

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**PHÂN HIỆU TẠI THÀNH PHỐ HỒ CHÍ MINH**

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO BÀI TẬP LỚN**

**MÔN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

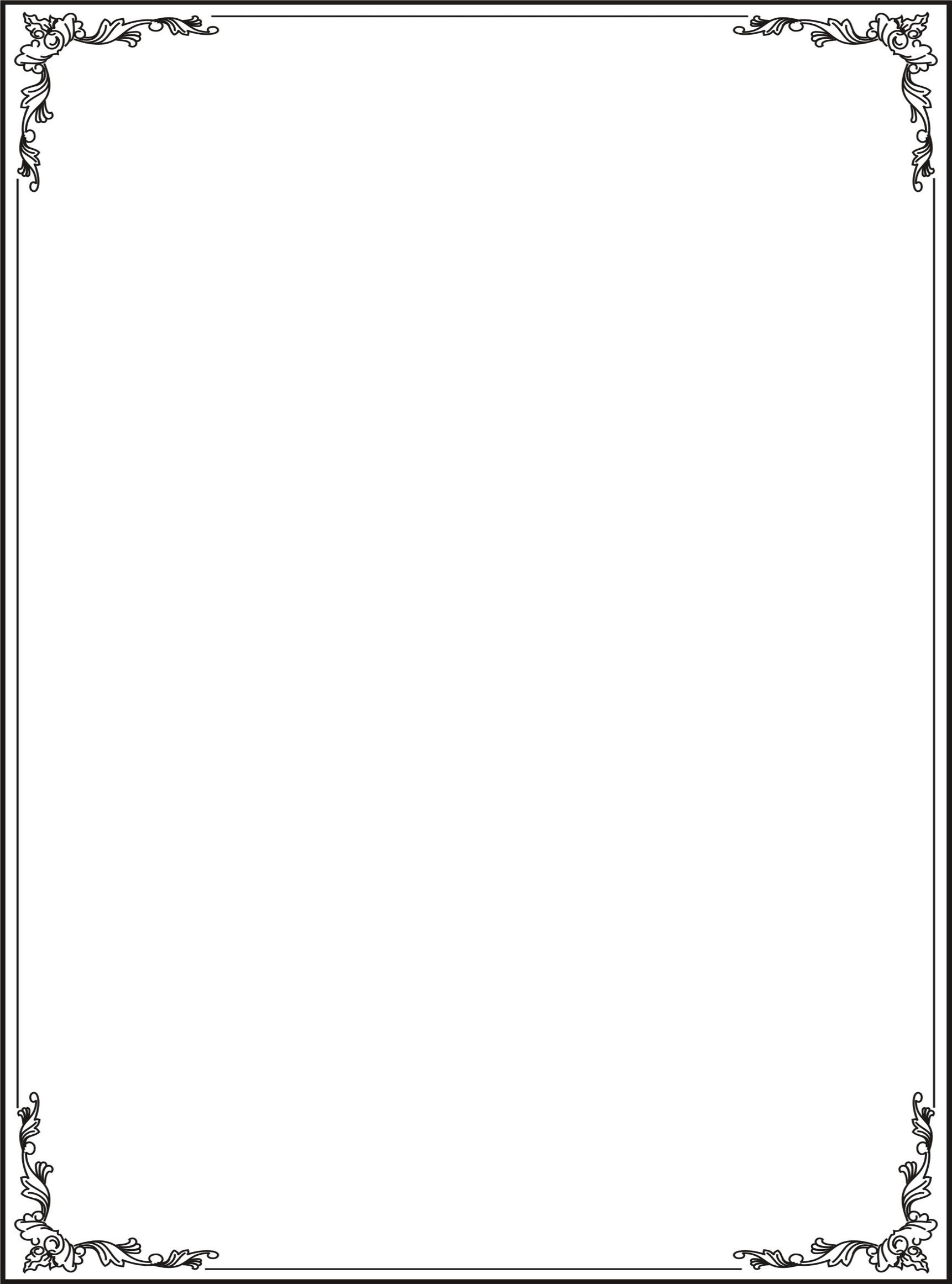
**ĐỀ TÀI: QUẢN LÝ TIỆM COFFEE**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | Trần Thị Dung |
| **Sinh viên:**  Hà Diễm Quỳnh | **MSSV:**  6151071021 |
| Nguyễn Thị Vân Khánh  Nguyễn Ngân Trúc  **Lớp:** | 6151071008  61510710  Công nghệ thông tin |
| **Khóa:** | K61 |
|  |  |

CHƯƠNG TRÌNH QUẢN LÝ NHÀ XE

Thành phố Hồ Chí Minh, năm 2021

**CỘNG HÒA XÃ HỘI CHỦ NGHĨA VIỆT NAM**

Độc lập - Tự do - Hạnh phúc

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**NHIỆM VỤ THIẾT KẾ BÁO CÁO**

BỘ MÔN: **CÔNG NGHỆ THÔNG TIN**

**-------\*\*\*-------**

**Mã sinh viên:** 6151071008 **Họ và tên:** Nguyễn Thị Vân Khánh

**Mã sinh viên:** 6151071021 **Họ và tên:** Hà Diễm Quỳnh

**Mã sinh viên:** 61510710 **Họ và tên:** Nguyễn Ngân Trúc

**Khóa:** Khóa 61 **Lớp:** CQ.61.CNTT

1. **Tên đề tài**: **Quản lý tiệm Coffee**
2. **Mục đích, yêu cầu:**
3. **Mục đích:**

Nhằm hiểu rõ hơn về phần mềm quản lý tiệm coffee thông qua ngôn ngữ lập trình hướng đối tượng.

1. **Yêu cầu:**

Phần 1: Dịch sách **The C++ Programming Language Fourth Edition**

Tóm tắt( chương 16,17,18,19,20,21)

Phần 2: Chương trình ứng dụng

Lí do chọn đề tài, mô tả bài toán, nội dung lý thuyết( tính chất hướng đối tượng, cấu trúc dữ liệu dùng trong bài toán, đồ họa), giao diện chương trình, hướng dẫn sử dụng chương trình , phụ lục: link github, cách cài đặt chương trình, hướng dẫn sử dụng chương trình.

**Lời nói đầu!**

Bạn đọc thân mến!

*Bạn có muốn biết thế giới đã thay đổi như thế nào sau một thập niên qua?*

Vậy thì hãy nối máy tính của bạn vào mạng Internet và gõ vào máy cái chữ mà bạn cần( xin vui lòng dùng tiếng Anh).Bạn sẽ nhận khoảng 30.000 từ. Sau một lúc tìm kiếm bạn sẽ thấy rằng ngày nay bạn có thể tham khảo hầu như tất cả mọi thứ trên mạng, từ các hàng hóa buôn bán ở Sydney đến các khóa đào tạo tin học qua email ở Ấn Độ. Hãy thử bấm vào một địa chỉ ở Thành phố Hồ Chí Minh và đăng kí mua coffee chắc chắn bạn sẽ nhận được hàng không lâu sau đó. Hãy ấn thử lần nữa vào một địa chỉ ở Thành phố Hồ Chí Minh và tìm một cửa hàng chuyên quản lí coffee, bạn sẽ được đáp ứng yêu cầu. Hãy tìm thử tên, thể loại và giá hay bất cứ thứ gì liên quan về những thức uống coffee v.v…chắc chắn bạn sẽ không thất vọng.

Mọi người đang tưởng tượng về một tương lai của những chiếc máy tính mặc/đeo ngược vào người, còn ứng dụng của chúng thì phát triển đến mức bất cứ ai cũng có một “thư kí ảo” đáng yêu bên cạnh với những khả năng vô bờ bến. Giấc mơ ấy hoàn toàn không phải hoang tưởng.

Còn điều gì nữa mà bạn chưa biết?

Bạn có muốn biết về phần mềm quản lí tiệm coffee hay tầm bao quát của con người đối với một hệ thống coffee vô cùng rộng lớn? Và nếu bạn muốn tìm hiểu thì cuốn sách nhỏ này sẽ giúp bạn am hiểu hơn một đôi điều. Cuốn sách này sẽ đưa bạn đến với những li kì và hình ảnh bắt mắt của thế giới coffee hiện nay . Dạo quanh thung lũng Silicon, cái nôi của nghành CNTT và gặp gỡ nhưng lập trình viên hàng đầu thế giới, những người thông minh nhất trong số những người thông minh đang nắm trong tay sức mạnh có thể thay đổi Thế giới. Và dĩ nhiên, sau khi đọc những trang tiếp theo, dù ít hay nhiều bạn cũng sẽ yêu và hứng thú với CNTT hơn.

Mong rằng cuốn sách này sẽ mang đến cho bạn đọc những giây phút thư giãn và bổ ích!

