ĐỀ CƯƠNG BÀI GIẢNG

BÀI 1: TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

1.1 Giải thuật và cấu trúc dữ liệu

1.1.1. Thông tin và dữ liệu

Dữ liệu là gì?

Dữ liệu là một khái niệm rất trừu tượng, chính là thông tin đã được đưa vào máy tính. Dữ liệu sau khi tập hợp và xử lý sẽ cho ta thông tin. Nói một cách khác, dữ liệu là thông tin đã được mã hóa trong máy tính. Ví dụ như con số điểm thi là một dữ liệu hoặc con số về nhiệt độ trong ngày là một dữ liệu, hình ảnh về con người hay phong cảnh cũng là những dữ liệu, …

Theo wikipedia, khái niệm dữ liệu là gì còn được định nghĩa là chuỗi bất kỳ của một hoặc nhiều ký tự có ý nghĩa thông qua việc giải thích một hành động cụ thể nào. Dữ liệu phải được thông dịch để trở thành thông tin. Để dữ liệu trở thành thông tin, cần xem xét một số nhân tố bao gồm người hoặc vật tạo ra dữ liệu.

Dữ liệu biểu diễn số lượng, tính chất hoặc ký hiệu hoạt động được máy tính lưu trữ trên ổ cứng từ, đĩa quang và được truyền đi dưới dạng tín hiệu điện.

Thông tin là gì?

Thông tin chính là sự thông báo, trao đổi, giải thích về một đối tượng nào đó và thường được thể hiện dưới dạng các tín hiệu như chữ viết, âm thanh, dòng điện, … Nói một cách khái quát, thông tin chính là sự hiểu biết của con người về một thực thể nào đó, có thể thu nhập, lưu trữ và xử lý được.

Thông tin sẽ được liên kết với dữ liệu vì dữ liệu đại diện cho các giá trị được quy cho các tham số và thông tin là dữ liệu theo ngữ cạnh và có ý nghĩa kèm theo. Thông tin cũng liên quan đến kiến thức bởi kiến thức biểu thị sự hiểu biết về một khái niệm trừu tượng hoặc cụ thể.

Thông tin có thể được mã hóa thành nhiều dạng khác nhau để truyền vài giải thích, nó cũng có thể được mã hóa để lưu trữ và liên lạc an toàn.

Thông tin được chia ra làm 2 loại chính đó là số (số nguyên, số thực) và phi số (là các văn bản, hình ảnh, âm thanh, …)

Đơn vị đo thông tin là gì?

Đơn vị đo thông tin chính là bit (Binary digit). Bit là dung lượng nhỏ nhất có thể ghi được hoặc kí hiệu là 0 hoặc 1. Hai ký hiệu này được dùng để biểu diễn thông tin trong máy tính. Ngoài đơn vị trên, byte cũng được coi là một đơn vị đo của thông tin, 1 byte = 8 bit.

1 byte = 8 bit.

1 kilôbai (kB) = 1024 byte = 210 byte.

1 megabit (MB)= 1024 kB = 210 kB.

1 gigabyte (GB) = 1024 MB = 210 MB.

1 têrabai (TB) = 1024 GB = 210 GB.

1 petabyte (PB)= 1024 TB = 210 TB.

Mã hóa thông tin trong máy tính như thế nào?

Để máy tính xử lý được thông tin thì thông tin cần phải được biến đổi thành một dãy bit. Cách biến đổi như vậy được biết đến với tên gọi là mã hóa thông tin.

Bộ mã ASCII được sử dụng để mã hóa thông tin dạng văn bản, sử dụng 8 bit để mã hóa ký tự. Trong bộ mã ASCII, các ký tự sẽ được đánh số từ 0 cho tới 255 và các ký tự này được gọi là mã ASCII thập phân của ký tự.

Bộ mã Unicode sử dụng 16 bit để mã hóa vì bộ mã ASCII chỉ mã hóa được 256 ký tự, chưa đủ để đồng hóa với bảng chữ cái của ngôn ngữ trên thế giới. Bộ Unicode mã hóa được 65536 ký tự khác nhau. Đây là bộ mã hóa chung được sử dụng để thể hiện các văn bản hành chính. Thông tin có nhiều dạng khác nhau nhưng đều được lưu trữ và xử lý trong máy tính ở một dạng chung nhất đó là mã nhị phân.

