CHƯƠNG 1

SỰ TRỪU TƯỢNG HOÁ DỮ LIỆU

Khi thiết kế thuật giải cho một vấn đề, chúng ta cần sử dụng sự trừu tượng hoá dữ liệu. Sự trừu tượng hoá dữ liệu được hiểu là chúng ta chỉ quan tâm tới một tập các đối tượng dữ liệu (ở mức độ trừu tượng) và các phép toán (các hành động) có thể thực hiện được trên các đối tượng dữ liệu đó. Với mỗi phép toán chúng ta cũng chỉ quan tâm tới điều kiện có thể sử dụng nó và hiệu quả mà nó mang lại, không cần biết nó được thực hiện như thế nào. Sự trừu tượng hoá dữ liệu được thực hiện bằng cách tạo ra các kiểu dữ liệu trừu tượng. Trong chương này chúng ta sẽ trình bày khái niệm kiểu dữ liệu trừu tượng, các phương pháp đặc tả và cài đặt kiểu dữ liệu trừu tượng.

1.1 BIỂU DIỄN DỮ LIỆU TRONG CÁC NGÔN NGỮ LẬP TRÌNH

Trong khoa học máy tính, dữ liệu được hiểu là bất kỳ thông tin nào được xử lý bởi máy tính. Dữ liệu có thể là số nguyên, số thực, ký tự, … Dữ liệu có thể có cấu trúc phức tạp, gồm nhiều thành phần dữ liệu được liên kết với nhau theo một cách nào đó. Trong bộ nhớ của máy tính, mọi dữ liệu đều được biểu diễn dưới dạng nhị phân (một dãy các ký hiệu 0 và 1). Đó là dạng biểu diễn cụ thể nhất của dữ liệu (dạng biểu diễn vật lý của dữ liệu).

Trong các ngôn ngữ lập trình bậc cao (Pascal, C, C+ +…), dữ liệu được biểu diễn dưới dạng trừu tượng, tức là dạng biểu diễn của dữ liệu xuất phát từ dạng biểu diễn toán học của dữ liệu (sử dụng các khái niệm toán học, các mô hình toán học để biểu diễn dữ liệu). Chẳng hạn, nếu dữ liệu là các điểm trong mặt phẳng, thì chúng ta có thể biểu diễn nó như một cặp số thực (x, y), trong đó số thực x là hoành độ, còn số thực y là tung độ của điểm. Do đó, trong ngôn ngữ C + +, một điểm được biểu diễn bởi cấu trúc:

```
struct   point
{   double     x;
    double     y;
 };
```

Trong các ngôn ngữ lập trình bậc cao, các dữ liệu được phân thành các lớp dữ liệu (kiểu dữ liệu). Kiểu dữ liệu của một biến được xác định bởi một tập các giá trị mà biến đó có thể nhận và các phép toán có thể thực hiện trên các giá trị đó. Ví dụ, có lẽ kiểu dữ liệu đơn giản nhất và có trong nhiều ngôn ngữ lập trình là kiểu boolean, miền giá trị của kiểu này chỉ gồm hai giá trị false và true, các phép toán có thể thực hiện trên các giá trị này là các phép toán logic mà chúng ta đã quen biết.

Mỗi ngôn ngữ lập trình cung cấp cho chúng ta một số **kiểu dữ liệu cơ bản (basic data types).** Trong các ngôn ngữ lập trình khác nhau, các kiểu dữ liệu cơ bản có thể khác nhau. Ngôn ngữ lập trình Lisp chỉ có một kiểu cơ bản, đó là các S-biểu thức. Song trong nhiều ngôn ngữ lập trình khác (chẳng hạn Pascal, C / C + +, Ada, …), các kiểu dữ liệu cơ bản rất phong phú. Ví dụ, ngôn ngữ C + + có các kiểu dữ liệu cơ bản sau:

Các kiểu ký tự (char, signed char, unsigned char)

Các kiểu nguyên (int, short int, long int, unsigned)

Các kiểu thực (float, double, long double)

Các kiểu liệt kê (enum)

Kiểu boolean (bool)

Gọi là các kiểu dữ liệu cơ bản, vì các dữ liệu của các kiểu này sẽ được sử dụng như các thành phần cơ sở để kiến tạo nên các dữ liệu có cấu trúc phức tạp. Các kiểu dữ liệu đã cài đặt sẵn (build-in types) mà ngôn ngữ lập trình cung cấp là không đủ cho người sử dụng. Trong nhiều áp dụng, người lập trình cần phải tiến hành các thao tác trên các dữ liệu phức hợp. Vì vậy, mỗi ngôn ngữ lập trình cung cấp cho người sử dụng một số quy tắc cú pháp để tạo ra các kiểu dữ liệu mới từ các kiểu cơ bản hoặc các kiểu khác đã được xây dựng. Chẳng hạn, C + + cung cấp cho người lập trình các luật để xác

định các kiểu mới: kiểu mảng (array), kiểu cấu trúc (struct), kiểu con trỏ, …

Ví dụ. Từ các kiểu đã có $T_1, T_2, ..., T_n$ (có thể khác nhau), khai báo sau

```
struct   S {
   T1    M1 ;
   T2    M2 ;
   ……
   Tn    Mn ;
}
```

xác định một kiểu cấu trúc với tên là S, mỗi dữ liệu của kiểu này gồm n thành phần, thành phần thứ i có tên là M_i và có giá trị thuộc kiểu T_i (i = 1,…, n).

