Phần 1 – PHẦN MỞ ĐẦU

Chương 1 – GIỚI THIỆU

1.1. Về phương pháp phân tích thiết kế hướng đối tượng

Thông thường phần quan trọng nhất của quá trình phân tích thiết kế là chia vấn đề thành nhiều vấn đề nhỏ dễ hiểu và chi tiết hơn. Nếu chúng vẫn còn khó hiểu, chúng lại được chia nhỏ hơn nữa. Trong bất kỳ một tổ chức nào, người quản lý cao nhất cũng không thể quan tâm đến mọi chi tiết cũng như mọi hoạt động. Họ cần tập trung vào mục tiêu và các nhiệm vụ chính, họ chia bớt trách nhiệm cho những người cộng sự dưới quyền của họ. Việc lập trình trong máy tính cũng tương tự. Ngay cả khi dự án đủ nhỏ cho một người thực hiện từ đầu tới cuối, việc chia nhỏ công việc cũng rất quan trọng. Phương pháp phân tích thiết kế hướng đối tượng dựa trên quan điểm này. Cái khó nhất là định ra các lớp sao cho mỗi lớp sau này sẽ cung cấp các đối tượng có các hành vi đúng như chúng ta mong đợi. Việc lập trình giải quyết bài toán lớn của chúng ta sẽ được tập trung vào những giải thuật lớn. Chương trình khi đó được xem như một kịch bản, trong đó các đối tượng sẽ được gọi để thực hiện các hành vi của mình vào những lúc cần thiết. Chúng ta không còn phải lo bị mất phương hướng vì những chi tiết vụn vặt khi cần phải phác thảo một kịch bản đúng đắn, một khi chúng ta đã tin tưởng hoàn toàn vào khả năng hoàn thành nhiệm vụ của các lớp mà chúng ta đã giao phó.

Các lớp do người lập trình định nghĩa đóng vai trò trung tâm trong việc hiện thực giải thuật.

1.2. Giới thiệu môn học Cấu trúc dữ liệu (CTDL) và giải thuật

Theo quan điểm của phân tích thiết kế hướng đối tượng, mỗi lớp sẽ được xây dựng với một số chức năng nào đó và các đối tượng của nó sẽ tham gia vào hoạt động của chương trình. Điểm mạnh của hướng đối tượng là tính đóng kín và tính sử dụng lại của các lớp. Mỗi phần mềm biên dịch cho một ngôn ngữ lập trình nào đó đều chứa rất nhiều thư viện các lớp như vậy. Chúng ta thử điểm qua một số lớp mà người lập trình thường hay sử dụng: các lớp có nhiệm vụ đọc/ ghi để trao đổi dữ liệu với các thiết bị ngoại vi như đĩa, máy in, bàn phím,...; các lớp đồ họa cung cấp các chức năng vẽ, tô màu cơ bản; các lớp điều khiển cho phép xử lý việc giao tiếp với người sử dụng thông qua bàn phím, chuột, màn hình; các lớp phục vụ các giao dịch truyền nhận thông tin qua mạng;...Các lớp CTDL mà chúng ta sắp bàn đến cũng không là một trường hợp ngoại lệ. Có thể chia tất cả các lớp này thành hai nhóm chính:

- Các lớp dịch vụ.
- Các lớp có khả năng lưu trữ và xử lý lượng dữ liệu lớn.

Nhóm thứ hai muốn nói đến các lớp CTDL (CTDL). Vậy có gì giống và khác nhau giữa các lớp CTDL và các lớp khác?

- Điểm giống nhau giữa các lớp CTDL và các lớp khác: mỗi lớp đều phải thực hiện một số chức năng thông qua các hành vi của các đối tượng của nó. Một khi chúng ta đã xây dựng xong một lớp CTDL nào đó, chúng ta hoàn toàn tin tưởng rằng nó sẽ hoàn thành xuất sắc những nhiệm vụ mà chúng ta đã thiết kế và đã giao phó cho nó. Điều này rất khác biệt so với những tài liệu viết về CTDL theo quan điểm hướng thủ tục trước đây: việc xử lý dữ liệu không hề có tính đóng kín và tính sử dụng lại. Tuy về mặt thực thi thì các chương trình như thế có khả năng chạy nhanh hơn, nhưng chúng bộc lộ rất nhiều nhược điểm: thời gian phát triển giải thuật chính rất chậm gây khó khăn nhiều cho người lập trình, chương trình thiếu tính trong sáng, rất khó sửa lỗi và phát triển.
- Đặc trưng riêng của các lớp CTDL: Nhiệm vụ chính của các lớp CTDL là nắm giữ dữ liệu sao cho có thể đáp ứng mỗi khi được chương trình yêu cầu trả về một dữ liệu cụ thể nào đó mà chương trình cần đến. Những thao tác cơ bản đối với một CTDL thường là: thêm dữ liệu mới, xóa bỏ dữ liệu đã có, tìm kiếm, truy xuất.

Ngoài các thao tác dữ liệu cơ bản, các CTDL khác nhau sẽ khác nhau về các thao tác bổ sung khác. Chính vì điều này mà khi thiết kế những giải thuật để giải quyết các bài toán lớn, người ta sẽ lựa chọn CTDL nào là thích hợp nhất.

Chúng ta thử xem xét một ví dụ thật đơn giản sau đây.

Giả sử chúng ta cần viết một chương trình nhận vào một dãy các con số, và in chúng ra theo thứ tự ngược với thứ tự nhập vào ban đầu.

