên kết đôi

Ví dụ:

```
void DList::setSize(int size){
    this->size = size;
}
void DList::Delete(){
    DNode* k = NULL;
    while(head != NULL){
        k = head;
        head = head->next;
        delete k;
    }
    setSize(0);
}
```

Đầu tiên khai báo Node k = NULL. Cho chạy vòng lặp với điều kiện head không bằng NULL, ta gán head cho k, rồi gán head->next cho head, sau đó xóa k đi. Cuối cùng ta set size của danh sách = 0.

5.3 Thuật toán sắp xếp

Có nhiều loại sắp xếp tiêu biểu hiện nay như Bubble sort, Insertion sort, Selection sort, … Nhưng để chương trình được diễn ra nhanh hơn, nhóm em quyết định sử dụng 2 loại sắp xếp có độ phức tạp thấp đó là Quick sort & Merge sort với độ phức tạp trung bình O(nlogn) và được sử dụng trong danh sách liên kết đôi.

5.3.1 Sắp xếp nhanh (Quick sort)

Thuật toán sắp xếp quick sort là một thuật toán chia để trị (Divide and Conquer algorithm). Nó chọn một phần tử trong danh sách làm điểm đánh dấu(pivot). Thuật toán sẽ thực hiện chia danh sách đó thành các danh sách nhỏ hơn dựa vào pivot đã chọn. Việc lựa chọn pivot ảnh hưởng rất nhiều tới tốc độ sắp xếp. Nhưng máy tính lại không thể biết khi nào thì nên chọn theo cách nào. Và một số cách để chọn pivot thường được sử dụng:

1. Luôn chọn phần tử đầu tiên của danh sách.
2. Luôn chọn phần tử cuối cùng của danh sách. (Được sử dụng trong bài)
3. Chọn một phần tử random.

4. Chọn một phần tử có giá trị nằm giữa danh sách. (median element).

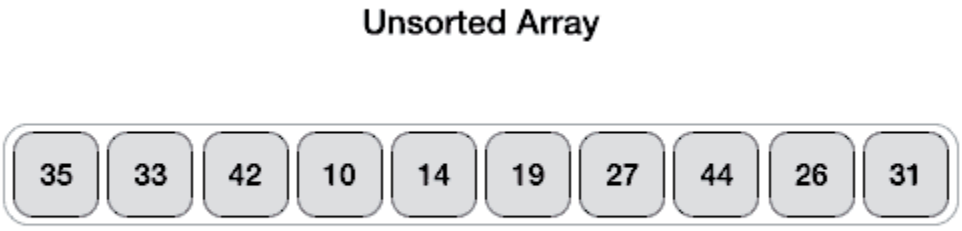
Vì để thuận tiện và phù hợp nên nhóm em đã sử dụng "Luôn chọn phần tử cuối cùng của danh sách"

Mấu chốt chính của thuật toán quick sort là việc phân đoạn (Hàm partition()). Mục tiêu của công việc này là: Cho một danh sách và một phần tử x là pivot. Đặt x vào đúng vị trí của mảng đã sắp xếp. Di chuyển tất cả các phần tử của danh sách mà nhỏ hơn x sang bên trái vị trí của x, và di chuyển tất cả các phần tử của danh sách mà lớn hơn x sang bên phải vị trí của x.

Khi đó ta sẽ có 2 danh sách con: danh sách bên trái của x và bên phải của x. Tiếp tục công việc với mỗi danh sách con(chọn pivot, phân đoạn) cho tới khi danh sách được sắp xếp.

Thuật toán phân đoạn

Đặt pivot là phần tử cuối cùng của danh sách. Ta bắt đầu từ phần tử trái nhất của dãy số có chỉ số là left, và phần tử phải nhất của dãy số có chỉ số là right -1(bỏ qua phần tử pivot). Chừng nào left < right mà left->data > pivot và right->data < pivot thì đổi chỗ hai phần tử left và right. Sau cùng, ta đổi chỗ hai phần tử left và pivot cho nhau. Khi đó, phần tử left đã đứng đúng vị trí và chia danh sách làm đôi (bên trái và bên phải).