Các kiểu dữ liệu được tạo thành từ nhiều kiểu dữ liệu khác (các kiểu này có thể là kiểu cơ bản hoặc kiểu dữ liệu đã được xây dựng) được gọi là **kiểu dữ liệu có cấu trúc**. Các dữ liệu thuộc kiểu dữ liệu có cấu trúc được gọi là các **cấu trúc dữ liệu** (data structure). Ví dụ, các mảng, các cấu trúc, các danh sách liên kết, … là các cấu trúc dữ liệu (CTDL).

Từ các kiểu cơ bản, bằng cách sử dụng các qui tắc cú pháp kiến tạo các kiểu dữ liệu, người lập trình có thể xây dựng nên các kiểu dữ liệu mới thích hợp cho từng vấn đề. Các kiểu dữ liệu mà người lập trình xây dựng nên được gọi là các kiểu dữ liệu được xác định bởi người sử dụng (user- defined data types).

Như vậy, một CTDL là một dữ liệu phức hợp, gồm nhiều thành phần dữ liệu, mỗi thành phần hoặc là dữ liệu cơ sở (số nguyên, số thực, ký tự,…) hoặc là một CTDL đã được xây dựng. Các thành phần dữ liệu tạo nên một CTDL được liên kết với nhau theo một cách nào đó. Trong các ngôn ngữ lập trình thông dụng (Pascal, C/ C+ +), có ba phương pháp để liên kết các dữ liệu:

1. Liên kết các dữ liệu cùng kiểu tạo thành mảng dữ liệu.
2. Liên kết các dữ liệu (không nhất thiết cùng kiểu) tạo thành cấu trúc trong C/ C+ +, hoặc bản ghi trong Pascal.

3. Sử dụng con trỏ để liên kết dữ liệu. Chẳng hạn, sử dụng con trỏ chúng ta có thể tạo nên các danh sách liên kết, hoăc các CTDL để biểu diễn cây. (Chúng ta sẽ nghiên cứu các CTDL này trong các chương sau)

Ví dụ. Giả sử chúng ta cần xác định CTDL biểu diễn các lớp học. Giả sử mỗi lớp học cần được mô tả bởi các thông tin sau: tên lớp, số tổ của lớp, danh sách sinh viên của mỗi tổ; mỗi sinh viên được mô tả bởi 3 thuộc tính: tên sinh viên, tuổi và giới tính. Việc xây dựng một CTDL cho một đối tượng dữ liệu được tiến hành theo nguyên tắc sau: từ các dữ liệu có kiểu cơ sở tạo ra kiểu dữ liệu mới, rồi từ các kiểu dữ liệu đã xây dựng tạo ra kiểu dữ liệu phức tạp hơn, cho tới khi nhận được kiểu dữ liệu cho đối tượng dữ liệu mong muốn. Trong ví dụ trên, đầu tiên ta xác định cấu trúc Student

```
struct   Student
{
    string     StName;
    int        Age;
    bool       Sex;
}
```

Danh sách sinh viên của mỗi tổ có thể lưu trong mảng, hoặc biểu diễn bởi danh sách liên kết. Ở đây chúng ta dùng danh sách liên kết, mỗi tế bào của nó là cấu trúc sau:

```
struct     Cell
{
      Student       Infor;
      Cell*         Next;
}
```

Chúng ta sử dụng một mảng để biểu diễn các tổ, mỗi thành phần của mảng lưu con trỏ trỏ tới đầu một danh sách liên kết biểu diễn danh sách các sinh viên của một tổ. Giả sử mỗi lớp có nhiều nhất 10 tổ, kiểu mảng GroupArray được xác định như sau:

```
typedef     Cell*     GroupArray[10];
```

Cuối cùng, ta có thể biểu diễn lớp học bởi cấu trúc sau:

```
struct      StudentClass
{
    string      ClassName;
    int         GroupNumber;
    GroupArray  Group;
 }
```

1.2 SỰ TRỪU TƯỢNG HOÁ DỮ LIỆU

Thiết kế và phát triển một chương trình để giải quyết một vấn đề là một quá trình phức tạp. Thông thường quá trình này cần phải qua các giai đoạn chính sau:

1. Đặc tả vấn đề.
2. Thiết kế thuật toán và cấu trúc dữ liệu.
3. Cài đặt (chuyển dịch thuật toán thành các câu lệnh trong một ngôn ngữ lập trình, chẳng hạn C+ +)
4. Thử nghiệm và sửa lỗi.

Liên quan tới nội dung của sách này, chúng ta chỉ đề cập tới hai giai đoạn đầu. Chúng ta muốn làm sang tỏ vai trò quan trọng của sự trừu tượng hoá (abstraction) trong đặc tả một vấn đề, đặc biệt là sự trừu tương hoá dữ liệu (data abstraction) trong thiết kế thuật toán.

Vấn đề được đặt ra bởi người sử dụng thường được phát biểu không rõ ràng, thiếu chính xác. Do đó, điều đầu tiên chúng ta phải làm là chính xác hoá vấn đề cần giải quyết, hay nói một cách khác là mô tả chính xác vấn đề. Điều đó được gọi là đặc tả vấn đề. Trong giai đoạn này, chúng ta phải trả lời chính xác các câu hỏi sau. Chúng ta được cho trước những gì? Chúng ta cần tìm những gì? Những cái đã biết và những cái cần tìm có quan hệ với nhau như thế nào? Như vậy, trong giai đoạn đặc tả, chúng ta cần mô tả chính xác các dữ liệu vào (inputs) và các dữ liệu ra (outputs) của chương trình. Toán học là một ngành khoa học trừu tượng, chính xác, các khái niệm toán học, các mô hình toán học là sự trừu tượng hoá từ thế giới hiện thực. Sử dụng trừu tượng hoá trong đặc tả một vấn đề đồng nghĩa với việc chúng ta sử dụng các khái niệm toán học, các mô hình toán học và logic để biểu diễn chính xác một vấn đề.

