Lời cảm ơn

Trước tiên, tôi xin bày tỏ lòng biết ơn sâu sắc tới thầy giáo hướng dẫn TS Đoàn Văn Ban, phòng CSDL&LT Viện Công Nghệ Thông Tin thuộc trung tâm Khoa Học Tự Nhiên và Công Nghệ Quốc Gia đã tận tình giúp đỡ tôi hoàn thành bài luận văn này.

Tôi xin chân thành cảm ơn các thầy, cô giáo khoa Công Nghệ Thông Tin trường ĐHDL Đông Đô đã giảng dạy và giúp đỡ em trong quá trình học tập ở trường.

Cuối cùng, xin chân thành cảm ơn những người thân trong gia đình và bạn bè đã giúp đỡ, động viên trong quá trình học tập.

Hà nội tháng 6 năm 2000

Mục lục

Phần B

CHƯƠNG I

NGÔN NGỮ C++ VÀ LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

I.1. Lập trình hướng đối tượng là gì?

Lập trình hướng đối tượng dựa trên nền tảng là các đối tượng. Đối tượng được xây dựng trên cơ sở gắn cấu trúc dữ liệu với các phép toán sẽ thể được đúng cách mà chúng ta suy nghĩ, bao quát về thế giới thực. [3]

Lập trình hướng đối tượng cho phép chúng ta kết hợp những tri thức bao quát về các quá trình với những khái niệm trừu tượng được sử dụng trong máy tính .

Lập trình hướng đối tượng là phương pháp lập trình lấy đối tượng làm nền tảng để xây dựng thuật giải, xây dựng chương trình, là cách tiếp cận để phân chia chương trình thành các đơn thể (modul) bằng cách tạo ra các vùng bộ nhớ cho cả dữ liệu lẫn hàm và chúng sẽ được sử dụng như các mẫu để tạo ra bản sao từng đơn thể khi cần thiết. Đối tượng ở đây được xem như là vùng phân chia chia bộ nhớ trong máy tính để lưu trữ dữ liệu và tập các hàm tác động trên dữ liệu gắn với chúng.

Khái niệm "Hướng đối tượng" được xây dựng trên nền tảng của khái niệm "Lập trình có cấu trúc" và "Sự trừu tượng hoá dữ liệu" sự thay đổi căn bản là ở chỗ một chương trình hướng đối tượng được thiết kế xoay quanh các dữ liệu mà ta làm việc trên nó, hơn là theo bản thân chức năng của chương trình.

Lập trình hướng đối tượng đặt trọng tâm vào đối tượng, yếu tố quan trọng trong quá trình phát triển chương trình và nó không cho phép dữ liệu chuyển động tự do trong hệ thống. Dữ liệu được gắn chặt với từng hàm thành các vùng riêng mà các hàm đó tác động lên và nó được bảo vệ cấm các hàm ngoại lai truy nhập tuỳ tiện. Tuy nhiên các đối tượng có thể trao đổi thông tin với nhau thông qua việc trao đổi thông báo.[5]

Tóm lại, so sánh lập trình cấu trúc lấy chương trình con làm nền tảng:

Chương trình = Cấu trúc dữ liệu + Giải Thuật

Trong lập trình hướng đối tượng chúng ta có :

Đối tượng =Dữ Liệu + Hành vi của dữ liệu

Lập trình hướng đối tượng có những đặc tính chủ yếu sau:

- Tập trung vào dữ liệu thay cho các hàm.
- Chương trình được chia thành tập các lớp đối tượng.
- Cấu trúc dữ liệu được thiết kế sao cho đặc tả các đối tượng.
- Các hàm được xác định trên các vùng dữ kiệu của đối tượng được gắn với nhau trên cấu trúc của dữ liệu đó.
- Dữ liệu được bao bọc, che dấu và không cho phép các hàm ngoại lai truy nhập tự do.
- Các đối tượng trao đổi thông tin với nhau qua các hàm.
- Dữ liệu và các hàm mới có thể dễ dàng bổ xung vào đối tượng nào đó khi cần thiết.
- Chương trình được thiết kế theo cách tiếp cận bottom-up.

I.2. Các ưu điểm của lập trình hướng đối tượng

- Thông qua nguyên lý thừa kế, chúng ta có thể loại bỏ được những đoạn chương trình lặp lại, dư thừa trong quá trình mô tả các lớp và khả năng sử dụng các lớp đã được xây dựng.
- Chương trình được xây dựng từ các đơn thể (module) trao đổi với nhau nên việc thiết kế và lập trình sẽ được thực hiện theo quy trình

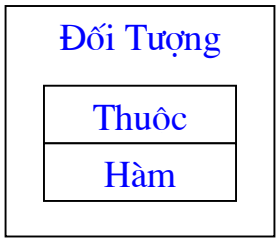
nhất định chứ không phải dựa vào kinh nghiệm và kỹ thuật như trước. Điều này đảm bảo rút ngắn được thời gian xây dựng hệ thống và tăng năng xuất lao động.

- Nguyên lý che dấu thông tin giúp người lập trình tạo ra được những chương trình an toàn không bị thay đổi bởi những chương trình khác.
- Có thể xây dựng được các ánh xạ đối tượng của bài toán vào đối tượng của chương trình.
- Cách tiếp cận thiết kế đặt trọng tâm vào dữ liệu giúp ta xây dựng được mô hình chi tiết và gần với dạng cài đặt hơn.
- Những hệ thống hướng đối tượng dễ mở rộng, nâng cấp thành những hệ thống lớn hơn.
- Kỹ thuật truyền thông báo trong việc tao trao đổi thông tin giữa các đối tượng giúp cho việc mô tả giao diện với các hệ thống bên ngoài đơn giản hơn.
- Có thể quản lý độ phức tạp của những sản phẩm phần mềm.

I.3. Đối tượng

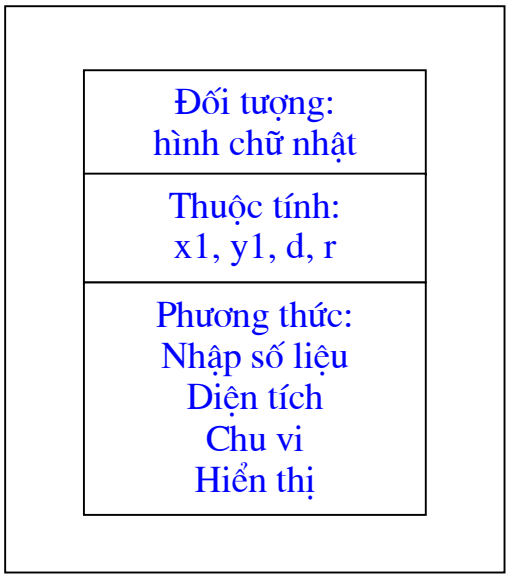
Đối tượng là thực thể được xác định trong thời hạn hệ thống hướng đối tượng hoạt động. Như vậy đối tượng là con người, sự vật, thiết bị, bảng dữ liệu cần xử lý trong chương trình. Mỗi đối tượng gồm có: tập các thuộc tính (attribute) và tập các hàm (function) để xử lý các thuộc tính đó.[5]

Một đối tượng có thể được minh hoạ như sau :



Hình 1. Cấu trúc tổng quát của một đối tượng.

Chẳng hạn chúng ta xét đối tượng hình chữ nhật bao gồm các thuộc tính (x1,y1) toạ độ góc trên bên trái, d, r là chiều dài chiều rộng của hình chữ nhật. Các hàm: nhập số liệu cho hình chữ nhật, hàm tính diện tích, chu vi và hàm hiển thị. Như vậy đối tượng hình chữ nhật có thể được mô tả như sau:



Hình 2. Mô tả đối tượng hình chữ nhật.

I.4. Các lớp đối tượng

Một tập dữ liệu và các hàm của một tập đối tượng có thể được xem như một kiểu dữ liệu được định nghĩa bởi người sử dụng. Kiểu dữ liệu ở đây được gọi là lớp (class), đó là một tập các thuộc tính và các hàm mô tả thế giới thực, một đối tượng là thể hiện của một lớp. Lớp là khái niệm trung tâm của lập trình hướng đối tượng, nó là sự mở rộng cấu trúc (struct) của C và bản ghi (record) của Pascal. Trong lập trình hướng đối tượng, lớp hầu như đồng nhất với kiểu dữ liệu trừu tượng. Lớp là khái niệm tĩnh, có thể nhận biết ngay từ văn bản chương trình. Ngược lại đối tượng là khái niệm động, nó được xác định trong bộ nhớ của máy tính nơi đối tượng chiếm một vùng của bộ nhớ lúc thực hiện chương trình. Đối tượng được tạo ra để xử lý thông tin, thực hiện nhiệm vụ được thiết kế và sau đó bị huỷ bỏ khi đối tượng đó hết vai trò. Khi một lớp được định nghĩa, thì nó có thể tạo ra số

lượng các đối tượng tuỳ ý của lớp đó. Như vậy lớp là tập hợp các đối tượng cùng kiểu. Sự khác biệt giữa lớp và đối tượng cũng giống như sự khác biệt giữa tập hợp các phần tử và một phần tử trong tập hợp.[5]

I.5. Trừu tượng hoá dữ liệu và bao gói thông tin

Việc đóng gói dữ liệu và các hàm vào một đơn vị cấu trúc được xem như một nguyên tắc (che dấu) thông tin, dữ được tổ chức sao cho thế giới bên ngoài không truy nhập được vào mà chỉ cho phép các hàm trong cùng lớp hoặc trong những lớp có quan hệ thừa với nhau được quền truy nhập. Chính các hàm thành phần của lớp sẽ đóng vai trò như là giao diện giữa dữ liệu của đối tượng và phần còn lại của chương trình. Nguyên tắc bao gói dữ liệu để ngăn cấm sự truy nhập trực tiếp trong lập trình được gọi là che dấu thông tin.

Trừu tượng hoá là cách biểu diễn những đặc tính chính và bỏ qua những chi tiết vụn vặt hoặc những giải thích. Để xây dựng các lớp chúng ta phải sử dụng khái niệm trừu tượng hoá. Trong lập trình hướng đối tượng lớp được sử dụng như dữ liệu trừu tượng. Ví dụ như chúng ta có thể định nghĩa một lớp là danh sách các thuộc tính trừu tượng như là kích thước, hình dáng mầu và các hàm xác định trên các thuộc tính này để mô tả các đối tượng trong không gian hình học.

I.6. Thừa Kế

Thừa kế là quá trình trong đó các đối tượng của lớp này được quyền sử dụng một số tính chất của các đối tượng của các lớp khác.

Nguyên lý thừa kế hỗ trợ cho việc tạo ra cấu trúc phân cấp các lớp. Nó được thực hiện dựa trên nguyên lý tổng quát hoá hoặc chi tiết hoá các đặc tính của các đối tượng trong các lớp.

Trong lập trình hướng đối tượng, khái niệm thừa kế kéo theo ý tưởng sử dụng lại. Nghĩa là một lớp đã được xây dựng (lớp cha hay lớp cơ sở) của chúng có thể bổ sung thêm các tính chất mới để tạo các lớp mới (lớp con hay lớp dẫn xuất) mô tả chi tiết hơn về một nhóm đối tượng cụ thể (theo nguyên lý chi tiết hoá) hoặc từ một nhóm lớp có số đặc tính giống nhau gộp chung các đặc tính đó lại để tạo ra một lớp mới, được gọi là lớp trừu tượng (nguyên lý tổng quát hoá).

Khái niệm kế thừa được hiểu như cơ chế sao chép ảo không đơn điệu. Trong thực tế, mọi việc xảy ra tựa như những lớp cơ sở đều được sao vào trong lớp dẫn xuất mặc dù điều này không được cài đặt tường minh (gọi là sao chép ảo) và việc sao chép chỉ được xác định trong lớp cơ sở (sao chép không đơn điệu).

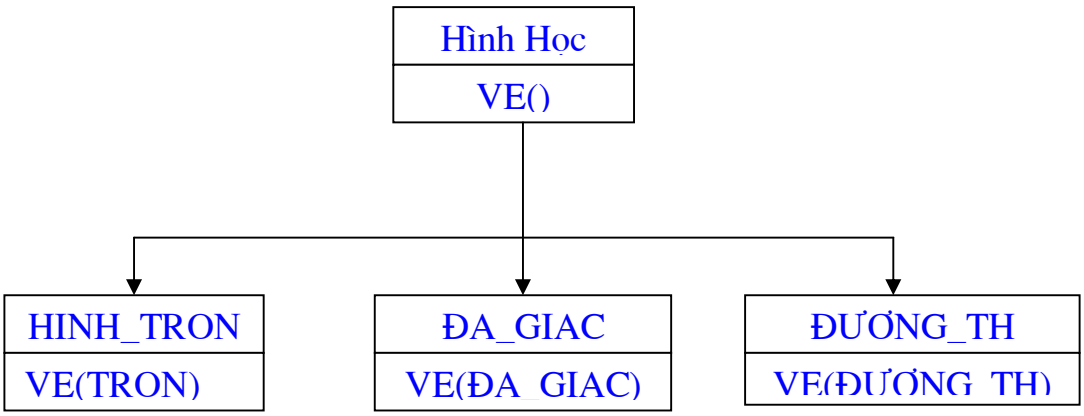
Một lớp có thể kế thừa các tính chất của một hay nhiều lớp cơ sở ở các mức khác nhau, do đó có năm dạng kế thừa được sử dụng trong lập trình hướng đối tượng là: kế thừa đơn, kế thừa bội, kế thừa phân cấp, kế thừa đa mức và kế thừa phức hợp (chương sau sẽ nói rõ về các dạng kế thừa này).[3]

I.7. Tương ứng bội

Tương ứng bội là một khái niệm có khả năng như các phép toán có thể được thực hiện ở nhiều dạng khác nhau. Hành vi của các phép toán tương ứng bội phụ thuộc vào kiểu dữ liệu mà nó sử dụng để xử lý. Tương ứng bội đóng vai trò quan trọng trong việc tạo ra các đối tượng có cấu trúc bên trong khác nhau nhưng có khả năng dùng chung một giao diện bên ngoài (như tên gọi). Điều này có nghĩa là một lớp các phép toán được định nghĩa theo những thuật toán khác nhau, nhưng có khả năng sử dụng theo cùng một cách giống nhau.Tương ứng bội là sự mở rộng khái niệm sử dụng lại trong nguyên lý kế thừa. Liên kết động là dạng liên kết các hàm, thủ tục khi chương trình thực hiện các lời gọi tới các hàm, thủ tục đó. Như vậy, trong liên kết động nội dung của đoạn chương trình ứng với thủ tục, hàm cho đến khi thực hiện các lời gọi tới các thủ tục và hàm đó. Nó cho phép chúng ta can thiệp vào sự hoạt động của các thực thể mà không cần biên dịch lại toàn bộ chương trình, chúng ta có thể truyền và nhận thông tin từ các đối tượng mới này giống như các đối tượng đã có. Liên kết động liên

quan chặt chẽ tới tương ứng bội và kế thừa, đôi khi liên kết động còn gọi là ***liên kết trễ*** hay liên kết vào lúc chạy (vì các phương thức chỉ được gọi vào lúc chương trình biên dịch chương trình biên dịch ra ngôn ngữ máy).[3]

Chẳng hạn như hàm VE() trong hình 3, theo nguyên lý kế thừa thì mọi đối tượng đều có thể sử dụng hàm này để vẽ theo yêu cầu. Tuy nhiên, thuật toán thực hiện hàm VE() là duy nhất đối với từng đối tượng HÌNH_TRÒN, ĐA_GIÁC, ĐƯỜNG_TH và vì vậy hàm VE() sẽ được định nghĩa lại khi các đối tượng tương ứng được xác định.



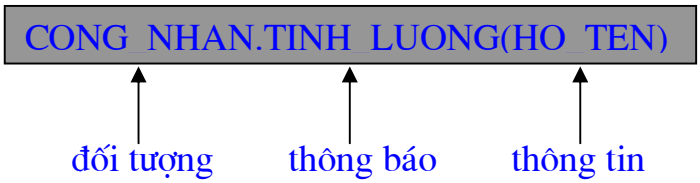
Hình 3. Tương ứng bội của hàm VE().

I.8. Truyền thông báo

Các đối tượng gửi và nhận thông tin với nhau giống như con người trao đổi thông tin với nhau. Chính nguyên lý trao đổi thông tin với nhau bằng cách truyền thông báo cho phép chúng ta dễ dàng xây dựng được hệ thống mô phỏng gần những hệ thống trong thế giới thực. Truyền thông báo cho một đối tượng tức là báo cho nó phải thực hiện một việc gì đó. Cách ứng xử cả đối tượng sẽ được mô tả ở trong lớp thông qua các hàm (hay còn được gọi là lớp dịch vụ). Thông báo truyền đi phải chỉ ra được hàm cần thực

hiện trong đối tượng nhận thông báo. Hơn thế nữa thông báo truyền đi phải xác định tên đối tượng, tên hàm và thông tin truyền đi.

Ví dụ: Lớp CONG_NHAN có thể là đối tượng cụ thể được xác định bởi HO_TEN nhận được thông báo cần TINH_LUONG đã được xác định trong lớp CONG_NHAN. Thông báo đó sẽ được xử lý như sau:



Mỗi đối tượng chỉ tồn tại trong một thời gian nhất định. Đối tượng tạo ra khi nó được khai báo và sẽ bị huỷ bỏ khi chương trình ra khỏi miền xác định của đối tượng đó. Sự trao đổi thông tin chỉ có thể thực hiện trong thời gian đối tượng tồn tại.

I.9. Những ứng dụng của lập trình hướng đối tượng

Lập trình hướng đối tượng là một trong những thuật ngữ được nhắc đến nhiều nhất trong công nghệ phần mềm và nó được ứng dụng để phát triển phần mềm và nhiều lĩnh vực khác nhau. Trong số đó có ứng dụng quan trọng và nổi tiếng nhất hiện nay là lĩnh vực thiết kế giao diện với người sử dụng. Ví dụ như Windows, hàng trăm hệ thống với giao diện Windows đã được phát triển dựa trên kỹ thuật lập trình hướng đối tượng. Những hệ thông tin doanh nghiệp trong thực tế rất phức tạp, chứa nhiều đối tượng, các thuộc tính và hàm. Để giải quyết những hệ thống phức hợp như thế thì lập trình hướng đối tượng lại tỏ ra khá hiệu quả. Tóm lại những lĩnh vực ứng dụng của kỹ thuật lập trình hướng đối tượng bao gồm:

- Những hệ thông tin làm việc theo thời gian thực.
- Trong lĩnh vực mô hình hoá hoặc mô phỏng quá trình.
- Các cơ sở dữ liệu hướng đối tượng.
- Hệ siêu văn bản và đa phương tiện.

- Lĩnh vực trí tuệ nhân tạo và các hệ chuyên gia.
- Lập trình song song và các mạng nơ_ron.
- Những hệ tự động hoá văn phòng và trợ giúp quyết định.
- Những hệ CAD/CAM.

Với nhiều đặc tính của lập trình hướng đối tượng nói riêng, cả phương pháp phát triển hướng đối tượng nói chung, chúng ta hy vọng nền công nghiệp phần mềm sẽ cải tiến không những về chất lượng mà còn gia tăng nhanh về số lượng trong tương lai. Kỹ nghệ hướng đối tượng sẽ là thay đổi cách suy nghĩ và cách thực hiện quá tình phân tích, thiết kế và cài đặt các hệ thống, góp phần giải quyết những vấn đề tồn tại trong công nghệ phần mềm.

C++ là một công cụ lập trình hướng đối tượng
C++ là công cụ lập trình hướng đối tượng. Ban đầu được gọi là "C with class" (C với các lớp) sau đó C++ được phát triển vào những năm đầu thập kỉ 80 ở AT&T Bell Laboratories. Nó được phát triển trên nền ngôn ngữ C. C++ là một tập mở rộng của C, vì thế hầu hết các tính chất của C vẫn được sử dụng trong C++. Điều này có nghĩa là hầu như toàn bộ các chương trình được viết bằng C thì cũng là chương trình của C++ . Tuy nhiên cũng có một số khác biệt làm cho chương trình C không thực được dưới chương trình C++.