Để giải quyết bài toán này, nếu chúng ta nghĩ đến việc phải khai báo các biến để lưu các giá trị nhập vào như thế nào, và sau đó là thứ tự in ra sao để đáp ứng yêu cầu bài toán, thì dường như là chúng ta đã quên áp dụng nguyên tắc lập trình hướng đối tượng: chúng ta đã phải bận tâm đến những việc quá chi tiết. Đây chỉ là một ví dụ vô cùng đơn giản, nhưng nó có thể minh họa cho vai trò của CTDL. Nếu chúng ta nhớ rằng, việc tổ chức và lưu dữ liệu như thế nào là một việc quá chi tiết và tỉ mỉ không nên thực hiện vào lúc này, thì đó chính là lúc chúng ta đã bước đầu hiểu được vai trò của các lớp CTDL.

Môn CTDL và giải thuật sẽ giúp chúng ta hiểu rõ về các lớp CTDL có sẵn trong các phần mềm. Hơn thế nữa, trong khi học cách xây dựng các lớp CTDL từ đơn giản đến phức tạp, chúng ta sẽ nắm được các phương pháp cũng như các kỹ năng thông qua một số nguyên tắc chung. Từ đó, ngoài khả năng hiểu rõ để có thể lựa chọn một cách đúng đắn nhất những CTDL có sẵn, chúng ta còn có khả năng xây dựng những lớp CTDL phức tạp hơn, tinh tế và thích hợp hơn trong mỗi bài toán mà chúng ta cần giải quyết. Khả năng thừa kế các CTDL có sẵn để phát triển thêm các tính năng mong muốn cũng là một điều đáng lưu ý.

Với ví dụ trên, những ai đã từng tiếp xúc ít nhiều với việc lập trình đều không xa lạ với khái niệm "ngăn xếp". Đây là một CTDL đơn giản nhất nhưng lại rất thông dụng, và dĩ nhiên chúng ta sẽ có dịp học kỹ hơn về nó. Ở đây chúng ta muốn mượn nó để minh họa, và cũng nhằm giúp cho người đọc làm quen với một phương pháp tiếp cận hoàn toàn nhất quán trong suốt giáo trình này.
Giả sử CTDL ngăn xếp của chúng ta đã được giao cho một nhiệm vụ là cất giữ những dữ liệu và trả về khi có yêu cầu, theo một quy định bất di bất dịch là dữ liệu đưa vào sau phải được lấy ra trước. Bằng cách sử dụng CTDL ngăn xếp, chương trình trở nên hết sức đơn giản và được trình bày bằng ngôn ngữ giả như sau:

```
Lặp cho đến khi nhập đủ các con số mong muốn
{
  Nhập 1 con số.
  Cất vào ngăn xếp con số vừa nhập.
}
Lặp trong khi mà ngăn xếp vẫn còn dữ liệu
{
  Lấy từ ngăn xếp ra một con số.
  In số vừa lấy được.
}
```

Chúng ta sẽ có dịp gặp nhiều bài toán phức tạp hơn mà cũng cần sử dụng đến đặc tính này của ngăn xếp. Tính đóng kín của các lớp giúp cho chương trình vô cùng trong sáng. Đoạn chương trình trên không hề cho chúng ta thấy ngăn xếp đã làm việc với các dữ liệu được đưa vào như thế nào, đó là nhiệm vụ mà chúng ta đã giao phó cho nó và chúng ta hoàn toàn yên tâm về điều này. Bằng cách này, khi đã có những CTDL thích hợp, người lập trình có thể dễ dàng giải quyết các bài toán lớn. Họ có thể yên tâm tập trung vào những điểm mấu chốt để xây dựng, tinh chế giải thuật và kiểm lỗi.

Trên đây chúng ta chỉ vừa mới giới thiệu về phần CTDL nằm trong nội dung của môn học "CTDL và giải thuật". Vậy giải thuật là gì? Đứng trên quan điểm thiết kế và lập trình hướng đối tượng, chúng ta đã hiểu vai trò của các lớp. Vậy khi đã có các lớp rồi thì người ta cần xây dựng kịch bản cho các đối tượng hoạt động nhằm giải quyết bài toán chính. Chúng ta cần một cấu trúc chương trình để tạo ra kịch bản đó: việc gì làm trước, việc gì làm sau; việc gì chỉ làm trong những tình huống đặc biệt nào đó; việc gì cần làm lặp lại nhiều lần. Chúng ta nhắc đến giải thuật chính là quay về với khái niệm của "lập trình thủ tục" trước kia. Ngoài ra, chúng ta cũng cần đến giải thuật khi cần hiện thực cho mỗi lớp: xong phần đặc tả các phương thức - phương tiện giao tiếp của lớp với bên ngoài - chúng ta cần đến khái niệm "lập trình thủ tục" để giải quyết phần hiện thực bên trong của

các phương thức này. Đó là việc chúng ta phải xử lý những dữ liệu bên trong của chúng như thế nào mới có thể hoàn thành được chức năng mà phương thức phải đảm nhiệm.

Như vậy, về phần giải thuật trong môn học này, chủ yếu chúng ta sẽ tìm hiểu các giải thuật mà các phương thức của các lớp CTDL dùng đến, một số giải thuật sắp xếp tìm kiếm, và các giải thuật trong các ứng dụng minh họa việc sử dụng các lớp CTDL để giải quyết một số bài toán đó.

Trong giáo trình này, ý tưởng về các giải thuật sẽ được trình bày cặn kẽ, phần chương trình dùng ngôn ngữ C++ hoặc ngôn ngữ giả theo quy ước ở cuối chương này. Phần đánh giá giải thuật chỉ nêu những kết quả đã được chứng minh và kiểm nghiệm, sinh viên có thể tìm hiểu kỹ hơn trong các sách tham khảo.