Ví dụ. Giả sử chúng ta cần viết chương trình lập lịch thi. Vấn đề như sau. Mỗi người dự thi đăng kí thi một số môn trong số các môn tổ chức thi. Chúng ta cần xếp lịch thi, mỗi ngày thi một số môn trong cùng một thời gian, sao cho mỗi người dự thi có thể thi tất cả các môn họ đã đăng kí.

Chúng ta có thể đặc tả inputs và outputs của chương trình như sau:

Inputs: danh sách các người dự thi, mỗi người dự thi được biểu diễn bởi danh sách các môn mà anh ta đăng kí.

Outputs: danh sách các ngày thi, mỗi ngày thi được biểu diễn bởi danh sách các môn thi trong ngày đó sao cho hai môn thi bất kì trong danh sách này không thuộc cùng một danh sách các môn đăng kí của một người dự thi.

Trong mô tả trên, chúng ta đã sử dụng khái niệm danh sách (khái niệm dãy trong toán học). Các khái niệm toán học, các mô hình toán học hoặc logic được sử dụng để mô tả các đối tượng dữ liệu tạo thành các **mô hình dữ liệu (data models**). Danh sách là một mô hình dữ liệu. Chú ý rằng, lịch thi cần thoả mãn đòi hỏi: người dự thi có thể thi tất cả các môn mà họ đăng kí. Để dễ dàng đưa ra thuật toán lập lịch, chúng ta sử dụng một mô hình dữ liệu khác: đồ thị. Mỗi môn tổ chức thi là một đỉnh của đồ thị. Hai đỉnh có cạnh nối, nếu có một người dự thi đăng kí thi cả hai môn ứng với hai đỉnh đó. Từ

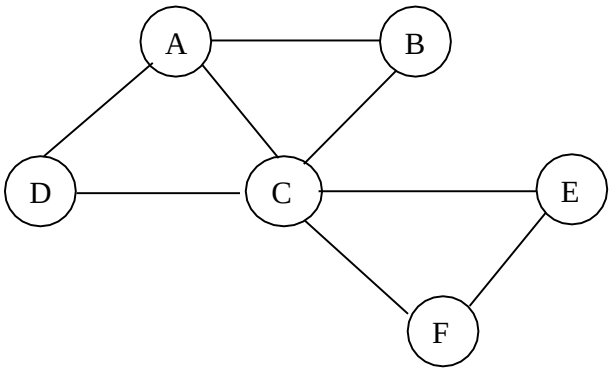
mô hình dữ liệu đồ thị này, chúng ta có thể đưa ra thuật toán lập lịch như sau:

Bước 1: Chọn một đỉnh bất kì, đưa môn thi ứng với đỉnh này vào danh sách các môn thi trong một ngày thi (danh sách này ban đầu rỗng). Đánh dấu đỉnh đã chọn và tất cả các đỉnh kề nó. Trong các đỉnh chưa đánh dấu, lại chọn một đỉnh bất kì và đưa môn thi ứng với đỉnh này vào danh sách các môn thi trong ngày thi. Lại đánh dấu đỉnh vừa chọn và các đỉnh kề nó. Tiếp tục quá trình trên cho tới khi tất cả các đỉnh của đồ thị được đánh dấu, chúng ta nhận được danh sách các môn thi trong một ngày thi.

Bước 2: Loại khỏi đồ thị tất cả các đỉnh đã xếp vào danh sách các môn thi trong một ngày thi ở bước 1 và loại tất cả các cạnh kề các đỉnh đó. Các đỉnh và các cạnh còn lại tạo thành đồ thị mới.

Bước 3: Lặp lại bước 1 và bước 2 cho tới khi đồ thị trở thành rỗng.

Chẳng hạn, giả sử các môn tổ chức thi là A, B, C, D, E, F và đồ thị xây dựng nên từ các dữ liệu vào được cho trong hình sau:



Khi đó lịch thi có thể như sau:

Ngày thi 1: A, F.

Ngày thi 2: D, E, B.

Ngày thi 3: C.

Sau khi đặc tả vấn đề, chúng ta chuyển sang giai đoạn thiết kế thuật toán để giải quyết vấn đề. Ở mức độ cao nhất của sự trừu tượng hoá, thuật toán được thiết kế như là **một dãy các hành động trên các đối tượng dữ**

liệu được thực hiện theo một trình tự logic nào đó. Thuật toán lập lịch thi ở trên là một ví dụ. Các đối tượng dữ liệu có thể là số nguyên, số thực, ký tự; có thể là các điểm trên mặt phẳng; có thể là các hình hình học; có thể là con người, có thể là danh sách các đối tượng (chẳng hạn, danh sách các môn thi trong ví dụ lập lịch thi); có thể là đồ thị, cây, …

Các hành động trên các đối tượng dữ liệu cũng rất đa dạng và tuỳ thuộc vào từng loại đối tượng dữ liệu. Chẳng hạn, nếu đối tượng dữ liệu là điểm trên mặt phẳng, thì các hành động có thể là: quay điểm đi một góc nào đó, tịnh tiến điểm theo một hướng, tính khoảng cách giữa hai điểm, … Khi đối tượng dữ liệu là danh sách, thì các hành động có thể là: loại một đối tượng khỏi danh sách, xen một đối tượng mới vào danh sách, tìm xem một đối tượng đã cho có trong danh sách hay không, …