Ba khái niệm quan trọng của C++ được bổ xung vào C là: lớp, hàm tải bội và toán tử tải bội. Những khái niệm cho phép chúng ta tạo ra những kiểu dữ liệu trừu tượng, kế thừa nhiều tính chất của những kiểu dữ liệu đã xây dựng và hỗ trợ cho việc sử dụng cơ chế tương ứng bội cho C++ trở thành ngôn ngữ hướng đối tượng thực sự. Các đặc tính của C++ cho phép người lập trình dễ dàng xây dựng được các chương trình lớn, có tính mở, dễ thích nghi, công việc bảo trì ít tốn kém hơn. C++ là công cụ thích ứng cho vệc xây dựng các chương trình lớn như các hệ soạn thảo chương, trình dịch, các hệ cơ sở dữ liệu, những hệ thống truyền tin và nhiều ứng dụng phức tạp khác.

C++ hỗ trợ cho việc tạo ra cấu trúc phân cấp các đối tượng giúp chúng ta có thể xây dựng những thư viện các đối tượng để cho nhiều người lập sử

dụng được. Do C++ là ngôn ngữ lập trình hướng đối tượng nên tất nhiên nó sẽ có những gì mà "hướng đối tượng" có.

CHƯƠNG II

THIẾT KẾ VÀ CÀI ĐẶT CÁC LỚP ĐỐI TƯỢNG TRONG C++

II.1. Định nghĩa lớp

II.1.1. Khai báo lớp tên đối tượng

Khai báo lớp là mô tả kiểu và nhiều miền xác định các thành phần của lớp, khai báo lớp cũng như khai báo các kiểu dữ liệu quen thuộc khác, nó có dạng như sau:

```
class class_name
{
private: // vùng riêng (mặc định)
... // khai báo dữ liệu và các hàm
public:// vùng công cộng
... // khai báo dữ liệu và các hàm
};
// kết thúc bằng dấu ";"
```

Từ khoá class định nghĩa một kiểu dữ liệu trừu tượng có tên gọi là class_name. Trong nội dung lớp có các biến và hàm, các biến và hàm được tổ chức thành hai nhóm: dùng chung (public) và sở hữu riêng (private, protected). Những thành phần được khai báo private, protected chỉ được truy nhập bởi các hàm thành phần của lớp, riêng đối với các hàm kiểu protected thì cho phép các hàm thành phần trong các lớp có quan hệ kế thừa (lớp dẫn xuất) mới được truy nhập. Ngược lại các thành phần kiểu public có thể được truy nhập từ bên ngoài lớp. Nguyên lí che dấu thông tin của C++ được thực hiện bằng cách sử dụng dạng khai báo private hoặc protected. Từ khoá private là tuỳ chọn, nếu mặc định thì những thành phần không được khai báo là public sẽ là private.

Những biến được khai báo trong các lớp được gọi là dữ liệu thành phần, còn các hàm được gọi là hàm thành phần. Các hàm thành phần kiểu private chỉ có thể truy nhập được các dữ liệu và hàm trong vùng private, còn các hàm thành phầm kiểu public thì truy nhập được tất cả các kiểu dữ liệu và các hàm trong cùng lớp.

Trong lớp Point thì các biến được khai báo trong vùng private (vùng riêng) là dữ liệu thành phần, còn các hàm trong vùng public (vùng chung) là các thành phần. Dữ liệu thành phần của lớp không thể có kiểu của chính lớp đó, nhưng có thể là kiểu con trỏ của lớp này, ví dụ:

```
class A
```

```
{
A x; // không cho phép, vì x không có kiểu lớp A
A *p; // cho phép vì p là con trỏ kiểu lớp A
};
```

dữ liệu thành phần của lớp cũng có thể là liểu của lớp khác.

Ví dụ:

```
class A
{
-   - - -
};
class B
{
A a:
-   - - -
};
```

II.1.2. Tạo lập các đối tượng

Trong C++, một đối tượng là một phần tử dữ liệu được khai báo kiểu là một class. Trong ví dụ trên, khai báo lớp Point mới chỉ xác định các thành phần của lớp Point chứ chưa tạo ra một đối tượng cụ thể. Lớp là một kiểu đối tượng trừu tượng, nên sau khi định nghĩa lớp chúng ta có thể khai báo các biến giống như đối với kiểu được định nghĩa bởi người sử dụng.

Khi các đối tượng được tạo lập thì sẽ có một chùm bộ nhớ được cấp phát để chứa các thành phần dữ liệu của mỗi đối tượng. Các đối tượng của cùng một lớp có thể được khởi đầu và gán cho một đối tượng khác. theo ngầm định, việc sao một đối tượng là tương đương với việc sao một thành phần của nó.

Các đối tượng còn có thể được cấp phát động trong heap, giống như những phần tử khác.

II.1.3. Các thành phần dữ liệu

Các thành phần dữ liệu trong một lớp không thể được khai báo kiểu auto, register, hay extern. Chúng có thể là các enum, nhóm bit và các kiểu dữ liệu có sẵn hoặc của người sử dụng định nghĩa. Thành phần dữ liệu cũng có thể là một đối tượng, tuy nhiên chúng chỉ có thể là những đối tượng của các lớp đã được khai báo hoặc đã được định nghĩa trước đó.

**Dữ liệu thành phần kiểu private*

Khi các thành phần dữ liệu một lớp được khai báo theo kiểu private thì chỉ có thánh phần của chính lớp đó hoặc các lớp bạn của nó mới được truy nhập đến các thành phần này.

**Dữ liệu thành phần public*

Nếu ta khai báo lại các thành phần dữ liệu trong lớp Point sang dạng public, tức là: Lúc đó mội thành phần trong lớp Point đều có thể được truy nhập từ thế giới bên ngoài.

**Dữ liệu thành phần protected*

Để sử dụng được các thành phần dữ liệu của một lớp từ các lớp khác nhưng phải bảo đảm nguyên lý che dấu thông tin thì chúng ta phải khai báo kiểu protected. Các thành phần dữ liệu trong protected chỉ cho phép các thành phần trong cùng lớp và trong dẫn xuất truy nhập đến.

**Dữ liệu thành phần tĩnh static*

Dữ liệu thành phần tĩnh là dữ liệu được khai báo với từ khoá static ở đầu. Khi dữ liệu thành phần tĩnh thì tất cả các thể hiện của lớp đó đều được phép dùng chung thành phần dữ liệu này. Dữ liệu theo kiểu static được phân bố ở một vùng bộ nhớ cố định trong quá trình liên kết cũng giống như các biến được khai báo theo kiểu tổng thể (global).

Biến dữ liệu tĩnh có những tính chất sau:

- Khi đối tượng đầu tiên của lớp được tạo lập thì các biến dữ liệu tĩnh được gán là 0.
- Chỉ có một bản sao của biến tĩnh được tạo ra cho cả lớp và sẽ được sử dụng chung cho tất cả các đối tượng trong cùng lớp.
- Chỉ được sử dụng trong lớp, nhưng nó tồn tại trong suốt thời gian hoạt động của chương trình.

II.2. Tính tương ứng bội

Như trên đã nói tương ứng bội là khả năng sử dụng một tên gọi dưới nhiều dạng khác nhau. Hàm và toán tử tải bội là hai trường hợp điển hình của tương ứng bội. Hàm, toán tử tải bội sẽ được phân tích để đối sánh về kiểu, số lượng tham biến trong thời gian dịch để chọn ra hàm, toán tử tương ứng. Liên kết được thực hiện trong thời gian biên dịch được gọi là thời gian tĩnh.

Thế mạnh của tương ứng bội có hai cấp:

- Cho phép chúng ta xử lý các khái niệm có liên hệ nhau theo một cách giống nhau, làm cho chương trình tổng quát hơn và dễ hiểu hơn.
- Tính tương ứng bội có thể được dùng để viết chương trình có thể mở rộng nhiều hơn. Khi một loại mới được thêm vào có liên hệ với các kiểu đang có thì bản chất tương ứng bội của nó sẽ làm cho loại mới này thích hợp ngay vào hệ thống mà không đòi hỏi phải thay phần còn lại của chương trình.

Cơ chế thực hiện tương ứng bội:

II.2.1. Hàm tải bội

Ngôn ngữ C không cho phép người lập trình sử dụng cùng một tên cho nhiều hàm trong một chương trình. Tuy vậy trong C++, việc định nghĩa này là hoàn toàn hợp lệ, các hàm này khác nhau về số lượng hoặc kiểu của các đối số cho hàm. Các hàm như vậy được gọi là hàm tải bội (kể cả cấu từ).

II.2.2. Toán tử tải bội.

Đã nhiều lần chúng ta khẳng định rằng trong C++ chúng ta có thể tạo ra những kiểu dữ liệu mới có hành vi giống như các kiểu dữ liệu cơ sở. Hơn thế nữa chúng ta còn có thể đưa thêm định nghĩa mới cho những toán tử được định nghĩa trước. Những toán tử cùng tên thức hiện được nhiều chức năng khác nhau được gọi là *toàn tử tải bội*.

Quy tắc xây dựng toán tử tải bội:

1. Chỉ có thể xây dựng những toán tử đã có trong C++ để thành toán tử bội . Không thể tự ý tạo ra những toán tử mới .

2. Toán tử tải bội phải có ít nhất một toán hạng có kiểu là kiểu được định nghĩa bởi người sử dụng .

3. Chúng ta không thể tự ý làm thay đổi ý nghĩa cơ bản của toán tử đã được định nghĩa trước . Ví dụ, chúng ta không thể định nghĩa lại được các phép +, - đối với các kiểu cơ sở (int, float).

4. Toán tử tải bội được xây dựng và sử dụng tuân theo quy tắc cú pháp của toán tử cơ sở như đã được định nghĩa trong ngôn ngữ.

5. Một số toán tử không thể định nghĩa thành toán tử tải bội được (bảng 3-1).

6. Một số toán tử không thể sử sụng với friend để thành hàm toán tử tải bội(bảng 3-2), nhưng có thể sử dụng hàm thành phần để đổi thành hàm toán tử tải bội.

7. Hàm thành phần toán tử tải bội một ngôi không có tham biến và trả lại giá trị tường minh. Hàm thân thiện toán tử tải bội một ngôi có tham biến là đối tượng.

8. Hàm thành phần là toán tử tải bội hai ngôi có một tham biến và hàm thân thiện là toán tử tải bội nhị nguyên có hai tham biến.

9. Khi sử dụng hàm thành phần là toán tử tải bội nhị nguyên thì toán hạng bên trái của toán tử phải là đối tượng trong lớp chứa hàm thành phần đó .

10. Những toán tử số học nhị nguyên +, -, *. Và / phải trả lại giá trị một cách tường minh.

sizeof	Toán tử xác định kích thước
.	Toán tử xác định thành phần
.*	Toán tử xác định thành phần mà con trỏ tới
::	Toán tử phân giải miền xác định
?:	Toán tử điều kiện

Bảng 3-1. Những toán tử không thể tải bội

=	Toán tử gán
( )	Toán tử gọi thực hiện một hàm
[ ]	Toán tử xác định một phần tử của bảng
- >	Toán tử truy cập tới phần tử của lớp

Bảng 3-2. Những toán tử không sử dụng được với friend

Định nghĩa toán tử tải bội

Để đưa thêm một chức năng mới cho toán tử, chúng ta phải biết được ý nghĩa của chức năng đó liên quan như thế nào với các lớp mà toán tử đó sẽ được áp dụng.

Định nghĩa tổng quát của toán tử tải bội là:

```
return-type class-name::operator op(arg-list)
 {
 ------------;// phần thân của các hàm toán tử
 }
```

Trong đó return-type là kiểu của kết quả thực hiện các phép toán, op là toán tử tải bội đứng sau là từ khoá operator op được gọi là hàm các toán tử với các tham biến là arg-list.

Hàm các toán tử tải bội phải là hàm thành phần (không phải là hàm tĩnh) hoặc là hàm thân thiện (friendly). Sự khác nhau cơ bản giữa chúng là:

- Hàm thành phần tải bội không có đối số cho hàm toán tử một ngôi và một đối số cho hàm toán tử hai ngôi .
- Hàm tải bội thân thiện có một đối số cho hàm toán tử một ngôi , hai đối số cho hàm toán tử hai ngôi .

Có sự khác nhau này bởi vì đối với hàm thành phần thì đối tượng được sử dụng liên quan đến hàm thành phần được tự động truyền vào tham số cho nó còn hàm thân thiện thì không làm được điều đó.

Ví dụ:

```
class Vector
{
private:
        int v[100];
public:
openrator + (vector); // cộng vector, hàm thành phần.
openrator - (vector & a); // trừ vector, hàm thành phần.
openrator = (const vector & a); // phép gán vector, hàm thành phần.
openrator -(); // đổi dấu vector, hàm thành phần.
friend vector+(vector, vector);// cộng vector hàm thân thiện.
int oprator == (vector);// so sánh hàm thành phần.
friend int == (vector, vector);// so sánh, hàm thân thiện.
};
```

Trong đó các vector là lớp các đối tượng.

Quá trình xác định các hàm toán tử tải bội được thực hiện như sau:

- Định nghĩa lớp để xác định kiểu dữ liệu sẽ được xác định trong phép toán tải bội.
- Khai báo hàm operator op trong vùng chung public của lớp. Nó có thể là hàm thành phần, hoặc là hàm thân thiện.

- Định nghĩa nội dung công việc cần thực hiện.

Hàm thành phần op x (hoặc x op) sẽ là:

operator op(x) đối với hàm thân thiện. Biểu thức x op y sẽ được chuyển sang sạng: x.operator op(y) đối với hàm thành phần và operator op (x,y) đối với hàm thân thiện.

Toán tử tải bội sử dụng với friend:

Để sử dụng hàm toán tử tải bội thân thiện ta chỉ việc khai báo từ khoá friend ở đầu, sau đó định nghĩa lại toán tử. Trong nhiều trường hợp, kết quả của việc sử dụng hàm thành phần giống hệt như hàm thân thiện. Một câu hỏi đặt ra là tại sao phải phân biệt hai trường hợp như thế? Hiển nhiên là vì có những tình huống đòi hỏi phải sử dụng hàm thân thiện mà không sử dụng hàm thành phần được.

II.2.2 Chuyển đổi kiểu

Chuyển đổi kiểu là một trong những đặc tính mạnh của C mà các ngôn ngữ khác hầu như không có được. Một biểu thức có thể có những hằng, biến ở nhiều kiểu khác nhau và khi thực hiện sẽ áp dụng quy tắc chuyển đổi kiểu tự động được chương trình dịch cài đặt sẵn, kiểu dữ liệu ở bên phải phép gán sẽ tự động chuyển đổi sang kiểu ở bên trái. Tuy nhiên điều này sẽ khó thực hiện được đối với kiểu dữ liệu do người sử dụng định nghĩa, bây giờ ta xét các phép toán chuyển kiểu trên lớp.

- *Từ kiểu cơ sở sang kiểu lớp*. Để thực hiện đổi kiểu dữ liệu sang lớp chúng ta sử dụng toán tử khởi tạo đối tượng.

- Từ kiểu lớp sang kiểu cơ sở. C++ cho phép định nghĩa toán tử quy hồi kiểu tải bội để chuyển dữ liệu kiểu lớp sang dạng kiểu cơ sở. Dạng tổng quát của toán tử quy hồi kiểu tải bội ở dạng hàm: operator double() chuyển lớp vector sang kiểu double. Khi đó trong chương trình chúng ta có thể viết:

double len = double(v1); // v1 là đối tượng trong lớp vector hoặc:

double len = v1;

Hàm toán tử kiểu quy hồi phải thỏa mãn những tính chất sau:

- Nó phải là hàm thành phần của một lớp.
- Nó không chỉ định kiểu giá trị quay trở lại.
- Nó không có đối số.

♦ *Từ một lớp sang một lớp khác:* việc chuyển đổi kiểu giữa những đối tượng ở nhiều lớp khác nhau được thực hiện thông qua toán tử khởi tạo đối tượng hoặc hàm đổi kiểu. Hàm đổi kiểu phải là hàm thành phần: operator type_name0. Chuyển đối tượng trong lớp chứa nó sang kiểu type_name. Type_name có thể là kiểu bất kỳ. Nếu chuyển kiểu giữa các đối tượng thông qua cấu tử thì thực chất là sao chép dữ liệu giữa các đối tượng.

II.3. Kế thừa và sự mở rộng các lớp

Khả năng sử dụng lại là đặc tính quan trọng của lập trình hướng đối tượng. Việc sử dụng lại những đơn thể chương trình, những lớp đã được phát triển tốt, đã được kiểm nghiệm không những tiếp kiệm được tiền của, thời gian mà còn làm tăng thêm những khả năng tương thích, độ tin cậy của hệ thống.

C++ hỗ trợ rất mạnh cho những khái niệm về sử dụng lại. Lớp được thiết kế trong C++ luôn luôn có thể được sử dụng lại theo nhiều cách khác nhau. Khi một lớp đã được định nghĩa, được kiểm nghiệm thì người lập trình khác có thể sử dụng nó trong các mục đích riêng của mình. Những lớp này là lớp cơ sở để tạo ra những lớp mới (lớp dẫn xuất), sử dụng lại những tính chất trong những lớp đã được xác định. Cơ chế dẫn xuất ra những lớp mới từ những lớp trước gọi là sự kế thừa (hoặc dẫn xuất).

Lớp dẫn xuất có thể kế thừa một số hoặc tất cả những đặc tính của lớp cơ sở, và có thể kế thừa một hay nhiều lớp ở các mức khác nhau. C++ cung cấp cho chúng ta 5 loại kế thừa sau:[3]

1. Kế thừa đơn.
2. Kế thừa đa mức.

3. Kế thừa phân cấp.
4. Kế thừa bội.
5. Kế thừa kép (phức hợp).

Cách tạo ra lớp dẫn xuất (derrved_class_name) từ một lớp cơ sở (base_class_name):

```
class derrved_class_name: mode base_class_name
{
  ----------
----------// các thành phần của lớp dẫn xuất
----------
};
```

Dấu ':' chỉ ra rằng lớp derrved_class-name được dẫn ra từ base_class_name. Từ mode là thể khai báo tuỳ chọn, có thể là private hoặc là public. Mặc nhiên là private (không có mode).

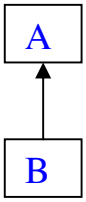
Khi lớp dẫn xuất khai báo kế thừa theo kiểu private thì tất cả những thành phần chung public của lớp cơ sở trở thành phần riêng private của lớp dẫn xuất và vì vậy những thành phần public của lớp cơ sở chỉ có thể truy nhập được thông qua hàm thành phần của lớp kế thừa.

Ngược lại, khi khai báo kế thừa theo kiểu public thì các thành phần chung của lớp cơ sở cũng trở thành phần chung của lớp dẫn xuất, nên các đối tượng của lớp dẫn xuất có thể truy nhập đến thành phần public của, lớp cơ sở (hình 3-3).

Trong cả hai trường hợp, thành phần private của lớp cơ sở hoàn toàn không được kế thừa. Vì thế những thành phần private của lớp cơ sở không khi nào trở thành thành phần của lóp dẫn xuất.

II.3.1. Kế thừa đơn

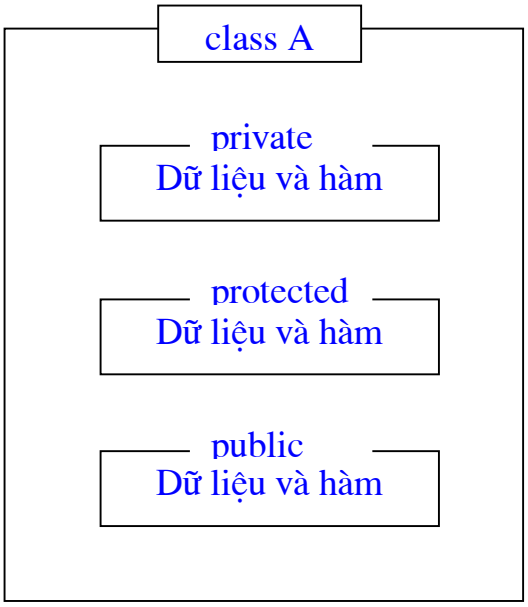
Kế thừa đơn là một lớp chỉ kế thừa một lớp đã có.