1.1.2. Khái niệm cấu trúc dữ liệu và giải thuật

Các thành phần dữ liệu thực tế rất đa dạng, phong phú và thường chứa đựng những quan hệ nào đó với nhau. Do đó, trong mô hình tin học của bài toán, cần phải biểu diễn chúng một cách thích hợp nhất, để vừa có thể phản ánh chính xác các dữ liệu thực tế này, vừa có thể dễ dàng dùng máy tính để xử lý. Công việc này gọi là xây dựng cấu trúc dữ liệu cho bài toán.

Từ những yêu cầu xử lý trong thực tế, cần tìm ra các giải pháp tương ứng để giải quyết yêu cầu, mỗi giải pháp cần phải xác định trình tự các thao tác máy tính phải thi hành để cho ra kết quả mong muốn. Đây là bước xây dựng giải thuật cho bài toán. Có thể sử dụng các giải thuật có sẵn, hoặc tự xây dựng.

Tuy nhiên, khi giải quyết bài toán, ta thường có khuynh hướng chỉ chú trọng đến việc xây dựng các giải thuật mà quên đi tầm quan trọng của việc tổ chức dữ liệu trong bài toán. Giải thuật phản ánh các phép xử lý, còn đối tượng xử lý của giải thuật lại là dữ liệu. Chính dữ liệu chứa đựng các thông tin cần thiết để thực hiện giải thuật. Để xác định được giải thuật phù hợp cần ta cần phải biết nó tác động đến loại dữ liệu nào và khi lựa chọn cấu trúc dữ liệu cũng cần phải hiểu rõ những thao tác nào tác động lên dữ liệu đó.

1.1.3. Khái niệm giải thuật

Giải thuật

Giải thuật (algorithm) là một trong những khái niệm quan trọng nhất trong tin học. Thuật ngữ giải thuật xuất phát từ nhà toán học Arập Abu Ja'far Mohammed ibn Musa al Khowarizmi (khoảng năm 825). Tuy nhiên, trong lúc bấy giờ và trong nhiều thế kỷ sau, nó không mang nội dung như ngày nay chúng ta quan niệm. Giải thuật nổi tiếng nhất, có từ thời cổ Hy Lạp là giải thuật Euclid, giải thuật tìm ước chung lớn nhất của hai số nguyên. Có thể mô tả giải thuật này như sau

* Giải thuật Euclid

Thông tin vào: m, n nguyên dương

Thông tin ra: d, ước chung lớn nhất của m và n.

Phương pháp

Bước 1: Tìm r, phần dư của phép chia m cho n

Bước 2: Nếu r = 0, thì d ← n (gán giá trị của n cho d) và dừng lại

Ngược lại, thì m ← n, n ← r và quay lại bước 1

Khái niệm về giải thuật

Giải thuật là một dãy hữu hạn các bước, mỗi bước mô tả chính xác các phép toán hoặc hành động cần thực hiện để giải quyết vấn đề đặt ra.

1.1.4 Mối quan hệ giữa CTDL và giải thuật

Cấu trúc dữ liệu và giải thuật có mối quan hệ chặt chẽ với nhau, được thể hiện qua 'công thức'

Cấu trúc dữ liệu + Giải thuật = Chương trình

Với một cấu trúc dữ liệu đã chọn, sẽ có những giải thuật tương ứng, phù hợp. Khi cấu trúc dữ liệu thay đổi thường giải thuật cũng phải thay đổi theo để tránh việc xử lý gượng ép, thiếu tự nhiên trên một cấu trúc không phù hợp. Hơn nữa, một cấu trúc dữ liệu tốt sẽ giúp giải thuật xử lý trên đó có thể phát huy tác dụng tốt hơn, vừa nhanh vừa tiết kiệm, giải thuật cũng đơn giản và dễ hiểu hơn.

1.2 Phân tích và đánh giá thuật toán

1.2.1. Phân tích giải thuật

Giả sử đối với một bài toán nào đó chúng ta có một số giải thuật giải. Một câu hỏi đặt ra là, chúng ta cần chọn giải thuật nào trong số giải thuật đã có để giải bài toán một cách hiệu quả nhất. Sau đây ta phân tích giải thuật và đánh giá độ phức tạp tính toán của nó.