Khi thiết kế thuật toán như là một dãy các hành động trên các đối tượng dữ liệu, chúng ta cần sử dụng **sự trừu tượng hoá dữ liệu (data abstraction).**

Sự trừu tượng hoá dữ liệu có nghĩa là chúng ta chỉ quan tâm tới một tập các đối tượng dữ liệu (ở mức độ trừu tượng) và các hành động (các phép toán) có thể thực hiện trên các đối tượng dữ liệu đó (với các điều kiện nào thì hành động có thể được thực hiện và sau khi thực hiện hành động cho kết quả gì), chúng ta không quan tâm tới các đối tượng dữ liệu đó được lưu trữ như thế nào trong bộ nhớ của máy tính, chúng ta không quan tâm tới các hành động được thực hiện như thế nào.

Sử dụng sự trừu tượng hoá dữ liệu trong thiết kế thuật toán là phương pháp luận thiết kế rất quan trọng. Nó có các ưu điểm sau:

- Đơn giản hoá quá trình thiết kế, giúp ta tránh được sự phức tạp liên quan tới biểu diễn cụ thể của dữ liệu .
- Chương trình sẽ có tính mođun (modularity). Chẳng hạn, một hành động trên đối tượng dữ liệu phức tạp được cài đặt thành một mođun (một hàm). Chương trình có tính mođun sẽ dễ đọc, dễ phát hiện lỗi, dễ sửa, …

Sự trừu tượng hoá dữ liệu được thực hiện bằng cách xác định các **kiểu dữ liệu trừu tượng (Abstract Data Type).** Kiểu dữ liệu trừu tượng (KDLTT) là một tập các đối tượng dữ liệu cùng với các phép toán có thể thực hiện trên các đối tượng dữ liệu đó. Ví dụ, tập các điểm trên mặt phẳng với các phép toán trên các điểm mà chúng ta đã xác định tạo thành KDLTT điểm.

Chúng ta có thể sử dụng các phép toán của các KDLTT trong thiết kế thuật toán khi chúng ta biết rõ các điều kiện để phép toán có thể thực hiện và hiệu quả mà phép toán mang lại. Trong nhiều trường hợp, các KDLTT mà chúng ta đã biết sẽ gợi cho ta ý tưởng thiết kế thuật toán . Đồng thời trong quá trình thiết kế, khi thuật toán cần đến các hành động trên các loại đối tượng dữ liệu mới chúng ta có thể thiết kế KDLTT mới để sử dụng không chỉ trong chương trình mà ta đang thiết kế mà còn trong các chương trình khác.

Phần lớn nội dung trong sách này là nói về các KDLTT. Chúng ta sẽ nghiên cứu sự thiết kế và cài đặt một số KDLTT quan trọng nhất được sử dụng thường xuyên trong thiết kế thuật toán .

1.3 KIỂU DỮ LIỆU TRỪU TƯỢNG

Mục này trình bày phương pháp đặc tả và cài đặt một KDLTT.

1.3.1 Đặc tả kiểu dữ liệu trừu tượng

Nhớ lại rằng, một KDLTT được định nghĩa là một tập các đối tượng dữ liệu và một tập các phép toán trên các đối tượng dữ liệu đó. Do đó, đặc tả một KDLTT gồm hai phần: đặc tả đối tượng dữ liệu và đặc tả các phép toán.

- **Đặc tả đối tượng dữ liệu**. Mô tả bằng toán học các đối tượng dữ liệu. Thông thường các đối tượng dữ liệu là các đối tượng trong thế giới hiện thực, chúng là các thực thể phức hợp, có cấu trúc nào

đó. Để mô tả chúng, chúng ta cần sử dụng sự trừu tượng hoá (chỉ quan tâm tới các đặc tính quan trọng, bỏ qua các chi tiết thứ yếu). Nói cụ thể hơn, để mô tả chính xác các đối tượng dữ liệu , chúng ta cần sử dụng các khái niệm toán học, các mô hình toán học như tập hợp, dãy, đồ thị, cây, … Chẳng hạn, đối tượng dữ liệu là sinh viên, thì có thể biểu diễn nó bởi một tập các thuộc tính quan trọng như tên, ngày sinh, giới tính, …

- **Đặc tả các phép toán**. Việc mô tả các phép toán phải đủ chặt chẽ, chính xác nhằm xác định đầy đủ kết quả mà các phép toán mang lại, nhưng không cần phải mô tả các phép toán được thực hiện như thế nào để cho kết quả như thế. Cách tiếp cận chính xác để đạt được mục tiêu trên là khi mô tả các phép toán, chúng ta xác định một tập các tiên đề mô tả đầy đủ các tính chất của các phép toán. Chẳng hạn, các phép toán cộng và nhân các số nguyên phải thoả mãn các tiên đề: giao hoán, kết hợp, phân phối, … Tuy nhiên, việc xác định một tập đầy đủ các tiên đề mô tả đầy đủ bản chất của các phép toán là cực kỳ khó khăn, do đó chúng ta mô tả các phép toán một cách không hình thức. Chúng ta sẽ mô tả mỗi phép toán bởi một hàm (hoặc thủ tục), tên hàm là tên của phép toán, theo sau là danh sách các biến. Sau đó chỉ rõ nhiệm vụ mà hàm cần phải thực hiện.