Ví dụ: Sắp xếp tên sử dụng Quick sort

```
DNode* partition(DNode* left, DNode* right){//hàm ngoài
    Patient x = right->data;//pivot ở đuôi
    DNode* i = left->prev;
    DNode* j = left;
    while(j != right){
        if(in_name(j->data.getName()) <= in_name(x.getName())){
            i = (i == NULL)? left : i->next;//i++
            swap(i, j);
        }
        j = j->next;
    }
    i = (i == NULL)? left : i->next;//i++
    swap(i, right);//pivot ở giữa (phía trước < pivot và sau > pivot)
    return i;//trả về pivot
}
void quicksort(DNode* left, DNode* right){//hàm ngoài
    if(right != NULL && left != right && left != right->next){
        DNode* p = partition(left, right);
        quicksort(left, p->prev);
        quicksort(p->next, right);
    }
}
void DList::SortByName(){
    quicksort(head, tail);
}
```

Thuật toán quick sort trong trường hợp trung bình và tốt nhất có độ phức tạp là O(nlogn). Tuy nhiên nếu dãy đã được sắp xếp và pivot được chọn ở đầu dãy thì lúc này phân đoạn sẽ không cân bằng, khi đó rơi vào trường hợp xấu nhất của thuật toán này với độ phức tạp là O(n^2). Nhưng việc đó cũng ít khi xảy ra và ta được không gian bộ nhớ sử dụng nhỏ là O(logn). Vậy nhóm em quyết định vận dụng thêm một thuật toán sắp xếp cũng khá nhanh như Quick sort mà không bị chậm khi trong trường hợp xấu nhất đó là Merge sort sẽ được nói trong phần sau.

5.3.2 Sắp xếp trộn (Merge sort)

Giống như Quick sort, Merge sort là một thuật toán chia để trị. Thuật toán này chia danh sách cần sắp xếp thành 2 nửa. Tiếp tục lặp lại việc này ở các nửa danh sách đã chia. Sau cùng gộp các nửa đó thành mảng đã sắp xếp. Hàm split() làm nhiệm vụ tách danh sách ra thành 2 nửa. Hàm merge() được sử dụng để gộp hai nửa mảng và sắp xếp.

Cách thuật toán hoạt động:

6 5 3 1 8 7 2 4

Thuật toán tách danh sách (split())

Gọi hai con trỏ fast và slow ở head, fast sẽ duyệt qua 2 phần tử còn slow sẽ duyệt chậm hơn là từng phần tử. Cho chạy vòng lặp while với điều kiện fast→next và fast→next→next không NULL, đến cuối danh sách ta sẽ có slow→next là phần tử cần tìm (phần tử ở giữa tách danh sách thành 2 danh sách con).

Thuật toán trộn danh sách (merge())

Lần lượt truyền vào 2 danh sách cần trộn cho first và second. Nếu first→data < second→data ta sẽ gọi đệ quy hàm merge cho first→next rồi gán first lại cho pre của first→next (vì khi trước khi gộp sẽ có 2 danh sách khác nhau, cần phải gán lại) và trả về first. Trường hợp ngược lại first→data >second→data, ta cũng làm tương tự cho second→next và trả về second.