1.2.2. Tính hiệu quả của giải thuật

Khi giải quyết một vấn đề, chúng ta cần chọn trong số các giải thuật, một giải thuật mà chúng ta cho là tốt nhất. Vậy ta cần lựa chọn giải thuật dựa trên cơ sở nào? Thông thường ta dựa trên hai tiêu chuẩn sau đây:

1. Giải thuật đơn giản, dễ hiểu, dễ cài đặt (dễ viết chương trình)

2. Giải thuật sử dụng tiết kiệm nhất nguồn tài nguyên của máy tính, và đặc biệt, chạy nhanh nhất có thể được.

Tiêu chuẩn (2) được xem là tính hiệu quả của giải thuật. Tính hiệu quả của giải thuật bao gồm hai nhân tố cơ bản

- Dung lượng nhớ cần thiết để lưu giữ các dữ liệu vào, các kết quả tính toán trung gian và các kết quả của giải thuật.

- Thời gian cần thiết để thực hiện giải thuật (ta gọi là thời gian chạy chương trình, thời gian này không phụ thuộc vào các yếu tố vật lý của máy tính như tốc độ xử lý của máy tính, ngôn ngữ viết chương trình v.v...).

Chúng ta sẽ chỉ quan tâm đến thời gian thực hiện giải thuật. Vì vậy khi nói đến đánh giá độ phức tạp của giải thuật, có nghĩa là ta nói đến đánh giá thời gian thực hiện. Một giải thuật có hiệu quả được xem là giải thuật có thời gian chạy ít hơn các giải thuật khác.

1.2.3. Đánh giá thời gian thực hiện của giải thuật

Có hai cách tiếp cận để đánh giá thời gian thực hiện của một giải thuật

a. Phương pháp thử nghiệm:

Chương trình được viết và cho chạy với các dữ liệu vào khác nhau trên một máy tính nào đó. Thời gian chạy chương trình phụ thuộc vào các yếu tố sau đây:

1. Các dữ liệu vào

2. Chương trình dịch, để chuyển chương trình mã nguồn thành chương trình mã máy.

3. Tốc độ thực hiện các phép toán của máy tính được sử dụng để chạy chương trình.

Vì thời gian chạy chương trình phụ thuộc vào nhiều yếu tố, nên ta không thể biểu diễn chính xác thời gian chạy là bao nhiêu đơn vị thời gian chuẩn, chẳng hạn nó là bao nhiêu giây.

b. Phương pháp lý thuyết:

Ta sẽ coi thời gian thực hiện của giải thuật như là một hàm số của kích thước dữ liệu vào. Kích thước của dữ liệu vào là một tham số đặc trưng cho dữ liệu vào, nó có ảnh hưởng quyết định đến thời gian thực hiện chương trình. Đơn vị tính kích thước của dữ liệu vào phụ thuộc vào các giải thuật cụ thể. Chẳng hạn, đối với các giải thuật sắp xếp mảng, thì kích thước của dữ liệu vào là số thành phần của mảng, đối với giải thuật giải hệ n phương trình tuyến tính với n ẩn, ta chọn n là kích thước. Thông thường dữ liệu vào là một số nguyên dương n. Ta sẽ sử dụng hàm số T(n),

trong đó n là kích thước dữ liệu vào, để biểu diễn thời gian thực hiện của một giải thuật.

Ta có thể xác định thời gian thực hiện T(n) là số phép toán sơ cấp cần phải tiến hành khi thực hiện giải thuật. Các phép toán sơ cấp là các phép toán mà thời gian thực hiện bị chặn trên bởi một hằng số chỉ phụ thuộc vào cách cài đặt được sử dụng. Chẳng hạn các phép toán số học +, -, *, /, các phép toán so sánh =, !=... là các phép toán sơ cấp.

1.2.4. Đánh giá độ phức tạp tính toán của giải thuật

Khi đánh giá thời gian thực hiện bằng phương pháp toán học, chúng ta sẽ bỏ qua yếu tố phụ thuộc vào cách cài đặt, chỉ tập trung vào xác định độ lớn của thời gian thực hiện T(n). Ký hiệu toán học O (đọc là ô lớn) được sử dụng để mô tả độ lớn của hàm T(n).