1.3. Cách tiếp cận trong quá trình tìm hiểu các lớp CTDL

1.3.1. Các bước trong quá trình phân tích thiết kế hướng đối tượng

Quá trình phân tích thiết kế hướng đối tượng khi giải quyết một bài toán gồm các bước như sau:

1. Định ra các lớp với các chức năng mà chúng ta mong đợi. Công việc này cũng giống như công việc phân công công việc cho các nhân viên cùng tham gia một dự án.
2. Giải quyết bài toán bằng cách lựa chọn các giải thuật chính. Đó là việc tạo ra một môi trường để các đối tượng của các lớp nêu trên tương tác lẫn nhau. Giải thuật chính được xem như một kịch bản dẫn dắt các đối tượng thực hiện các hành vi của chúng vào những thời điểm cần thiết.
3. Hiện thực cho mỗi lớp.

Ý tưởng chính ở đây nằm ở bước thứ hai, dẫu cho các lớp chưa được hiện thực, chúng ta hoàn toàn có thể sử dụng chúng sau khi đã biết rõ những chức năng mà mỗi lớp sẽ phải hoàn thành. Trung thành với quan điểm này của hướng đối tượng, chúng ta cũng sẽ nêu ra đây phương pháp tiếp cận mà chúng ta sẽ sử dụng một cách hoàn toàn nhất quán trong việc nghiên cứu và xây dựng các lớp CTDL.

Ứng dụng trong chương 18 về chương trình Game Of Life là một dẫn chứng về các bước phân tích thiết kế trong quá trình xây dựng nên một chương trình. Sinh viên có thể tham khảo ngay phần này. Riêng phần 18.4.2 phiên bản thứ hai của chương trình sinh viên chỉ có thể tham khảo sau khi đọc qua chương 4 về danh sách và chương 12 về bảng băm.

1.3.2. Quá trình xây dựng các lớp CTDL

Chúng ta sẽ lần lượt xây dựng từ các lớp CTDL đơn giản cho đến các lớp CTDL phức tạp hơn. Tuy nhiên, quá trình thiết kế và hiện thực cho mọi lớp CTDL đều tuân theo đúng các bước sau đây:

1. Xuất phát từ một mô hình toán học hay dựa vào một nhu cầu thực tế nào đó, chúng ta định ra các chức năng của lớp CTDL chúng ta cần có. Bước này giống bước thứ nhất ở trên, vì lớp CTDL, cũng như các lớp khác, sẽ cung cấp cho chúng ta các đối tượng để hoạt động trong chương trình chính. Và như vậy, những nhiệm vụ mà chúng ta sẽ giao cho nó sẽ được chỉ ra một cách rõ ràng và chính xác ở bước kế tiếp sau đây.

2. Đặc tả đầy đủ cách thức giao tiếp giữa lớp CTDL đang được thiết kế với môi trường ngoài (các chương trình sẽ sử dụng nó). Phần giao tiếp này được mô tả thông qua định nghĩa các phương thức của lớp. Mỗi phương thức là một hành vi của đối tượng CTDL sau này, phần đặc tả gồm các yếu tố sau:
   - Kiểu của kết quả mà phương thức trả về.
   - Các thông số vào / ra.
   - Các điều kiện ban đầu trước khi phương thức được gọi (*precondition*).
   - Các kết quả mà phương thức làm được (*postcondition*).
   - Các lớp, hàm có sử dụng trong phương thức (*uses*).

   Thông qua phần đặc tả này, các CTDL hoàn toàn có thể được sử dụng trong việc xây dựng nên những giải thuật lớn trong các bài toán lớn. Phần đặc tả này có thể được xem như những cam kết mà không bao giờ được quyền thay đổi. Có như vậy các chương trình có sử dụng CTDL mới không bị thay đổi một khi đã sử dụng chúng.

3. Tìm hiểu các phương án hiện thực cho lớp CTDL. Chúng ta cũng nên nhớ rằng, có rất nhiều cách hiện thực khác nhau cho cùng một đặc tả của một lớp CTDL. Về mặt hiệu quả, có những phương án gần như giống nhau, nhưng cũng có những phương án khác nhau rất xa. Điều này phụ thuộc rất nhiều vào cách tổ chức dữ liệu bên trong bản thân của lớp CTDL, vào các giải thuật liên quan đến việc xử lý dữ liệu của các phương thức.

4. Chọn phương án và hiện thực lớp. Trong bước này bao gồm cả việc kiểm tra để hoàn tất lớp CTDL như là một lớp để bổ sung vào thư viện, người lập trình có thể sử dụng chúng trong nhiều chương trình sau này. Công việc ở bước này kể cũng khá vất vả, vì chúng ta sẽ phải kiểm tra thật kỹ lưỡng, trước khi đưa sản phẩm ra như những đóng gói, mà người khác có thể hoàn toàn yên tâm khi sử dụng chúng.

Để có được những sản phẩm hoàn hảo thực hiện đúng những điều đã cam kết, bước thứ hai trên đây được xem là bước quan trọng nhất. Và để có được một định nghĩa và một đặc tả đầy đủ và chính xác nhất cho một CTDL mới nào đó, bước thứ hai phải được thực hiện hoàn toàn độc lập với hai bước sau nó. Đây là nguyên tắc vô cùng quan trọng mà chúng ta sẽ phải tuân thủ một cách triệt để. Vì trong trường hợp ngược lại, việc xem xét sớm các chi tiết cụ thể sẽ làm cho chúng ta dễ có cái nhìn phiến diện, điều này dễ dẫn đến những đặc tả mang nhiều sơ suất.