Ví dụ: sắp xếp nơi ở theo Merge sort

```
DNode* split(DNode* head){//hàm ngoài (tách list thành 2 nửa)
    DNode *fast = head,*slow = head;
    while (fast->next && fast->next->next)
    {
        fast = fast->next->next;
        slow = slow->next;
    }
    DNode *temp = slow->next;
    slow->next = NULL;
    return temp;
}
DNode* merge(DNode* first, DNode* second){//hàm ngoài (trộn 2 list)
    //Nếu list thứ nhất empty
    if (!first)
        return second;
    //Nếu list thứ 2 empty
    if (!second)
```

```
        return first;
    //Lấy giá trị nhỏ nhất
    if (first->data.getPlace() < second->data.getPlace()){
        first->next = merge(first->next,second);
        first->next->prev = first;
        first->prev = NULL;
        return first;
    }
    else{
        second->next = merge(first,second->next);
        second->next->prev = second;
        second->prev = NULL;
        return second;
    }
}
DNode* mergeSort(DNode *head){//hàm ngoài (thực hiện merge sort)
    if (!head || !head->next)
        return head;
    DNode *second = split(head);
    //đệ quy cho 2 nửa
    head = mergeSort(head);
    second = mergeSort(second);
    //trộn lại
    return merge(head,second);
}
void DList::SortByPlace(){
    head = mergeSort(head);
}
```

Đối với thuật toán Merge sort thì trong trường hợp nào độ phức tạp cũng O(nlogn) dù là trường hợp xấu nhất, ta có thể thấy thuật toán này ổn định hơn so với Quick sort (O(n^2) với trường hợp xấu). Tuy nhiên nó tốn bộ nhớ hơn Quick sort với không gian bộ nhớ sử dụng là O(n) (ta cũng có thể thấy qua ví dụ).

5.4 Chức năng thống kê

Phần này sẽ nói về chức năng "thống kê F" được sử dụng trong bài.

- Đầu tiên ta cho duyệt cây để đẩy vào một danh sách (cụ thể là danh sách liên kết đôi), mặc định phần tử phải có Infection là "NO" (tương đương F0). Ta cần thêm một mảng để chứa các số (như F1 F2 F3…).

- Gọi Node p và gán ở đầu danh sách liên kết và i = 0. Cho chạy vòng lặp while với điều kiện p không NULL và i < size. Với phần tử đầu tiên sẽ có Infection là

“NO”, ta gán arr[i] = 0. Gọi thêm biến j = 0 và Node q cũng ở đầu danh sách, tiếp tục chạy vòng lặp while ở trong với điều kiện q != p và j<i, nếu Infection của p == với Id của q, ta sẽ cho arr[i] = arr[j]+1 và break ra khỏi vòng while này, rồi cho tăng j và q = q->next, lặp lại vòng while lớn đến khi sai điều kiện.

- Hàm sortId là để sắp xếp lại theo id của bệnh nhân tăng dần. Sau đó gán lại p cho head và xuất danh sách với arr[i] ra màn hình.

```
void BST::F(Node* root){
    change.Delete();
    DList d = preOrder(root);
    int arr[size];
    int i = 0, stt = 1;
    DNode* p = d.getHead();
    while(p != NULL && i<size){
        if(p->data.getInfect() == "NO"){
        arr[i] = 0;
        }else{
            int j = 0;
            DNode* q = d.getHead();
            while(q != p && j<i){
                if(p->data.getInfect() == q->data.getId()){
                    arr[i] = arr[j] + 1;
                    break;
                }
                j++;
                q = q->next;
            }
        }
        i++;
        p = p->next;
    }
    sortId(d,arr);
    i = 0;
    p = d.getHead();
    while(p!=NULL && i<size){
        cout<< p->data.getId()<<" |\t"<<p->data.getName()<<"\t| "<<arr[i]<< endl;
        i++;
        stt++;
        p = p->next;
    }
}
```

6. Kết luận

6.1 Kết quả đạt được

- Nhóm chúng em đã hoàn thành xong chương trình quản lý bệnh nhân mắc covid với các chức năng cơ bản khác nhau như: thêm, sửa, xóa, tìm kiếm, lưu file, xuất file,...