Ví dụ. Sau đây là đặc tả KDLTT số phức. Trong sách này, chúng ta sẽ đặc tả các KDLTT khác theo khuôn mẫu của ví dụ này.

Mỗi số phức là một cặp số thực (x, y), trong đó x được gọi là phần thực (real), y được gọi là phần ảo (image) của số phức.

Trên các số phức, có thể thực hiện các phép toán sau:

1. Create (a, b). Trả về số phức có phần thực là a, phần ảo là b.
2. GetReal (c). Trả về phần thực của số phức c.
3. GetImage (c). Trả về phần ảo của số phức c.
4. Abs (c). Trả về giá trị tuyệt đối (modun) của số phức c.
5. Add (c_1,c_2). Trả về tổng của số phức c_1 và số phức c_2.
6. Multiply (c_1 , c_2). Trả về tích của số phức c_1 và số phức c_2 .

7. Print (c). Viết ra số phức c dưới dạng a + i b trong đó a là phần thực, b là phần ảo của số phức c.

Trên đây chỉ là một số ít các phép toán số phức. Còn nhiều các phép toán khác trên số phức, chẳng hạn các phép toán so sánh, các phép toán lượng giác, …, để cho ngắn ngọn chúng ta không liệt kê ra hết.

1.3.2 Cài đặt kiểu dữ liệu trừu tượng

Trong giai đoạn đặc tả, chúng ta chỉ mới mô tả các phép toán trên các đối tượng dữ liệu, chúng ta chưa xác định các phép toán đó thực hiện nhiệm vụ của mình như thế nào. Trong chương trình, để sử dụng được các phép toán của một KDLTT đã đặc tả, chúng ta cần phải cài đặt KDLTT đó trong một ngôn ngữ lập trình.

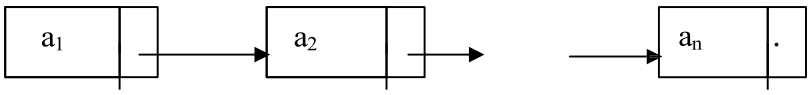
Công việc đầu tiên phải làm khi cài đặt một KDLTT là chọn một CTDL để biểu diễn các đối tượng dữ liệu. Cần lưu ý rằng, một CTDL là một dữ liệu phức hợp được tạo nên từ nhiều dữ liệu thành phần bằng các liên kết nào đó. Chúng ta có thể mô tả các CTDL trong một ngôn ngữ lập trình (chẳng hạn, C/ C + +) bằng cách sử dụng các phương tiện có sẵn trong ngôn ngữ lập trình đó, chẳng hạn sử dụng các qui tắc cú pháp mô tả mảng, cấu trúc, … Một CTDL cũng xác định cho ta cách lưu trữ dữ liệu trong bộ nhớ của máy tính.

Ví dụ. Chúng ta có thể biểu diễn một số phức bởi cấu trúc trong C + +

```
struct      complex
{
        float    real;
        float    imag;
 }
```

Cần chú ý rằng, một đối tượng dữ liệu có thể cài đặt bởi các CTDL khác nhau. Chẳng hạn, một danh sách (a_1, a_2, …, a_n) có thể cài đặt bởi mảng A, các thành phần của danh sách lần lượt được lưu trong các thành phần liên

tiếp của mảng A[0], A[1],…, A[n-1]. Nhưng chúng ta cũng có thể cài đặt danh sách bởi CTDL danh sách liên kết sau:



Sau khi đã chọn CTDL biểu diễn đối tượng dữ liệu, bước tiếp theo chúng ta phải thiết kế và cài đặt các hàm thực hiện các phép toán của KDLTT.

Trong giai đoạn thiết kế một hàm thực hiện nhiệm vụ của một phép toán, chúng ta cần sử dụng **sự trừu tượng hoá hàm (functional abstraction).** Sự trừu tượng hoá hàm có nghĩa là cần mô tả hàm sao cho người sử dụng biết được hàm thực hiện công việc gì, và sao cho họ có thể sử dụng được hàm trong chương trình của mình mà không cần biết đến các chi tiết cài đặt, tức là không cần biết hàm thực hiện công việc đó như thế nào.

Sự trừu tượng hoá hàm được thực hiện bằng cách viết ra **mẫu hàm (function prototype)** kèm theo các chú thích.

Mẫu hàm gồm tên hàm và theo sau là danh sách các tham biến. Tên hàm cần ngắn ngọn, nói lên được nhiệm vụ của hàm. Các tham biến cần phải đầy đủ: các dữ liệu vào cần thiết để hàm có thể thực hiện được công việc của mình và các dữ liệu ra sau khi hàm hoàn thành công việc.

Chú thích đưa ra sau đầu hàm là rất cần thiết (đặc biệt trong các đề án lập trình theo đội). Trong chú thích này, chúng ta cần mô tả đầy đủ, chính xác nhiệm vụ của hàm. Sau đó là hai phần: Preconditions (các điều kiện trước) và Postconditions (các điều kiện sau).

- Preconditions gồm các phát biểu về các điều kiện cần phải thoả mãn trước khi hàm thực hiện.
- Postconditions gồm các phát biểu về các điều kiện cần phải thoả mãn sau khi hàm hoàn thành thực hiện.