Lớp cơ sở

Lớp dẫn xuất

Chúng ta có thể mô tả như sau:

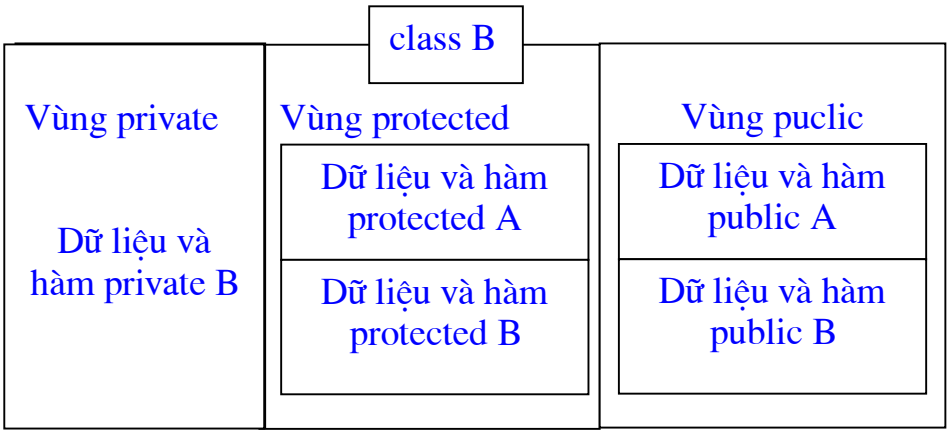


*Lớp B kế thừa kiểu public từ lớp A:

```
class B: public A
{
---------// Dữ liệu và hàm vùng private
```

```
---------// Dữ liệu và hàm vùng protected
---------// Dữ liệu và hàm vùng public
};
```

lớp B được mô tả như sau:



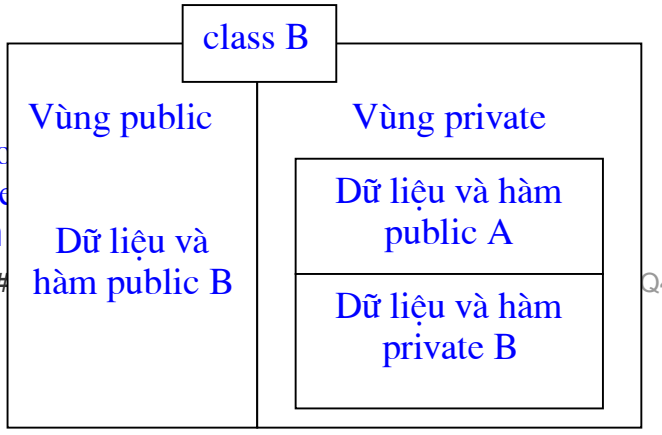
Lớp B kế thừa những thành phần chung của A. Những thành phần chung của A không được kế thừa. Để truy nhập được tới thành private của A như a (những thành phần không được kế thừa) ở trong B thì chúng ta phải sử dụng tới hàm thành phần được kế thừatừ A là get_a().

* Lớp B kế thừa kiểu private từ lớp A. Lúc đó dữ liệu và hàm của lớp B được mô tả như trong hình trên.

Ta thấy với cơ chế kế thừa kiểu private thì đối tượng X của lớp B không thể sử dụng trực tiếp những thành phần public của A. do vậy có lệnh:

```
X.get_ab();
X.get_a();
X.show_a();
```

đều không thực hiện được, bởi vì chúng đã trở thành private của lớp B.



Hình 3-5 Bổ sung thành phần kế thừa vào vùng private.

Chúng ta thấy rằng, những thành phần khai báo ở vùng private của lớp cơ sở không được kế thừa. Do vậy mà lớp kế thừa của lớp dẫn xuất không sử dụng được những thành phần mà nó kế thừa. Chúng ta sẽ thực hiện như thế nào khi trong lớp dẫn xuất có nhu cầu kế thừa những thành phần dữ liệu private của lớp cơ sở. Điều này có thể thực hiện được bằng cách chuyển dữ kiệu thành phần đặc khai báo trong vùng private sang vùng public. Nhưng khi đó thì dữ liệu đó lại có thể truy nhập bởi tất cả những thành phần khác trong chương trình. Điều này phá vỡ nguyên lý che dấu thông tin mà chúng ta cần thực hiện.

Để giải quyết vấn đề trên, C++ đưa thêm thể khai báo protected (được bảo vệ) cho những thành phần của lớp cơ sở cần được kế thừa chỉ trong những lớp dẫn xuất trực tiếp. Những thành phần được khai báo protected có thể được truy nhập bởi những thành phần trong cùng lớp và trong những lớp dẫn xuất trực tiếp từ lớp cơ sở. Như vậy, để có những thành phần protected ta chỉ cần khai báo như sau:

```
class A
 {
      private:
    _ _ _ _ _
      protected:
      _ _ _ _ _
      public:
```

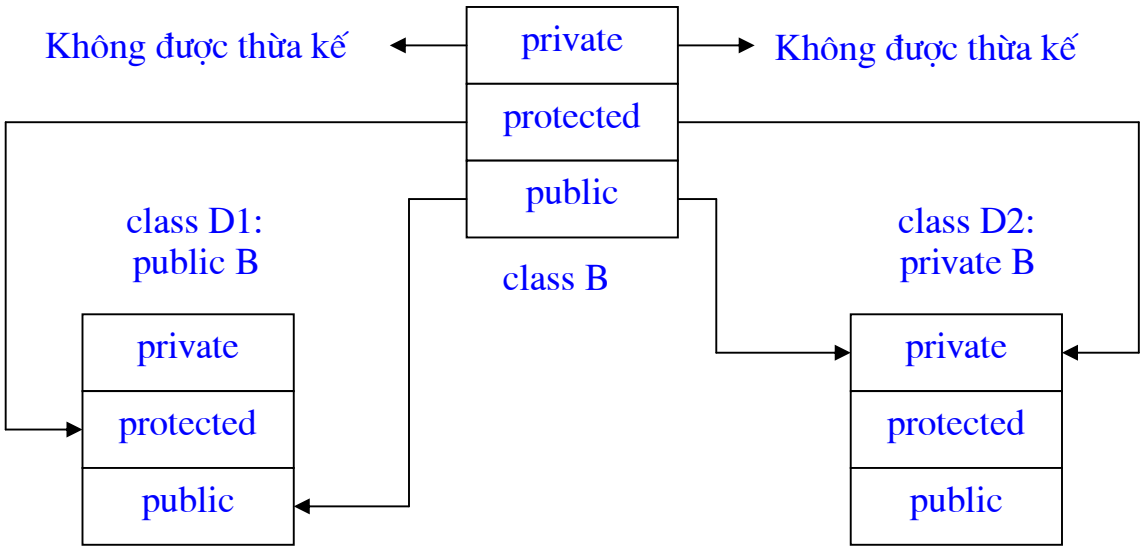
_ _ _ _ _

};

Mối quan hệ giữa các thành phần của lớp cơ sở và lớp dẫn xuất được mô tả trong bảng 3-3 và hình 3-7.

Lớp cơ sở	Lớp dẫn xuất	
	Kiểu dẫn xuất public	Kiểu dẫn xuất private
private	Không được kế thừa	Không được kế thừa
protected	protected	private
public	public	private

Bảng 3-3. Những thành phần được kế thừa.

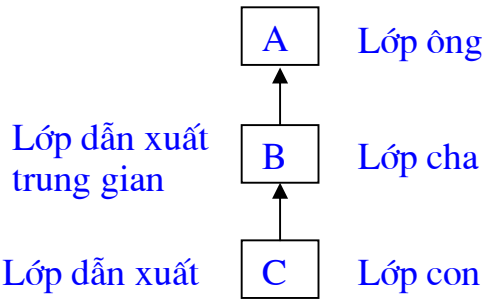


class X: public D1: public D2

Hình 3-6 Sự tương ứng trong kế thừa

II.3.2. Kế thừa đa mức

Trong kế thừa đa mức, các lớp được tổ chức như sau:



Hình 3-7 Kế thừa đa mức

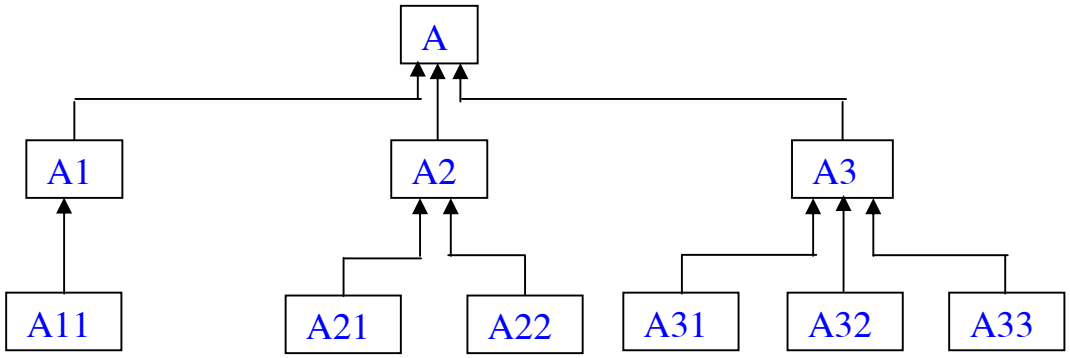
Lớp dẫn xuất của kế thừa đa mức được khai báo như sau:

```
class A {. . .};// lớp cơ sở

class B: public A {. . .};// B dẫn xuất từ A

class C: public B {. . . }// C dẫn xuất từ B
```

Quá trình kế thừa có thể được mở rộng tuỳ ý, nghĩa các lớp kế thừa nhau với số mức tuỳ ý. Trong kế thừa đa mức thì nguyên tắc truy nhập của các lớp dẫn xuất đối với các lớp cơ sở cũng giống như trong kế thừa đơn và nó không bị hạn chế bởi các lớp trung gian. Có nghĩa là lớp B truy nhập được đến thành phần (dữ liệu và hàm trong vùng protected và public) của lớp A, lớp C truy nhập được đến thành phần của lớp B và cũng truy nhập trực tiếp được đến các thành phần của lớp A mà không cần phải thông qua lớp B.

II.3.3. Kế thừa phân cấp

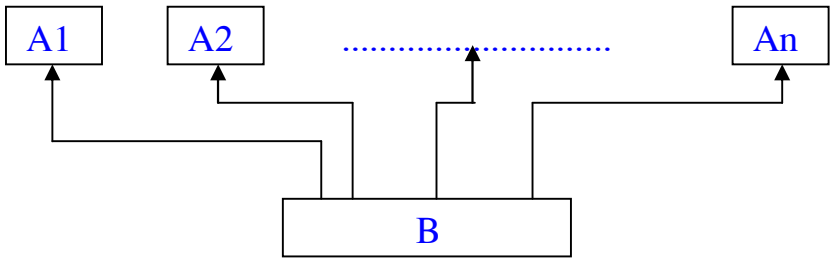
Kế thừa là sự phân cấp các lớp, các lớp kế thừa nhau theo một cấu trúc phân cấp được gọi là kế thừa phân cấp. C++ hỗ trợ cho việc sử dụng cơ chế kế thừa để phân cấp các lớp mô tả những cấu trúc được thiết kế theo cách tiếp cận phân cấp. Có thể mô tả tổng quá như sau:



Hình 3-8 Kế thừa phân cấp

II.3.4. Kế thừa bội

Một lớp có thể kế thừa thuộc tính của nhiều lớp được gọi là kế thừa bội. Kế thừa bội cho phép chúng ta kết hợp đặc trưng của một số lớp tạo ra lớp mới.



Hình 3-9. Kế thừa bôi

Một lớp được dẫn xuất từ nhiều lớp cơ sở được khai báo như sau:

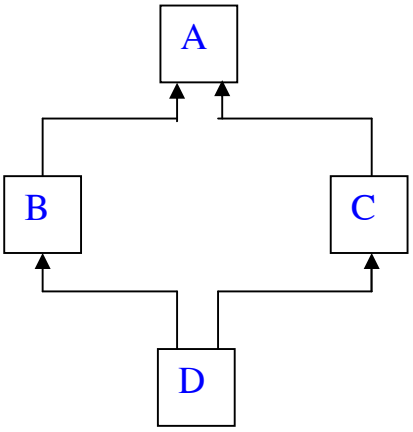
class B: mode A1, mode A2, ..., mode An

{- - - - };

với mode là kiểu khai báo: public hoặc private, và các lớp cơ sở phải được khai báo trước.

II.3.5. Kế thừa kép

Kế thừa kép là sự kết hợp của kế thừa đa thức và kế thừa bội. Nghĩa là một lớp dẫn xuất có thể kế thừa nhiều lớp cơ sở và ở nhiều mức khác nhau.

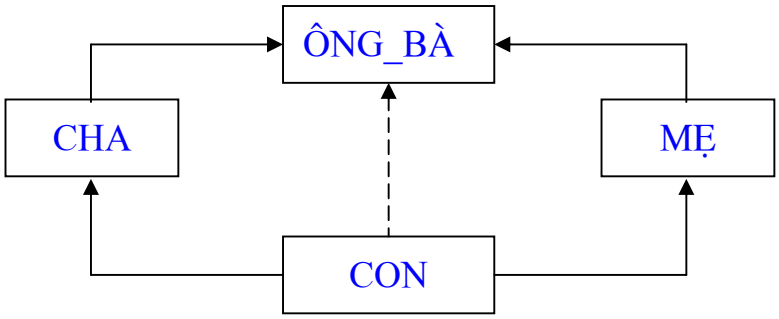


Hình 3-10. Kế thừa kép

II.3.6. Các lớp cơ sở ảo

Trong thức tế đôi khi chúng ta gặp tình huống đòi hỏi phải sử dụng kết hợp cả ba loại kế thừa: kế thừa bội, đa mức và phân cấp gọi là kế thừa lai ghép.

Ví dụ:



Hình 3-11 Kế thừa lai kép

Lớp kế thừa trực tiếp hai lớp con CHA và MẸ. Ngoài ra nó còn kế thừa từ ÔNG BÀ theo hai đường khác nhau. Nó kế thừa trực tiếp, một số đặc tính của ÔNG BÀ (theo đường - - -) và gán tiếp qua CHA, MẸ (hai lớp cơ sở trung gian). Như vậy những thành phần public và protected của ÔNG BÀ sẽ được kế thừa đúp ở lớp CON, lần thứ nhất là kế thừa từ CHA, lần thứ hai kế thừa từ MẸ. Nghĩa là lớp CON sẽ có hai tập các thành phần kế thừa giống nhau. Do vậy chúng ta phải loại bỏ dư thừa. Trong C++ cho phép loại bỏ những thành phần dư thừa này bằng cách khai báo lớp cơ sở ảo (virtual base class). Lúc đó chúng ta khai báo như sau:

```
class ONG_BA
        {------};
class CHA: virtual ONG_BA // hoặc public virtual
        {------};
class ME: virtual public ONG_BA
        {------};
class CON: public CHA, public ME
        {------};
```

Lúc này chỉ có một bản sao của ONG_BA sẽ được kế thừa ở lớp CON. Khi một lớp được khai báo kế thừa từ các lớp cơ sở ảo thì chỉ có một bản sao của những thành phần kế thừa được chuyển cho lớp dẫn xuất, bất luận bao nhiêu đường kế thừa cũng thế.

II.3.7 Cấu tử trong các lớp dẫn xuất

Cấu tử được sử dụng để tạo lập các đối tượng. Nhưng việc tạo lập các đối tượng trong những lớp kế thừa được thực hiện như thế nào? Chúng ta thấy có những điểm khác biệt như sau: nếu trong lớp cơ sở không có một cấu tử nào có tham biến thì trong lớp dẫn xuất không phải có hàm cấu tử. Song nếu một lớp cơ sở nào đó có chứa cấu tử có tham biến thì lớp dẫn xuất cũng phải có cấu tử và các tham số thực sự sẽ được truyền tương ứng cho các cấu tử có tham biến trong các lớp cơ sở. Cả hai lớp cơ sở và dẫn xuất đều có cấu tử thì cấu tử của lớp cơ sở thực hiện trước rồi sau đó đến lớp cấu tử của lớp dẫn xuất. Trong trường hợp kế thừa bội thì các cấu tử của lớp được thực hiện lần lượt theo thứ tự mà chúng được khai báo trong lớp dẫn xuất.

Trong trường hợp kế thừa đa mức thì các cấu tử được thực hiện theo thứ tự kế thừa theo đa mức.

Cấu tử của lớp dẫn xuất được định nghĩa như sau:

```
Phương_thức_thiếp_lập_dẫn_xuất (DS1, DS2, ..., DSn, DSDX):
        Cơ_sở_1 (DS1),
        Cơ_sở_1 (DS2),
        ----------
        Cơ_sở_1 (DSn),
        {
        ------// Nội dung cấu tử ở lớp dẫn xuất
```

Trong đó phần đầu của phương_thức_thiết_lập_tử_dẫn_xuất bao gồm hai thành phần cách nhau bằng hai dấu chấm ':'.Đầu tiên là khai báo danh sách những tham biến sẽ được truyền cho các cấu tử ở lớp dẫn xuất là phương_thức_thiết_lập_dẫn_xuất (DS1, DS2, ...,DSn, DSDX), sau đó là các lời gọi tới các hàm cấu tử đã được định nghĩa ở những lớp cơ sở là Cơ_sở_1(DS1), ..., Cơ_sở_n(DSn). Trong đó DS1, DS2, ..., DSn là những danh sách tham biến được sử dụng để tạo lập các đối tượng. Ví dụ:

```
D (int a1, a2, float b1, float b2, int d1):
        A(a1, a2), // gọi cấu tử A trong lớp A
        B(b1, b2), // gọi cấu tử B trong lớp B
        {
                d = d1;
        }
```

Cấu tử D() cung cấp các giá trị a1, a2 và b1, b2 để thực hiện cấu tử A(a1, a2) trong lớp A, B(b1, b2) trong lớp B.

II.3.8. Hàm ảo

Tương ứng bội là cấu trúc cho phép những đối tượng của nhiều lớp khác nhau có khả năng xử lý cùng một thông tin giống nhau nhưng ở nhiều dạng khác nhau. Một nhu cầu đặt ra là làm thế nào xử lý được các đối tượng đó mà không cần chú ý đến các lớp của chúng. để thực hiện được yêu cầu trên đối với tương ứng bội ta có thể sử dụng *hàm ảo* (**virtual function).**

Cách định nghĩa hàm ảo

Giả sử A là lớp cơ sở, các lớp B, C, D dẫn xuất (trực tiếp hoặc gián tiếp) từ A. giả sử trong 4 lớp trên đều có phương thức trùng tên, kiểu, các đối số. để định nghĩa các phương thức này là phương thức ảo, ta chỉ cần:

- Hoặc thêm từ khoá virtual vào dòng tiêu đề của phương thức bên trong định nghĩa của lớp cơ sở.
- Hoặc thêm từ khoá virtual vào dòng tiêu đề bên trong định nghĩa của tất cả các lớp A, B, C, D.

Quy tắc gọi phương thức ảo

Phương thức ảo được gọi thông qua một con trỏ của lớp cơ sở, lúc đó phương thức của lớp nào được gọi phụ thuộc vào đối tượng mà con trỏ đang trỏ tới.

Với các lớp A, B, C, D đã được định nghĩa, để truy nhập đến hàm hiển_thị() của các lớp ta khai báo như sau:

```
A*p; // p là con trỏ kiểu A

A a; B b; Cc; Dd; //khai báo các đối tượng

p=&a;// p trỏ tới đối tượng a của lớp A

p->hiển_thị(); // gọi tới A:: hiển_thị()

p=&b; // p trỏ tới đối tượng b của lớp B

p->hiển_thị(); // gọi tới B::hiển_thị()

p=&c;// p trỏ tới đối tượng c của lớp C

p->hiển_thị(); // gọi tới C::hiển_thị()
```

Hàm ảo phải tuân theo những quy tắc sau:

1. Hàm ảo phải là hàm thành phần của lớp.
2. Những thành phần tĩnh không thể khai báo ảo được.

3. Sử dụng con trỏ để truy nhập đến các hàm ảo.
4. Hàm ảo có thể là hàm thân thiện (friend) của lớp khác.
5. Hàm ảo phải được định nghĩa trong lớp cơ sở hoặc cả trong lớp cơ sở và lớp dẫn xuất, ngay cả khi không sử dụng nó.
6. Mẫu của tất cả các phiên bản (lớp cơ sở và lớp dẫn xuất) phải giống nhau. Nếu hai hàm cùng tên nhưng giống nhau thì C++ xem là toán tử tải bội.
7. Không được tạo ra toán tử ảo, nhưng có thể tạo ra được phương thức huỷ bỏ ảo.
8. Con trỏ kiểu lớp cơ sở có thể dùng để xác định đối tượng lớp dẫn xuất, nhưng ngược lại thì không.
9. Không dùng được phép tăng giảm giá trị con trỏ đối với lớp dẫn xuất, mà chỉ có tác dụng với lớp cơ sở.