6.2 Nhược điểm

- Chưa hiểu sâu về lý thuyết của cây nhị phân tìm kiếm và danh sách liên kết đôi 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 chương trình quản lý bệnh nhân mắc covid với nhiều chức năng hơn, ứng dụng vào thực tế nhiều hơn, thỏa mãn người sử dụng trong xã hội.

6.4 Tài liệu tham khảo

- stackoverflow, nguyenvanhieu.vn, geeksforgeeks, giáo trình một số trường đại học khác tiêu biểu là giáo trình cấu trúc dữ liệu và giải thuật của Học viện Công nghệ Bưu chính Viễn thông, giải thuật và lập trình (Lê Minh Hoàng),…

Phụ Lục:

Hướng dẫn sử dụng

Bước 1: Đăng nhập bằng mật khẩu **12345**, nếu người dùng đăng nhập sai thì còn lại 3 lần xác minh mật khẩu. Nếu hết lần đăng nhập mà người dùng không điền đúng thì chương trình sẽ tự động thoát.

```
############################   LOGIN   ############################
#===============================================================#
Please enter password (5 characters): *****
---------------------------------------------------------------
                      YOU STILL HAVE 3 ACCESS
############################   LOGIN   ############################
#===============================================================#
Please enter password (5 characters): *****
---------------------------------------------------------------
                      YOU STILL HAVE 2 ACCESS
############################   LOGIN   ############################
#===============================================================#
Please enter password (5 characters): *****
---------------------------------------------------------------
                      YOU STILL HAVE 1 ACCESS
############################   LOGIN   ############################
#===============================================================#
Please enter password (5 characters):
```

Bước 2: Khi người dùng đã đăng nhập thành công, thì sẽ thấy một bản menu gồm 12 chức năng, các chức năng được đánh dấu từ 1->12. Để lựa chọn các chức năng, người dùng chỉ cần nhập các số tương ứng với các chức năng, nếu nhập sai hoặc không hợp lệ thì sẽ in ra thông báo "`Choice is non-valid.`"

```
####################### MANAGEMENT PATIENTS ######################
#================================================================#
######    Class: CNTT K61  BM CNTT - DH GTVT PH.TPHCM       ######
#================================================================#
#================================================================#
# 1-Edit patient                              7-Check healthy   #
# 2-Statistics F by id                        8-Print file      #
# 3-Status statistic F0                       9-Print Patient   #
# 4-Sort by name                              10-Remove Patient #
# 5-Patients have Q-Day > 21                  11-Add Patinet    #
# 6-Patients at Q-Place                       12-Find Patinet   #
######                    0-Exit program                    ######
#================================================================#
##################################################################
What is your choice:
```

Chức năng 1: Nhập ID của một bệnh nhân để in ra toàn bộ thông tin của người bệnh đó.

```
Enter id of patient need to find: 1
|STT|  ID   |      Full Name      |      Birth          |      Addrress       |      Status |      Infect |      Inject |      Q_Place |      Q_Day  |
| 1 |   1   |      Nguyen Van D   |      21/12/1895     |      Quan 3         |      1      |      2      |      1      |      Quan 9  |      21     |
```

Nếu mà ID không hợp lệ thì sẽ in ra thông báo "`No such patient has been found !!`" và bắt người dùng nhập lại.

```
Enter id of patient need to find: 100
No such patient has been found !!
Enter id of patient need to find:
```

Chức năng 2: thống kê F của bệnh nhân theo ID, giúp truy vết nguồn lây.

```
<-------- THE LIST OF F-PATIENTS BY ID -------->
      ---------------------------------
      |STT|IDs|   Full Name       | F |
      ---------------------------------
      | 1 | 4 |   Nguyen Van A    | 0 |
      ---------------------------------
      | 2 | 6 |   Le Van C        | 0 |
      ---------------------------------
      | 3 | 2 |   Tran Thi B      | 1 |
      ---------------------------------
      | 4 | 7 |   Tran Thi G      | 1 |
      ---------------------------------
      | 5 | 1 |   Nguyen Van D    | 2 |
      ---------------------------------
      | 6 | 3 |   Ngo Van E       | 3 |
      ---------------------------------
      | 7 | 5 |   Ly Thi F        | 4 |
      ---------------------------------
```