Hai phần Preconditions và Postconditions tạo thành **hợp đồng** giữa một bên là người sử dụng hàm và một bên là hàm. Preconditions là trách

nhiệm của người sử dụng, còn Postconditions là trách nhiệm của hàm. Một khi sử dụng hàm (gọi hàm), người sử dụng phải có trách nhiệm cung cấp cho hàm các dữ liệu vào thoả mãn các điều kiện trong Preconditions. Sau khi hoàn thành thực hiện, hàm phải cho ra các kết quả thoả mãn các điều kiện trong Postconditions. Sau đây là ví dụ một mẫu hàm:

```
void        Sort (int  A[ ], int  n)
//  Sắp xếp mảng A theo thứ tự không giảm.
//  Preconditions: A là mảng số nguyên có cỡ Max ≥ n.
//  Postconditions: A[0] ≤ A[1] ≤ …. ≤ A[n-1],
//  n không thay đổi.
```

Bước tiếp theo, chúng ta phải thiết kế thuật toán thực hiện công việc của hàm khi mà đối tượng dữ liệu được biểu diễn bởi CTDL đã chọn. Việc cài đặt hàm bây giờ là chuyển dịch thuật toán thực hiện nhiệm vụ của hàm sang dãy các khai báo biến địa phương cần thiết và các câu lệnh. Tất cả các chi tiết mà hàm cần thực hiện này là công việc riêng tư của hàm, người sử dụng hàm không cần biết đến, và không được can thiệp vào. Làm được như vậy có nghĩa là chúng ta đã thực hành **nguyên lý che dấu thông tin (the principle of information hiding)** - một nguyên lý quan trọng trong phương pháp luận lập trình môđun.

Trên đây chúng ta mới chỉ trình bày các kỹ thuật liên quan đến thiết kế CTDL cho đối tượng dữ liệu, thiết kế và cài đặt các hàm cho các phép toán của KDLTT. Câu hỏi được đặt ra là: Chúng ta phải tổ chức CTDL và các hàm đó như thế nào? Có hai cách: cách cài đặt cổ điển và cách cài đặt định hướng đối tượng. Mục sau sẽ trình bày phương pháp cài đặt KDLTT trong ngôn ngữ C. Cài đặt KDLTT bởi lớp trong C + + sẽ được trình bày trong chương 3.

1.4 CÀI ĐẶT KIỂU DỮ LIỆU TRỪU TƯỢNG TRONG C

Trong mục này chúng ta sẽ trình bày phương pháp cài đặt KDLTT theo cách truyền thống (cài đặt không định hướng đối tượng) trong C. Trong cách cài đặt này, chúng ta sẽ xây dựng nên một thư viện các hàm thực hiện các phép toán của một KDLTT sao cho bạn có thể sử dụng các hàm này trong một chương trình bất kỳ giống như bạn sử dụng các hàm trong thư viện chuẩn. Sự xây dựng một thư viện điển hình được tổ chức thành hai file: file đầu (header file) và file cài đặt (implementation file).

- File đầu chứa các mệnh đề # include, các định nghĩa hằng, … cần thiết và khai báo CTDL. Theo sau là các mẫu hàm cho mỗi phép toán của KDLTT.
- File cài đặt cũng chứa các mệnh đề # include cần thiết và chứa các định nghĩa của các hàm đã được đặc tả trong file đầu.

Tên file đầu có đuôi là .h, file cài đặt có đuôi là .c (hoặc .cpp, .cxx). File cài đặt được dịch và được kết nối vào file thực hiện được mỗi khi cần thiết. Với cách tổ chức này, bạn có thể sử dụng các hàm của một KDLTT giống hệt như bạn sử dụng các hàm trong thư viện chuẩn. Chỉ có một việc bạn cần nhớ là bạn phải include file đầu vào trong chương trình của bạn bởi mệnh đề:

include "tên file đầu"

Người sử dụng các hàm của một KDLTT chỉ cần biết các thông tin trong file đầu, không cần biết các hàm này được cài đặt như thế nào (các thông tin trong file cài đặt).

Ví dụ. Cài đặt KDLTT số phức đã đặc tả trong mục 2.3. File đầu complex.h cho trong hình 2.1. Nội dung của file này nằm giữa các mệnh đề

ifndef … # define … và # endif. Đây là các chỉ định tiền xử lý cần thiết phải có nhằm đảm bảo file đầu chỉ có mặt một lần trong file nguồn chương trình của bạn.

```
// File : complex.h
# ifndef  COMPLEX_H
```

```
# define    COMPLEX_H
          struct  Complex
          {
             double  real;
             double  image;
          };

 Complex      CreateComplex (double a, double b) ;
// Postcondition: Trả về số phức có phần thực là a, phần ảo là b.

double     GetReal (Complex c);
// Postcondition: Trả về phần thực của số phức c.

double     GetImag (Complex c);
// Postcondition: Trả về phần ảo của số phức c.

double     GetAbs (Complex c);
// Postcondition: Trả về giá trị tuyệt đối (mođun) của số phức c.

Complex     Add (Complex c1, Complex c2);
// Postcondition: Trả về số phức là tổng của số phức c1 và số phức c2.

Complex     Multiply (Complex c1, Complex c2);
// Postcondition: Trả về số phức là tích của số phức c1 và số phức c2.

void    Print (Complex c);
// Postcondition: số phức c được viết ra dưới dạng a + ib, trong đó a là
// phần thực, b là phần ảo của số phức c.

// Mẫu hàm của các phép toán khác.
# endif
```

Hình 1.1. File đầu của sự cài đặt không định hướng đối tượng của KDLTT số phức trong C/C + +.