Giả sử n là số nguyên không âm, T(n) và f(n) là các hàm thực không âm. Ta viết T(n) = O(f(n)) (đọc: T(n) là ô lớn của f(n)), nếu và chỉ nếu tồn tại các hằng số dương c và no sao cho $T(n) \leq c.f(n)$, với $\forall n > no$.

Nếu một giải thuật có thời gian thực hiện T(n) = O(f(n)), chúng ta sẽ nói rằng giải thuật có thời gian thực hiện cấp f(n).

Ví dụ: Giả sử $T(n) = 10n^2 + 4n + 4$

Ta có: $T(n) \leq 10n^2 + 4n^2 + 4n^2 = 12\ n^2$, với $\forall n \geq 1$

Vậy $T(n) = O(n^2)$. Trong trường hợp này ta nói giải thuật có độ phức tạp (có thời gian thực hiện) cấp n^2.

Bảng sau đây cho ta các cấp thời gian thực hiện giải thuật được sử dụng rộng rãi nhất và tên gọi thông thường của chúng.

Ký hiệu ô lớn (O)	Tên gọi thông thường
O(1)	Hằng
$O(\log_2 n)$	logarit
O(n)	Tuyến tính
$O(n\log_2 n)$	$n\log_2 n$
$O(n^2)$	Bình phương
$O(n^3)$	Lập phương
$O(2^n)$	Mũ

Danh sách trên sắp xếp theo thứ tự tăng dần của cấp thời gian thực hiện.

Các hàm như $\log_2 n$, n, $n\log_2 n$, n^2, n^3 được gọi là các hàm đa thức. Giải thuật với thời gian thực hiện có cấp hàm đa thức thì thường chấp nhận được.

Các hàm như 2^n, n!, nn được gọi là hàm loại mũ. Một giải thuật mà thời gian thực hiện của nó là các hàm loại mũ thì tốc độ rất chậm.

1.2.5. Xác định độ phức tạp tính toán

Xác định độ phức tạp tính toán của một giải thuật bất kỳ có thể dẫn đến những bài toán phức tạp. Tuy nhiên, trong thực tế, đối với một số giải thuật ta cũng có thể phân tích được bằng một số qui tắc đơn giản.

a. Qui tắc tổng:

Giả sử T1(n) và T2(n) là thời gian thực hiện của hai giai đoạn chương trình P1 và P2 mà T1(n) = O(f(n); T2(n) = O(g(n)) thì thời gian thực hiện đoạn P1 rồi P2 tiếp theo sẽ là T1(n) + T2(n) = O(max(f(n), g(n))).

b. Qui tắc nhân:

Nếu tương ứng với P1 và P2 là T1(n) = O(f(n)), T2(n) = O(g(n)) thì thời gian thực hiện P1 và P2 lồng nhau sẽ là: T1(n).T2(n) = O(f(n).g(n))

Trong sách báo quốc tế các giải thuật thường được trình bầy dưới dạng các thủ tục hoặc hàm trong ngôn ngữ tựa Pascal/C. Để đánh giá thời gian thực hiện giải thuật, ta cần biết cách đánh giá thời gian thực hiện các câu lệnh của Pascal/C. Các câu lệnh trong Pascal/C được định nghĩa đệ qui như sau:

1. Các phép gán, đọc, viết, **goto** là các câu lệnh. Các lệnh này gọi là lệnh đơn

2. Nếu $S_1, S_2, ..., S_n$ là các câu lệnh thì

$\{S_1, S_2, ..., S_n\}$

là câu lệnh và được gọi là lệnh hợp thành (hoặc khối lệnh).