Chức năng 3: thống kê tình trạng sức khỏe của bệnh nhân F0.

```
      <------ STATUS OF F0 PATIENT ------>
 ------------------------------------------------
 |STT|IDs|    Full Name       |       Status  |
 ------------------------------------------------
 | 1 | 6 |   Le Van C         |       Healthy |
 ------------------------------------------------
 | 2 | 9 |    Tran Van A      |       Heavily |
 ------------------------------------------------
```

Chức năng 4: sắp xếp danh sách bệnh nhân theo tên bệnh nhân (từ A->Z).

```
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  9     |      Tran Van A     |      21/5/2002      |      Quan 9         |      2       |      NO      |      2       |      Quan 9   |      30      |
| 2 |  2     |      Tran Thi B     |      23/5/1994      |      Quan 9         |      2       |      4       |      0       |      Quan 9   |      42      |
| 3 |  6     |      Le Van C       |      2/1/1990       |      Quan 7         |      0       |      NO      |      1       |      Quan 4   |      35      |
| 4 |  1     |      Nguyen Van D   |      21/12/1895     |      Quan 3         |      1       |      2       |      1       |      Quan 9   |      21      |
| 5 |  3     |      Ngo Van E      |      16/2/2000      |      Quan 2         |      1       |      1       |      2       |      Quan 12  |      14      |
| 6 |  5     |      Ly Thi F       |      23/5/2002      |      Quan 12        |      2       |      3       |      2       |      Quan 12  |      14      |
| 7 |  7     |      Tran Thi G     |      30/6/1996      |      Quan 10        |      0       |      6       |      1       |      Quan 5   |      28      |
Press any key to continue . . .
```

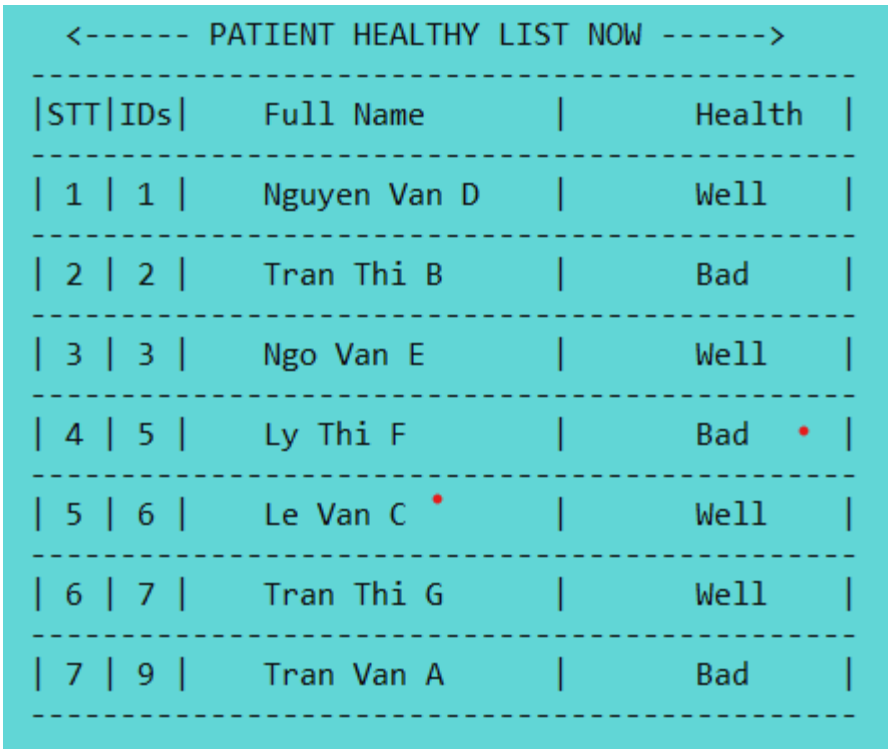
Chức năng 5: thống kê số bệnh nhân có trên 21 ngày cách ly.

```
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  2     |      Tran Thi B     |      23/5/1994      |      Quan 9         |      2       |      4       |      0       |      Quan 9   |      42      |
| 2 |  6     |      Le Van C       |      2/1/1990       |      Quan 7         |      0       |      NO      |      1       |      Quan 4   |      35      |
| 3 |  7     |      Tran Thi G     |      30/6/1996      |      Quan 10        |      0       |      6       |      1       |      Quan 5   |      28      |
| 4 |  9     |      Tran Van A     |      21/5/2002      |      Quan 9         |      2       |      NO      |      2       |      Quan 9   |      30      |
-->The number of people have quarantine day > 21: 4
Press any key to continue . . .
```