*Lời cảm ơn*

*L*

*ời nói đầu tiên, chúng em xin gửi lời cảm ơn chân thành đến Trường Đại học Giao thông Vận tải Phân hiệu tại Thành phố Hồ Chí Minh đã tạo điều kiện để sinh viên chúng em có một môi trường học tập thoải mái cả về vật chất lẫn tinh thần dù tình hình dịch bệnh diễn ra vô cùng phức tạp.*

*Cảm ơn toàn thể Bộ môn Công Nghệ Thông Tin đã tận tình giúp đỡ, hướng dẫn giúp chúng em hoàn thành đề tài một cách nhanh chóng và hiệu quả nhất.*

*Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến giảng viên bộ môn - Cô Trần Thị Dung, đã dạy dỗ, truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học Lập trình hướng đối tượng của cô, chúng em đã được trang bị cho mình những kiến thức bổ ích, với tinh thần học tập hiệu quả, nghiêm túc. Đây chắc chắn sẽ là những kiến thức quý báu, là hành trang để chúng em có thể vững bước sau này.*

*Có thể nói đây là môn học thú vị, vô cùng bổ ích cho chuyên ngành chúng em sau này. Đảm bảo cung cấp đủ kiến thức nền tảng, gắn liền với nhu cầu của ngành Công Nghệ Thông Tin. Tuy nhiên, do vốn hiểu biết còn nhiều hạn chế và khả năng tiếp thu thực tế còn nhiều bỡ ngỡ. Mặc dù chúng em đã cố gắng hết sức nhưng chắc chắn bài báo cáo khó có thể tránh khỏi những thiếu sót và nhiều vấn đề còn chưa chính xác, kính mong cô xem xét và góp ý để bài báo cáo của chúng em được hoàn thiện hơn.*

*Lời sau cùng chúng em một lần nữa kính chúc quý thầy, cô bộ môn Công nghệ thông tin Trường Đại học Giao thông Vận tải – Phân hiệu tại thành phố Hồ Chí Minh thật nhiều sức khỏe để chèo chống đón đưa thêm nhiều thế hệ sinh viên qua sông.*

*Xin chân thành cảm ơn quý thầy cô rất nhiều!*

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

**MỤC LỤC**

Nhiệm vụ thiết kế báo cáo 1

Lời nói đầu:…………………………………… 2

Lời tri ân:……………………………………… 3

**CHƯƠNG 1: TỔNG QUAN**

**I.1.TỔNG QUAN VỀ DỊCH SÁCH**

I.1.1 Tóm tắt chương 16 …………………………………………… 5

I.1.2 Tóm tắt chương 17 ……………………………………………… 6

I.1.3 Tóm tắt chương 18……………………………………………… 6

I.1.4 Tóm tắt chương 19 …………………………………………… 7

I.1.5 Tóm tắt chương 20 …………………………………………. 7

I.1.6 Tóm tắt chương 21 …………………………………………. 7

**I.2.TỔNG QUAN VỀ CHƯƠNG TRÌNH QUẢN LÝ ỨNG DỤNG**

I.2.1 Lý do chọn đề tài ……………………………………….. 8

I.2.2 Mô tả bài toán ………………………………….……… 9

**I.3. DANH MỤC CHỮ VIẾT TẮT**

**I.4. BẢNG BIỂU PHÂN CÔNG CÔNG VIỆC**

**CHƯƠNG II: CHƯƠNG TRÌNH ỨNG DỤNG**

II.1 Cơ sở lý thuyết: …………………………………………… 27

II.2 Hướng dẫn cài đặt:

II.3 Hướng dẫn sử dụng: ………………………….. 27

**CHƯƠNG III: KẾT LUẬN**

III.1 Kết quả đạt được: ……………………………………………. 33

III.2 Nhược điểm: ………………………………………………….. 33

III.3 Hướng phát triển: ……………………………….……………. 33

**CHƯƠNG IV: PHỤ LỤC:**

Tài liệu tham khảo: …………………………………………… 33

**CHƯƠNG 1: TỔNG QUAN**

**I.1.TỔNG QUAN VỀ DỊCH SÁCH**

**I.1.1 Tóm tắt chương 16**

**CHƯƠNG 16**

**Classes**

Một lớp là một kiểu do người dùng định nghĩa.

Các hàm thành viên có thể xác định ý nghĩa của khởi tạo (tạo), sao chép, di chuyển và dọn dẹp (phá hủy).

Các thành viên được truy cập bằng cách sử dụng **.**(dấu chấm) cho các đối tượng và **->**(mũi tên) cho con trỏ.

Các nhà khai thác, ví dụ như **+**, **!**, và **[]**, có thể được định nghĩa cho một lớp.

Một lớp là một không gian tên chứa các thành viên của nó.

Một **struct**là một **lớp**nơi các thành viên theo public.

Ví dụ:

**class X {**

**private:**

**int m;**

**public:**

**X(int i =0) :m{i} { }**

**int mf(int i)**

**{**

**int old = m;**

**m = i;**

**return old;**

**}**

**};**

**X var {7};**

**int user(X var, X ptr)**

**{**

**int x = var.mf(7);**

**int y = ptr−>mf(9);**

**int z = var.m;**

**}**

**16.2.1 Hàm thành viên**

Các *hàm thành viên* sử dụng cú pháp chuẩn để truy cập thành viên cấu trúc.

Ví dụ:

**Date my\_bir thday;**

**void f()**

**{**

**Date today;**

**today.init(16,10,1996);**

**my\_bir thday.init(30,12,1950);**

**Date tomorrow = today;**

**tomorrow.add\_day(1);**

**}**

Bởi vì các cấu trúc khác nhau có thể có các hàm thành viên có cùng tên, chúng ta phải chỉ định tên cấu trúc khi xác định một hàm thành viên.

**16.2.2 Sao chép mặc định**

Theo mặc định, các đối tượng có thể được sao chép. Đặc biệt, một đối tượng lớp có thể được khởi tạo bằng một bản sao của một đối tượng thuộc lớp của nó. Ví dụ:

**Date d1 = my\_bir thday;**

**Date d2 {my\_bir thday};**

Tương tự, theo mặc định, các đối tượng của lớp có thể được sao chép theo phép gán. Ví dụ:

**void f(Date& d)**

**{**

**d = my\_bir thday;**

**}**

**16.2.3 Kiểm soát truy cập**

Private: chỉ có thể được sử dụng bởi các hàm thành viên.

Public : *công khai*, tạo thành giao diện công khai cho các đối tượng của lớp.

Tuy nhiên, các chức năng nonmember bị cấm sử dụng các thành viên riêng tư. Ví dụ:

**void timewarp(Date& d)**

**{**

**d.y −= 200;** **//** *error : Date::y is private*

**}**

Có một số lợi ích thu được từ việc hạn chế quyền truy cập vào cấu trúc dữ liệu đối với danh sách các hàm được khai báo rõ ràng.

Bảo vệ dữ liệu riêng tư dựa trên việc hạn chế sử dụng tên thành viên trong lớp.

**16.2.4 Lớp và cấu trúc**

class **X {...};**

được gọi là *định nghĩa lớp*; nó định nghĩa một loại gọi là **X**.Một định nghĩa lớp có thể được sao chép trong các tệp nguồn khác nhau bằng cách sử dụng **#include**mà không vi phạm quy tắc một định nghĩa .

Theo định nghĩa, một **cấu trúc**là một lớp trong đó các thành viên được mặc định là công khai; đó là,

**struct S {/ \****...***\* /};**

chỉ đơn giản là viết tắt của

**class S {public: / \****...***\* /};**

Theo mặc định, các thành viên của một classlà riêng tư:

Không bắt buộc phải khai báo dữ liệu trước trong một class. Trên thực tế, thường hợp lý khi đặt các thành viên dữ liệu cuối cùng để nhấn mạnh các chức năng cung cấp giao diện người dùng công khai. Ví dụ:

**class Date3 {**

**public:**

**Date3(int dd, int mm, int yy);**

**void add\_year(int n);** **//** *add n years*

**private:**

**int d, m, y;**

**};**

Các chỉ định truy cập có thể được sử dụng nhiều lần trong một khai báo lớp duy nhất. Ví dụ:

**class Date4 {**

**public:**

**Date4(int dd, int mm, int yy);**

**private:**

**int d, m, y;**

**public:**

**void add\_year(int n);** **//** *add n years*

**};**

Tuy nhiên, có nhiều hơn một phần công khai, có xu hướng lộn xộn và có thể ảnh hưởng đến bố cục đối tượng .

**16.2.5 Hàm tạo**

Việc sử dụng các hàmđể cung cấp khởi tạo cho các đối tượng lớp là không phù hợp và dễ xảy ra lỗi. Bởi vì không có chỗ nào nói 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 thảm hại như nhau). Một cách tiếp cận tốt hơn là cho phép pro-grammer khai báo một hàm với mục đích rõ ràng là khởi tạo các đối tượng. Bởi vì một hàm như vậy xây dựng các giá trị của một kiểu nhất định, nó được gọi là một hàm *tạo*.

Hàm tạo có cùng tên với class.

Khi một lớp có một phương thức khởi tạo, tất cả các đối tượng của lớp đó sẽ được khởi tạo bởi một lời gọi phương thức khởi tạo. Nếu hàm tạo yêu cầu các đối số, các đối số này phải được cung cấp:

Vì một phương thức khởi tạo xác định quá trình khởi tạo cho một lớp, chúng ta có thể sử dụng ký hiệu **{}**:

Sử dụng ký hiệu **{} thay** ký hiệu **()**để khởi tạo vì nó rõ ràng về những gì đang được thực hiện (khởi tạo), tránh một số lỗi tiềm ẩn và có thể được sử dụng nhất quán. Có những trường hợp phải sử dụng ký hiệu **()**, nhưng chúng rất hiếm.

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.

-Hàm tạo mặc định

-Hàm tạo không đầy đủ tham số

-Hàm tạo không đầy đủ tham số

-Hàm tạo sao chép

Ví dụ:

**class Date {**

**int d, m, y;**

**public:**

**//** *...*

**Date(int, int, int);**

**Date(int, int);**

**Date(int);**

**Date();**

**Date(const char );**

Với các hàm tạo, các hàm thành viên khác không còn phải đối phó với khả năng dữ liệu chưa được khởi tạo.