3. Nếu S_1 và S_2 là các câu lệnh và E là biểu thức logic thì

if **(**E) S_1 ***else*** S_2

và ***if*** (E) S_1

là câu lệnh và được gọi là lệnh ***if*** **– lệnh rẽ nhánh điều kiện.**

4. Nếu $S_1, S_2, ..., S_{n+1}$ là các câu lệnh, E là biểu thức có kiểu thứ tự đếm được, và $v_1, v_2, ..., v_n$ là các giá trị cùng kiểu với E thì

switch (E)

{

v_1: S_1 ; break;

v_2: S_2 ; break;

.....

v_n: S_n ; break;

[**default:** S_{n+1};]

}

là câu lệnh và được gọi là lệnh **switch – lệnh rẽ nhánh lựa chọn.**

5. Nếu S là câu lệnh và E là biểu thức logic thì

while (E) S;

là câu lệnh và được gọi là lệnh ***while* – vòng lặp.**

6. Nếu S_1, S_2, ..., S_n là các câu lệnh, E là biểu thức logic thì

do { S_1, S_2,..., S_n **}** ***while*** (E);

là câu lệnh và được gọi là lệnh **do … *while* – vòng lặp**

7. Với S là câu lệnh, E_1 và E_2 là các biểu thức có cùng một kiểu thứ tự đếm được, thì

for (i= E_1; i<=E_2; i++) S;

là câu lệnh, và

for (i= E_2; i>=E_1; i--) S

là câu lệnh. Lệnh này được gọi là lệnh **for – vòng lặp.**

Giả sử rằng, các lệnh gán không chứa các lời gọi hàm. Khi đó để đánh giá thời gian thực hiện một chương trình, ta có thể áp dụng phương pháp đệ qui sau

1. Thời gian thực hiện các lệnh đơn: gán, đọc, viết là O(1)

2. Lệnh hợp thành: thời gian thực hiện lệnh hợp thành được xác định bởi luật tổng.

3. Lệnh ***if***: Giả sử thời gian thực hiện các lệnh S_1, S_2 là O(f(n)) và O(g(n)) tương ứng. Khi đó thời gian thực hiện lệnh ***if*** là O(max (f(n), g(n)))

4. Lệnh **switch**: Lệnh này được đánh giá như lệnh ***if***

5. Lệnh ***while***: Giả sử thời gian thực hiện lệnh S (thân của ***while***) là O(f(n)) và g(n) là số tối đa các lần thực hiện lệnh S, khi đó thời gian thực hiện lệnh ***while*** là O(f(n).g(n)).

6. Lệnh **do…*while***: Giả sử thời gian thực hiện khối **{**S_1, S_2, ...,S_n**}** là O(f(n)) và g(n) là số lần lặp tối đa. Khi đó thời gian thực hiện lệnh **do…*while*** là O(f(n).g(n)).

7. Lệnh **for**: Lệnh này được đánh giá tương tự như lệnh **do…*while*** và ***while***.

Ví dụ 3: Tìm trong dãy số $s_1, s_2,..., s_n$ một phần tử có giá trị bằng x cho trước

Đầu vào: Dãy $s_1, s_2,..., s_n$ và khoá cần tìm x

Đầu ra: Vị trí phần tử có khoá x hoặc là n + 1 nếu không tìm thấy.

```
int  linear_search( day s, int n, kdl  x)
/*trong đó day, kdl là các kiểu dữ liệu đã được định nghĩa từ trước*/
{       int  i;
        i = 0;
        do
        {
                i = i + 1;
        }
        while (i<= n || s[i] != x);
        return  i;
}
```

Ví dụ này ta không thể đánh giá như hai ví dụ trên. Do quá trình tìm kiếm không những phụ thuộc vào kích thước của dữ liệu vào, mà còn phụ thuộc vào tình trạng của dữ liệu. Tức là thời gian thực hiện giải thuật còn phụ thuộc vào vị trí của phần tử trong dãy bằng x. Quá trình tìm kiếm chỉ dừng khi tìm thấy phần tử bằng x, hoặc duyệt hết dãy mà không tìm thấy. Vì vậy, trong những trường hợp như trên ta cần phải đánh giá thời gian tính tốt nhất, tồi nhất và trung bình của giải thuật với kích thước đầu vào n. Rõ ràng thời gian tính của giải thuật có thể được đánh giá bởi số lần thực hiện câu lệnh i = i + 1 (gọi là phép toán tích cực).

Nếu s[1] = x thì câu lệnh i = i + 1 trong thân vòng lặp do…while thực hiện 1 lần. Do đó thời gian tính tốt nhất của giải thuật là O(1).

Nếu x không xuất hiện trong dãy khoá đã cho, thì câu lệnh i = i + 1 được thực hiện n lần. Vì thế thời gian tính tồi nhất là O(n).