1.4. Một số định nghĩa cơ bản

Chúng ta bắt đầu bằng định nghĩa của một kiểu dữ liệu (*type*):

1.4.1. Định nghĩa kiểu dữ liệu

Định nghĩa: Một kiểu dữ liệu là một tập hợp, các phần tử của tập hợp này được gọi là các trị của kiểu dữ liệu.

Chúng ta có thể gọi một kiểu số nguyên là một tập các số nguyên, kiểu số thực là một tập các số thực, hoặc kiểu ký tự là một tập các ký hiệu mà chúng ta mong muốn sử dụng trong các giải thuật của chúng ta.

Lưu ý rằng chúng ta đã có thể chỉ ra sự phân biệt giữa một kiểu dữ liệu trừu tượng và cách hiện thực của nó. Chẳng hạn, kiểu int trong C++ không phải là tập của tất cả các số nguyên, nó chỉ chứa các số nguyên được biểu diễn thực sự bởi một máy tính xác định, số nguyên lớn nhất trong tập phụ thuộc vào số bit người ta dành để biểu diễn nó (thường là một từ gồm 2 bytes tức 16 bits). Tương tự, kiểu float và `double` trong C++ biểu diễn một tập các số thực có dấu chấm động nào đó, và đó chỉ là một tập con của tập tất cả các số thực.

1.4.2. Kiểu nguyên tố và các kiểu có cấu trúc

Các kiểu như `int, float, char` được gọi là các kiểu nguyên tố (*atomic*) vì chúng ta xem các trị của chúng chỉ là một thực thể đơn, chúng ta không có mong muốn chia nhỏ chúng. Tuy nhiên, các ngôn ngữ máy tính thường cung cấp các công cụ cho phép chúng ta xây dựng các kiểu dữ liệu mới gọi là các kiểu có cấu trúc (*structured types*). Chẳng hạn như một struct trong C++ có thể chứa nhiều kiểu nguyên tố khác nhau, trong đó không loại trừ một kiểu có cấu trúc khác làm thành phần. Trị của một kiểu có cấu trúc cho chúng ta biết nó được tạo ra bởi các phần tử nào.

1.4.3. Chuỗi nối tiếp và danh sách

Định nghĩa: Một chuỗi nối tiếp (*sequence*) kích thước 0 là một chuỗi rỗng. Một chuỗi nối tiếp kích thước $n \geq 1$ các phần tử của tập T là một cặp có thứ tự (Sn-1, t), trong đó Sn-1 là một chuỗi nối tiếp kích thước $n - 1$ các phần tử của tập T, và t là một phần tử của tập T.

Từ định nghĩa này chúng ta có thể tạo nên một chuỗi nối tiếp dài tùy ý, bắt đầu từ một chuỗi nối tiếp rỗng và thêm mỗi lần một phần tử của tập T.

Chúng ta phân biệt hai từ: nối tiếp (*sequential*) ngụ ý các phần tử thuộc một chuỗi nối tiếp về mặt luận lý, còn từ liên tục (*contiguous*) ngụ ý các phần tử nằm kề nhau trong bộ nhớ. Trong định nghĩa trên đây chúng ta chỉ dùng từ nối tiếp mà thôi, chúng ta chưa hề quan tâm về mặt vật lý.

Từ định nghĩa chuỗi nối tiếp hữu hạn cho phép chúng ta định nghĩa một danh sách (*list*):

Định nghĩa: Một danh sách các phần tử thuộc kiểu T là một chuỗi nối tiếp hữu hạn các phần tử kiểu T.

1.4.4. Các kiểu dữ liệu trừu tượng

Định nghĩa: CTDL (*Data Structure*) là một sự kết hợp của các kiểu dữ liệu nguyên tố, và/ hoặc các kiểu dữ liệu có cấu trúc, và/ hoặc các CTDL khác vào một tập, cùng các quy tắc về các mối quan hệ giữa chúng.

Trong định nghĩa này, cấu trúc có nghĩa là tập các quy tắc kết nối các dữ liệu với nhau. Mặt khác, đứng trên quan điểm của hướng đối tượng, chúng ta sẽ xây dựng mỗi CTDL như là một lớp mà ngoài khả năng chứa dữ liệu, nó còn có các hành vi đặc trưng riêng, đó chính là các thao tác cho phép cập nhập, truy xuất các giá trị dữ liệu cho từng đối tượng. Nhờ đó, chúng ta có được một khái niệm mới: kiểu dữ liệu trừu tượng (*abstract data type*), thường viết tắt là ADT.

Nguyên tắc quan trọng ở đây là một định nghĩa của bất kỳ một kiểu dữ liệu trừu tượng nào cũng gồm hai phần: phần thứ nhất mô tả cách mà các phần tử trong kiểu liên quan đến nhau, phần thứ hai là sự liệt kê các thao tác có thể thực hiện trên các phần tử của kiểu dữ liệu trừu tượng đó.

Lưu ý rằng khi định nghĩa cho một kiểu dữ liệu trừu tượng chúng ta hoàn toàn không quan tâm đến cách hiện thực của nó. Một định nghĩa cho một kiểu dữ liệu trừu tượng phụ thuộc vào những nhiệm vụ mà chúng ta trông đợi nó phải thực hiện được. Dưới đây là một số vấn đề chúng ta thường hay xem xét:

- Có quan tâm đến thứ tự thêm vào của các phần tử hay không?
- Việc truy xuất phần tử phụ thuộc thứ tự thêm vào của các phần tử, hay có thể truy xuất phần tử bất kỳ dựa vào khóa cho trước?
- Việc tìm kiếm phần tử theo khóa, nếu được phép, là hoàn toàn như nhau đối với bất kỳ khóa nào, hay phụ thuộc vào thứ tự khi thêm vào, hay phụ thuộc vào tần suất mà khóa được truy xuất?
- ...