File cài đặt KDLTT số phức được cho trong hình 1.2. Trong file cài đặt, ngoài các mệnh đề # include cần thiết cho sự cài đặt các hàm, nhất thiết phải có mệnh đề

include "tên file đầu"

```
// File: Complex.cxx
# include  "complex.h"
# include  < math.h >          // cung cấp hàm sqrt
# include  < iostream.h >     // cung cấp đối tượng cout

Complex   CreateComplex (double a, double b)
{
    Complex  c ;
    c.real = a ;
    c.imag = b ;
    return c ;
}

double  GetReal (Complex c)
{
   return  c.real ;
}

double  GetImag (Complex c)
{
   return c.imag ;
}
```

```
double  GetAbs (Complex c)
{
   double  result ;
   result = sqrt ( c.real * c.real + c.imag * c.imag);
   return  result ;
}

Complex  Add (Complex c1, Complex c2)
{
   Complex  c ;
   c.real = c1.real + c2.real ;
   c.imag = c1.imag + c2.imag ;
   return c ;
}

Complex  Multiply (Complex c1, Complex c2)
{
   Complex  c ;
   c.real = (c1.real * c2.real) – (c1.imag * c2.imag) ;
   c.imag = (c1.real *c2.imag) + (c1.imag * c2.real) ;
   return  c ;
}

void  Print (Complex c)
{
  cout << c.real << " +i " << r.imag << " \ n" ;
}
```

Hình 1.2. File cài đặt của KDLTT số phức.

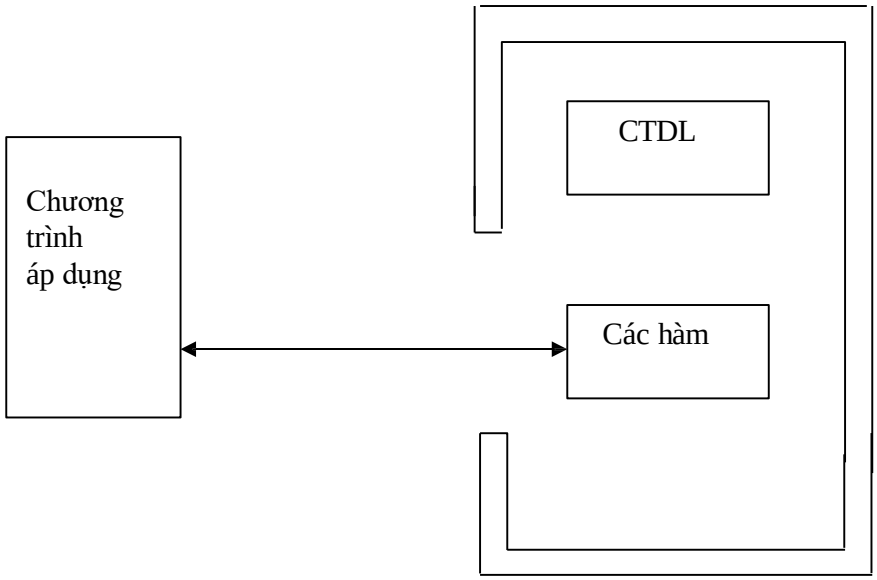
1.5 TRIẾT LÝ CÀI ĐẶT KIỂU DỮ LIỆU TRỪU TƯỢNG

Trong giai đoạn thiết kế chương trình, chúng ta cần xem xét dữ liệu dưới cách nhìn của sự trừu tượng hoá dữ liệu. Cụ thể hơn là, trong giai đoạn thiết kế chương trình, chúng ta chỉ cần quan tâm tới đặc tả của các KDLTT. Cài đặt KDLTT có nghĩa là biểu diễn các đối tượng dữ liệu bởi các CTDL và cài đặt các hàm thực hiện các phép toán trên dữ liệu. Cần nhấn mạnh rằng, chỉ có một cách nhìn logic đối với dữ liệu, nhưng có thể có nhiều cách tiếp cận để cài đặt nó. Nói một cách khác, ứng với mỗi KDLTT có thể có nhiều cách cài đặt.

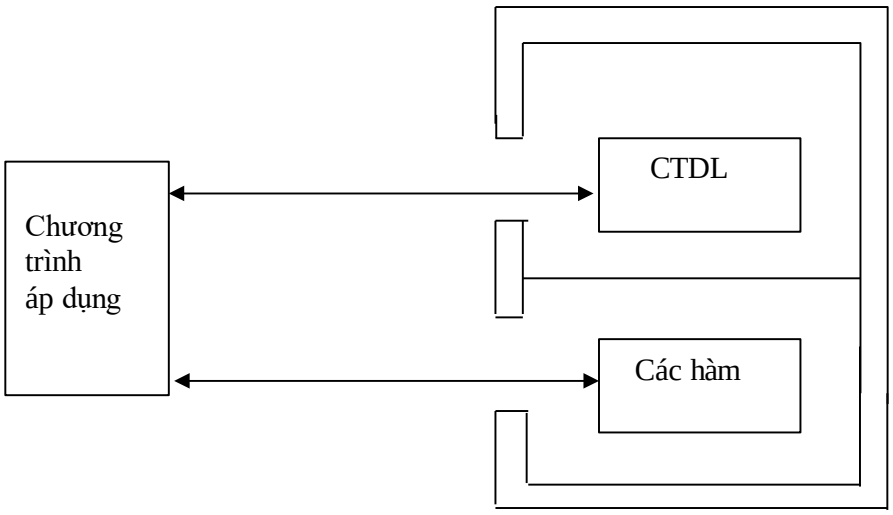
Một chương trình áp dụng cần phải sử dụng các phép toán trên dữ liệu dưới dạng biểu diễn trừu tượng, chứ không phải dưới dạng mà dữ liệu được lưu trữ trong bộ nhớ của máy tính (dạng cài đặt). Chẳng hạn, chúng ta đã sử dụng các số nguyên trong chương trình dưới dạng biểu diễn toán học của các số nguyên, và sử dụng các phép toán +, - , *, / các số nguyên với cách viết và ý nghĩa của chúng trong toán học mà không cần biết các số nguyên và các phép toán +, -, *, / được cài đặt như thế nào trong máy tính. (Cần biết rằng, các số nguyên cùng với các phép toán trên số nguyên: +, - , *, / tạo thành một KDLTT và KDLTT này đã được cài đặt sẵn, chúng ta chỉ việc sử dụng.)