**16.2.6 Hàm tạo rõ ràng**

Một construc-tor được khai báo với từ khóa **explicit** chỉ có thể được sử dụng để khởi tạo và chuyển đổi rõ ràng. Ví dụ:

**struct X {**

**explicit X();**

**explicit X(int,int);**

|  |  |  |  |
| --- | --- | --- | --- |
| **};** |  |  |  |
| **X x1 = {};** | | | **//** *error: implicit* |
| **X x2** | **= {1,2};** | | **//** *error: implicit* |
| **X x3** | **{};** | | **//** *OK: explicit* |
| **X x4** | **{1,2};** | | **//** *OK: explicit* |
| **int f(X);** | | |  |
| **int i1 = f({});** | | | **//** *error: implicit* |
| **int i2** | | **= f({1,2});** | **//** *error: implicit* |
| **int i3** | | **= f(X{});** | **//** *OK: explicit* |
| **int i4** | | **= f(X{1,2});** | **//** *OK: explicit* |

Một khởi tạo với dấu **=**được coi là một *khởi tạo sao chép*.

Nếu bỏ dấu = thì được coi là khởi tạo trực tiếp.

Sự phân biệt giữa khởi tạo trực tiếp và sao chép được duy trì cho việc khởi tạo danh sách

**16.2.7 Bộ khởi tạo trong lớp**

Khi chúng ta sử dụng một số hàm tạo, việc khởi tạo thành viên có thể trở nên lặp đi lặp lại. Ví dụ:

**class Date {**

**int d, m, y;**

**public:**

**Date(int, int, int);**

**Date(int, int);**

**Date(int);**

**Date();**

**Date(const char );**

**//** *...*

**//** *day, month, year*

**//** *day, month, today’s year*

**//** *day, today’s month and year*

**//** *default Date: today*

**//** *date in string representation*

**};**

Chúng ta có thể giải quyết vấn đề đó bằng cách giới thiệu các đối số mặc định để giảm số lượng hàm tạo (§16.2.5). Ngoài ra, chúng ta có thể thêm trình khởi tạo vào các thành viên dữ liệu:

**class Date {**

**int d {today.d};**

**int m {today.m};**

**int y {today.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*

**//** *...*

**16.2.8 Định nghĩa hàm trong lớp**

Một thành viên có thể tham chiếu đến một thành viên khác trong lớp của mình một cách độc lập với nơi thành viên đó được xác định (§6.3.4). ví dụ:

**class Date {**

**public:**

**void add\_month(int n) { m+=n; }** **//** *increment the Date’s m*

**private:**

**int d, m, y;**

**};**

**16.2.9 Thành viên tĩnh**

Các biến thành viên tĩnh là các biến thành viên mà thuộc về class, chứ không phải là thuộc về các đối tượng của class. Tương tự, một hàm cần truy cập vào các thành viên của một lớp, nhưng không cần phải được gọi cho một đối tượng cụ thể, được gọi là hàm thành viên **tĩnh**.

Nếu được sử dụng, một thành viên **tĩnh**- một hàm hoặc một thành viên dữ liệu - phải được định nghĩa ở đâu đó. Từ khóa **static**không được lặp lại trong định nghĩa của một thành viên **static**. Ví dụ:

**Date Date::default\_date {16,12,1770};**

**//** *definition of Date::default\_date*

**void Date::set\_default(int d, int m, int y)**

**{**

**//** *definition of Date::set\_default*

**default\_date = {d,m,y};**

**//** *assign new value to default\_date*

**}**

**16.2.10 Các loại thành viên**

Các kiểu và bí danh kiểu có thể là thành viên của một lớp. Ví dụ:

**template<typename T>**

**class Tree {**

**using value\_type = T;**

**//** *member alias*

**enum Policy { rb, splay, treeps }; class Node {**

**//** *member enum*

**//** *member class*

**Node right;**

**Node left;**

**value\_type value;**

**public:**

**void f(Tree );**

**};**

**Node top;**

**public:**

**void g(const T&);**

**//** *...*

**};**

Một lớp không có bất kỳ quyền truy cập đặc biệt nào đối với các thành viên của lớp lồng nhau của nó. Ví dụ:

**template<typename T>**

**void Tree::g(Tree::Node p)**

**{**

**value\_type val = right−>value;**

**//** *error : no object of type Tree::Node*

**value\_type v = p−>right−>value;**

**//** *error: Node::right is private*

**p−>f(this);**

**//** *OK*

**}**

**16.3 Lớp Trừu tượng**

**16.3.1 Chức năng thành viên**

Đương nhiên, một triển khai cho mỗi chức năng thành viên phải được cung cấp ở đâu đó. Ví dụ:

**Date::Date(int dd, Month mm, int yy)**

**:d{dd}, m{mm}, y{yy}**

**{**

**if (y == 0) y = default\_date().year();**

**if (m == Month{}) m = default\_date().month();**

**if (d == 0) d = default\_date().day();**

**if (!is\_valid()) throw Bad\_date();**

**}**

**I.1.2 Tóm tắt chương 17**

**CHƯƠNG 17**

**17,1** Hàm tạo và Hàm hủy

Một lớp thường sẽ có hàm tạo và hàm hủy.

**17.2.1 Hàm tạo**

Một thành viên có cùng tên với lớp của nó được gọi là một phương thức *khởi tạo*. Ví dụ:

**class Vector {**

**public:**

**Vector(int s);**

**};**

Một khai báo phương thức khởi tạo chỉ định một danh sách đối số (chính xác như đối với một hàm) nhưng không có kiểu trả về. Tên của một lớp không thể được sử dụng cho một hàm thành viên thông thường, thành viên dữ liệu, kiểu thành viên, v.v., trong lớp. Ví dụ: 

**struct S {**

**S();**

**void S(int);**

**int S;**

**enum S { foo, bar };**

**//** *fine*

**//** *error : no type can be specified for a constructor*

**//** *error : the class name must denote a constructor*

**//** *error : the class name must denote a constructor*

**};**

**17.2.2 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 lấy một tài nguyên - chẳng hạn như tệp, khóa hoặc một số bộ nhớ - phải được giải phóng sau khi sử dụng (§5.2, §13.3). Do đó, một số lớp cần một hàm được đảm bảo sẽ được gọi khi một đối tượng bị hủy theo cách tương tự như cách một phương thức khởi tạo được 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, đối với exam-ple**˜Vector ()**. Một ý nghĩa của **˜**là '' phần bổ sung '' (§11.1.2), và hàm hủy cho một lớp hoàn thành các hàm tạo của nó. Một hàm hủy không nhận đối số và một lớp chỉ có thể có một bộ hủy cấu trúc.

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*

1. *...*

**private:**

**double elem; //** *elem points to an array of sz doubles*

**int sz;** **//** *sz is non-negative*

**};**

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 ++.

Một lập trình viên khai báo hàm hủy cho một lớp cũng phải quyết định xem các đối tượng của lớp đó có thể được sao chép hoặc di chuyển hay không (§17.6).

**17.2.3 Tạo và hủy thành viên**

Bộ tạo và bộ hủy tương tác chính xác với cấu trúc phân cấp lớp (§3.2.4, Chương 20). Con-structor xây dựng một đối tượng lớp '' từ dưới lên '':

Đầu tiên, hàm tạo gọi các hàm tạo lớp cơ sở của nó,

Sau đó, nó gọi các hàm tạo thành viên, và cuối cùng, nó thực thi cơ thể của chính nó.

Đầu tiên, trình hủy thực thi phần thân của chính nó, sau đó, nó gọi các trình hủy thành viên của nó, và cuối cùng, nó gọi các hàm hủy lớp cơ sở của nó.

Nếu một lớp được sử dụng để cần một hàm tạo mặc định và nếu lớp đó không có các cấu trúc con khác, thì trình biên dịch sẽ cố gắng tạo một hàm tạo mặc định. Ví dụ:

**struct S1 {**

**string s;**

**};**

**S1 x;** **//** *OK: x.s is initialized to ""*

**17.2.4 Gọi hàm tạo và hủy**

Một hàm hủy được gọi ngầm khi thoát khỏi phạm vi hoặc bằng cách **xóa**. Thông thường không chỉ cần gọi rõ ràng một hàm hủy một cách rõ ràng; làm như vậy sẽ dẫn đến các lỗi khó chịu. Tuy nhiên, có những trường hợp hiếm hoi (nhưng quan trọng) mà một hàm hủy phải được gọi một cách rõ ràng.

**17.2.5 Bộ hủy ảo**

Một hàm hủy có thể được khai báo là **ảo**và thường phải dành cho một lớp có hàm ảo.

Ví dụ:

**class Shape {**

**public:**

**virtual void draw() = 0;**

**virtual ˜Shape();**

**};**

**class Circle {**

**public:**

**//** *...*

**void draw();**

**˜Circle();**

**//** *…*

**//** *overr ides ˜Shape()*

**};**

Lý do chúng ta cần một hàm hủy **ảo**là một đối tượng thường được thao tác thông qua giao diện được cung cấp bởi một lớp cơ sở cũng thường bị **xóa** thông qua giao diện đó:

**17.3** **Khởi tạo đối tượng lớp**

**17.3.1 Khởi tạo không có trình tạo**

Chúng ta không thể định nghĩa một phương thức khởi tạo cho một kiểu dựng sẵn, nhưng chúng ta có thể khởi tạo nó với một giá trị của kiểu phù hợp. Ví dụ:

Khởi tạo mặc định của việc sử dụng **{}**được định nghĩa là khởi tạo của từng thành viên bởi **{}**. Vì vậy, **không có nào**được khởi tạo thành **{{}, {}, {}}**, là **{"", "", 0}**(§17.3.3).