Một đặc tả cho một kiểu dữ liệu trừu tượng hoàn toàn có thể có nhiều cách hiện thực khác nhau. Mỗi cách hiện thực mang lại tính khả thi và tính hiệu quả khác nhau. Điều này phụ thuộc vào yêu cầu về thời gian và không gian của bài toán. Nhưng cần nhấn mạnh rằng, mọi cách hiện thực của một kiểu dữ liệu trừu tượng đều luôn trung thành với đặc tả ban đầu về các chức năng của nó.

Nhiệm vụ của chúng ta trong việc hiện thực CTDL trong C++ là bắt đầu từ những khái niệm, thường là định nghĩa của một ADT, sau đó tinh chế dần để có được hiện thực bằng một lớp trong C++. Các phương thức của lớp trong C++ tương ứng một cách tự nhiên với các thao tác dữ liệu trên ADT, trong khi những thành phần dữ liệu của lớp trong C++ tương ứng với CTDL vật lý mà chúng ta chọn để biểu diễn ADT.

1.5. Một số nguyên tắc và phương pháp để học tốt môn CTDL và giải thuật

1.5.1. Cách tiếp cận và phương hướng suy nghĩ tích cực

Mỗi CTDL đều được trình bày theo đúng các bước sau đây:

- Định nghĩa lớp.
- Đặc tả lớp.
- Phân tích các phương án hiện thực.
- Hiện thực lớp.
-

Lưu ý rằng, sự trừu tượng và đặc tả dữ liệu phải luôn đi trước sự lựa chọn cách thức tổ chức lưu trữ dữ liệu và cách hiện thực chúng.

Trong phần định nghĩa và đặc tả lớp, để có thể hiểu sâu vấn đề và cảm thấy hứng thú hơn, sinh viên nên tự xem mình là một trong những người tham gia vào công việc thiết kế. Vì chức năng của lớp hoàn toàn phụ thuộc vào quan điểm của người thiết kế. Nếu chúng ta giới hạn cho mỗi lớp CTDL một số chức năng thao tác dữ liệu cơ bản nhất, chúng ta có một thư viện gọn nhẹ. Ngược lại, thư viện sẽ rất lớn, nhưng người lập trình có thể gọi thực hiện bất cứ công việc nào mà họ muốn từ những phương thức đã có sẵn của mỗi lớp. Thư viện các lớp CTDL trong VC++ là một minh họa cho thấy mỗi lớp CTDL có sẵn rất nhiều phương thức đáp ứng được nhu cầu của nhiều người dùng khác nhau.

Các phương thức được đặc tả kỹ càng cho mỗi lớp trong giáo trình này cũng chỉ là để minh họa. Sinh viên có thể tự ý chọn lựa để đặc tả một số phương thức bổ sung khác theo ý muốn.

Trước khi tìm hiểu các phương án hiện thực được trình bày trong giáo trình dành cho mỗi lớp CTDL, sinh viên cũng nên tự phác họa theo suy nghĩ của riêng

bản thân mình. Với cách chủ động như vậy, sinh viên sẽ dễ dàng nhìn ra các ưu nhược điểm trong từng phương án. Và nếu có được những ý tưởng hoàn toàn mới mẻ so với những gì được trình bày trong giáo trình, sinh viên sẽ tự tin hơn khi cần giải quyết các bài toán. Những CTDL nhằm phục vụ cho các bài toán lớn đôi khi được hình thành từ sự ghép nối của một số CTDL đơn giản. Chính sự ghép nối này làm nảy sinh vô vàn phương án khác nhau mà chúng ta phải chọn lựa thật thận trọng, để bảo đảm tính khả thi và hiệu quả của chương trình. Một khi gặp một bài toán cần giải quyết, nếu sinh viên biết chọn cho mình những phương án ghép nối các CTDL đơn giản, biết cách sử dụng lại những gì đã có trong thư viện, và biết cách làm thế nào để hiện thực bổ sung những gì thuộc về những ý tưởng mới mẻ vừa nảy sinh, xem như sinh viên đã học tốt môn CTDL và giải thuật.

So với nhiều giáo trình khác, giáo trình này tách riêng phần ứng dụng các CTDL nhằm làm gọn nhẹ hơn cho phần II là phần chỉ trình bày về các CTDL. Như vậy sẽ thuận tiện hơn cho sinh viên trong việc tìm hiểu những phần căn bản hay là tra cứu mở rộng kiến thức. Hơn nữa, có nhiều ứng dụng liên quan đến nhiều CTDL khác nhau.

1.5.2. Các nguyên tắc

1. Trước khi hiện thực bất kỳ một lớp CTDL nào, chúng ta cần chắc chắn rằng chúng ta đã định nghĩa CTDL và đặc tả các tác vụ cho nó một cách thật đầy đủ. Có như vậy mới bảo đảm được rằng:
   - Chúng ta đã hiểu về nó một cách chính xác.
   - Trong khi hiện thực chúng ta không phải quay lại sửa đổi các đặc tả của nó, vì việc sửa đổi này có thể làm cho chúng ta mất phương hướng, CTDL sẽ không còn đúng như những ý tưởng ban đầu mà chúng ta đã dự định cho nó.
   - Các chương trình ứng dụng không cần phải được chỉnh sửa một khi đã sử dụng CTDL này.
   - Nếu chúng ta cung cấp nhiều hiện thực khác nhau cho cùng một CTDL, thì khi đổi sang sử dụng một hiện thực khác, chương trình ứng dụng không đòi hỏi phải được chỉnh sửa lại. Các hiện thực khác nhau của cùng một CTDL luôn có cùng một giao diện thống nhất.