Do vậy, khi cài đặt KDLTT chúng ta cần tạo ra một giao diện (interface) giữa chương trình áp dụng và sự cài đặt KDLTT. Giao diện này bao gồm các phép toán đã xác định trong KDLTT. Người sử dụng chỉ được phép giao tiếp với sự cài đặt KDLTT thông qua giao diện này. Nói một cách hình ảnh thì cách cài đặt KDLTT cần phải sao cho tạo thành bức tường giữa chương trình áp dụng và sự cài đặt KDLTT. Bức tường này che chắn CTDL, chỉ có thể truy cập tới CTDL thông qua các phép toán đã đặc tả trong KDLTT (hình 1.3.a). Nếu thực hiện được sự cài đặt KDLTT như thế, thì khi ta thay đổi CTDL biểu diễn đối tượng dữ liệu và thay đổi cách cài đặt các hàm cũng không ảnh hưởng gì đến chương trình áp dụng sử dụng KDLTT này. Điều này tương tự như khi ta sử dụng máy bán nước giải khát tự động.

Thiết kế bên ngoài và các chỉ dẫn sẽ cho phép người sử dụng mua được loại nước mà mình mong muốn. Chẳng hạn, có ba nút ấn: cam, côca, cà phê. Bỏ 5 xu vào lỗ dưới nút cam và ấn nút cam, người sử dụng sẽ nhận được cốc cam ,… Người sử dụng không cần biết đến cấu tạo bên trong của máy.Chúng ta có thể thay đổi cấu tạo bên trong của máy, miễn là nó vẫn đáp ứng được mong muốn của người sử dụng khi họ thực hiện các thao tác theo chỉ dẫn.



a. Cài đặt định hướng đối tượng



b. Cài đặt không định hướng đối tượng

Hình 1.3. Chương trình áp dụng và sự cài đặt KDLTT.

Cách cài đặt truyền thống (cài đặt không định hướng đối tượng) đã tách biệt CTDL và các hàm. Người sử dụng vẫn có thể truy cập trực tiếp đến CTDL không cần thông qua các hàm (hình 1.3.b). Nói một cách khác, cách cài đặt truyền thống không tạo thành bức tường vững chắc che chắn CTDL. Điều này có thể dẫn tới các lỗi không kiểm soát được, khó phát hiện. Chỉ có cách cài đặt định hướng đối tượng mới đảm bảo yêu cầu tạo ra bức tường giữa chương trình áp dụng và sự cài đặt KDLTT.

Trong cách cài đặt định hướng đối tượng sử dụng C + +, CTDL và các hàm thực hiện các phép toán trên dữ liệu được đóng gói (encapsulation) vào một thực thể được gọi là lớp. Lớp có cơ chế điều khiển sự truy cập đến CTDL. Mỗi lớp cung cấp cho người sử dụng một giao diện. Chương trình áp dụng chỉ có thể truy cập đến CTDL qua giao diện này (bao gồm các hàm trong mục public). Cài đặt KDLTT bởi lớp C + + cho phép ta khi viết các chương trình ứng dụng, có thể biểu diễn các phép toán trên các đối tượng dữ liệu dưới dạng các biểu thức toán học rất sáng sủa, dễ hiểu. Cách cài đặt KDLTT bởi lớp C + + còn cho phép thực hành sử dụng lại phần mềm

(reusability). Chúng ta sẽ nghiên cứu phương pháp cài đặt KDLTT bởi lớp trong chương sau.

BÀI TẬP

Trong các bài tập sau đây, hãy đặc tả các KDLTT bằng cách thực hiện hai phần sau:

- Mô tả đối tương dữ liệu bằng cách sử dụng các ký hiệu và các khái niệm toán học.
- Mô tả các phép toán trên các đối tượng dữ liệu đó. Cần biểu diễn mỗi phép toán bởi hàm, nói rõ mục tiêu của hàm, các điều kiện để hàm thực hiện được mục tiêu đó, và hiệu quả mà hàm mang lại.

1. KDLTT tập hợp với các phép toán: hợp, giao, hiệu, kiểm tra một tập có rỗng không, kiểm tra một đối tượng có là phần tử của một tập, kiểm tra hai tập có bằng nhau không, kiểm tra một tập có là tập con của một tập khác không.
2. KDLTT điểm (điểm trên mặt phẳng). Các phép toán gồm: tịnh tiến điểm, quay điểm đi một góc, tính khoảng cách giữa hai điểm, xác định điểm giữa hai điểm.
3. KDLTT hình cầu.
4. KDLTT ma trận.
5. KDLTT phân số.
6. KDLTT xâu ký tự.

Trong các bài tập 3 → 6, hãy tự xác định các phép toán (càng đầy đủ càng tốt), và hãy đặc tả các phép toán bởi các hàm.