CHƯƠNG III

HÀM VÀ LỚP MẪU

III.1. Hàm mẫu

III.1.1. Định nghĩa

Khi chúng ta muốn tạo ra một chương trình C++ thực hiện các công việc nào đó cho tất cả các kiểu dữ liệu tất nhiên ta sẽ sử dụng hàm nạp chồng (function overloading). Chẳng hạn ta muốn tạo các hàm tính bình phương số int và float, ta sẽ phải tạo hai hàm:

```
int Square(int x){
  return x*x;
}
```

và

```
float Square(float x){
 return x*x;
}
```

Tương tự như vậy, cho mỗi kiểu dữ liệu khác nhau phải tạo ra một hàm Square mới. Ta nhận thấy rằng mã lệnh không đổi chỉ có kiểu dữ liệu là khác nhau. Chính vì vậy, ngôn ngữ lập trình hướng đối tượng C++ cho phép cài đặt hàm tổng quát về dữ liệu, khi sử dụng sẽ chỉ định cụ thể để trình biên dịch hiểu được. Hàm như vậy được gọi là hàm mẫu (template) như ví dụ trên ta có thể viết lại như sau:

```
template <class T>
T Square (T x){
   return x*x;
}
```

Trong đó T là tên gọi mẫu của kiểu dữ liệu sử dụng trong hàm. Với cách viết này, có thể sử dụng một kiểu dữ liệu bất kỳ nào khác.

Ví dụ:

```
void main(){
  cout << "Square(2)="<<Square(2)<< endl;
 cout << "Square(7.1)="<<Square(7.1)<< endl;
}
```

Trong lần gọi thứ nhất, hàm Square có tham số là int nên trình biên dịch sẽ tạo ra hàm int Square (int), lần hai tạo ra hàm có dạng float Square (float).

III.1.2. Hàm mẫu có nhiều tham số hình thức

Trường hợp hàm mẫu có nhiều tham số cùng kiểu, khi trình biên dịch biết được kiểu thứ nhất thì các tham số còn lại mang cùng kiểu mẫu sẽ nhận kiểu của tham số thứ nhất.

Ví dụ:

```
template <class T>
T max <T a, T b>       {
   return (a>b)?a:b;
}
void main(){
 float a,b;
cout << " Vào 2 số:"<<endl;
cin >> a>>b>>endl;
cout << " Số lớn hơn:"<< max(a,b);
```

```
}
```

Trong trường hợp này trình biên dịch tạo ra hàm float max(float, float) theo dạng của mẫu

III.1.3. Hàm mẫu có nhiều tham số khác nhau

Hàm template có nhiều tham số với kiểu khác nhau. Tuy nhiên nó vẫn mang đầy đủ tính chất của hàm mẫu đơn giản.

Ví dụ:

```
template <class A, class B>

A factorial (B  n){

if (n>0) return n*factorial(n-1);

else return 1;

}

void main(){

float f;

f=factorial(20);

cout <<"20!="<< f<< endl;

}
```

Chú ý:

- Trong việc tạo ra hàm mẫu, chúng ta có thể sử dụng bất kỳ kiểu dữ liệu nào kể cả do người sử dụng tạo ra.
- Chúng ta vẫn có thể xây dựng hàm trùng tên với hàm template như hiện tương nạp chồng.

```
template <class T>
T abs (T a){
  return (a<0)?-a:a;
}
float abs(float a){
for (int i=0,float s=0;i<5;i++)
 s+=abs(a[i]);
 return s;
}
void main(){
  float a[5]={2,-4,-3,5,-6};
  cout <<"chuẩn của vecto a là:"<<abs(a);
  cout << "abs(-5.3)="<<abs(-5.3)<<endl;
}
```

Đầu tiên chương trình sẽ tìm hàm có tham số phù hợp nếu không tìm thấy nó sẽ gọi hàm mẫu. Trong chương trình có hai hàm abs nhưng do cách truyền tham số thực và sự hiện diện của hàm float abs(float). Chương trình dịch sẽ lựa chọn đúng để thực hiện và biên dịch.

- Một hàm mẫu ra lệnh cho chương trình dịch cách tạo ra một tập các hàm nạp chồng. Chương trình dịch chỉ sinh ra mã cho các kiểu dữ liệu khi nó gọi hàm mẫu.

III.2. Lớp mẫu

III.2.1. Định nghĩa

Cũng giống như hàm mẫu, lớp mẫu cũng mang đầy đủ tính chất, tư tưởng của hàm mẫu bằng cách cài đặt một lớp chung tổng quát nào đó. Khi sử dụng cho từng trường hợp cụ thể lớp sẽ mang kiểu dữ liệu xác định. Đây là công cụ mạnh để xây dựng các lớp chứa.

Vi dụ:

```
Template<class K ,class D>
    class Record {
      K key ;
     D data;
    public:
      Record(K kx,D dx);
    K GetKey(){return key;}
   DgetData(){return data;}
    }
  template<class K ,class D>
    Record <K,D>::Record(K kx,D dx){
    key=kx;
    data=dx;
```

```
}
```

Lưu ý:

- Khi nội dung của phương thức viết bên ngoài lớp chúng ta phải mô tả lại khai báo template <class K, class D> phía trước tất cả kể cả kiểu trả về của phương thức. Còn các tham chiếu bên trong lớp mẫu không phải khai báo.
- Các lớp mẫu có thể thừa kế từ các lớp mẫu khác.

  Ví dụ:

```
template <class D>
class Object: Record<int, D>
{...}
```

  - Lớp mẫu có thể là tham số cho một lớp mẫu khác:

```
Record <int, Object<string>> rec;
```

III.2.2. Lớp mẫu có tham số

Ngôn ngữ lập trình hướng đối tượng C++ cho phép tạo ra các lớp mẫu linh động bằng cách cho phép thay đổi giá trị của các thành phần bên trong lớp. Khi đó lớp có dạng của một hàm với tham số hình thức.

```
template <class T, size_t MAX>
class Stack{...}
```

Chú ý:

Khi định nghĩa một lớp mẫu, mỗi lớp mẫu có phương thức và dữ liệu riêng của nó. Cho nên càng nhiều lớp mẫu thì chương trình càng cần nhiều bộ nhớ. Phần lớn các mẫu sẽ được định nghĩa trong các tệp chứa khai báo và sẽ được dịch gộp (include) khi sử dụng lớp đó.

III.3. Kết luận

Các hàm và lớp mẫu cho phép tạo ra một họ các hàm và các lớp để tác động đến các kiểu dữ liệu khác nhau. Với những đặc điểm nổi bật này, chương trình hướng đối tượng hoàn toàn linh động và mang tính kế thừa cao. Đồng thời nó giúp cho người lập trình không phải viết lại những đoạn mã giống nhau, cũng như làm thiểu mã nguồn một cách đáng kể.

CHƯƠNG IV

CẤU TRÚC DỮ LIỆU & CÁC LỚP MẪU

IV. Cấu trúc dữ liệu

Các chương trình thường chứa hai phần: giải thuật và cấu trúc dữ liệu. Một chương trình tốt là chương trình hoà hợp được cả hai vấn đề này. Sự chọn lựa và thi hành của một cấu trúc dữ liệu được xem là quan trọng ngang với các trình vận dụng nó. Do đó, việc có phương pháp đúng lưu và truy xuất dữ liệu trong một số trường hợp là rất quan trọng.

Cấu trúc dữ liệu được xem như một tập các dịch vụ cung cấp cho thế giới bên ngoài, Người sử dụng không quan tâm đến việc lưu trữ cụ thể trên các thiết bị vật lý như thế nào,mà chỉ quan tâm đến việc truy cập – tức là lưu vào và phục hồi như thế nào.

Các kiểu cấu trúc dữ liệu cơ bản sau mà tôi đề cập trong cuốn tiểu luận này:

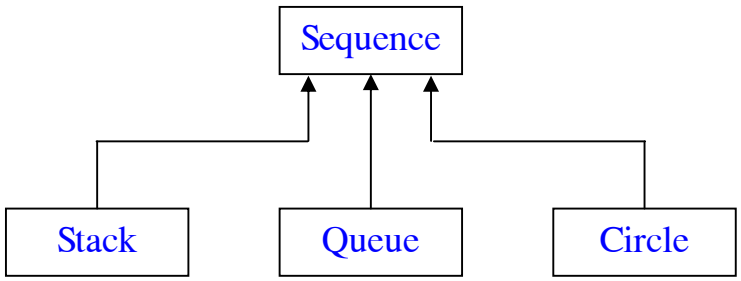
- Hàng đợi (Queue).
- Hàng quay tròn (Circle).
- Ngăn xếp (Stack).
- Danh sách liên kết đơn (Simple List).

- Danh sách liên kết đôi (double List).
- Cây nhị phân (Binary Tree).

IV.1.1. Lớp chứa

Do các tính chất chung cơ bản của các biểu cấu trúc dữ liệu tuyến tính ngăn xếp hàng và hàng xoay nên tôi đã xây dựng lớp chứa thuần ảo Sequence vì các lý do sau:

- Sử dụng được sự thừa kế, tránh phải viết lại các phương thức dữ liệu ở các lớp khác nhau.
- Tận dụng được sức mạnh cũng như tính linh hoạt của phương thức ảo vào các ứng dụng đa dạng sau này.



Hình Lớp chứa thuần ảo Sequence

IV.1.2. Lớp chứa thuần ảo

```
// SEQUENCE
template<class T,int size>
class Sequence
  {
```

```
    protected:
        size_t count;
        T   data[size];
        void Copy(const Sequence<T,size> & seq);
    public:
        Sequence()              { count=0;}
        Sequence(const Sequence<T,size> & seq){ Copy(seq);}
        virtual int Put(const T & item)=0;//phương thức ảo
        virtual int Get(T & item)=0;
        virtual int See(T & item,int i)=0;//xem phần tử thứ i
        size_t GetSize()    { return size;}
        size_t GetCount()   { return count;}
        void Flush()      { count=0;}
  };
//---------------------------------------------------------------------
template<class T,int size>
void Sequence<T,size>::Copy(const Sequence<T,size> & seq)
  {
   count=seq.count;
   for(int i=0;i<size;i++) data[i]=seq.data[i];
  }
```

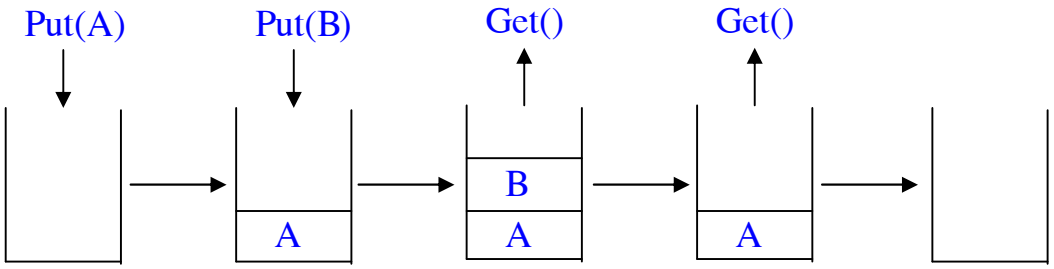
IV.2.1. Ngăn xếp

Ngăn xếp là một cấu trúc tuyến tính đặc biệt nó sử dụng cách truy xuất vào sau ra trước, thường được gọi là LIFO(Last In First Out). Nói chung ngăn xếp không được sử dụng để lưu trữ dữ liệu tĩnh. Khi thông tin được lấy ra nó sẽ bị xoá khỏi ngăn xếp. Một ngăn xếp có thể rỗng hoặc có các phần tử.

Có thể hình dung ngăn xếp như một chồng đĩa. Cái đặt lên bàn đầu tiên sẽ được sử dụng sau cùng. Còn đĩa cuối cùng nằm trên đầu chồng được sử dụng đầu tiên. Ngăn xếp thường được dùng để truyền tham số cho hàm.

IV.2.2. Lưu trữ ngăn xếp bằng mảng

Có thể lưu trữ ngăn xếp vào một mảng (véc tơ) gồm n phần tử nhớ liên tiếp. Khi bổ xung một phần tử số lượng các phần tử bằng lên một, còn khi lấy ra một phần tử số lượng phần tử của ngăn xếp giảm đi một, điều này được minh hoạ trong hình sau:



Chú ý:

- Vì khối lượng lưu trữ của ngăn xếp là hữu hạn (bằng n) nên khi ta bổ sung thêm một phần tử cần phải kiểm tra số lượng đã đạt đến giới hạn chưa.

- Ngược lại, khi lấy ra một phần tử từ ngăn xếp, ta phải kiểm tra còn phần tử nào trong ngăn xếp không.

IV.2.3. Xây dựng lớp ngăn xếp mẫu

Lớp dữ liệu mẫu ngăn xếp gồm một số phương thức cơ bản như:

- Lấy một phần tử Get().
- Lưu trữ một phần tử Put().
- Số phần tử của ngăn xếp Getcount().
- Cỡ của ngăn xếp Getsize().

Ngoài ra còn có một số phương thức tiện ích khác hỗ trợ cho các trường hợp đặc biệt:

- Xoá ngăn xếp Flush().
- Xem phần tử thứ i mà không ảnh hưởng đến trật tự của ngăn xếp See().

Lớp ngăn xếp được thừa kế từ lớp chứa Sequence.

```
//STACK
template<class T,int size>
class  Stack:public Sequence<T,size>
  {
    public:
      Stack():Sequence<T,size>() {}
      Stack(const Stack<T,size> & sta):Sequence<T,size>(sta){}
      virtual int Put(const T & item);
      virtual int Get(T & item);
      virtual int See(T & item,int i);
      void operator =(const Stack<T,size> & sta){ Copy(sta);}
  };
//------Ghi một phần tử vào ----------------------------------------
template<class T,int size>
int Stack<T,size>::Put(const T & item)
  {
    if( count==size) return 1;// Tràn Stack
    data[count++]=item;
    return 0;
  }
//-------Lấy một phần tử-------------------------------------------
template<class T,int size>
int Stack<T,size>::Get(T & item)
  {
    if (!count) return 1;//Stack rỗng
    item=data[--count];
    return 0;
  }
//-------Xem một phần tử bất kỳ-------------------------------------
template<class T,int size>
int Stack<T,size>::See(T & item,int i)
  {
    if(!i || i>count) return 1;
    item=data[--i];
    return 0;
  }
```

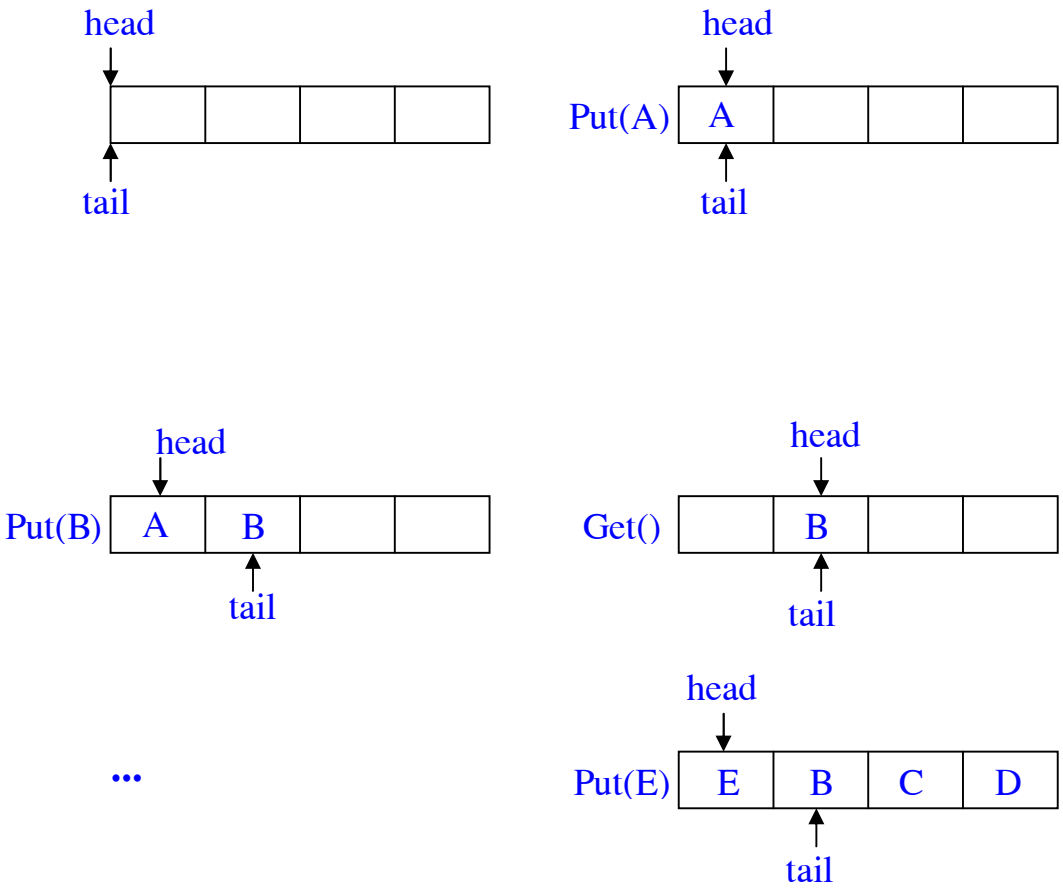
IV.3. Hàng đợi

Ngược lại với ngăn xếp, hàng đợi cho phép truy nhập theo cơ chế vào trước ra sau, thường gọi là FIFO (first in first out), tức là bổ sung dữ liệu ở một đầu, và lấy dữ liệu ra ở đầu khác. Khi dữ liệu được lấy ra nó sẽ bị xoá khỏi hàng đợi vì vậy một hàng đợi có thể rỗng hoặc có nhiều phần tử.

Hàng đợi được sử dụng nhiều trong lập trình, một trường hợp thông thường là việc mô phỏng các hàng đợi cũng được dùng bởi bộ định lịch cho các tác vụ của hệ điều hành và xuất nhập.

IV.3.1. Lưu trữ hàng đợi bằng mảng

Cũng giống như ngăn xếp, hàng đợi được lưu trữ trong mảng có kích cỡ n. Ngoài ra còn cần hai biến chỉ số để ghi vị trí đầu (head) mà một phần tử có thể được lấy ra, và vị trí đuôi (tail) nơi phần tử cuối có thể được bổ sung. Vị trí đuôi được tăng lên khi bổ sung một phần tử mới và khi lấy ra một phần tử thì đầu (head) tăng lên một. Cứ như vậy vị trí đầu đuổi theo vị trí đuôi. Nếu vị trí đuôi vượt qua chỉ số lớn nhất của mảng nó sẽ quay về chỉ số nhỏ nhất. Tương tự như vậy với vị trí đầu.



Hình Mô tả cách hoạt động của Sequence

Chú ý: Trước khi lấy ra một phần tử phải kiểm tra hàng có rỗng không, ngược lại khi thêm vào một phần tử phải kiểm tra hàng đã đầy chưa.