2. Mỗi phương thức của lớp luôn có năm phần mô tả (kiểu trả về, thông số vào/ra, *precondition*, *postcondition*, *uses*)

   Đây là yêu cầu chung trong việc lập tài liệu cho một hàm. Các CTDL của chúng ta sẽ được sử dụng trong rất nhiều ứng dụng khác nhau. Do đó chúng ta cần xây dựng sao cho người lập trình bớt được mọi công sức có thể. Lời khuyên ở đây là: phần *precondition* chỉ nhằm giải thích ý nghĩa các thông số

vào, chứ không nên ràng buộc những trị hợp lệ mà thông số vào phải thoả. Nhiệm vụ trong phần hiện thực của phương thức là chúng ta phải lường hết mọi khả năng có thể có của thông số vào và giải quyết thỏa đáng từng trường hợp.

> *Chúng ta xem các CTDL cũng như các dịch vụ, chúng được viết một lần và được sử dụng trong rất nhiều ứng dụng khác nhau. Do đó CTDL cần được xây dựng sao cho người sử dụng bớt được công sức mọi lúc có thể.*
>
> *Các phương thức public của các CTDL nên được hiện thực không có precondition.*

3. Đảm bảo tính đóng kín (*encapsulation*) của lớp CTDL. Dữ liệu có tính đóng kín khi chúng chỉ có thể được truy xuất bởi các phương thức của lớp.

Ưu điểm của việc sử dụng một lớp có tính đóng kín là khi chúng ta đặc tả và hiện thực các phương thức, chúng ta không phải lo lắng đến các giá trị không hợp lệ của các dữ liệu đang được lưu trong đối tượng của lớp.

> *Các thành phần dữ liệu của CTDL nên được khai báo private.*

4. Ngoại trừ các *constructor* có chủ đích, mỗi đối tượng của CTDL luôn phải được khởi tạo là một đối tượng rỗng và chỉ được sửa đổi bởi chính các phương thức của lớp. Với các phương thức đã được hiện thực và kiểm tra kỹ lưỡng, chúng ta luôn an tâm rằng các đối tượng CTDL luôn chứa những dữ liệu hợp lệ. Điều này giúp chúng luôn hoàn thành nhiệm vụ được giao, và đó cũng là nguyên tắc của hướng đối tượng.

1.5.3. Phong cách lập trình (*style of programming*) và các kỹ năng:

1. Vấn đề xử lý lỗi:

Việc xử lý lỗi cung cấp một mức độ an toàn cần thiết mà chúng ta nên thực hiện mọi lúc có thể trong CTDL của chúng ta. Có vài cách khác nhau trong việc xử lý lỗi. Chẳng hạn chúng ta có thể in ra thông báo hoặc ngưng chương trình khi gặp lỗi. Hoặc thay vào đó, chúng ta dành việc xử lý lỗi lại cho người lập trình sử dụng CTDL của chúng ta bằng cách trả về các mã lỗi thông qua trị trả về của các phương thức. Cách cuối cùng này hay hơn vì nó cung cấp khả năng lựa chọn cho người lập trình. Có những tình huống mà người lập trình thấy cần thiết phải ngưng ngay chương trình, nhưng cũng có những tình huống lỗi có thể bỏ qua để chương trình tiếp tục chạy. Bằng cách này, người lập trình khi sử dụng các CTDL hoàn toàn có thể chủ động đối

phó với mọi tình huống. Hơn nữa, các CTDL của chúng ta sẽ được xây dựng như là các thư viện dùng chung cho rất nhiều chương trình.

> *Khi sử dụng một phương thức của một lớp CTDL, người lập trình cần phải xem xét lại mã lỗi mà phương thức trả về để xử lý lỗi khi cần thiết.*
>
> *Các lớp CTDL cần phải được thiết kế sao cho có thể cho phép người lập trình chọn lựa cách thức xử lý lỗi theo ý muốn.*

Chúng ta sẽ dùng khai báo `ErrorCode` như một kiểu dữ liệu kiểu liệt kê tập các trị tương ứng các tình huống có thể xảy ra khi một phương thức của một lớp được gọi: thành công hay thất bại, tràn bộ nhớ, trị thông số không hợp lệ,... Chúng ta sẽ cố gắng thiết kế một cách thật nhất quán: hầu hết các phương thức của các lớp trả về kiểu `ErrorCode` này.

> *Sự nhất quán bao giờ cũng tạo ra thói quen rất tốt trong phong cách lập trình. Điều này tiết kiệm rất nhiều công sức và thời gian của người lập trình.*

2. Cách truyền nhận dữ liệu giữa đối tượng CTDL với chương trình sử dụng

Các giao tiếp truyền nhận dữ liệu khác giữa chương trình sử dụng và các lớp CTDL dĩ nhiên cũng thông qua danh sách các thông số. Trong phương thức của lớp CTDL sẽ không có việc chờ nhận dữ liệu trực tiếp từ bàn phím. Chúng ta nên dành cho người lập trình quyền chuyển hướng dòng nhập xuất dữ liệu với các thiết bị bên ngoài như bàn phím, màn hình, tập tin, máy in,...

3. Vấn đề kiểu của dữ liệu được lưu trong CTDL.

Mỗi lớp CTDL mà chúng ta xây dựng đều có tính tổng quát cao, nó có thể chấp nhận bất kỳ một kiểu dữ liệu nào cho dữ liệu được lưu trong nó. Trong C++ từ khóa `template` cho phép chúng ta làm điều này. Các kiểu dữ liệu này thường được yêu cầu phải có sẵn một số thao tác cần thiết như so sánh, nhập, xuất,...