**17.3.2 Trình tạo mặc định**

Một phương thức khởi tạo có thể được gọi mà không cần đối số được gọi là một phương thức *khởi tạo mặc định*. Cấu trúc con mặc định rất phổ biến. Ví dụ:

 **class Vector {**

**public:**

**Vector(); //** *default constructor : no elements*

**};**

Một hàm tạo mặc định được sử dụng nếu không có đối số nào được chỉ định hoặc nếu danh sách bộ khởi tạo trống được cung cấp:

**Vectơ v1;** **//***OK*

**Vectơ v2 {};** **//***OK*

Một đối số mặc định (§12.2.5) có thể tạo một hàm tạo nhận các đối số vào một hàm tạo mặc định. Ví dụ:

**class String {**

**public:**

**String(const char p = ""); //** *default constructor : empty string* **//** *...*

**};**

**String s1;** **//** *OK*

**String s2 {};** **//** *OK*

**17,4 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**; có nghĩa là, hiệu quả là các giá trị của **x**và **y**là cả hai bằng **y**‘s giá trị trước khi chuyển nhượng.

*Move*lá **x**với **y**‘s giá trị cũ và **y**với một số *di chuyển-từ nhà nước.*Đối với hầu hết các trường hợp lồng vào nhau, vùng chứa, trạng thái chuyển từ đó là '' trống ''.

Sự phân biệt logic đơn giản này bị nhầm lẫn bởi truyền thống và thực tế là chúng ta sử dụng cùng một ký hiệu cho cả di chuyển và sao chép.

Thông thường, một nước đi không thể ném, trong khi một bản sao có thể (vì nó có thể cần lấy tài nguyên), và một nước đi thường hiệu quả hơn một bản sao. Khi bạn viết 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.

Để giúp chúng ta khỏi công việc lặp đi lặp lại tẻ nhạt, sao chép và di chuyển có các định nghĩa mặc định (§17.6.2).

**17.5.1 Sao chép**

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 &)**

Bạn có thể xác định hai phép toán này bằng các loại đối số mạo hiểm hơn, chẳng hạn như **X dễ bay hơi &**, nhưng không; bạn sẽ chỉ nhầm lẫn chính bạn và những người khác. Một phương thức khởi tạo sao chép được cho là tạo một bản sao của một đối tượng mà không cần sửa đổi nó. Tương tự, bạn có thể sử dụng **const X &**làm kiểu trả về của gán bản sao.

Hàm tạo sao chép và phép gán sao chép khác nhau ở chỗ hàm tạo sao chép khởi tạo bộ nhớ chưa khởi tạo, trong khi toán tử gán bản sao phải xử lý chính xác một đối tượng đã được xây dựng và có thể sở hữu tài nguyên.

**17.5.1.3 Ý nghĩa của bản sao**

Một hàm tạo bản sao hoặc phép gán bản sao phải làm gì để được coi là '' một công cụ sao chép thích hợp ''? Ngoài việc được khai báo với một kiểu chính xác, một hoạt động sao chép phải có ngữ nghĩa sao chép thích hợp. Hãy xem xét một hoạt động sao chép, **x = y**, của hai đối tượng cùng loại. Để phù hợp với lập trình hướng giá trị nói chung (§16.3.4) và để sử dụng với thư viện chuẩn nói riêng (§31.2.2), hoạt động phải đáp ứng hai tiêu chí:

*Tính tương đương*: Sau **x = y**, các phép toán trên **x**và **y**sẽ cho kết quả giống nhau. Đặc biệt, nếu **==**được xác định cho kiểu của chúng, chúng ta sẽ có **x == y**và **f (x) == f (y)**cho bất kỳ hàm **f ()**nào chỉ phụ thuộc vào các giá trị của **x**và **y**(trái ngược với có hành vi của nó phụ thuộc vào địa chỉ của **x**và **y**).

*Độc lập*: Sau **x = y**, các phép toán trên **x**không được thay đổi hoàn toàn trạng thái của **y**, nghĩa là **f (x)**không thay đổi giá trị của **y**miễn là **f (x)**không tham chiếu đến **y**.

**17.5.2 Di chuyển**

Cách truyền thống để nhận giá trị từ **a**đến **b**là sao chép nó. Đối với một số nguyên trong mem-ory của máy tính, đó chỉ là điều duy nhất có ý nghĩa: đó là những gì phần cứng có thể làm với một lệnh duy nhất. Tuy nhiên, theo quan điểm chung và logic thì không phải như vậy. Hãy xem xét việc triển khai rõ ràng **swap ()**trao đổi giá trị của hai đối tượng. Ví dụ:

**template<class T> void swap(T& a, T& b) {**

**const T tmp = a;**

**a = b;**

**b = tmp;**

**};**

Vấn đề cơ bản là chúng ta thực sự không muốn sao chép gì cả: chúng ta chỉ muốn trao đổi các cặp giá trị.

Chúng ta cũng có thể nhìn vấn đề sao chép từ một quan điểm hoàn toàn khác: chúng ta thường không sao chép những thứ vật chất, trừ khi chúng ta hoàn toàn phải làm như vậy. Nếu bạn muốn mượn điện thoại của tôi, tôi sẽ chuyển điện thoại của tôi cho bạn chứ không phải biến đện thoại của tôi thành bản sao của riêng bạn. Nếu tôi cho bạn mượn xe của tôi, tôi sẽ đưa cho bạn một chìa khóa và bạn lái xe đi trong xe của tôi, chứ không phải trong bản sao mới tạo của bạn về chiếc xe của tôi. Một khi tôi đã đưa cho bạn một đối tượng, bạn có nó và tôi không còn làm nữa.

**I.1.3 Tóm tắt chương 18**

**CHƯƠNG 18: NẠP CHỒNG**

1. **Nạp chồng hàm:**

Là các hàm có tên giống nhau, nhưng khác nhau về kiểu dữ liệu của tham số. Khi gọi tên, hàm sẽ được truy cập tới dựa vào tham số truyền vào.

VD:

void tong( int a, int b)

{

return a+b;

}

void tong( float a, float b)

{

return a+b;

}

int main()

{

tong(2,3); // truy cap ham tong( int a, int b)

tong(3.5,6.8); //truy cap ham tong( float a, float b)

1. **Nạp chồng toán tử:**

Toán tử có thể nạp chồng:

|  |  |  |
| --- | --- | --- |
| **Ngôi** | **Nhóm** | **Toán tử** |
| 1 ngôi | Tăng giảm | ++, -- |
|  | Dấu số học | +, - |
|  | Logic | !, ~ |
|  | Con trỏ | \*, & |
|  | Ép kiểu | int, float, double,… |
|  |  |  |
| 2 ngôi | Số học | +, -, \*, /, %, +=, -=, \*=, /=, %=,… |
|  | So sánh | <, >, ==, >=, <=, != |
|  | Logic | &&, ||, &, | |
|  | Nhập xuất | <<, >> |
|  | Gán | = |
|  | Lấy chỉ số mảng | [ ] |

Tóan tử không thể nạp chồng:

|  |  |
| --- | --- |
| **Nhóm** | **Toán tử** |
| Xác định phạm vi | :: |
| Lựa chọn thành viên | . |
| Lựa chọn thành viên thông qua con trỏ | .\* |
| Kích thước | sizeof |
| Căn chỉnh | alignof |
| Trả về thông tin lúc chạy | typeid |
| Toán tử điều kiện | ? : |
| Chỉ thị tiền xử lý | # |
| Chỉ thị tiền xử lý | ## |

Tên của hàm toán tử:

operator@(); Với @ là toán tử

VD:

class phanso

{

private:

int tu;

int mau;

puplic:

void operator\*(phanso a,phanso b);

};

void phanso::operator\*(phanso a,phanso b){

phanso t;

tu=a.tu\*b.tu;

mau=a.mau\*b.mau;

return t;

}

int main(){

phanso x,y;

cout<<”Tong”<<x\*y; //goi ham tinh tong

}

Hàm nạp chồng toán tử có ít nhất 1 đối số, đối số đầu tiên bao giờ cũng thuộc lớp do người dùng định nghĩa.

Đối với các phép toán tăng giảm một đối tượng:

Có 2 trường hợp: ++a hoặc a++

Đối với trường hợp ++a ta viết tên hàm : operator++a();

Đối với trường hợp a++ ta viết tên hàm : operator++a(int);

Có những phép toán mà đối với dữ liệu có sẵn, chương trình có thể tự hiểu và chuyển đổi như –a tương tương với a -= 1 hay là a=a-1. Nhưng với nạp chồng khi ta viết opertator +=() không có nghĩa nó là sự kết hợp giữ operator+() và operator=().

Với toán tử gán( =) giá trị được cung cấp cho đối tượng trước khi gán sẽ được xóa.