Cuối cùng ta tính thời gian tính trung bình của giải thuật. Nếu x được tìm thấy ở vị trí thứ i của dãy thì câu lệnh i = i + 1 phải thực hiện i lần (i = 1, 2, ..., n), còn nếu x không xuất hiện trong dãy thì câu lệnh i = i + 1 phải thực hiện n lần.

Từ đó suy ra số lần trung bình phải thực hiện câu lệnh i = i + 1 là

$$[(1 + 2 + ... + n) + n] /(n + 1)$$

Ta có:

$[(1 + 2 + ... + n) + n] /(n + 1) \leq (n^2 + n)/(n + 1) = n$

Vậy thời gian tính trung bình của giải thuật là O(n).

Nhận xét:

Việc xác định T(n) trong trường hợp trung bình thường gặp nhiều khó khăn vì sẽ phải dùng tới công cụ toán đặc biệt, hơn nữa tính trung bình có nhiều cách quan niệm. Trong các trường hợp mà T(n) trung bình thường khó xác định người ta thường đánh giá giải thuật qua giá trị xấu nhất của T(n). Hơn nữa, trong một số lớp giải thuật, việc xác định trường hợp xấu nhất là rất quan trọng.

1.3 Ngôn ngữ diễn đạt giải thuật

Dưới đây là 3 cách diễn đạt giải thuật phổ biến:

a. Ngôn ngữ tự nhiên: Liệt kê danh sách các bước của giải thuật bằng một ngôn ngữ nào đó phù hợp với người học kể cả sử dụng ngôn ngữ mẹ đẻ.
b. Lược đồ khối: Sử dụng ký hiệu, hình vẽ để biểu diễn giải thuật.
c. Mã giả: Biểu diễn giải thuật dựa theo cú pháp của ngôn ngữ lập trình (dự định cài đặt) chương trình.

Tuy nhiên, để đảm bảo tính xác định của giải thuật thì nên biểu diễn nó bằng ngôn ngữ lập trình cụ thể.

1.4 Các cấu trúc dữ liệu cơ sở

Máy tính thực sự chỉ có thể lưu trữ dữ liệu ở dạng nhị phân thô sơ. Nếu muốn phản ánh được dữ liệu thực tế vốn rất đa dạng và phong phú, cần phải xây dựng những phép ánh xạ, những qui tắc tổ chức phức tạp che lên tầng dữ liệu thô, nhằm đưa ra những khái niệm logic về hình thức lưu trữ khác nhau thường được gọi là kiểu dữ liệu. Như đã phân tích ở phần đầu, giữa hình thức lưu trữ và các thao tác xử lý trên đó có quan hệ mật thiết với nhau. Từ đó có thể đưa ra một định nghĩa cho kiểu dữ liệu như sau.

Kiểu dữ liệu T được xác định bởi bộ <V,O>, với :

- V: tập các giá trị hợp lệ mà đối tượng kiểu T có thể lưu trữ.
- O: tập các thao tác xử lý có thể thi hành trên đối tượng kiểu T.

Ví dụ: Kiểu dữ liệu **Số nguyên** = $<V_i, O_i>$

V_i = **{**-32768 ÷32767**}**

O_i = **{**+, -, *, /, %, các phép so sánh, các phép toán luận lý nhị phân**}**

Như vậy, muốn sử dụng một kiểu dữ liệu cần nắm vững cả nội dung dữ liệu được phép lưu trữ và các xử lý tác động trên đó.

Thông thường trong một hệ kiểu của ngôn ngữ lập trình sẽ có một số kiểu dữ liệu được gọi là *kiểu dữ liệu đơn* hay *kiểu dữ liệu phân tử* (atomic).

Thông thường, các kiểu dữ liệu cơ bản bao gồm :

- Kiểu có thứ tự rời rạc: số nguyên, ký tự, logic, liệt kê, miền con
- Kiểu không rời rạc: số thực

Tuỳ từng ngôn ngữ lập trình, các kiểu dữ liệu định nghĩa sẵn này có thể khác nhau đôi chút. Chẳng hạn, với ngôn ngữ C, các kiểu dữ liệu này chỉ gồm số nguyên, số thực, ký tự. Và theo quan điểm của C, kiểu ký tự thực chất cũng là kiểu số nguyên về mặt lưu trữ, chỉ khác về cách sử dụng. Ngoài ra, giá trị logic đúng (TRUE) và giá trị logic sai (FALSE) được biểu diễn trong ngôn ngữ C như là các giá trị nguyên khác 0 và bằng 0. Trong khi đó Pascal định nghĩa tất cả các kiểu dữ liệu đã liệt kê ở trên và phân biệt chúng một cách chặt chẽ.