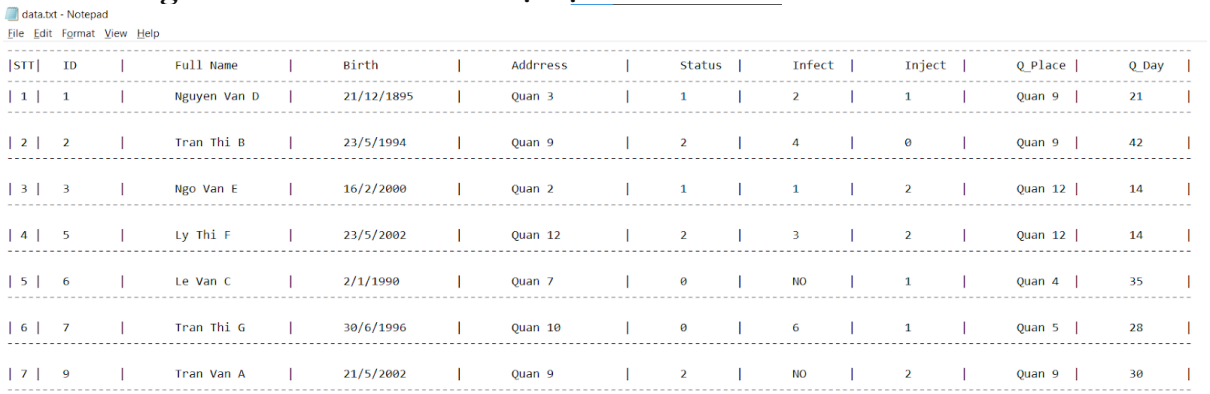
Chức năng 6: thống kê số bệnh nhân ở các khu cách ly tập trung.

```
Place: Quan 9
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  1     |      Nguyen Van D   |      21/12/1895     |      Quan 3         |      1       |      2       |      1       |      Quan 9   |      21      |
Place Quan 9 has 1 patients
Place: Quan 5
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  7     |      Tran Thi G     |      30/6/1996      |      Quan 10        |      0       |      6       |      1       |      Quan 5   |      28      |
Place Quan 5 has 1 patients
Place: Quan 4
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  6     |      Le Van C       |      2/1/1990       |      Quan 7         |      0       |      NO      |      1       |      Quan 4   |      35      |
Place Quan 4 has 1 patients
Place: Quan 12
|STT|  ID    |      Full Name      |      Birth          |      Addrress       |      Status  |      Infect  |      Inject  |      Q_Place  |      Q_Day   |
| 1 |  5     |      Ly Thi F       |      23/5/2002      |      Quan 12        |      2       |      3       |      2       |      Quan 12  |      14      |
| 2 |  3     |      Ngo Van E      |      16/2/2000      |      Quan 2         |      1       |      1       |      2       |      Quan 12  |      14      |
Place Quan 12 has 2 patients
Press any key to continue . . .
```