Một hàm nạp chồng toán tử phải là một thành viên của class và có ít nhất một đối số do người dùng định nghĩa

Có 2 cách để truyền giá trị cho toán tử nạp chồng là truyền bằng trị và truyền bằng tham trị, với các đối tượng không thay đổi trong suốt quá trình thực hiện chương trình thì ta có thể thêm const.

|  |  |
| --- | --- |
| **Truyền bằng trị** | **Truyền tham trị** |
| operator +=( int a); | operator +=( int &a); |

VD: Hàm truyền bằng tham trị

ostream& operator << (ostream & os, Phanso &ps)

{

os<< ps.tu<<"/"<<ps.mau<<endl;

return os;

}

VD: Hàm truyền bằng trị

operator <(Phanso ps)

{

if((float)(tu/mau)<(float)(ps.tu/ps.mau))

return 1;

}

Hàm nạp chồng toán tử là thành viên của lớp, nhưng có những hàm nạp chồng toán tử cùng tên, giống nhau cả về tham số nhưng khác nhau về kết quả trả về. Vì vậy khi gọi tên hàm chương trình sẽ không biết phải sử dụng hàm nào. Do đó namespace được triển khai để giúp chương trình tìm được hàm cần sử dụng. Toán tử nạp chồng khi được gọi ra sẽ được chương trình tìm kiếm trong không gian tên của nó thông qua các tham số.

class complex {

private:

double re, im;

public:

complex& operator+=(complex a){

re += a.re;

im += a.im;

return ∗this;

}

complex& operator+=(double a){

re += a;

return ∗this;

}

};

complex operator+(complex a, complex b)

{

return a += b;

}

complex operator+(complex a, double b)

{

return {a.real()+b,a.imag()};

}

complex operator+(double a, complex b)

{

return {a+b.real(),b.imag()};

}

void f(complex x, complex y){

auto r1 = x+y;

auto r2 = x+2;

auto r3 = 2+x;

auto r4 = 2+3;

}

Ta cũng có thể xây dựng một hàm chuyển kiểu giống như khi người ta cho complex{3} có nghĩa là cho một số phức với phần thực là 3 và phần ảo là 0.

Ta xây dựng lớp như sau để định nghĩa:

class complex

{

private:

double re, im;

public:

complex(): re{0}, im{0} { }

complex(double r): re{r}, im{0} { }

complex(double r, double i): re{r}, im{i} { }

};

Tuy nhiên cách nào sẽ trỏ nên phức tạp nếu trong trường hợp hàm nạp chồng có 3 đối số, sẽ có rất nhiều trường hợp xảy ra. Nên chúng ta có thể viết theo cách khác:

bool operator==(complex,complex);

void f(complex x, complex y)

{

x==y;

x==3;

3==y;

}

1. **Literal**

Literal là giá trị cử chính nó đang mang. Ví dụ trong câu lệnh cout<<”Hello world”;

Thì literal chính là “Hello world”. Một số phức có thể được tạo trong quá trình biên dịch khi ta sử dụng constexpt:

class complex {

public:

constexpr complex(double r =0, double i =0) : re{r}, im{i} { }

};

void main()

{

complex z1 {1.2+12e3i};

}

Trong 1 vài trường hợp ta sẽ cần sử dụng dữ liệu ở private. Nhưng nếu dùng hàm thông thường thì chương trình sẽ không cho phép thực hiện. Do đó, ta phải dùng hàm thành viên lấy ra giá trị thuộc tính ở private.

class complex

{

double re,im;

public:

constexp double real() const {return re;}

constexp double image() const {return im;}

void real (double r) {re=r;}

void hinh (double i) {im=i:}

}

1. **Ép kiểu trong c++**

Chúng ta có thể sử dụng toán tử operator để ép kiểu dữ liệu trong c++

**I.1.4 Tóm tắt chương 19**

**CHƯƠNG 19: TOÁN TỬ ĐẶC BIỆT**

1. **Toán tử đặc biệt**

Đó là các loại toán tử sau: [], (), ++, --,new delete

Nạp chồng toán tử truy cập đến phần tử mảng

#include <iostream>

using namespace std;

const int MAX = 15;

class Mang

{

private:

int A[MAX];

public:

Mang()

{

register int i;

for (i = 0; i < MAX; i++)

{ A[i] = i; }

}

int& operator[](int i)

{

if (i > MAX)

{

cout << "\n======================\n" << endl;

  cout << "Chi muc vuot gioi han!" << endl;

return mang[0];

}

return mang[i];

}

};

int main()

{

Mang V;

cout << "Gia tri cua V[3] la: " << V[3] << endl;

cout << "Gia tri cua V[6] la: " << V[6] << endl;

cout << "Gia tri cua V[16]la:"<<V[16]<<endl;

return 0;

}

Toán tử () được dùng với hàm toán tử truyền tham số bất kì.

#incluce<iostream>

using namespace std;

class Sosanh{

private:

int a;

int b;

public:

int operator() (int a, int b)

{

if( a>b)

{

return a;

}

}

int oprator() (int a, int b, int c)

{

return a+b+c;

}

};

int main()

{

Sosanh f;

cout<< f( 3,5)<<endl;

cout<< f( 7,8,3);

}

Tăng giảm tiền tố hậu tố:

class Phanso{

private :

int tu;

int mau;

public:

Phanso operator ++(int);

Phanso operator ++();

}

Phanso Phanso::operator ++(int)

{

tu=tu+mau;

return \*this;

}

Phanso Phanso::operator ++()

{

tu=tu+mau;

return \*this;

}

Xác định new và delete cho 1 lớp cụ thể:

void \*operator new(size\_t);

void operator delete(void\*, size\_t);

Kiểu dữ liệu do người dùng định nghĩa:

VD: nhị phân, 5giây, 10km, 20phút,…

Những kiểu dữ liệu này luôn có hậu tố đi theo như giây, phút, giờ, m, km,…

Khi đó ta sẽ dùng operator để đọc 1 thông tin nhập vào với hậu tố:

operator “” <hậu tố> (tham số);

VD:Ta nhập hai quãng đường 10km và 5km. khi tính toán thì máy tính không thể lấy 10km+5km mà lấy như sau 10+5 rồi thêm hậu tố km vào kết quả.

* KM operator “” km(float d)

{

return KM(d);

}

Trả về giá trị trước hậu tố để tính toán.

1. **Lớp chuỗi:**

Các phương thức với lớp chuỗi

* Tạo chuỗi mặc định.
* Sao chép chuỗi.
* Di chuyển chuỗi.
* Hủy chuỗi.

Truy cập vào các kí tự trong chuỗi bằng cách sử dụng [].

Trình bày chuỗi đáp ứng 3 yêu cầu: dễ dàng truy cập vào từng kí tự, dễ thêm kí tự vào cuối và tiết kiệm bộ nhớ.

Hàm thành viên của class chuỗi gồm có:

Hàm tạo chuỗi rỗng

String::String()  
 :sz{0}, ptr{ch}

{

ch[0]=0;

}

Hàm tạo copy chuỗi

String::String(const char∗ p)

:sz{strlen(p)},

ptr{(sz<=short\_max) ? ch : new char[sz+1]},

space{0}

{

strcpy(ptr,p);

}

Hàm tạo di chuyển chuỗi

String::String(String&& x)

{

move\_from(x);

}

Toán tử copy:

String& String::operator=(const String& x)

{

if (this==&x) return ∗this;

char∗ p = (shor t\_max<sz) ? ptr : 0;

copy\_from(x);

delete[] p;

return ∗this;

}

Toán tử di chuyển:

String& String::operator=(String&& x)

{

if (this==&x) return ∗this;

if (short\_max<sz) delete[] ptr;

move\_from(x); // does not throw

return ∗this;

}

Thêm kí tự vào cuối:

String& String::operator+=(char c)

{

if (sz==short\_max) {

int n = sz+sz+2;

ptr = expand(ptr,n);

space = n−sz−2;

}

else if (short\_max<sz)

{

if (space==0)

{

int n = sz+sz+2;

char∗ p = expand(ptr,n);

delete[] ptr;

ptr = p;

space = n−sz−2;

}

else

−−space;

}

ptr[sz] = c;

ptr[++sz] = 0;

return ∗this;

}

1. Hàm bạn:

Là cho phép hàm ngoài class có thể truy cập vào thuộc tính private.

class Phanso{

private :

int tu;

int mau;

};

friend istream& operator >> (istream & is, Phanso &ps);

Hàm bạn sẽ được gọi thông qua đối số của nó.

Cách sử dụng hàm bạn và hàm thành viên:

Hàm thành viên chỉ sử dụng cho một đối tượng thuộc lớp của nó, còn hàm bạn được sử dụng chỉ với mục đích truy cập vào một phần của thuộc tính lớp nhằm mục đích khác chứ không thể hiện rõ nó là một phương thức của đối tượng.

**I.1.5 Tóm tắt chương 20**

**CHƯƠNG 20 : LỚP DẪN XUẤT**

**1. Giới thiệu**

Từ Simula, C ++ đã vay mượn ý tưởng của các lớp và cấu trúc phân cấp lớp. Ngược lại, việc sử dụng các tính năng ngôn ngữ hỗ trợ cho các ý tưởng thiết kế sẽ phân biệt việc sử dụng C ++ hiệu quả. Một khái niệm (ý tưởng, khái niệm, v.v.) không tồn tại một cách cô lập. Ví dụ, cố gắng giải thích ô tô là gì. 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, dầu, vé chạy quá tốc độ, nhà nghỉ, v.v. Vì chúng tôi sử dụng các lớp để biểu thị các khái niệm nên vấn đề trở thành cách thể hiện mối quan hệ giữa các khái niệm.