Các kiểu dữ liệu có cấu trúc

Khi giải quyết các bài toán phức tạp, ta chỉ sử dụng các dữ liệu các dữ liệu đơn là không đủ, ta phải cần đến các *cấu trúc dữ liệu.* Một cấu trúc dữ liệu bao gồm một tập hợp các *dữ liệu phân tử,* các thành phần này móc nối với nhau bởi một phương thức được qui định bởi ngôn ngữ lập trình. Đa số các ngôn ngữ lập trình đều cài đặt sẵn một số kiểu có cấu trúc cơ bản như mảng, chuỗi, tập tin, cấu trúc v.v ... và cung cấp cơ chế cho lập trình viên tự định nghĩa kiểu dữ liệu mới.

Mảng một chiều

Trong C và trong nhiều ngôn ngữ thông dụng khác có một cách đơn giản nhất để tạo để móc nối các đối tượng trong một tập hợp, đó là cách sắp xếp các đối tượng đó thành một dãy. Để lưu trữ dãy đối tượng trong máy tính người ta sử dụng mảng một chiều. Khi đó ta có một cấu trúc dữ liệu được gọi là mảng (array). Như vậy, có thể nói một mảng là một cấu trúc dữ liệu gồm một dãy xác định các dữ liệu thành phần cùng một kiểu (mảng số nguyên, mảng số thực, mảng các cấu trúc v.v...).

Trong C việc khai báo một mảng là khá đơn giản, cần chỉ ra kiểu dữ liệu của phần tử, tên mảng, kích thước mảng, mẫu như sau:

<Kiểu phần tử> <tên mảng> <[kích thước]>;

Ví dụ: ***int*** A[10]; //khai báo mảng A chứa 10 số nguyên

Chuỗi

Trong C, chuỗi thực chất là một mảng các ký tự, tuy nhiên có khác là trong chuỗi có chứa ký tự kết thúc ‘\0’. Việc nhập và hiển thị chuỗi cũng đơn giản hơn

mảng, ta có thể sử dụng các hàm nhập xuất chuẩn trong thư viện "stdio.h" như *scanf(), gets(), printf(), puts()*.

C cũng định nghĩa một số hàm xử lý chuỗi (thư viện string.h) như: *strcpy(), strlen(), strcmp(), strchr(), strcat(), strstr()…*

Mảng nhiều chiều

Mảng nhiều chiều được sử dụng nhiều nhất là mảng 2 chiều, có thể hình dung mảng 2 chiều giống như một bảng gồm các dòng và các cột, chẳng hạn, bảng ghi nhiệt độ trung bình trong 5 năm ở năm thành phố.

	2003	***2004***	***2005***	***2006***	***2007***
Hà Nội	27	27,5	28,5	30	27
TP Hồ Chí Minh	32,5	32	30,5	31	30
Huế	30	31	32	28	29
Hải Phòng	27,5	26,5	26	27,5	27
Hạ Long	25	27	26	28	27

Cấu trúc khai báo của mảng nhiều chiều được viết như sau:

<Kiểu phần tử> <tên mảng ><[kích thước chiều 1]…[kích thước chiều n]>;

Sau tên mảng, mỗi cặp ngoặc vuông [ ] được tính là một chiều. Chữ số ghi trong cặp ngoặc [ ] là số phần tử của chiều đó.

Ví dụ: ***float*** temp [5][5]; // mảng 2 chiều temp, kích thước 5x5, các số thực.

Cấu trúc

Cấu trúc là tập hợp các mẫu dữ liệu khác nhau của một đối tượng (các mẫu dữ liệu có thể có kiểu khác nhau). Các mẫu dữ liệu đó được gọi là thành phần dữ liệu của cấu trúc. Các cấu trúc có thể được sử dụng để tạo nên các kiểu dữ liệu khác, chẳng hạn như mảng cấu trúc.