4. Các khai báo bên trong một lớp CTDL.

Lớp CTDL cung cấp các thao tác dữ liệu thông qua các phương thức được khai báo là `public`.
Tất cả những thuộc tính và các hàm còn lại luôn được khai báo `private` hoặc `protected`.
Các thuộc tính của một lớp CTDL có thể được phân làm hai loại:

- Thuộc tính bắt buộc phải có để lưu dữ liệu.

- Thuộc tính mà đối tượng cần có để tự quản lý, trong số này có thuộc tính được bổ sung chỉ để đẩy nhanh tốc độ của các thao tác dữ liệu.

Các hàm được che dấu bên trong lớp được gọi là các hàm phụ trợ (auxilary function), chúng chỉ được sử dụng bởi chính các phương thức của lớp CTDL đó mà thôi.

Việc mở rộng thêm các tác vụ cho một lớp có sẵn có thể theo một trong hai cách:

- Bổ sung thêm phương thức mới.
- Xây dựng lớp thừa kế.

5. Một số hướng dẫn cần thiết trong việc thử nghiệm chương trình.

✓ Bộ chương trình thử (*driver*): Đây là đoạn chương trình thường được viết trong hàm `main` và chứa một thực đơn (*menu*) cho phép thử mọi phương thức của lớp CTDL đang được xây dựng.
Chúng ta sẽ viết, thử nghiệm, và hoàn chỉnh nhiều lớp CTDL khác nhau. Do đó ngay từ đầu chúng ta nên xây dựng một driver sao cho tổng quát, khi cần thử với một CTDL nào đó chỉ cần chỉnh sửa lại đôi chút mà thôi.
Trong driver chúng ta nên chuẩn hóa việc đọc ghi tập tin, xử lý các thao tác đọc từ bàn phím và xuất ra màn hình. Phần còn lại là một menu cho phép người sử dụng chạy chương trình chọn các chức năng như tạo đối tượng CTDL mới, gọi các thao tác thêm, xóa, tìm kiếm, truy xuất,... trên CTDL đó.

✓ Các mẩu tạm (*stub*): đây là một mẹo nhỏ nhưng rất hữu ích. Để dịch và chạy thử một vài phần nhỏ đã viết, những phần chưa viết của chương trình sẽ được tạo như những mẩu nhỏ và chỉ cần hợp cú pháp (Xem ứng dụng tính toán các đa thức trong chương 15).
Ví dụ: Trong đoạn chương trình nào đó chúng ta đang muốn chạy thử mà có sử dụng lớp A, hàm B,..., chúng ta sẽ tạm khai báo các *stub*:

```
class A
{
}; // Một lớp chưa có thuộc tính vì chúng ta chưa quyết định nên chọn
      kiểu      thuộc tính như thế nào.
void B()
{
} // Một hàm với thân hàm còn rỗng mà chúng ta hẹn sẽ viết sau.
```

Nếu một hàm đã có định nghĩa thì chỉ cần trả về sao cho hợp lệ:

```
int C()
{
    return 1;
} // chỉ cần cẩn thận trong trường hợp nếu như giá trị trả về lại được
  dùng trong một biểu thức luận lý để xét điều kiện lặp vòng thì có
  khả năng vòng lặp không được thực hiện hoặc lặp vô tận.
```

- ✓ Cách thức theo dõi một chương trình đang chạy hoặc nhu cầu khảo sát cách làm việc của một trình biên dịch nào đó:

  Ví dụ gợi ý:

```
void D()
{
    count << "\n Hàm D đang được gọi \n";
}
```

  Trong C++ các hàm *constructor* và *destructor* được trình biên dịch gọi khi một đối tượng vừa được tạo ra hoặc sắp bị hủy. Vậy nếu có thắc mắc về thứ tự gọi các hàm này của một lớp thừa kế từ lớp khác, chúng ta có thể dùng cách tương tự để viết trong *constructor* và *destructor* của từng lớp cha, con.

  Nếu chúng ta có thắc mắc về cách ứng xử của trình biên dịch khi gọi các hàm này hay các hàm được định nghĩa đè (*overloaded*, *overwriten*) trong trường hợp các lớp thừa kế lẫn nhau, hoặc một số trường hợp khác nào đó, thì đây là cách hay nhất để chúng ta tự kiểm nghiệm lấy.

  Phần lớn các giải thuật được nghiên cứu trước hết chỉ dựa trên ý tưởng (biểu diễn bằng ngôn ngữ giả và độc lập với mọi ngôn ngữ lập trình). Tuy nhiên khi hiện thực chúng ta thường gặp vướng mắc ở chỗ mỗi ngôn ngữ lập trình có một số đặc điểm khác nhau, và ngay cả khi dùng chung một ngôn ngữ, các trình biên dịch khác nhau (khác hãng sản xuất hay khác phiên bản) đôi khi cũng ứng xử khác nhau. Điều đó gây rất nhiều khó khăn và lãng phí thời gian của nhiều sinh viên.

  Chỉ cần lấy một ví dụ đơn giản, đó là việc đọc ghi file, việc thường xuyên phải cần đến khi muốn thử nghiệm một giải thuật nào đó. Các vòng lặp thường nhầm lẫn ở điều kiện kết thúc file trong ngôn ngữ C++, mà điều này hoàn toàn phụ thuộc vào việc xử lý con trỏ file của trình biên dịch. Ngay một phần mềm như Visual C++ hiện tại cũng chứa cùng lúc trong thư viện không biết bao nhiêu lớp phục vụ cho việc khai báo và đọc ghi file. Chúng ta chỉ có thể sử dụng một trong các thư viện đó một cách chính xác sau khi đã tìm hiểu thật kỹ! Một ví dụ khác cũng hay gây những lỗi mất rất nhiều thời gian, đó là việc so sánh các trị: NULL, '0', '\0', 0, … mà nếu không khảo sát kỹ chúng ta sẽ bị trả giá bởi sự chủ quan cho rằng mình đã hiểu đúng quy ước của trình biên dịch.