IV.3.2 Xây dựng lớp hàng đợi mẫu

Cũng giống như ngăn xếp, hàng đợi có những phương thức cơ bản và một số phương thức tiện ích đặc biệt khác, được thừa kế từ lớp chứa Sequence.

```
//QUEUE
template<class T,int size>
class Queue:public Sequence<T,size>
  {
    protected:
      int head;
      int tail;
    public:
      Queue():Sequence<T,size>(){ head=tail=0;}
      Queue(const Queue<T,size> & que);
      virtual int Put(const T & item);
      virtual int Get(T & item);
      virtual int See(T & item,int i);
      void operator = (const Queue<T,size> & que);
  };
//---------Phương thứ thiết lập sao chép-------------------------------
template<class T,int size>
Queue<T,size>::Queue(const Queue<T,size> & que):Sequence<T,size>(que)
  {
    head=que.head;
    tail=que.tail;
  }
//----------Ghi một phần tử vào ----------------------------------------
template<class T,int size>
int Queue<T,size>::Put(const T & item)
  {
    if(count==size)  return 1;//tràn hàng
    if(!count){
       head=tail=0;
       data[0]=item;
     }
    else{
       tail++;
       if(tail==size) tail=0;
       data[tail]=item;
     }
    count++;
    return 0;
```

```
  }
//---------Lấy một phần tử ----------------------------------------
template<class T,int size>
int Queue<T,size>::Get(T & item)
  {
   if(!count) return 1;//hàng rỗng
   item=data[head];
   count--;
   if(count){
     head++;
     if(head==size) head=0;
    }
   return 0;
  }
//-------Xem một phần tử----------------------------------------
template<class T,int size>
int Queue<T,size>::See(T & item,int i)
  {
   if(i>count || !count) return 1;
   i--;
   int ind=i+head;
   if(ind>=size) ind-=size;
   item=data[ind];
   return 0;
  }
//---------Toán tử gán----------------------------------------
template<class T,int size>
void Queue<T,size>::operator = (const Queue<T,size> & que)
  {
   head=que.head;
   tail=que.tail;
   Copy(que);
  }
```

IV.4. Hàng quay tròn

Hàng quay tròn là một cấu trúc đặc biệt khi mà đạt đến vị trí cuối cùng thì nó lặp lại ở vị trí đầu tiên.

Hàng quay tròn thường được sử dụng trong các hệ điều hành, trong các chương trình ứng dụng thời gian thực.

```
//CIRCLE
template<class T,int size>
```

```
class Circle:public Sequence<T,size>
  {
    protected:
      int current;
    public:
      Circle():Sequence<T,size>(){ current=0;}
      Circle(const Circle<T,size> & cir):Sequence<T,size>(cir){ current=cir.current;}
      virtual int Put(const T & item);
      virtual int Get(T & item);
      virtual int See(T & item,int i);
      void operator = (const Circle<T,size> & cir);
  };
//---------Ghi một phần tử----------------------------------------
template<class T,int size>
int Circle<T,size>::Put(const T & item)
  {
    if(count==size)  return 1;
    data[count++]=item;
    return 0;
  }
//-----------Lấy một phần tử----------------------------------------
template<class T,int size>
int Circle<T,size>::Get(T & item)
  {
    if(!count) return 1;
    item=data[current++];
    if(current==size) current=0;
    return 0;
  }
//------Xem một phần tử----------------------------------------
template<class T,int size>
int Circle<T,size>::See(T & item,int i)
  {
    if(!count) return 1;
    if(i>count)  i%=count;
    if(!i) i=count;
    item=data[--i];
    current=i;
    return 0;
  }
//----------Toán tử gán----------------------------------------
template<class T,int size>
void Circle<T,size>::operator = (const Circle<T,size> & cir)
  {
    current=cir.current;
    Copy(cir);
  }
```

IV.5. Danh sách liên kết

more information and additional documents
connect with me here: http://facebook.com/ngphutien/
file đồ án kèm theo: Cau truc du lieu voi C++.rar 259 KB
https://mega.co.nz/#!FgU3nAKa!-VgkoAGeGZtP3Q2LwN2LKMQXE4Uj67Q4nbAJrT4aGB8

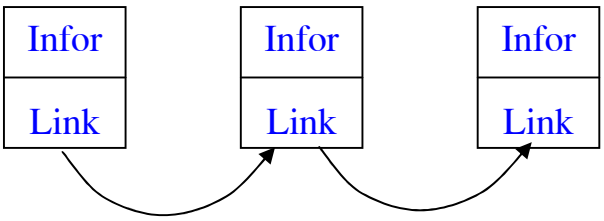
Như chúng ta đã biết, mảng (sử dụng các vùng bộ nhớ liên tục) được sử dụng rộng rãi trong nhiều ứng dụng. Tuy nhiên nó có hạn chế: kích thước phải biết trước, thời gian dịch, dữ liệu trong mảng được chia với những khoảng cách như nhau trong bộ nhớ. Điều này có nghĩa là việc chèn một phần tử vào mảng thì phải dịch chuyển các phần tử khác trong mảng, ngoài ra còn lãng phí bộ nhớ khi không còn sử dụng. Sự giới hạn này được khắc phục bằng việc sử dụng cấu trúc liên kết. Cấu trúc liên kết là một tập các nút (node), lưu trữ dữ liệu và các liên kết (links) bởi các nút khác. Bằng cách này, các nút có thể đặt bất kỳ nơi nào trong bộ nhớ và việc đi qua từ nút này sang nút khác được thực hiện bằng lưu trữ các địa chỉ của nút khác trong danh sách.

IV.6. Danh sách liên kết đơn

Danh sách liên kết đơn yêu cầu mỗi nút chứa một liên kết đến phần tử kế tiếp trong danh sách. Riêng nút cuối cùng không có phần tử đứng sau nên con trỏ của nút này chứa giá trị đặc biệt để đánh dấu kết thúc danh sách gọi là nút rỗng (NULL).

Để truy nhập được tất cả các nút trong danh sách thì cần truy nhập nút đầu tiên.

Theo quan niệm danh sách liên kết đơn giống như hình minh hoạ sau:



IV.6.1. Thêm một phần tử vào danh sách

Có hai cách để xây dựng một danh sách liên kết đơn:

- Mỗi phần tử mới vào cuối danh sách.

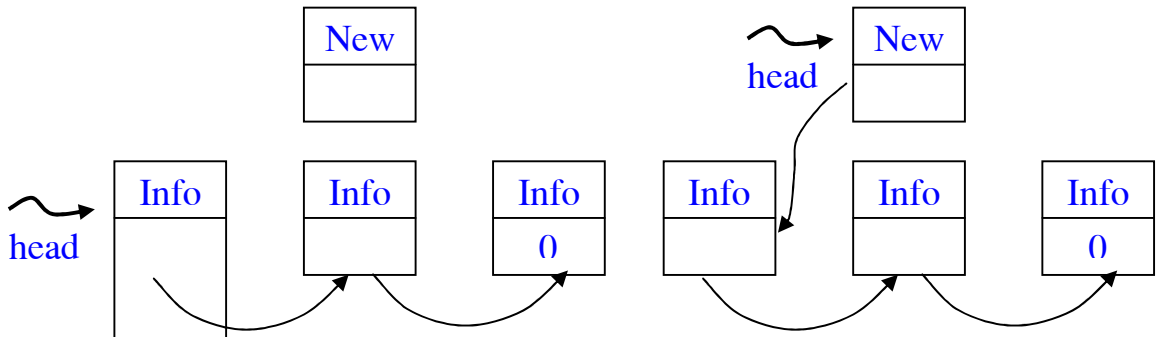
- Thêm phần tử mới vào một vị trí đặc biệt trong danh sách.

Có ba trường hợp khi chèn thêm một phần tử mới vào một danh sách liên kết đơn:

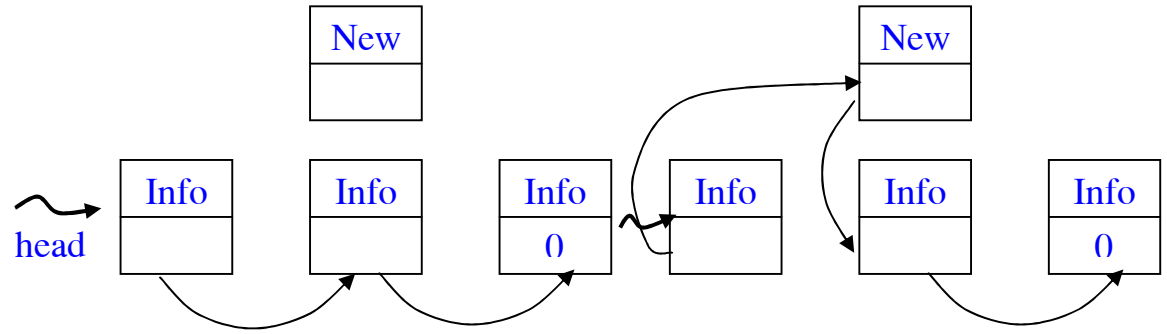
- Nút mới trở thành phần tử đầu tiên mới.
- Đứng giữa hai phần tử khác nhau.
- Trở thành phần tử cuối cùng.

Chú ý:

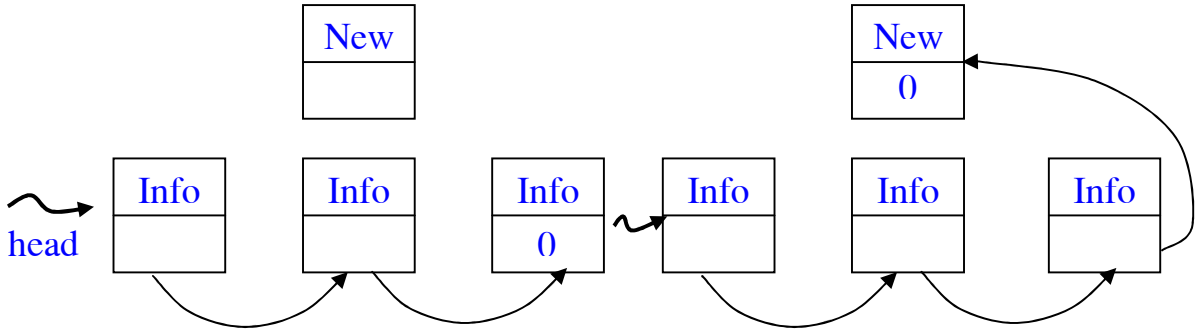
- Khi thay đổi phần tử đầu tiên hoặc cuối cùng, ta phải thay đổi địa chỉ mới cho các biến lưu trữ tương ứng.
- Để dễ hình dung tôi sẽ minh hoạ bằng hình ảnh các trường hợp này:



Hình Nút trở thành phần tử đầu tiên mới



Hình Nút chèn giữa hai nút khác

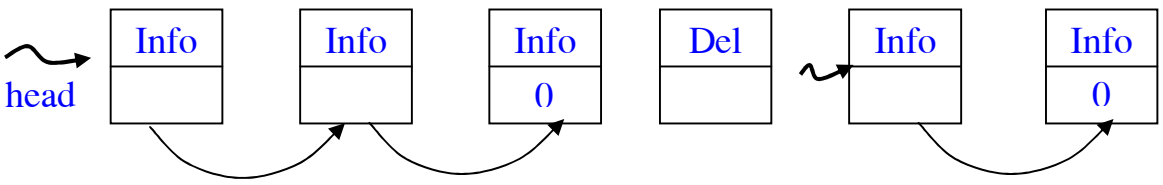


Hình Nút trở thành phần tử cuối cùng

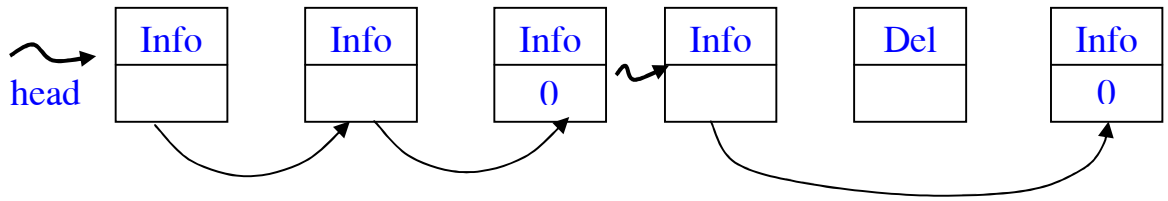
IV.6.2. Xoá một phần tử khỏi danh sách

Tương tự như thêm một phần tử có ba trường hợp:

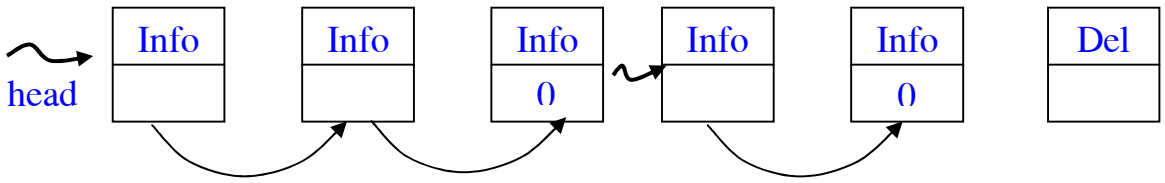
- Xoá phần tử đầu tiên.
- Xoá phần tử giữa.
- Xoá phần tử ở cuối danh sách.



Hình Xoá phần tử đầu tiên



Hình Xoá phần tử ở giữa



Hình Xoá phần tử ở cuối

IV.6.3. Xây dựng lớp danh sách liên kết đơn mẫu

Ngoài các phương thức cơ bản như thêm và xoá phần tử như đã trình bày ở trên. Lớp danh sách liên kết đơn mẫu còn có một số phương thức và dữ liệu khác để tiện cho việc quản lý cũng như sử dụng. Đặc biệt là phương thức Find() tìm kiếm và Sort() sắp xếp, đây là hai phương thức tìm kiếm và sắp xếp đơn giản, dễ hiểu. Nó hoạt động tốt với dữ liệu cơ bản như: int, float, char,... còn đối với kiểu dữ liệu tự định nghĩa nếu muốn dùng các phương thức này, phải định nghĩa các toán tử: <, >, == cho kiểu dữ liệu.

Ví dụ:

```
struct String

{

private:

        char *txt;

public:
```

```
        operator > (coust String & str);

        operator < (coust String & str);

        operator == (coust String & str);

    };
```

Lớp danh sách liên kết đơn mẫu

```
/*SLIST*/
enum direct {ASC,DESC};//hướng sắp xếp
template<class T>
class SList
 {
  protected:
    struct Node {    //định nghĩa một phần tử của danh sách
          T data;
          Node * next;
     } * head,* tail,*current;
     size_t counter;
  public:
    SList() {SetNull();}
    SList(const SList<T> & sli) {SetNull();Copy(sli);}
    ~SList() { Erase();}
    int operator = (const SList<T> & sli);
    int Append(const T & item);
    int Append(const SList<T> & sli);
    int Insert(const T & item);
    int Insert(const SList<T> & sli);
    void Erase();
    int Delete();
    int Get(T & item);
    size_t Count()  {  return counter;}
    int Next();
    void ToHead()  { current = head;}
    int AtTail()       { return current==tail;}
    int Find(const T & item);
    void Sort(direct dir= ASC);
  private:
    void SetNull();
    int Copy(const SList<T> & sli);
 };
//---------------------------------------------------
```

more information and additional documents
connect with me here: http://facebook.com/ngphutien/
file đồ án kèm theo: Cau truc du lieu voi C++.rar 259 KB
https://mega.co.nz/#!FgU3nAKa!-VgkoAGeGZtP3Q2LwN2LKMQXE4Uj67Q4nbAJrT4aGB8

```
template<class T>
inline void SList<T>::SetNull()
  {
   head=tail= current=NULL;
   counter=0;
  }
//-------------------------------------------------
template<class T>
int SList<T>::Copy(const SList<T> & sli)
  {
   if(!sli.counter) return 1;
   Node * tmp=sli.head;
   do{
     if(Append(tmp->data)) return 1;
     tmp=tmp->next;
      }
   while(tmp);
   current = sli->current;
return 0;
  }
//-----toán tử gán-----------------------------------------
template<class T>
int SList<T>::operator = (const SList<T> & sli)
  {
  Erase();
  if(Copy(sli))return 1;
  return 0;
  }
//------Phương thức xoá-----------------------------------
template<class T>
void SList<T>::Erase()
 {
   current=head;
   while(current){
     head=current->next;
     delete current;
     current=head;
    }
   SetNull();
 }
//-------Thêm một phần tử----------------------------------
template<class T>
int SList<T>::Append(const T & item)
 {
   Node * tmp=new Node;
   if(!tmp) return 1;
   tmp->data=item;
   tmp->next=NULL;
```

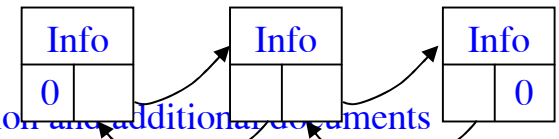
```
   if(!tail){
    head=tail=tmp;
    current=tmp;
    }
    else{
     tail->next=tmp;
     tail=tmp;
    }
    counter++;
    return 0;
 }
//---------ghép danh sách---------------------------------------
template<class T>
int SList<T>::Append(const SList<T> & sli)
 {
   const Node * tmp=sli.head;
   while(tmp){
     if(Append(tmp->data)) return 1;
     tmp=tmp->next;
    }
    return 0;
 }
//--------chèn một phần tử-------------------------------------
template<class T>
int SList<T>::Insert(const T & item)
  {
    if(!current) return 2;
    Node * tmp=new Node;
    if(!tmp) return 1;
    tmp->data=item;
    tmp->next=current->next;
    current->next=tmp;
    current=tmp;
    counter++;
    return 0;
  }
//-----chèn một danh sách-----------------------------------------
template<class T>
int SList<T>::Insert(const SList<T> & sli)
  {
    if(!current) return 2;
    Node * tmp;
    tmp=sli.head;
    while(tmp){
       if(Insert(tmp->data)) return 1;
       tmp=tmp->next;
     }
   return 0;
  }
```

```
//------Xoá một phần tử----------------------------------------
template<class T>
int SList<T>::Delete()
  {
    if(!current) return 2;
    if(current==head){
      if(AtTail()){
            delete current;
            SetNull();
            return 0;
       }
      else{
            head=current->next;
            delete current;
            current=head;
        }
      }
    else{
      Node * tmp=head;
      while(tmp->next != current) tmp=tmp->next;
       tmp->next=current->next;
       if(AtTail()) tail=tmp;
       delete current;
       current=tmp->next;
       }
      counter--;
      return 0;
  }
//-------Lấy giá trị của phần tử hiện tại------------------------------
template<class T>
inline int SList<T>::Get(T & item)
  {
    if(!current) return 1;
    item =current->data;
    return 0;
  }
//------chuyển đến phần tử kế tiếp-----------------------------------
template<class T>
inline int SList<T>::Next()
   {
    if(!current) return 2;
    current=current->next;
    return 0;
    }
//------Tim kiếm-----------------------------------------------
template<class T>
int SList<T>::Find(const T & item)
  {
    Node * tmp=head;
```

```
    while(tmp){
      if(tmp->data==item){
           current=tmp;
           return 1;
           }
      tmp=tmp->next;
     }
    return 0;
  }
//-------Sắp xếp-------------------------------------------
template<class T>
void SList<T>::Sort(direct dir=ASC)
 {
  Node *lap,*tmp;
  T buf;
  int exchange=0;
  ToHead();
  while(current){
     lap=current->next;
     buf=current->data;
     exchange=0;
     while(lap){
          if((buf > lap->data && dir ==ASC) ||
            (buf < lap->data && dir ==DESC)){
             buf=lap->data;
             tmp=lap;
             exchange=1;
            }
           lap=lap->next;
      }
   if(exchange){
     tmp->data=current->data;
     current->data=buf;
    }
   Next();
  }
  ToHead();
 }
```

IV.7. Danh sách liên kết đôi

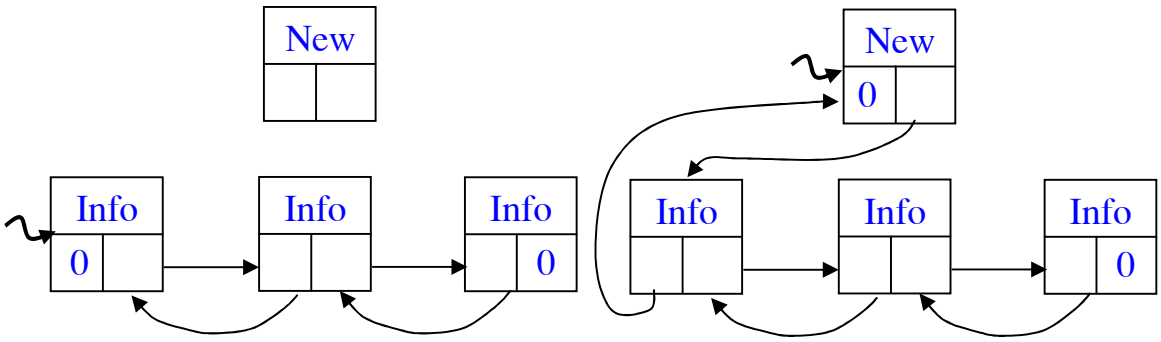
Danh sách liên kết đôi chứa dữ liệu và các liên kết đến phần tử kế tiếp và đến phần tử trước nó. Ưu điểm hơn so với liên kết đơn là có thể đọc cả hai chiều, đơn giản hoá việc quản lý danh sách làm cho việc chèn và xoá dễ hơn.