Giả sử T1, T2,...,T_n là các kiểu đã cho, và F_1, F_2,..., F_n là các tên thành phần. Khi đó ta có thể thành lập kiểu cấu trúc ST với n thành phần dữ liệu, thành phần thứ i có tên là F_i và các giá trị của nó có kiểu T_i vơi i = 1, 2,..., n.

```
struct  ST
{
        T1    F1 ;
        T2    F2 ;
        …
```

Tn Fn ;

} ;

struct ST s1, s2, d[20] ;

trong khai báo này s1, s2 là các biến cấu trúc, d là một mảng cấu trúc.

để truy nhập vào một thành phần dữ liệu của một cấu trúc ta viết theo mẫu:

<tên biến cấu trúc>.<tên thành phần>

Ví dụ : s1.F1, d[2].Fn

Mỗi giá trị của kiểu cấu trúc ST là một bộ k giá trị (t_1, t_2,..., t_k), trong đó $t_i \in T_i$ (i = 1, 2, ..., k).

Ví dụ khai báo cấu trúc lưu trữ phân số gồm tử số và mẫu số là các số nguyên

```
struct phan_so
{
    int tu_so;
    int mau_so;
};
//khai báo các biến lưu trữ phân số
struct phan_so p1, p2, ps[100];
```

Ở đây, p1, p2 là hai biến cấu trúc lưu trữ phân số, còn ps là mảng lưu trữ một dãy nhiều nhất là 100 phân số, ps được gọi là mảng cấu trúc.

Kiểu con trỏ

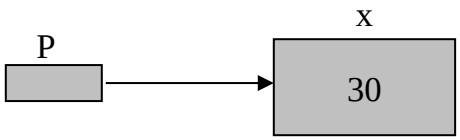
Một phương pháp quan trọng nữa để kiến tạo các cấu trúc dữ liệu, đó là sử dụng con trỏ.

Con trỏ là biến được sử dụng để lưu địa chỉ của một biến khác.

Con trỏ được khai báo theo mẫu: **<kiểu dữ liệu> *<tên con trỏ> ;**

Ví dụ : ***int*** *P, x ;

```
P=&x;  // P chứa địa chỉ của biến x, hay P trỏ vào x
```



Hình 1.1:Biểu diễn con trỏ

Sau này con trỏ được dùng để tạo ra kiểu danh sách móc nối, hoặc cây là các cấu trúc dữ liệu rất quan trọng, ta sẽ có dịp tìm hiểu kỹ hơn cách sử dụng con trỏ trong chương 3.

Kiểu file (tệp tin)

Khác với các kiểu dữ liệu trước đây, số nguyên, số thực, mảng, chuỗi v.v… dữ liệu được lưu ở bộ nhớ trong. Vì thế, khi chương trình kết thúc, dữ liệu cũng bị xóa. Để khắc phục trường hợp này, dữ liệu cần được lưu ở bộ nhớ ngoài, đó là các tệp tin.

- Theo cách lưu trữ này, khi thao tác cần một tên tệp tin (gồm cả đường dẫn), một con trỏ tệp (FILE * tên_con_trỏ).

- Khi thao tác với tệp tin cũng cần các hàm xử lý, bạn đọc có thể xem trong tài liệu (ngôn ngữ lập trình C – Quách Tuấn Ngọc).

Kết luận

Qua các nội dung trên chúng ta thấy rằng giai đoạn thiết kế hoặc lựa chọn các cấu trúc dữ liệu và các giải thuật khi phát triển dự án phần mềm là hết sức quan trọng, nó quyết định một phần sự thành bại của một dự án tin học. Chất lượng của dự án cũng phụ thuộc vào việc đánh giá các cấu trúc dữ liệu và các giải thuật khi chúng được thiết kế và lựa chọn.

Trong các nội dung sau, sẽ cung cấp cho người học một số các giải thuật và cấu trúc dữ liệu cơ bản, được áp dụng chủ yếu trong các bài toán quản lý như các giải thuật tìm kiếm, sắp xếp, cấu trúc danh sách tuyến tính, cây v.v… cùng với cách thiết kế, đánh giá và áp dụng chúng trong bài toán thực tế.