Khái niệm về một lớp dẫn xuất 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 định nghĩa rõ ràng lớp Circle và lớp Triangle để có chung lớp Shape. Trong trường hợp đó, lớp chung 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 lớp con. 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 lớp con. Chương này là sự khám phá ý nghĩa của ý tưởng đơn giản này, 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:

• Kế thừa triển khai: để tiết kiệm nỗ lực triển khai bằng cách chia sẻ các cơ sở 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 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**

**Employee**

**↑**

**Manager**

Một lớp dẫn xuất thường được cho là kế thừa các thuộc tính từ cơ sở của nó, còn được gọi là kế thừa. Một lớp cơ sở được gọi là lớp cha và lớp dẫn xuất là lớp con.

Ta có một chương trình định nghĩa cấu trúc nhân viên:

struct Employee{

string name,

char middle\_initial;

}

Và định nghĩa cấu trúc người quảnn lý

struct Manager {

Employee emp;

list<Employee\*> group;

}

**Hoặc sơ đồ:**

|  |
| --- |
| first\_name  family\_name  ... |

**Employee: Manager:**

|  |
| --- |
| first\_n*ame*  *family*\_name  ... |
| group  level  ... |

**3. Hàm thành viên**

Một hàm thành viên của một lớp dẫn xuất có thể truy cập vào các thành viên riêng của lớp cơ sở của nó. Thành viên riêng sẽ trở nên vô nghĩa nếu cho phép một lập trình viên truy cập vào phần riêng của một lớp chỉ đơn giản bằng cách dẫn xuất một lớp mới từ nó

**4. Phân cấp lớp**

Bản thân một lớp dẫn xuất có thể là một lớp cơ sở. Ví dụ:

**Class Employee{/ \* ... \* /};**

**Class Manager: public Employee {/ \* ... \* /};**

**Class Director: public Manager {/ \* ... \* /};**

**5. Chức năng ảo**

Các hàm ảo khắc phục các vấn đề với giải pháp trường kiểu bằng cách cho phép lập trình viên khai báo các hàm trong một lớp cơ sở có thể được định nghĩa lại trong mỗi lớp dẫn xuất.

**class Employee {**

**public:**

**Employee(const string& name, int dept);**

**virtual void print() const;**

**// ...**

**private:**

**string first\_name , family\_name;**

**short depar tment;**

**// ...};**

Một hàm ảo có thể được sử dụng ngay cả khi không có lớp nào được dẫn xuất từ lớp của nó và một lớp dẫn xuất không cần phiên bản riêng của hàm ảo thì không cần cung cấp một lớp.Ví dụ:

**class Manager : public Employee {**

**public:**

**Manager(const string& name, int dept, int lvl);**

**void print() const;**

**// ...**

**private:**

**list<Employee∗> group;**

**short level;**

**// ...};**

**void Manager::print() const{**

**Employee::print();**

**cout << "\tlevel " << level << '\n';**

**// ...}**

**5. Kiểm soát truy cập**

Thành viên của một lớp có thể là riêng tư, được bảo vệ hoặc công khai:

• Nếu nó là private, tên của nó chỉ có thể được sử dụng bởi các hàm thành viên và bạn bè của lớp mà nó được khai báo.

• Nếu nó được bảo vệ, tên của nó chỉ có thể được sử dụng bởi các hàm thành viên và bạn bè của lớp mà nó được khai báo và bởi các hàm thành viên và bạn bè của các lớp dẫn xuất từ lớp này .

• Nếu nó là công khai, tên của nó có thể được sử dụng bởi bất kỳ chức năng nào.

người dùng chung

các hàm thành viên và bạn bè của lớp dẫn xuất

chức năng thành viên riêng và bạn bè

public:

protected:

private:

**6.Quyền truy cập vào các lớp cơ sở**

Giống như một thành viên, một lớp cơ sở có thể được khai báo là riêng tư, được bảo vệ hoặc công khai. Ví dụ:

class X : public B { /\* ... \*/ };

class Y : protected B { /\* ... \*/ };

class Z : private B { /\* ... \*/

Các chỉ số truy cập khác nhau phục vụ các nhu cầu thiết kế khác nhau:

• công khai dẫn xuất làm cho lớp dẫn xuất trở thành một kiểu con của cơ sở của nó. Ví dụ, X là một loại B. Đây là dạng dẫn xuất phổ biến nhất.

• Các cơ sở riêng hữu ích nhất khi xác định một lớp bằng cách giới hạn giao diện cho một cơ sở để có thể cung cấp các đảm bảo mạnh mẽ hơn. Ví dụ, B là một chi tiết triển khai của Z .

• Các cơ sở được bảo vệ rất hữu ích trong các cấu trúc phân cấp lớp trong đó dẫn xuất thêm là tiêu chuẩn. Giống như dẫn xuất riêng, dẫn xuất được bảo vệ được sử dụng để biểu diễn các chi tiết triển khai.. Ví dụ:

class XX : B { /\* ... \*/ }; // B is a private base

struct YY : B { /\* ... \*/ }; // B is a public base

**I.1.6 Tóm tắt chương 21**

**CHƯƠNG 21: CẤU TRÚC 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. Đề thi được lấy từ thiết kế giao diện người dùng, nhưng tôi tránh chủ đề về lập trình hướng sự kiện thường được sử dụng cho các hệ thống giao diện người dùng đồ họa (GUI). Một cuộc thảo luận về chính xác cách một hành động trên màn hình được chuyển thành lời gọi của một hàm thành viên sẽ thêm ít vào các vấn đề của thiết kế phân cấp lớp. Để hiểu về GUI, hãy xem một trong nhiều thư viện C ++ GUI

**2. Thiết kế cấu trúc phân cấp lớp**

**User Ival\_box:**

Value

**(via ‘‘system’’) application**

**set\_value() get\_value()**

**3. Kế thừa triển khai**

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, chúng tôi khai báo dữ liệu cần thiết để triển khai khái niệm cơ bản:

**class Ival\_box {**

**protected:**

**int val;**

**int low, high;**

**bool chang ed {false}; // thay đổi bởi người dùng set\_value()**

**public:**

**Ival\_box(int ll, int hh) :val{ll}, low{ll}, high{hh} { }**

**virtual int get\_value() { chang ed = false; return val; } // cho ứng dụng**

**virtual void set\_value(int i) { chang ed = true; val = i; } // cho người dùng**

**virtual void reset\_value(int i) { chang ed = false; val = i; } // cho ứng dụng**

**virtual void prompt() { }**

**virtual bool was\_chang ed() const { return chang ed; }**

**virtual ̃Ival\_box() {};**

**};**

Việc triển khai mặc định của các chức năng là khá cẩu thả và được cung cấp ở đây chủ yếu để đánh lừa ngữ nghĩa dự định. Ví dụ, một lớp thực tế sẽ cung cấp một số kiểm tra phạm vi.

Một lập trình viên có thể sử dụng '' các lớp ival '' như thế này:

**void interact(Ival\_box∗ pb)// tương tác**

**{**

**pb−>prompt(); // aler t user**

**// ...**

**int i = pb−>get\_value();**

**if (pb−>was\_chang ed()) {**

**// ... new value; do something ...// giá trị khác , làm viễ gì đó**

**}**

**else {**

**// ... do something else ...// làm việc khác**

**}}**

**void some\_fct()**

**{**

**unique\_ptr<Ival\_box> p1 {new Ival\_slider{0,5}}; // Ival\_slider bắt nguồn Ival\_box**

**interact(p1.get());**

**unique\_ptr<Ival\_box> p2 {new Ival\_dial{1,12}};**

**interact(p2.get());**

**}**

Mã ứng dụng được viết dưới dạng (con trỏ tới) Ival\_boxes thuần túy như cách tương tác (). Bằng cách đó, ứng dụng không cần biết về số lượng lớn các biến thể tiềm năng của khái niệm Ival\_box. Điều này cách ly người dùng khỏi những thay đổi trong việc triển khai các lớp dẫn xuất.

Có thể chương trình thực sự đợi người dùng trong get\_value () (ví dụ: sử dụng get () trong tương lai, có thể chương trình liên kết Ival\_box với một sự kiện và chuẩn bị trả lời một lệnh gọi lại, hoặc có thể chương trình tạo ra một luồng cho Ival\_box và sau đó sẽ hỏi về trạng thái của luồng đó.

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 quyết định thanh trượt trông như thế nào, etc. ...**

**public:**

**Ival\_slider(int, int);**

**int get\_value() override; // lấy hàm từ người dùng và ứng dụng gửi vào val**

**void prompt() override;// ghi đè };**

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 một lớp và từ các thành viên của các lớp dẫn xuất, nhưng không phải đối với người dùng thông thường .

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 BBwid-get. 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 kiểu đồ họa, được thay đổi kích thước, được kéo xung quanh, v.v., phù hợp với tiêu chuẩn do hệ thống BBwidget thiết lập. Hệ thống phân cấp lớp của chúng ta sẽ trông như thế này:

**class Ival\_box : public BBwidget { /\* ... \*/ }; // viết lại người dùng để sử dụng Bbwidget**

**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 bằng đồ thị**:

BBwidget

Ival\_box

Ival\_slider Ival\_dial

Popup\_ival\_slider Flashing\_ival\_slider

**4.Nhiều lớp triển khai**

Một mô phỏng của các thiên thể quay quanh Trái đất, trong đó các vật thể quay xung quanh được thể hiện như đối tượng của lớp Vệ tinh. Một đối tượng Vệ tinh sẽ chứa quỹ đạo, kích thước, hình dạng, albedo, tham số mật độ, v.v. và cung cấp các hoạt động để tính toán quỹ đạo, sửa đổi thuộc tính, v.v.