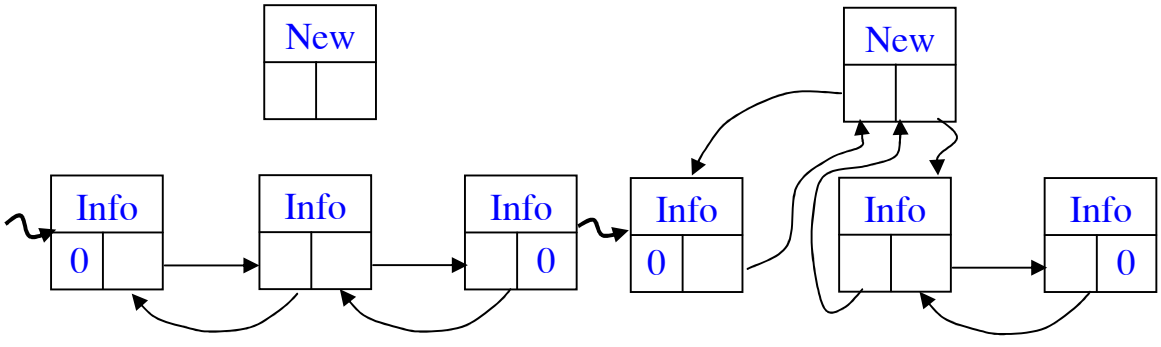
IV.7.1. Thêm một phần tử mới

Cũng giống như danh sách liên kết đơn có ba cách thêm một phần tử mới:

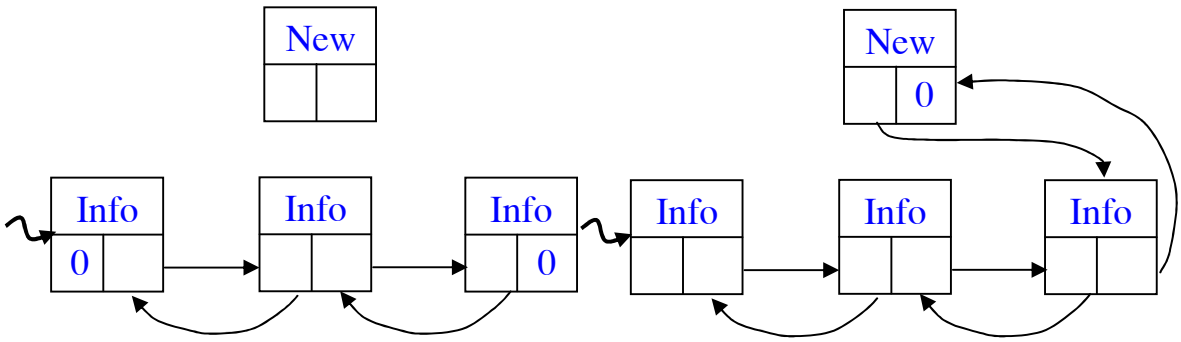
- Đầu danh sách.
- Giữa danh sách.
- Cuối danh sách.



Hình Thêm vào đầu danh sách



Hình Thêm vào giữa hai phần tử

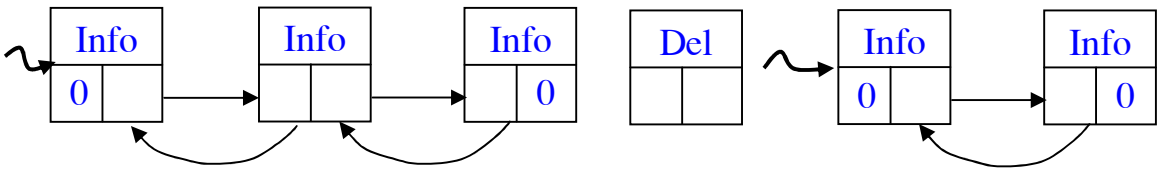


Hình Thêm vào cuối danh sách

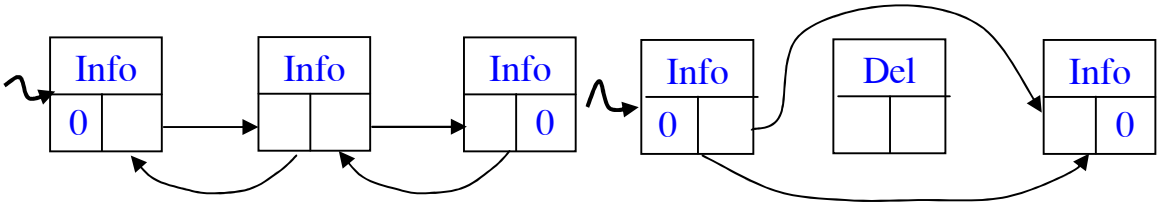
IV.7.2. Xoá một phần tử

Có ba trường hợp xoá một phần tử khỏi danh sách liên kết đôi:

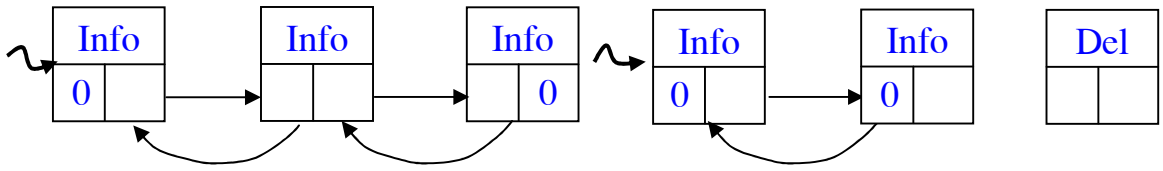
- Phần tử đầu tiên.
- Phần tử giữa.
- Phần tử cuối.



Hình Xoá phần tử ở đầu danh sách



Hình Xoá phần tử ở giữa.



Hình Xoá phần tử ở cuối danh sách

IV.7.3. Xây dựng lớp danh sách liên kết đôi mẫu

Cũng giống như danh sách liên kết đơn hai phương thức Find() và Sort() cần phải được định nghĩa các phép toán >, <, == đối với các kiểu dữ liệu do người sử dụng định nghĩa.

```
/*DLIST*/
enum direct {ASC,DESC};
template <class T>
class DList
  {
    protected:
      struct DNode {          //định nghĩa phần tử của danh sách
           T data;
           DNode * next,* prev;
           } *head,* tail,* current;
           size_t counter;
    public:
           DList() { SetNull();}
           DList(const DList<T> & dls) { SetNull(); Copy(dls);}
           ~DList() {Erase();}
           int operator = (const DList<T> & dls);
           int Append(const T & item);
           int Append(const DList<T> & dls);
           int Insert(const T & item);
           int Insert(const DList<T> & item);
           int InsertBefore(const T & item);
           int InsertBefore(const DList<T> & dls);
           void Erase();
           int Delete();
           int Get(T & item);
           size_t Count(){  return counter;}
           int AtHead() ;
           int AtTail()  ;
           void ToHead(){current=head;}
           void ToTail    {  current=tail;}
```

```
        int Prev();
        int Next();
        int Find(const T & item);
        void Sort(direct dir=ASC);
    private:
        void SetNull();
        int Copy(const DList<T> & dls);
  };
//-------------------------------------------------------
template <class T>
void DList<T>::SetNull()
 {
  head=tail=current=NULL;
  counter=0;
 }
//-------------------------------------------------------
template <class T>
int DList<T>::Copy(const DList<T> & dls)
  {
   if(!dls.counter)return 1;
   const DNode *tmp=dls.head;
   do{
     if(Append(tmp->data)) return 1;
     tmp=tmp->next;
     }
   while(tmp);
   current=dls.current;
   return 0;
  }
//------xoá danh sách-----------------------------------------
template <class T>
void DList<T>::Erase()
  {
   ToHead();
   while(current){
      head=current->next;
      delete current;
      current=head;
    }
   SetNull();
  }
//---------Toán tử gán-------------------------------------------
template <class T>
int DList<T>::operator = (const DList<T> & dls)
  {
   Erase();
   return Copy(dls);
  }
//------thêm một phần tử--------------------------------------
```

```
template <class T>
int DList<T>::Append(const T & item)
  {
    DNode *tmp= new DNode;
    if(!tmp) return 1;
    tmp->data=item;
    tmp->next=NULL;
    tmp->prev=tail;
      if(!Count())  head=tail=current=tmp;
       else {
            tail->next=tmp;
            tail=tmp;
            }
    counter++;
    return 0;
  }
//---------Ghép một danh sách----------------------------------------------
template <class T>
int DList<T>::Append(const DList<T> & dls)
  {
    const DNode *tmp=dls.head;
    while(tmp){
       if(Append(tmp->data)) return 1;
       tmp=tmp->next;
     }
    return 0;
  }
//---------chèn một phần tử-----------------------------------------------
template <class T>
int DList<T>::Insert(const T & item)
 {
   if(!current)return 2;
   DNode *tmp=new DNode;
   if(!tmp)return 1;
   tmp->data=item;
   if(current==tail){
      tmp->next=NULL;
      tmp->prev=tail;
      tail->next=tmp;
      tail=tmp;
    }
    else {
       current->next->prev=tmp;
       tmp->next=current->next;
       tmp->prev=current;
       current->next=tmp;
     }
     counter++;
     return 0;
```

```
 }
//-------------chèn một danh sách-----------------------------------------
template <class T>
int DList<T>::Insert(const DList<T> & dls)
  {
    if(!current) return 2;
    const DNode *tmp=dls.tail;
    while(tmp){
          if(Insert(tmp->data)) return 1;
          tmp=tmp->prev;
      }
     return 0;
  }
//------------chèn trước vào vị trí hiện tại-----------------------------------------
template <class T>
int DList<T>::InsertBefore(const T & item)
  {
    if(!current) return 2;
    DNode * tmp=new DNode;
    if(!tmp) return 1;
    tmp->data=item;
    if(current==head){
       tmp->next=head;
       tmp->prev=NULL;
       head->prev=tmp;
       head=tmp;
     }
     else{
          current->prev->next=tmp;
          tmp->prev=current->prev;
          tmp->next=current;
          current->prev=tmp;
      }
     counter++;
     return 0;
  }
//----------chèn trước vị trí hiện tại một danh sách-------------------------
template <class T>
int DList<T>::InsertBefore(const DList<T> & dls)
  {
    if(!current) return 2;
    const DNode * tmp= dls.head;
    while(tmp){
       if(InsertBefore(tmp->data))return 1;
       tmp=tmp->next;
     }
     return 0;
  }
//--------Xoá một phần tử-----------------------------------------
```

```
template <class T>
int DList<T>::Delete()
  {
   if(!current)return 2;
   if(current==head){
    if(!current->next){
           delete current;
           SetNull();
          }
         else{
           head=current->next;
           head->prev=NULL;
           delete current;
           current=head;
          }
     }
   else{
    if(current==tail){
          if(!current->prev){
            delete current;
            SetNull();
           }
          else{
           tail=current->prev;
           tail->next=NULL;
           delete current;
           current=tail;
          }
      }
    else{
         DNode * tmp=current->next;
         current->prev->next=current->next;
         current->next->prev=current->prev;
         delete current;
         current=tmp;
     }
   }
   counter--;
   return 0;
  }

//---------lấy giá trị ----------------------------------------------
template <class T>
int DList<T>::Get(T & item)
  {
    if(!current)return 2;
    item=current->data;
    return 0;
  }
```

```
//-------------------------------------------------------
template <class T>
inline int DList<T>::AtHead()
  {
    if(current==head) return 1;
    return 0;
  }

//-------------------------------------------------------
template <class T>
inline int DList<T>::AtTail()
 {
   if(current==tail)return 1;
   return 0;
 }
//-------------------------------------------------------
template <class T>
inline int DList<T>::Next()
 {
  if(!current)return 2;
  current=current->next;
  return 0;
 }
//-------------------------------------------------------
template <class T>
inline int DList<T>::Prev()
  {
    if(!current)return 2;
    current=current->prev;
    return 0;
  }
//-------------------------------------------------------
template <class T>
int DList<T>::Find(const T & item)
  {
    const DNode *tmp= head;
    while(tmp){
       if(tmp->data==item)return 1;
       tmp=tmp->next;
     }
     return 0;
   }
//-------------------------------------------------------
template <class T>
void DList<T>::Sort(direct dir=ASC)
  {
    T item;
    DNode *lap,*tmp;
    ToHead();
```

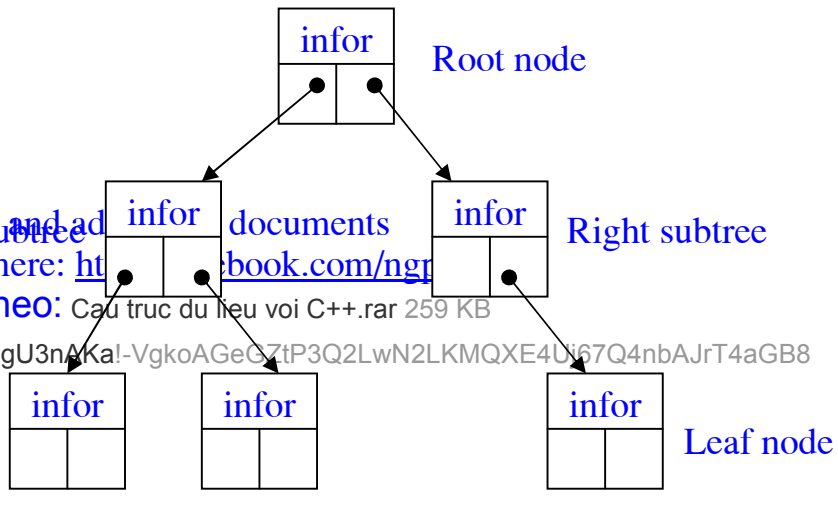
```
   while(current){
//      lap=current->next;
      lap=head;
      while(lap!=current){
            if((lap->data>current->data) && (dir==ASC) ||
              (lap->data<current->data) && (dir==DESC)) break;
            lap=lap->next;
           }
      if(lap!=current){
            item=current->data;
            Delete();
            tmp=current;
            current=lap;
            InsertBefore(item);
            current=tmp;
           }
      else Next();
    }
  }
```

IV.8. Cây nhị phân

Cây nhị phân gồm một tập hữu hạn các nút (hay đỉnh) và một tập hữu hạn các cạnh nối các cặp nút với nhau. Mỗi nút của cây bao gồm thông tin với một liên kết đến phần tử bên trái và một liên kết với phần tử bên phải.

- Trong đó có một nút đặc biệt gọi là nút gốc (root) là nút không có cạnh nào hướng tới nó.

- Các nút lá (leaf) là những nút ở đó không có cạnh nào đi ra.



- Nút truy cập đến nút khác là nút bố (parent) và nút được truy cập là nút con (child).
- Một cây nhị phân đầy đủ là cây mỗi nút đều có hai con (trừ lá).
- Bản thân một nút và con của nó cũng là một cây gọi là cây con (subtree).
- Mức của nút là số cung đi từ gốc đến lá cộng thêm một. Gốc của cây có số mức là một.
- Chiều cao một cây là sô mức lớn nhất của nút có trên cây đó.
- Số con của một nút có bốn trường hợp: có hai con, không có con nào (lá), có một con trái và một con phải.

IV.8.1. Giá trị khoá

Cây nhị phân có trật tự (ordered binary tree) mỗi nút đều có một khoá tuân theo nguyên tắc:

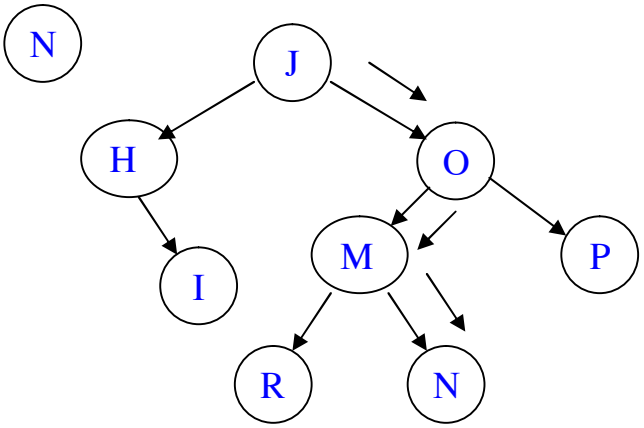
- Các giá trị khoá không trùng nhau.
- Đối với một nút: giá trị khoá của nó lớn hơn giá trị khoá của các nút con trái của nó và ngược lại nhỏ hơn so với các nút con phải của nó.

IV.8.2. Tìm kiếm

Việc tìm kiếm dựa trên khoá chứa trong các nút:

- Nếu khoá phù hợp với khoá tìm kiếm dừng --> đã tìm thấy.
- Nếu khoá nhỏ hơn khoá của nút --> tìm nút con trái của nó.
- Nếu khoá lớn hơn khoá của nút --> tìm nút con phải của nó.
- Quay lại bước một.

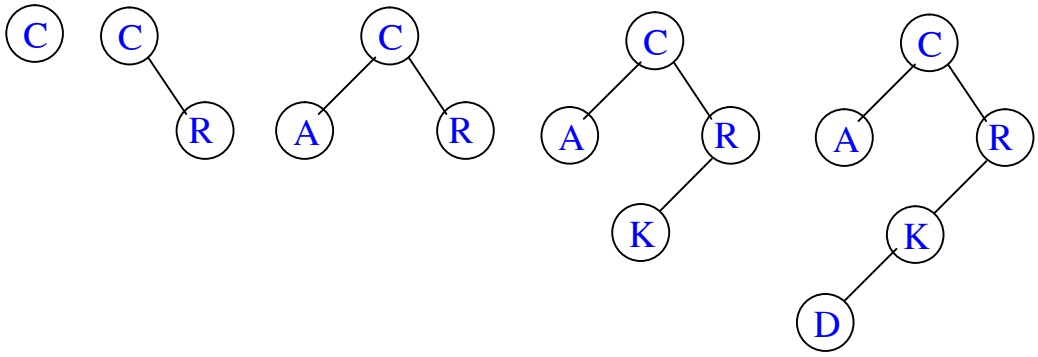
Việc tìm kiếm bắt đầu từ gốc và lặp theo quá trình ở trên cho đến khi tìm thấy hoặc nút con là rỗng là không tìm thấy.



Hình Tìm N trong cây nhị phân

IV.8.3. Thêm một nút mới

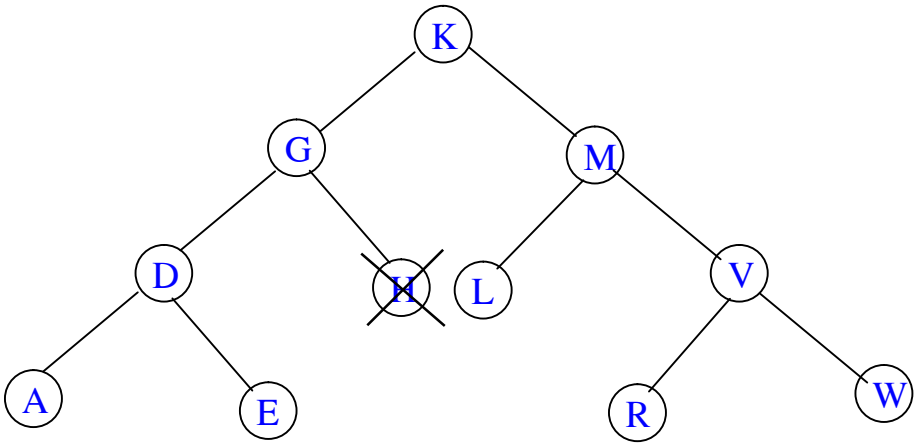
Thêm một nút mới vào cây rỗng nó trở thành gốc của cây. Quá trình tìm vị trí thích hợp thêm vào cây cũng giống như việc tìm kiếm. Nếu khoá của nút mới không có trong cây. Việc tìm kiếm kết thúc ở vị trí rỗng (NULL) nào đó. Khi đó ta nối nút mới vào vị trí này và trở thành lá của cây.