Việc tìm đọc tài liệu kèm theo trình biên dịch là một việc làm cần thiết, nó cho chúng ta sự hiểu biết đầy đủ và chính xác. Nhưng để rút ngắn thời gian thì gợi ý trên đây cũng là một lời khuyên quý báu. Không gì nhanh và chính xác bằng cách tìm câu trả lời trong thử nghiệm. Việc sửa đổi chương trình như thế nào để có được các *stub* thỏa những nhu cầu cần thử nghiệm là tùy thuộc vào sự tích cực, say mê và sáng tạo của sinh viên.

- ✓ Gỡ rối chương trình (*debug*)
  Đây là khả năng theo vết chương trình ở những đoạn mà người lập trình còn nghi ngờ có lỗi. Bất cứ người lập trình nào cũng có lúc cần phải chạy debug. Vì vậy tốt hơn hết là ngay từ đầu sinh viên nên tìm hiểu kỹ các khả năng của công cụ debug của trình biên dịch mà mình sử dụng (cho phép theo dõi trị các biến, lịch sử các lần gọi hàm,...).

  Một gợi ý trong phần này là sinh viên cần biết cách cô lập từng phần của chương trình đã viết bằng cách dùng ký hiệu dành cho phần chú thích (*comment*) để khóa bớt những phần chưa đến lượt kiểm tra. Hoặc khi lỗi do trình biên dịch báo có vẻ mơ hồ, thì cách cô lập bằng cách giới hạn dần đoạn chương trình đang dịch thử sẽ giúp chúng ta sớm xác định được phạm vi có lỗi. Cũng có thể làm ngược lại, chỉ dịch thử và chỉnh sửa từng đoạn chương trình nhỏ, cho đến khi hết lỗi mới nới dần phạm vi chương trình để dịch tiếp.

1.6. Giới thiệu về ngôn ngữ giả:

Phần lớn chương trình được trình bày trong giáo trình này đều sử dụng ngôn ngữ C++, sau khi ý tưởng về giải thuật đã được giải thích cặn kẽ. Phần còn lại có một số giải thuật được trình bày bằng ngôn ngữ giả.

Ngôn ngữ giả, hay còn gọi là mã giả (*pseudocode*), là một cách biểu diễn độc lập với mọi ngôn ngữ lập trình, nó không ràng buộc sinh viên vào những cú pháp nghiêm ngặt cũng như cách gọi sao cho chính xác các từ khóa, các hàm có trong thư viện một trình biên dịch nào đó. Nhờ đó sinh viên có thể tập trung vào ý tưởng lớn của giải thuật.

Các quy định về mã giả được sử dụng trong giáo trình này:

- ➢ Biểu diễn sự tuần tự của các lệnh chương trình: các lệnh được thực thi tuần tự lệnh này sang lệnh khác sẽ có cùng khoảng cách canh lề như nhau và được đánh số thứ tự tăng dần, luôn bắt đầu từ 1.

- Cấu trúc khối lồng nhau: một khối nằm trong một khối khác sẽ có khoảng cách canh lề lớn hơn.

  Trong giáo trình này, chỉ những phần được trình bày bằng mã giả mới có số thứ tự ở đầu mỗi dòng lệnh.

  Ví dụ:

```
1.
   1.
   2.
     1. // Đây là dòng lệnh có số thứ tự là 1.2.1
     2.
   3.
2.
   1.
3.
   1.
   2.
```

- Sự rẽ nhánh: chúng ta sử dụng các từ khóa:
  - `if` <biểu thức luận lý>

    …

    `endif`
  - `if` <biểu thức luận lý>

    …

    `else`

    …

    `endif`
  - `case`

    `case1:` …

    `case2:` …

    `case3:` …

    `else:` …

    `endcase`

- Sự lặp vòng:
  - `loop` <biểu thức luận lý>

    …

    `endloop` // lặp trong khi biểu thức luận lý còn đúng.
  - `repeat`

    …

    `until` <biểu thức luận lý> // lặp cho đến khi biểu thức luận lý đúng.

- Khai báo hàm:
  <kiểu trả về> tên hàm (danh sách thông số)
  trong đó danh sách thông số: `val/ ref` <tên kiểu> tên thông số, val/ ref <tên kiểu> tên thông số,...
  `val`: dành cho tham trị; `ref`: dành cho tham biến.

- Khai báo cấu trúc, lớp:
  `struct` tên kiểu dữ liệu cấu trúc
  `end struct`

  class tên kiểu dữ liệu cấu trúc
  `end class`

- Khai báo phương thức của lớp:
  <kiểu trả về> tên lớp::tên hàm (danh sách thông số);

- Khai báo biến:
  <tên kiểu> tên biến

<u>Một chút lưu ý về cách trình bày trong giáo trình</u>:

Do các đoạn chương trình sử dụng *font* chữ `Courier New`, nên các tên biến, tên lớp, tên đối tượng, tên các hàm khi được nhắc đến cũng dùng *font* chữ này. Các từ tiếng Anh khác được in nghiêng. Đặc biệt những phần có liên quan chặt chẽ đến những đặc thù của ngôn ngữ lập trình C++ thường dùng kích cỡ chữ nhỏ hơn, để phân biệt với những phần quan trọng khác khi nói về ý tưởng và giải thuật, và đó mới là mục đích chính của môn học này.
Có một số từ hay đoạn được in đậm hay gạch dưới nhằm giúp sinh viên đọc dễ dàng hơn.