**class Comm\_sat : public Satellite, public Displayed {**

**public: // ...**

**};**

**hoặc bằng đồ thị:**

Satellite Displayed

Comm\_sat

Việc thực hiện điều này rõ ràng liên quan đến một số kỹ thuật biên dịch (đơn giản) để đảm bảo rằng các chức năng mong đợi một Vệ tinh nhìn thấy một phần khác của Comm\_sat so với các chức năng mong đợi một Hiển thị. Các chức năng ảo hoạt động như bình thường. Ví dụ:

**class Satellite {**

**public:**

**virtual Pos center() const = 0; // center of gravity**

**// ...};**

**class Displayed {**

**public:**

**virtual void draw() = 0;**

**// ...};**

**class Comm\_sat : public Satellite, public Displayed {**

**public:**

**Pos center() const override; // overr ide Satellite::center()**

**void draw() override; // over ide Displayed::draw()**

**// ...};**

**I.2.TỔNG QUAN VỀ CHƯƠNG TRÌNH QUẢN LÝ ỨNG DỤNG**

**I.2.1 Lý do chọn đề tài**

Trong xã hội ngày càng phát triển hiện nay, ứng dụng công nghệ vào đời sống không còn gì là quá xa lạ đối với mỗi chúng ta. Với sự phát triển nhanh một cách không ngừng của công nghệ thông tin đã giúp giải quyết các công việc học tập, nguyên cứu, quản lý thông tin,… Một cách dễ dàng và tiện lợi. Thấy được tiềm năng đó các quốc gia, doanh nghiệp, trường học, các cá nhân, … đã ứng dụng nó vào thực tiển cuộc sống để giải quyết công việc, học tập, giải trí với những chiếc điện thoại thông minh nhỏ gọn. Trong những năm gần đây nhu cầu về các phần mềm, ứng dụng học tập, giải trí càng nhiều hơn do nhu cầu sử dụng điện thoại các thiết bị gọn nhẹ ngày càng cao và thuận tiện cho người sử dụng. Nhằm giúp đỡ người sử dụng cụ thể là các nhà quản trị tiện lợi trong việc rà soát, nắm thông tin của từng tiệm coffee mọi lúc mọi nơi mà không phụ thuộc quá nhiều vào sổ sách cồng kềnh. Ứng dụng giúp mọi người sử dụng một cách dễ dàng, giao diện thân thiện với người sử dụng.

**I.2.2 Mô tả bài toán**

**MÔ TẢ THUẬT TOÁN**

Class 1: Khách hàng

**Private:**

* Khách hàng gồm có:
* Mã khách hàng
* SĐT
* Ngày mua
* Số lượng hàng mua
* Mặt hàng

**Public:**

* Hàm nhập 1 khách hàng
* Hàm xuất một khách hàng
* Hàm định giá mặt hàng(tính luôn tổng tiền 1 người mua)
* Hàm lấy mã khách hàng từ private

**Ngoài class1 :**

Tạo một struct node

Tạo một struct list

Tạo một hàm list rỗng

Tạo hàm xuất 1 danh sách

Tạo hàm tạo 1 node

Tạo hàm thêm node vào cuối

Tạo hàm in hóa đơn khách hàng dựa vào mã

Class 2: Nguyên liệu( tổ chức theo kiểu dslk đơn gồm n ngày)

**Private:**

* Date( kiểu DL Date đã có cấu trúc định nghĩa ở file code chính)
* Cafee(kg)
* Trà(kg)
* Sữa(l)
* Đá(cây)
* Ly đựng(lốc)

**Public:**

* Hàm tạo nguyên liệu không đối số.

VD: NguyenLieu()

{

d.date=d.month=d.year=0;

Coffee=Tra=Sua=Da=LyDung=0;

}

* Hàm nhập số lượng nguyên liệu 1 ngày(1 node là 1 ngày)
* Hàm xuất số lượng nguyên liệu 1 ngày.
* Hàm tính tổng tiền nguyên liệu 1 ngày( Qui định café 80k/kg, trà 30k/kg, sữa 20k/l, đá 10k/1 cây, ly 10k/1 lốc)
* Hàm ghi file là hàm bạn của nguyên liệu

**Ngoài class 2:**

Tạo một struct node

Tạo một struct list

Tạo một hàm list rỗng

Tạo hàm tạo 1 node

Tạo hàm thêm node vào cuối

Hàm chỉnh sửa một node dựa vào tìm kiếm ngày cần chỉnh sửa

Hàm nhập 1 ds nguyên liệu

Hàm xuất ds nguyên liệu.

Hàm ghi File nguyên liệu ra txt.

Tạo hàm tính tổng tiền tất cả các ngày đã nhập( có thể viết hàm tính tổng tiền nguyên liệu 1 ngày trả về kiểu float sau đó cho chạy từ đầu đến cuối dslk đơn để tính tổng)

Lưu ý: thêm các âm thanh thông báo \7 khi có lỗi, Beep(523,500);cin.get(); khi xuất thông tin hoặc xuất tiền nguyên liệu tất cả thành công.

Định dạng các khoảng cách hiển thị như file code chính.

Xây dựng menu3 với hướng dẫn điều hướng như code chính.

**Nếu có thể áp dụng template vào chương trình**

Sửa lại hàm menu chính để đưa hàm menu 3 vào trong bảng điều hướng menu chính.

Class 3:Nhân viên

**Private:**

* Họ tên nhân viên
* Ngày sinh
* Địa chỉ
* SĐT
* Số lượng h đi làm
* Loại nhân viên(fulltime 25k/1 h parttime 17k/1h)

**Public:**

* Hàm nhập thông tin 1 nhân viên
* Hàm xuất thông tin 1 nhân viên
* Hàm tính tiền lương 1 nhân viên ( chú ý qui định mức lương 1 h dựa vào loại nhân viên)
* Hàm lấy tên nhân viên từ private

**Ngoài class**

Tạo một struct node

Tạo một struct list

Tạo một hàm list rỗng

Tạo hàm tạo 1 node

Tạo hàm thêm node vào cuối

Hàm tính lương nhân viên theo tuần( giống như cách tính tiền nguyên liệu theo tuần)

Hàm tìm nhân viên dựa và tên

Hàm menu: dựa vào các hàm chính ngoài class để thêm vào hàm menu

**I.3. DANH MỤC CHỮ VIẾT TẮT**

**I.4. BẢNG BIỂU PHÂN CÔNG CÔNG VIỆC**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Công việc  Họ và tên | Chương trình | | | Dịch sách |
| Code | Báo cáo | Powerpoint |
| Nguyễn Ngân Trúc | Class Nhân viên |  |  | Chương 20, 21 |
| Hà Diễm Quỳnh | Class  Khách hàng |  | Làm slide | Chương 18, 19 |
| Nguyễn Thị Vân Khánh | Class Nguyên liệu | Làm báo cáo |  | Chương 16, 17 |

**CHƯƠNG II: CHƯƠNG TRÌNH ỨNG DỤNG**

**II.1 Cơ sở lý thuyết:**

* **Tính chất hướng đối tượng:**

**Tính đóng gói**

Đây là tính chất không cho phép người dùng tác động trực tiếp đến dữ liệu bên trong Object, mà phải thông qua các phương thức nó cung cấp và luôn đảm bảo tính toàn vẹn của mọi đối tượng.

Ngoài ra, tính đóng gói còn giúp che giấu một số thông tin chi tiết về cài đặt nội bộ để bên ngoài không thể nhìn thấy.

**Tính kế thừa**

Tính kế thừa cho phép xây dựng một lớp mới dựa trên những định nghĩa đã có. Hiểu đơn giản như lớp cha có thể chia sẽ dữ liệu, phương thức cho lớp con. Tại đây, chúng không phải định nghĩa lại, ngoài ra có thể mở rộng các thành phần kế thừa, bổ sung thêm những hình thức mới. Đây là tính chất giúp con người tránh việc code lặp lại, dư thừa mà chỉ xử lý công việc tương tự.

* **Cấu trúc dữ liệu:**

**Danh sách liên kết đơn**

[1. Lý thuyết về danh sách liên kết](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "1-ly-thuyet-ve-danh-sach-lien-ket" \o "1. Lý thuyết về danh sách liên kết)

[2. Danh sách liên kết là gì?](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "2-danh-sach-lien-ket-la-gi" \o "2. Danh sách liên kết là gì?)