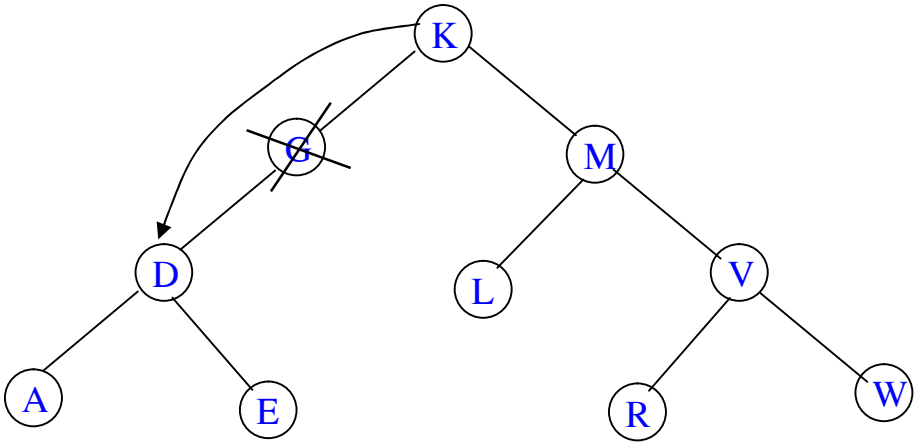
Hình minh hoạ quá trình tạo một cây

IV.8.4. Xoá một nút

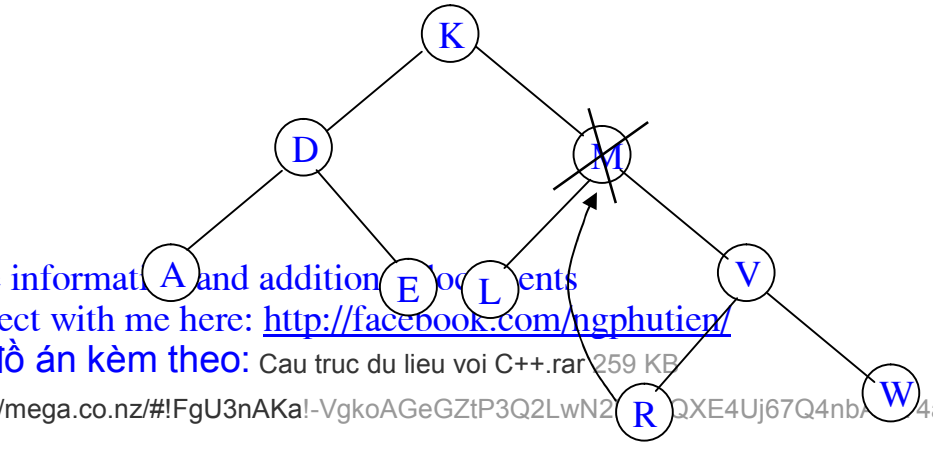
Xoá một nút trên cây nhị phân có ba trường hợp: nút cần xoá có hai con, một con và không có con nào. Việc xoá phải đảm bảo tất cả các nút vẫn theo quy tắc của cây nhị phân có thứ tự.



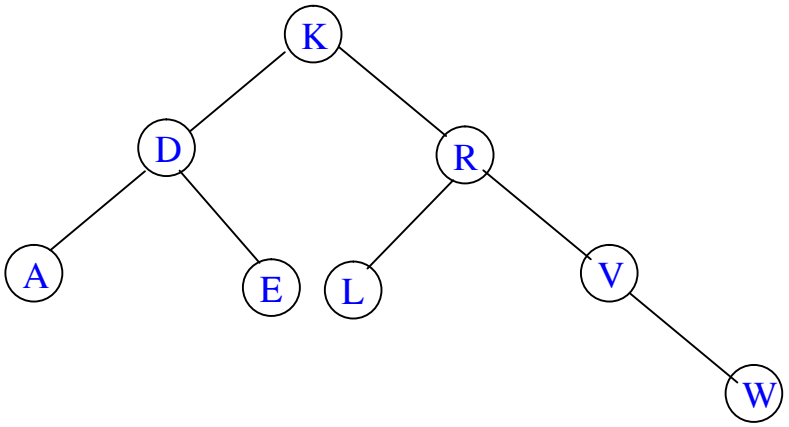
Hình xoá một lá



Hình xoá nút có một con



Hình xoá một nút có hai con



Hình kết quả

IV.8.5. Các phương pháp duyệt cây

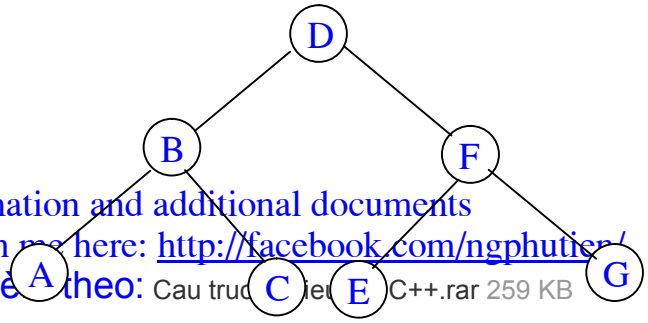
Có ba cách duyệt cây: trung tố (Inorder), tiền tố (Preorder) và hậu tố (Postorder).

Duyệt cây trung tố: duyệt cây con trái trước, duyệt gốc, duyệt cây con phải.

Duyệt cây tiền tố: duyệt gốc trước, duyệt cây con trái, duyệt cây con phải.

Duyệt cây hậu tố: duyệt con trái trước, duyệt con phải, duyệt gốc.

Ví dụ:



Hình minh hoạ

Duyệt trung tố: ABCDEFG.

Duyệt tiền tố: DBACFEG.

Duyệt hậu tố: ACBEGFD.

IV.8.6. Lớp dữ liệu mẫu

```
  enum TravType{inord,preord,postord};
//----------------------------------------
template<class K,class D>
struct TNode{
  K key;
  D data;
  TNode<K,D> *parent,*lchild,*rchild;
  TNode(const K & k,const D & d);
  TNode(const TNode<K,D> & node)  {NCopy(node);}
  void operator = (const TNode<K,D> & node) {NCopy(node);}
  private:
    void NCopy(const TNode<K,D> & node);
 };
//-----------------------------------------------
template<class K,class D>
TNode<K,D>::TNode(const K & k,const D & d)
  {
   key=k;
   data=d;
   parent=lchild=rchild=NULL;
  }
//-----------------------------------------------
template<class K,class D>
void TNode<K,D>::NCopy(const TNode<K,D> & node)
  {
   key=node.key;
   data=node.data;
   parent=node.parent;
   lchild=node.lchild;
   rchild=node.rchild;
  }
```

```
//-----------------------------------------------
//TREE
//-----------------------------------------------
template<class K,class D>
class BTree
  {
    protected:
          TNode<K,D> *root;
          int Copy(TNode<K,D>* node);
          void Erase(TNode<K,D> * node);
          void (*Travfunc)(const K & key,const D & data);
          void Inorder(TNode<K,D> * node);
          void Preorder(TNode<K,D> * node);
          void Postorder(TNode<K,D> * node);
    public:
          BTree()   {root=NULL;}
          BTree(const BTree<K,D> & tree);
          ~BTree()  {Erase(root);}
          int operator =(const BTree<K,D> & tree);
          int Insert(const K & key,const D & data);
          int Delete(const K & key);
          void Traverse(void(*func)(const K & key, const D & data),const TravType &
ord=inord);
          int Change(const K & key,const D & data);
          int Search(const K & key,D & data);
  };
//-----------------------------------------------------------------------
template<class K,class D>
int BTree<K,D> ::Copy(TNode<K,D> * node)
  {
   if(node){
     if(Insert(node->key,node->data))return 1;
     Copy(node->lchild);
     Copy(node->rchild);
    }
   return 0;
  }
//-----------------------------------------------------------------------
template<class K,class D>
void BTree<K,D>::Erase(TNode<K,D> * node)
  {
   if(node){
      Erase(node->lchild);
      Erase(node->rchild);
      delete node;
     }
  }

//-----------------------------------------------------------------------
```

```
template<class K,class D>
BTree<K,D> ::BTree(const BTree<K,D> & tree)
  {
    root=NULL;
    Copy(tree.root);
  }
//---------------------------------------------------------------------
template<class K,class D>
int BTree<K,D> ::operator=(const BTree<K,D> & tree)
 {
   Erase(root);
   root=NULL;
   if(Copy(tree.root))return 1;
   return 0;
 }
//---------------------------------------------------------------------
template<class K,class D>
int BTree<K,D> ::Insert(const K & key,const D & data)
  {
    TNode<K,D>* newnode= new TNode<K,D>(key,data);
    if(!newnode) return 1;
    if(!root) root=newnode;
    else{
      TNode<K,D> * node=root;
      while(1){
            if(node->key < newnode->key){
              if(!node->rchild){
                 node->rchild=newnode;
                 newnode->parent=node;
                 return 0;
               }
              else node=node->rchild;
             }
            else{
              if(!node->lchild){
                node->lchild=newnode;
                newnode->parent=node;
                return 0;
               }
              else node=node->lchild;
             }
            }
      }
    return 0;
  }

//---------------------------------------------------------------------
template<class K,class D>
int BTree<K,D>::Search(const K & key,D & data)
```

```
  {
   TNode<K,D> * node=root;
   while(node){
     if(key==node->key){
            data=node->data;
            return 0;
      }
     if(key>node->key) node=node->rchild;
     else node=node->lchild;
    }
   return 1;
  }
//---------------------------------------------------------------------
template<class K,class D>
int BTree<K,D> ::Change(const K & key,const D & data)
  {
    TNode<K,D> * node=root;
    while(node){
      if (key==node->key) break;
      if(key > node->key) node=node->rchild;
      else node=node->lchild;
    }
    if(node) {
       node->data= data;
       return 0;
     }
    else return 1;
  }
//---------------------------------------------------------------------
template<class K,class D>
void BTree<K,D> ::Traverse(void (*func)(const K & key,const D & data),const
TravType & ord)
  {
    Travfunc=func;
    switch(ord)
    {
      case preord:
            Preorder(root);
            break;
       case postord:
            Postorder(root);
            break;
      default :
             Inorder(root);
             break;
    }
  }
//---------------------------------------------------------------------
template<class K,class D>
```

```
void BTree<K,D> ::Inorder(TNode<K,D> * node)
  {
   if(node){
     Inorder(node->lchild);
     Travfunc(node->key,node->data);
     Inorder(node->rchild);
    }
  }
//---------------------------------------------------------------------
template<class K,class D>
void BTree<K,D> ::Preorder(TNode<K,D> * node)
  {
   if(node){
     Travfunc(node->key,node->data);
     Preorder(node->lchild);
     Preorder(node->rchild);
    }
  }
//---------------------------------------------------------------------
template<class K,class D>
void BTree<K,D> ::Postorder(TNode<K,D> * node)
  {
   if(node){
     Postorder(node->lchild);
     Postorder(node->rchild);
     Travfunc(node->key,node->data);
    }
  }
//---------------------------------------------------------------------
template<class K,class D>
int BTree<K,D>::Delete(const K & key)
  {
   TNode<K,D> * node=root;
   while (node){
      if(key==node->key) break;
      if(key > node->key) node=node->rchild;
      else node=node->lchild;
     }
  if(!node) return 1;
  if(!node->rchild){
      if(!node->lchild){
             if(node==root) root=NULL;
             else{
              if(node->parent->lchild==node) node->parent->lchild=NULL;
              else node->parent->rchild=NULL;
             }
           }
      else{
            if(node==root) root=node->lchild;
```

```
            else
             if(node->parent->lchild==node) node->parent->lchild=node->lchild;
             else node->parent->rchild=node->lchild;
            }
   }
   else{
    if(!node->lchild){
          if(node==root) root=node->rchild;
          else{
           if(node->parent->lchild==node)node->parent->lchild=node->rchild;
           else node->parent->rchild=node->rchild;
          }
    }
    else{
     TNode<K,D> *temp=node->rchild;
     while(temp->lchild) temp=temp->lchild;
     if(temp->parent->lchild==temp) temp->parent->lchild=temp->rchild;
     else temp->parent->rchild=temp->rchild;
    node->key=temp->key;
    node->data=temp->data;
    if(temp->rchild) temp->rchild->parent=temp->parent;
    node=temp;
     }
    }
  delete node;
  return 0;
}
```

IV.9. Nhận xét

Phương pháp biểu diễn móc nối nói chung cồng kềnh hơn phương pháp biểu diễn liên kết và việc truy nhập tới các phần tử cũng chậm hơn. Ngược lại, việc cập nhật không yêu cầu các động tác sao chép nên ít tốn kém hơn so với phương pháp biểu diễn liên tục. Ta thường chọn phương pháp biểu diễn móc nối khi vẫn để cập nhật trở nên quan trọng hơn vấn đề truy nhập và ngược lại.

Phần B

Các chương trình ứng dụng

more information and additional documents
connect with me here: http://facebook.com/ngphutien/
file đồ án kèm theo: Cau truc du lieu voi C++.rar 259 KB
https://mega.co.nz/#!FgU3nAKa!-VgkoAGeGZtP3Q2LwN2LKMQXE4Uj67Q4nbAJrT4aGB8

I. Quản lý sinh viên

Như đã biết, công việc quản lý sinh viên là rất quan trọng đối với bất kỳ một nhà trường nào. Đó là công việc phức tạp, đòi hỏi tính chính xác. Tuy nhiên tôi không muốn đi sâu vào chi tiết, chương trình tôi xây dựng chỉ mang tính mô phỏng, nhưng vẫn mang đầy đủ những chức năng cơ bản cần thiết.

Dữ liệu bao gồm: mã số, họ tên, ngày sinh, lớp, điểm trung bình và ghi chú.

Chương trình này được xây dựng trên hai lớp cấu trúc dữ liệu đã được thiết kế ở phần trước đó là: Danh sách liên kết đôi và ngăn xếp. Danh sách liên kết đôi dùng để lưu trữ và xử lý những bản ghi, còn ngăn xếp để lưu trữ những bản ghi đã bị xoá và có thể phục hồi khi cần thiết. Như đã biết ngăn

xếp hoạt động theo nguyên tắc "Vào sau ra trước" tức là nó sẽ phục hồi những bản ghi tuần tự theo thời gian gần nhất.

Chương trình có một số chức năng chính sau:

- Thêm mới một bản ghi, sử dụng phương thức thêm danh sách liên kết đôi (Append). Việc trùng khóa trong dữ liệu không được phép. Vì vậy, trước khi thêm mới phải kiểm tra đã có khoá (mã sinh viên) có trong dữ liệu chưa. Nếu có đưa ra thông báo, ngược lại bản ghi được cập nhật trong danh sách. Chức này sử dụng phương thức tìm kiếm (Find) của danh sách liên kết đôi.
- Chèn một bản ghi sử dụng phương thức chèn (Insert) cũng giống như việc thêm mới, trước khi chèn một bản ghi phải kiểm tra đã có khoá này trong dữ liệu chưa bằng phương thức tìm kiếm Find.
- Chuyển đến bản ghi kế tiếp sử dụng phương thức di chuyển (Next). Việc di chuyển sẽ bị huỷ bỏ nếu đang ở cuối danh sách. Phương thức AtTail cho biết có phải đang ở cuối danh sách không.
- Chuyển đến bản ghi trước sử dụng phương thức Prev của danh sách liên kết đôi. Việc di chuyển không được thực hiện nếu đang ở đầu danh sách. Phương thức AtHead cho biết có phải đang ở đầu danh sách không.
- Trở về bản ghi đầu tiên sử dụng phương thức Tohead.
- Trở về bản ghi cuối sử dụng phương thức ToTail.
- Xoá bản ghi hiện tại sử dụng phương thức Delete. Ngoài ra bản ghi bị xoá được lưu trữ vào ngăn xếp bằng phương thức đưa vào (Put). Giới hạn của ngăn xếp bằng 50. Nếu tràn ngăn xếp được tự động làm rỗng bằng phương thức Flush.
- Xoá hết danh sách sử dụng phương thức Erase. Các bản ghi bị xoá lưu trữ vào ngăn xếp.
- Sắp xếp danh sách tăng hoặc giảm theo tên sinh viên sử dụng phương thức Sort.

- Tìm kiếm một bản ghi theo mã hoặc theo tên sinh viên. Việc tìm kiếm theo họ tên có thể sử dụng '*' để thay cho các ký tự bất kỳ. Việc tìm kiếm được sử dụng phương thức Find.

- Phục hồi các bản ghi đã bị xoá sử dụng phương thức lấy dữ liệu của ngăn xếp (Get). Việc lấy dữ liệu ra được thực hiện khi ngăn xếp không rỗng, tức phương thức GetCount khác không.

Ngoài ra còn có một số chức năng khác như: mở tệp, ghi vào tệp, trợ giúp, thông báo, sử dụng chuột,...

Mã nguồn của chương trình có trong đĩa mềm kèm theo bài luận văn này.