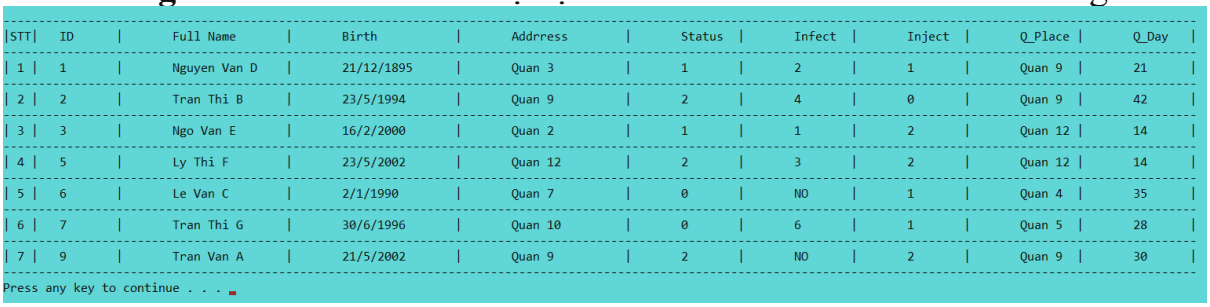
Chức năng 7: kiểm tra sức khỏe hiện tại của các bệnh nhân, để đưa ra phương án điều trị sớm nhất cho những người không khỏe và ngược lại cho những người khỏe mạnh có cơ hội về sớm để cách ly tại nhà.



Chức năng 8: In danh sách toàn bộ bệnh nhân nhiễm covid ra file data.txt.



Chức năng 9: In danh sách toàn bộ bệnh nhân nhiễm covid ra màn hình người dùng.



Chức năng 10: Nhập ID của một bệnh nhân, mà người dùng cần xóa. Sau khi xóa thành công sẽ in ra thông báo "`Remove success!` " và in ra danh sách bệnh nhân mới

ra màn hình. Nếu ID không hợp lệ thì sẽ in ra thông báo "`No such patient has been found !!`".

```
Enter id of patient need to remove: 2
Remove success!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
3	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
4	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
5	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
6	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

```
Press any key to continue . . .
```

Chức năng 11: Thêm một bệnh nhân vào danh sách quản lý. In ra thông báo "`Add success!`", nếu thêm thành công. In danh sách mới ra màn hình người dùng.

```
Enter information of patient need to add:
id: 6
Name: Tran Van C
-Birthday:
Day: 21
Month: 5
Year: 2002
Address: Quan 9
NOTE - input status: healthy=0, lightly=1, heavily =2
Status: 2
NOTE - input infection: 'NO' or id of before patients
Infection: NO
Injection: 2
Quarantine place: Quan 9
Day need to quarantine: 23
Add success!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
3	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
4	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
5	6	Tran Van C	21/5/2002	Quan 9	2	NO	2	Quan 9	23
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

```
Press any key to continue . . .
```

Chức năng 12: yêu cầu nhập đúng ID của bệnh nhân cần được chỉnh sửa. Nếu không nhập đúng sẽ in ra thông báo "No such patient has been found !!" và in ra danh sách bệnh nhân hiện tại để người dùng kiểm tra lại id của mình.

```
Enter id of patient need to edit: 100
No such patient has been found !!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	2	Tran Thi B	23/5/1994	Quan 9	2	4	0	Quan 9	42
3	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
4	4	Nguyen Van A	12/4/1996	Quan 5	1	NO	0	Quan 9	35
5	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
6	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
7	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28

```
Press any key to continue . . .
```

Nếu nhập đúng id bệnh nhân, chương trình sẽ cho người dùng chỉnh sửa thông tin của bệnh nhân đó:

```
Enter id of patient need to edit: 4
id: 9
Name: Tran Van A
-Birthday:
Day: 21
Month: 5
Year: 2002
Address: Quan 9
NOTE - input status: healthy=0, lightly=1, heavily =2
Status: 2
NOTE - input infection: 'NO' or id of before patients
Infection: NO
Injection: 2
Quarantine place: Quan 9
Day need to quarantine: 30
```

Lưu ý: + không được nhập id trùng.
+ điền "injection" theo đúng như phần "note" đã ghi rõ.
+ không được điền "injection" lớn hơn 2.

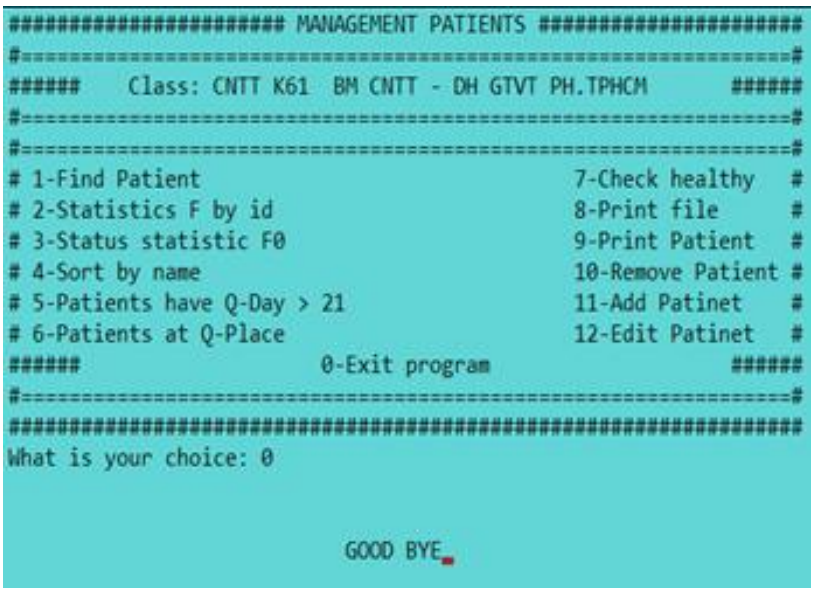
⇒ Nếu sai một trong 3 trường hợp trên, người dùng sẽ được yêu cầu nhập lại.

Khi người dùng chỉnh sửa một bệnh nhân xong, sẽ in ra thông báo "Edit success!" và in ra màn hình danh sách mới cho người dùng.

```
Edit success!
```

STT	ID	Full Name	Birth	Addrress	Status	Infect	Inject	Q_Place	Q_Day
1	1	Nguyen Van D	21/12/1895	Quan 3	1	2	1	Quan 9	21
2	2	Tran Thi B	23/5/1994	Quan 9	2	4	0	Quan 9	42
3	3	Ngo Van E	16/2/2000	Quan 2	1	1	2	Quan 12	14
4	5	Ly Thi F	23/5/2002	Quan 12	2	3	2	Quan 12	14
5	6	Le Van C	2/1/1990	Quan 7	0	NO	1	Quan 4	35
6	7	Tran Thi G	30/6/1996	Quan 10	0	6	1	Quan 5	28
7	9	Tran Van A	21/5/2002	Quan 9	2	NO	2	Quan 9	30

Bước 3: Nếu người dùng muốn thoát chương trình, thì chỉ cần bấm phím số 0 sẽ in ra thông báo "GOOD BYE" và thoát chương trình.


########################## MANAGEMENT PATIENTS ##########################
#===#
Class: CNTT K61 BM CNTT - DH GTVT PH.TPHCM
#===#
#===#
1-Find Patient   7-Check healthy
2-Statistics F by id   8-Print file
3-Status statistic F0   9-Print Patient
4-Sort by name   10-Remove Patient
5-Patients have Q-Day > 21   11-Add Patinet
6-Patients at Q-Place   12-Edit Patinet
0-Exit program
#===#
###
What is your choice: 0
GOOD BYE