[3. Cài đặt danh sách liên kết đơn](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "3-cai-dat-danh-sach-lien-ket-don" \o "3. Cài đặt danh sách liên kết đơn)

[3.1. Khai báo linked list](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "31-khai-bao-linked-list" \o "3.1. Khai báo linked list)

[3.2. Tạo mới 1 Node](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "32-tao-moi-1-node" \o "3.2. Tạo mới 1 Node)

[3.3. Thêm Node vào danh sách liên kết](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "33-them-node-vao-danh-sach-lien-ket" \o "3.3. Thêm Node vào danh sách liên kết)

[Thêm vào đầu](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "them-vao-dau" \o "Thêm vào đầu)

[Thêm vào cuối](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "them-vao-cuoi" \o "Thêm vào cuối)

[3.4. Tìm kiếm trong danh sách liên kết](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "36-tim-kiem-trong-danh-sach-lien-ket" \o "3.6. Tìm kiếm trong danh sách liên kết)

[3.5. Duyệt danh sách liên kết](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "37-duyet-danh-sach-lien-ket" \o "3.7. Duyệt danh sách liên kết)

[3.6. Một số hàm bổ trợ khác](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "38-mot-so-ham-bo-tro-khac" \o "3.8. Một số hàm bổ trợ khác)

[Hàm khởi tạo Node head](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "ham-khoi-tao-node-head" \o "Hàm khởi tạo Node head)

[Hàm lấy số phần tử của DSLK](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "ham-lay-so-phan-tu-cua-dslk" \o "Hàm lấy số phần tử của DSLK)

[Hàm nhập danh sách liên kết](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "ham-nhap-danh-sach-lien-ket" \o "Hàm nhập danh sách liên kết)

[4. Code hoàn chỉnh của danh sách liên kết đơn](https://nguyenvanhieu.vn/danh-sach-lien-ket-don/" \l "4-code-hoan-chinh-cua-danh-sach-lien-ket-don" \o "4. Code hoàn chỉnh của danh sách liên kết đơn)

## 1. Lý thuyết về danh sách liên kết

Về bản chất, danh sách liên kết có chức năng như một mảng, có thể thêm và xóa các phần tử ở bất kỳ vị trí nào khi cần thiết. Một số sự khác nhau giữa danh sách liên kết và mảng:

|  |  |  |
| --- | --- | --- |
| ****Nội dung**** | ****Mảng**** | ****Danh sách liên kết**** |
| Kích thước | * Kích thước cố định * Cần chỉ rõ kích thước trong khi khai báo | * Kích thước thay đổi trong quá trình thêm/ xóa phần tử * Kích thước tối đa phụ thuộc vào bộ nhớ |
| Cấp phát bộ nhớ | * Tĩnh: Bộ nhớ được cấp phát trong quá trình biên dịch | * Động: Bộ nhớ được cấp phát trong quá trình chạy |
| Thứ tự & sắp xếp | * Được lưu trữ trên một dãy ô nhớ liên tục | * Được lưu trữ trên các ô nhớ ngẫu nhiên |
| Truy cập | * Truy cập tới phần tử ngẫu nhiên trực tiếp bằng cách sử dụng chỉ số mảng: O(1) | * Truy cập tới phần tử ngẫu nhiên cần phải duyệt từ đầu/cuối đến phần tử đó: O(n) |
| Tìm kiếm | * Tìm kiếm tuyến tính hoặc tìm kiếm nhị phân | * Chỉ có thể tìm kiếm tuyến tính |

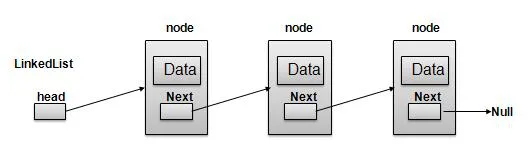
## 2. Danh sách liên kết là gì?

Danh sách liên kết đơn là một tập hợp các Node được phân bố động, được sắp xếp theo cách sao cho mỗi Node chứa “một giá trị”(Data) và “một con trỏ”(Next). Con trỏ sẽ trỏ đến phần tử kế tiếp của danh sách liên kết đó. Nếu con trỏ mà trỏ tới NULL, nghĩa là đó là phần tử cuối cùng của linked list.

Hình ảnh mô tả cho một Node trong danh sách liên kết đơn:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/data-node-danh-sach-lien-ket.png)

Và đây là hình ảnh mô phỏng một danh sách liên đơn kết đầy đủ:

[](https://nguyenvanhieu.vn/wp-content/uploads/2018/12/danh-sach-lien-ket-la-gi.jpg)

Mô phỏng của danh sách liên kết đơn

Danh sách các kiểu danh sách liên kết:

* Danh sách liên kết đơn(Single linked list): Chỉ có sự kết nối từ phần tử phía trước tới phần tử phía sau.
* Danh sách liên kết đôi(Double linked list): Có sự kết nối 2 chiều giữa phần tử phía trước với phần tử phía sau
* Danh sách liên kết vòng(Circular Linked List): Có thêm sự kết nối giữa 2 phần tử đầu tiên và phần tử cuối cùng để tạo thành vòng khép kín.

## 3. Cài đặt danh sách liên kết đơn

### 3.1. Khai báo linked list

Khai báo trên sẽ được sử dụng cho mọi Node trong linked list. Trường data sẽ lưu giữa giá trị và next sẽ là con trỏ để trỏ đến thằng kế tiếp của nó.

****Tại sao next lại là kiểu LinkedList của chính nó?****Bởi vì nó là con trỏ trỏ của chính bản thân nó, và nó trỏ tới một thằng Node kế tiếp cũng có kiểu LinkedList.

### 3.2. Tạo mới 1 Node

Mỗi một Node khi được khởi tạo, chúng ta cần cấp phát bộ nhớ cho nó, và mặc định cho con trỏ next trỏ tới NULL. Giá trị của Node sẽ được cung cấp khi thêm Node vào linked list.

* ****typedef**** được dùng để định nghĩa một kiểu dữ liệu trong C. VD: typeder long long LL;
* ****malloc**** là hàm cấp phát bộ nhớ của C. Với C++ chúng ta dùng new
* ****sizeof**** là hàm trả về kích thước của kiểu dữ liệu, dùng làm tham số cho hàm malloc

****Lưu ý:**** Không giống với mảng, cần khai báo arr[size]. Trong linked list, vì mỗi Node sẽ có con trỏ liên kết đến Node tiếp theo. Do đó, với danh sách liên kết đơn, bạn chỉ cần lưu giữ Node đầu tiên(HEAD). Có head rồi bạn có thể đi tới bất cứ Node nào.

### 3.3. Thêm Node vào danh sách liên kết

#### Thêm vào đầu

Việc thêm vào đầu chính là việc cập nhật lại thằng head. Ta gọi Node mới(temp), ta có:

* Nếu head đang trỏ tới NULL, nghĩa là linked list đang trống, Node mới thêm vào sẽ làm head luôn
* Ngược lại, ta phải thay thế thằng head cũ bằng head mới. Việc này phải làm theo thứ tự như sau:
  + Cho next của temp trỏ tới head hiện hành
  + Đặt temp làm head mới

#### Thêm vào cuối

Chúng ta sẽ cần Node đầu tiên, và giá trị muốn thêm. Khi đó, ta sẽ:

1. Tạo một Node mới với giá trị value
2. Nếu head = NULL, tức là danh sách liên kết đang trống. Khi đó Node mới(temp) sẽ là head luôn.
3. Ngược lại, ta sẽ duyệt tới Node cuối cùng(Node có next = NULL), và trỏ next của thằng cuối tới Node mới(temp).

Tổng quan hơn, chúng ta sẽ sẽ viết hàm thêm một Node vào vị trí bất kỳ nhé.

Lưu ý: Bạn phải làm theo thứ tự trên, nếu bạn cho p->next = temp trước. Khi đó, bạn sẽ không thể lấy lại phần sau của danh sách liên kết nữa(Vì next chỉ được được lưu trong p->next mà thay đổi p->next rồi thì còn đâu giá trị cũ).

### 3.4. Tìm kiếm trong danh sách liên kết

Hàm tìm kiếm này sẽ trả về chỉ số của Node đầu tiên có giá trị bằng với giá trị cần tìm. Nếu không tìm thấy, chúng ta trả về -1.

Chúng ta có thể sử dụng hàm này để xóa tất cả các Node trong danh sách liên kết có giá trị chỉ định như sau:

### 3.5. Duyệt danh sách liên kết

Việc duyệt danh sách liên kết cực đơn giản. Khởi tạo từ Node head, bạn cứ thế đi theo con trỏ next cho tới trước khi Node đó NULL.

### 3.6. Một số hàm bổ trợ khác

#### Hàm khởi tạo Node head

Đơn giản là cho con trỏ head = NULL thôi. Nếu bạn để ý, chúng ta vẫn check head = NULL để biết rằng danh sách liên kết chưa có phần tử nào ở các hàm phía trên.

#### Hàm lấy số phần tử của DSLK

Duyệt và đếm chừng nào các Node chưa NULL. Sau cùng, trả về giá trị đếm được.

#### Hàm nhập danh sách liên kết

## 4. Code hoàn chỉnh của danh sách liên kết đơn

Kết quả chạy thử:

* **Đồ họa:**

**II.2 Hướng dẫn cài đặt:**

**II.3 Hướng dẫn sử dụng:**

**CHƯƠNG III: KẾT LUẬN**

**III.1 Kết quả đạt được:**

Nhóm chúng em đã hoàn thành xong chương trình quản lý tiệm coffee với các chức năng cơ bản khác nhau gồm: In hóa đơn khách hàng dựa vào mã, ghi file nguyên liệu ra txt, tính lương nhân viên, …

**III.2 Nhược điểm:**

Mặc dù chúng em đã cố gắng hết sức nhưng chắc chắn bài tập lớn khó có thể tránh khỏi những thiếu sót và nhiều vấn đề còn chưa chính xác, kính mong cô xem xét và góp ý để bài báo cáo của chúng em được hoàn thiện hơn.

**III.3 Hướng phát triển:**

Nhóm chúng em sẽ cố gắng học hỏi và phát triển chương trình quản lý tiệm coffee với nhiều chức năng hơn, ứng dụng vào thực tế nhiều hơn, thoả mãn nhu cầu của người sử dụng trong xã hội.

**CHƯƠNG IV: PHỤ LỤC:**

**Tài liệu tham khảo:**

1. Website: <https://nguyenvanhieu.vn/>
2. Slide bài giảng môn Lập trình hướng đối tượng
3. Slide bài giảng môn Cấu trúc dữ liệu và giải thuật