Sau đây là một số hàm chính:

```
void Menu();
void Show();
void Hide();
void Info();
int AddNew();
void Key();
void File();
void Open(int name=0);
void SaveAs();
void Save();
void Find();
void SortAsc();
void SortDesc();
int Exit();
void Begin();
void Next();
void Prev();
void End();
void New();
void Ins();
void Del();
void Erase();
void Fresh();
void Act();
void Total();
int View();
char * Table( const SV & view );
char * Vert(char *txt,int len);
void InitMouse();
void ShowMouse();
void HideMouse();
void SetMousePos(int x,int y);
void MouseSpeed(int sp);
void GetPos(int & x,int & y);
```

```
int Click(int & x,int & y);
int Process(char ch);
char Select(int & x,int & y);
void Store(SV & sta);
void Restore();
void Help();

DList<SV> list;
SV tmp;
const int STK_LEN = 50;
Stack<SV,50> stack;
char filename[50];
FILE *f;
//-----------------------------------------------------
void main()
{
  Menu();
  InitMouse();
  ShowMouse();
//  HideMouse();
  char ch;
  int flag=1;
  int x,y;
  while(flag)
  {
      if(kbhit())
      {
        ch=getch();
        ch=toupper(ch);
        if(Process(ch)) flag=0;//ket thuc chuong trinh
       }
       if(Click(x,y)==1)
       {
            ch=Select(x,y);
            if(Process(ch)) flag=0;
       }
   }
}
//-----------------------------------------------------
int  Process(char ch) //tra lai 1 khi EXIT
{
      int flag=0;
      ch=toupper(ch);
      switch(ch)
      {
         case 'V':
               ch=View();
         case 'B':
               Begin();break;
         case 'N':
               Next();break;
         case 'P':
               Prev();break;
         case 'E':
               End();break;
         case 'W':
               New();break;
```

```
         case 'I':
               Ins();break;
         case 'D':
               Del();break;
         case 'L':
               Erase();break;
         case 'R':
               Restore();break;
         case 'O':
               File();break;
         case 'S':
               Save();break;
         case 'A':
              SaveAs();break;
         case 'F':
              Find();break;
         case '0':
              SortAsc();break;
         case '1':
              SortDesc();break;
         case 'H':Help();break;
         case 'X':
              if(Exit())  flag=1;
              break;
       }
   return flag;
}
void Store(SV & sta)
{
 if(stack.GetCount()==STK_LEN) stack.Flush();
 if(strlen(sta.ma)) stack.Put(sta);
}
//-----------------------------------------------------
void Restore()
{
      SV sv;
      char txt[100];
      if( !stack.GetCount())
         msg="Khong con ban ghi nao de phuc hoi...";
      if(stack.Get(sv))return;
      Input ques(10,10,"Ban co muon hoi phuc(C\\K) ");

      strcat(ques.disp,sv.hoten);
      strcat(ques.disp,"(ma so: ");
      strcat(ques.disp,sv.ma);
      strcat(ques.disp,")");
      char *str=ques.Text(1);
      if(str[0]=='c' || str[0]=='C')
      {
         list.Append(sv);
         list.ToTail();
       }
       else stack.Put(sv);
       ques.Hide();
       Act();
}
//-----------------------------------------------------
```

```
void Key()
{
  char *txt;
  Display guide(10,2,"Chon cach tim kiem",LIGHTGREEN);
  Display guid1(10,3,"1: Theo ma hoc sinh",YELLOW);
  Display guid2(10,4,"2: Theo ten hoc sinh",YELLOW);
  guide.Show();
  guid1.Show();
  guid2.Show();
  Display   choice(10,6,"Chon :  ",MAGENTA);
  choice.Show();
  char ch='0';
  int mx,my;
  while(ch=='0')
  {
    if(kbhit()){ch=getch();ch=toupper(ch);}
    if(Click(mx,my))
     {
      my-=2;
      if((my==3)&&(mx>10)&&(mx<10+strlen("1: Theo ma hoc sinh")))
ch='1';
      if((my==4)&&(mx>10)&&(mx<10+strlen("1: Theo ten hoc sinh")))
ch='2';
      }
   }
   txt[0]=ch;
   txt[1]='\0';
   strcat(choice.disp,txt);
   choice.Show();
  Input input(10,9,"La : ",LIGHTCYAN);
  if(ch=='1')
  {
   txt=input.Text();
   strcpy(tmp.ma,txt);
  }
  if(ch=='2')
   {
    Display guid3(10,8,"Co the dung '*' cho ki tu bat ky",CYAN);
    guid3.Show();
    txt=input.Text();
    strset(tmp.ma,NULL);
    strcpy(tmp.hoten,txt);
  }
  clrscr();
}
//-----------------------------------------------------
void Find()
{
  clrscr();
  Hide();
  Key();
  if(!strlen(tmp.ma) && !strlen(tmp.hoten)){
    Act();
    return;
  }
  if(list.Find(tmp))
  {
```

```
     Act();
     return;
  }
  else{
    msg=" Khong tim thay !";
    list.ToHead();
    Act();
  }
}
//---------------------------------------------------
void SortAsc()
{
  if(!list.Count())return;
  Hide();
  clrscr();
  list.Sort();
  list.ToHead();
  Act();
}
//---------------------------------------------------
void SortDesc()
{
  if(!list.Count()) return;
  list.Sort(DESC);
  Hide();
  clrscr();
  list.ToHead();
  Act();
}
//---------------------------------------------------
void Begin()
{
  Hide();
  list.ToHead();
  Act();
}
//---------------------------------------------------
void Next()
{
//  Hide();
  if(list.AtTail())
  {
     msg="Da den cuoi danh sach! ";
     Show();
     return;
  }
  list.Next();
  Act();
}
//---------------------------------------------------
void Prev()
{
  Hide();
  if(list.AtHead())
  {
  msg="Da den dau danh sach! ";
  Show();
```

```
  return;
  }
  list.Prev();
  Act();
}
//---------------------------------------------------
void End()
{
  Hide();
  list.ToTail();
  Act();
}
//---------------------------------------------------
int AddNew()
{
  char * txt;//[100];
  txt=nhapma.Text();
  if(!txt) return 1;
  strncpy(tmp.ma,txt,MA_LEN-1);
  if(list.Find(tmp))
  {
   msg="Da co ma hoc sinh nay trong du lieu! ";
   return 1;
  }
  txt=nhapten.Text();
  strncpy(tmp.hoten,txt,TEN_LEN-1);
  txt=nhaplop.Text();
  strncpy(tmp.lop,txt,LOP_LEN-1);
  txt=nhapdiem.Text();
  strncpy(tmp.dtb,txt,DTB_LEN-1);
  txt=nhapngay.Text();
  strncpy(tmp.ngaysinh,txt,NGAY_LEN-1);
  txt=nhapchu.Text();
  strncpy(tmp.ghichu,txt,GHI_LEN-1);
  return 0;
}

//---------------------------------------------------
void New()
{
   Hide();
   if(AddNew()){
     clrscr();
     Act();
     return;
   }
   if(!strlen(tmp.ma))
   {
     Act();
     return;
   }
   list.Append(tmp);
   list.ToTail();
   Act();
   Total();
}
//---------------------------------------------------
```

```
void Ins()
{
  Hide();
  if(AddNew())
  {
     clrscr();
     Act();
     return;
  }
  if(!strlen(tmp.ma))
  {
    Act();
    return;
  }
  list.Insert(tmp);
  Act();
  Total();
}
//-----------------------------------------------------
void Del()
{
  Hide();
  char ques[100];
  strcpy(ques,"Ban co muon xoa(C\K) :");
  strcat(ques,tmp.hoten);
  strcat(ques,"( ");
  strcat(ques,tmp.ma);
  strcat(ques," )");
  Input input(10,10,ques);
  char *txt=input.Text(1);
  if(txt[0]=='c' || txt[0]=='C')
  {
  Store(tmp);
  list.Delete();
  }
  clrscr();
  Act();
//  Total();
}
//-----------------------------------------------------
void Erase()
{
  Hide();
  Input flu(10,10,"Ban muon xoa het?(C/K) ",RED);
  char *txt=flu.Text(1);
  strcat(flu.disp,txt);
  flu.Hide();
  strcpy(flu.disp,flu.caption);
  if(*txt=='c' || *txt=='C')
  {
       list.ToHead();
       while(!list.Get(tmp))
       {
        stack.Put(tmp);
        list.Next();
       }
       list.Erase();
```

```
        tongso="0";
        Fresh();
        Show();
        Total();
   }
}
//---------------------------------------------------
```

II. Thống kê từ tiếng Việt

Trong các nghành khoa học, kinh tế, cũng như đối với tin học. Thống kê là một công việc vô cùng quan trọng, việc thống kê có bao nhiêu từ tiếng Việt cũng như số lần xuất hiện của nó không chỉ có ích cho các nhà ngôn ngữ học, mà còn giúp cho những người làm tin học có được cách mã hoá tối ưu và phù hợp nhất.

Trong chương trình. Tôi dựa vào tính đơn âm tiết cửa tiếng Việt để loại bỏ những từ không phù hợp. Sau khi thống kê có thể tìm kiếm và loại bỏ nốt những từ không phải là tiếng Việt. Kết quả được lưu vào đĩa theo thứ tự bảng chữ cái hoặc theo số lần xuất hiện.

Chương trình được xây dựng dựa trên các lớp dữ liệu mẫu: cây nhị phân và danh sách liên kết đơn.

Cây nhị phân là cấu trúc dữ liệu có tốc độ tìm kiếm nhanh nhất, rất thích hợp để lưu trữ một lượng dữ liệu lớn. Danh sách liên kết đơn được sử dụng để sắp xếp các từ theo thứ tự giảm dần. Kết quả thống kê được lưu vào tệp để nghiên cứu.

Thuật toán, khi phân tách một từ và phân tích từ đó thoả mãn là từ tiếng Việt từ đó sẽ tiếp tục được xử lý như sau: Kiểm tra trong cây nhị phân đã có từ này chưa bằng phương thức Search. Nếu có sẽ tăng số lần xuất hiện của từ này lên một bằng phương thức Change, ngược lại nếu chưa có thì thêm từ này vào cây nhị phân và gán số lần xuất hiện của từ này bằng một bằng phương thức Append. Sau khi thống kê có thể xem kết quả và những từ nào không phù hợp có thể xoá băng các phương thức duyệt cây nhị phân và phương thức Delete.Việc ghi kết quả vào tệp theo hai cách. Cách thứ nhất ghi số từ xuất hiện theo thứ tự chữ cái bằng phương thức duyệt trung thứ tự. Cách thứ hai gán kết quả vào một danh sách liên kết đơn, sau đó sắp

xếp theo thứ tự giảm dần theo số lần xuất hiện bằng phương thức Sort và lưu vào tệp.

Ngoài ra còn có một số chức năng khác như: mở tệp, ghi vào tệp, trợ giúp, thông báo, sử dụng chuột,...

Mã nguồn của chương trình có trong đĩa mềm kèm theo bài luận văn này

Sau đây là một số hàm quan trọng của chương trình này:

```
//-----------------------------------------------------
void Menu();
void Open(int name=0);
void Save();
void TravAlpha(const String & txt,const size_t & num);
void SaveFre(const String & txt,const size_t & num);
void SaveFre();
void TravFre(const String & txt,const size_t & num);
void Find();
void Del();
void View();
void TravView(const String & txt,const size_t & num);
int  Exit();
void InitMouse();
void ShowMouse();
void HideMouse();
void SetMousePos(int x,int y);
void MouseSpeed(int sp);
void GetPos(int & x,int & y);
int Click(int & x,int & y);
int Process(char ch);
char Select(int & x,int & y);
void Help();
void File();
BTree<String,size_t> tree;
SList<Freq> list;
char filename[100];

//-----------------------------------------
void main()
{
  Menu();
  InitMouse();
  ShowMouse();
//  HideMouse();
  char ch;
  int flag=1;
  int x,y;
  while(flag)
  {
      if(kbhit())
      {
```

```
         ch=getch();
         ch=toupper(ch);
         if(Process(ch)) flag=0;//ket thuc chuong trinh
        }
        if(Click(x,y)==1)
        {
             ch=Select(x,y);
             if(Process(ch)) flag=0;
        }
    }

}
//------------------------------------------------------
int Process(char ch)
{
   ch=toupper(ch);
   switch(ch)
   {
      case 'O': File();clrscr();break;
      case 'S': Save();clrscr();break;
      case 'V': View();clrscr();break;
      case 'F': Find();clrscr();break;
      case 'D': Del(); clrscr();break;
      case 'H': Help();break;
      case 'E': if(Exit()) return 1;
   }
   return 0;

}
//------------------------------------------------------
void Open(int name)
{
   clrscr();
   Input esc(3,5,"Ban muon dung chuong trinh ?(C/K) ",YELLOW);
   Display title(10,4,"Thong ke tu",LIGHTGREEN);
   title.Show();
   ActDisp dict(10,7,"Tu dien : ");
   ActDisp cou(10,8,"So tu da xu ly: ");
   ActDisp comp(10,9,"Hoan thanh : ");

   if(!name)
   {
   Input path(10,4," Ten tep:",YELLOW);
   char *txt=path.Text();
   strcat(path.disp,txt);
   strcpy(filename,txt);
   }
   FILE *f,*ftmp;
   ftmp=fopen("temp.dat","wb");
   if((f=fopen(filename,"rb"))==NULL)
   {
    Msg msg(10,7,"");
    msg="Khong mo duoc tep !";
    clrscr();
    return;
   }
   Display guide(10,14,"An ESC de thoat",YELLOW);
```

more information and additional documents
connect with me here: http://facebook.com/ngphutien/
file đồ án kèm theo: Cau truc du lieu voi C++.rar 259 KB
https://mega.co.nz/#!FgU3nAKa!-VgkoAGeGZtP3Q2LwN2LKMQXE4Uj67Q4nbAJrT4aGB8

```
 guide.Show();

const size_t MAXLEN=9;
const size_t MAXBUF=3000;

String tmp(MAXLEN);
char buf [ MAXBUF ];

size_t pos,numread,count;
int flag=0,l=0;
unsigned long sum=0;//,amount=0;

char *p;

fseek(f,sizeof(char),SEEK_END);
unsigned long  filesize;
filesize = ftell(f);
rewind(f);

while (!feof(f))
{
  flag=1;pos=0;
  numread=fread(buf,sizeof(char),MAXBUF,f);

  while(pos<numread)
  {
    p=(char *)tmp;l=0;count=0;
    if(buf[pos]==10)
    {
    pos++;
    continue;
    }
    while(IsChar(buf[pos]) && pos<numread)
    {
    if(l<MAXLEN-2) *(p+l)=buf[pos];
    l++;
    pos++;
    }
    *(p+l)=NULL;
    p=tmp;
    Lower(p);
    if(l && l<MAXLEN-1 && IsVNWord(tmp))
    {
    if(!tree.Search(tmp,count))
       tree.Change(tmp,++count);
    else{
       tree.Insert(tmp,1);
       String test;//err la bien kiem tra
       if(test.Err()){
         Msg over(5,10,"",LIGHTRED);
         over="Tran bo nho";
         fseek(f,filesize, SEEK_SET);
         break;
       }
     }
    fwrite(tmp,sizeof(char),l,ftmp);
    fputc(' ',ftmp);
```

```
      sum++;
      flag=1;
      }
      else
      if(flag)
      {
        fputc(10,ftmp);
        flag=0;
      }
     pos++;

      cou=sum;
      comp = ftell(f)*100/filesize;
      dict=tree.Count();
      char ch;
      ch='\0';
      if(kbhit()){
        ch=getch();
        if(ch==27) ch='C';
      }
      int x,y;
      if(Click( x, y)){
//    Display guide(10,14,"An ESC de thoat",YELLOW);
          y-=2;
          if((y==14) && ((x>10) &&(x< 10+strlen("An ESC de thoat"))))
ch='C';
        }
       if(ch=='C') {
           char * choice=esc.Text(1);
           if(choice[0]=='C'||choice[0]=='c')
           {
             fseek(f,filesize, SEEK_SET);
             esc.Hide();
             break;
           }
           esc.Hide();
         }
      }
    }
  fclose(f);
  fclose(ftmp);
  Msg suc(10,20,"",YELLOW);
  suc= " Da hoan thanh ";

}
//--------------------------------------
FILE *f;
//--------------------------------------
void Save()
{
   Input path(10,5,"Ghi vao tep: ");
   char *txt=path.Text();
   if((f=fopen(txt,"wb"))==NULL)
   {
     Msg msg(10,7,"");
     msg="Khong mo duoc tep! ";
     return;
```

more information and additional documents
connect with me here: http://facebook.com/ngphutien/
file đồ án kèm theo: Cau truc du lieu voi C++.rar 259 KB
https://mega.co.nz/#!FgU3nAKa!-VgkoAGeGZtP3Q2LwN2LKMQXE4Uj67Q4nbAJrT4aGB8

```
   }
  Display guide(10,2,"Chon mot trong hai cach ghi sau:",LIGHTGREEN);
  Display guid1(10,3,"1: Ghi theo thu tu chu cai",YELLOW);
  Display guid2(10,4,"2: Ghi theo so lan xuat hien",YELLOW);
  Display esc(3,2," An ESC de thoat",YELLOW);
  esc.Show();
  clrscr();
  guide.Show();
  guid1.Show();
  guid2.Show();
  Display   choice(10,6,"Chon :  ",MAGENTA);
  choice.Show();
  char ch='0';
  int mx,my;
  while(ch=='0')
  {
    if(kbhit()){ch=getch();ch=toupper(ch);}
    if(Click(mx,my))
     {
      my-=2;
      if((my==3)&&(mx>10)&&(mx<10+strlen("1: Ghi theo thu tu chu
cai"))) ch='1';
      if((my==4)&&(mx>10)&&(mx<10+strlen("2: Ghi theo so lan xuat
hien"))) ch='2';
      if((my==2)&&(mx>3)&&(mx<10+strlen("an ESC de thoat"))) ch='2';
      }
   }
   txt[0]=ch;
   txt[1]='\0';
   strcat(choice.disp,txt);
   choice.Show();
  Input input(10,9,"La : ",LIGHTCYAN);
   if(txt[0]=='1') tree.Traverse(TravAlpha);
   if(txt[0]=='2')  SaveFre();

    Msg succ(10,20,"",YELLOW);
    succ="Da hoan thanh";
    fclose(f);
    list.Erase();
}
//-----------------------------------------------------
int overflow;
void SaveFre()
{
  Freq tmp;
  char  str[10];
  overflow=0;
  tree.Traverse(TravFre);
  ActDisp warning(10,14,"",YELLOW);
  ActDisp over(10,15,"Trao bo nho",YELLOW);
  if(list.Count() !=tree.Count())
  {
    int num=tree.Count()-list.Count();
    if(overflow) over.Show();
    warning.CharNum("Du lieu co the bi mat khoang: ",num);
  }
  list.Sort(ASC);
```

```
  list.ToHead();
  if(!list.Count()) return ;
   while(1){
    if(list.Get(tmp))break;
    fputs(tmp.txt,f);
    ltoa(tmp.count,str,10);
    fputs(" ",f);
    fputs(str,f);
    fputc(10,f);
  }

}
//-------------------------------------------------------
void TravFre(const String & str,const size_t & num)
{
    Freq fre(str,num);
    if(!fre.Err())return;
    fre.txt=str;
    fre.count=num;
  list.Append(fre);
}
//-------------------------------------------------------
void TravAlpha(const String & str,const size_t & num)
{
   static char tmp[10];
   long l=long(num);
   ltoa(l,tmp,10);
   fputs(str,f);
   fputs("  ",f);
   fputs(tmp,f);
   fputc(10,f);
}
//-------------------------------------------------------
void Del()
{
  clrscr();
  Msg msg(10,7,"",LIGHTCYAN);
  Input input(10,5,"Nhap tu can xoa: ");
  String str;
  str=input.Text();
//  strcat(input.disp,str);
  if( !tree.Delete(str)) msg="Da xoa...";
  else msg="Khong xoa duoc";
}
//---------------------------------------
void Find()
{
  clrscr();
  Input input(10,5,"Tim tu:  ");
  ActDisp act(10,7,"So lan xuat hien: ");
  Msg msg(10,8,"",MAGENTA);
  char * txt=input.Text();
  size_t num;
  String str;
  str=txt;
  if(tree.Search(str,num)) msg="Khong tim thay";
  else{
```

```
    act=num;
    msg=" ";
   }
}
//-------------------------------------
```

Kết luận

Cùng với sự phát triển mạnh mẽ của công nghệ thông tin, thì cấu trúc dữ liệu như là nền tảng của sự phát triển này. Với sự ưu việt của phương pháp lập trình hướng đối tượng là tính kế thừa và sự linh động đối với các kiểu dữ liệu khác nhau của các lớp mẫu. Các lớp cấu trúc dữ liệu mẫu của C^{++} rất phù hợp để xây dựng các chương trình lớn và đa dạng.

Các lớp cấu trúc dữ liệu mẫu: ngăn xếp, hàng đợi, hàng quay tròn, danh sách liên kết đơn, liên kết đôi và cây nhị phân mà tôi đã xây dựng ở trên. Tôi hy vọng các lớp cấu trúc dữ liệu mẫu này được sử dụng hoặc có ích cho những người lập trình. Tuy rằng đã cố gắng xây dựng xây dựng các lớp một cách tổng quát và chi tiết nhất nhưng do thời gian và khả năng còn hạn chế nên không tránh khỏi những thiếu sót. Ngoài các kiểu cấu trúc đã trình bày còn có một số kiểu cấu trúc dữ liệu khác: bảng băm, cây n phân, ... mà trong khuôn khổ của một bài luận văn tốt nghiệp tôi không có điều kiện để trình bày. Tôi sẽ cố gắng hoàn thiện và nghiên cứu để làm giàu thêm kiến thức của mình. Và tôi hy vọng trong tương lai có thể xây dựng được các chương trình có giá trị sử dụng thực tế, góp phần thúc đẩy sự phát triển nền công nghệ thông tin nước ta.

Tài liệu tham khảo

[1] J.COURTIN & I.KOWARSKI Nhập môn thuật toán và cấu trúc dữ liệu (Tập II), Viện tin học – Khoa học Việt Nam 1993

[2] LARRY NYHOFF & SANFORD LEEDSTMA - Lập trình nâng cao bằng PASCAL với các cấu trúc dữ liệu (Tập II), NXB Đà Nẵng 1998

[3] TRẦN VĂN LĂNG - Lập trình hướng đối tượngC++ , NXB thống kê

[4] NGUYỄN CẨN - C Tham khảo toàn toàn diện, NXB Đồng Nai 1996

[5] NGÔ TRUNG VIỆT - Ngôn ngữ lập trình C & C++, nhà xuất bản Giao thông vận tải 1996

[6] SCOOT ROBERT LADD - C++ Components And Algosithms. M&T Books 1994

[7] SCOOT ROBERT LADD - C++ Template And Tools. M&T Books 1995