MỤC LỤC

Chương I:
TỔNG QUAN VỀ CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Chương này trình bày về tổng quan về cấu trúc dữ liệu và giải thuật, các tiêu chuẩn danh gia cấu trúc dữ liệu ,phản ánh đúng thực tếp, phù hợp với các thao tác trên đó, tiết kiệm tài nguyên hệ thống. Kiểu dữ liệu, kiểu dữ liệu cơ bản, các kiểu dữ liệu có cấu trúc, kiểu chuỗi ký tự, kiểu mảng, kiểu union, kiểu mẫu tin (cấu trúc), kiểu con trỏ, kiểu tập tin, mối quan hệ giữa cấu trúc dữ liệu và giải thuật.

I. KHÁI NIỆM THUẬT GIẢI VÀ ĐÁNH GIÁ ĐỘ PHỨC TẠP CỦA GIẢI THUẬT

1. Khái niệm thuật giải.

1.1. Thuật giải.

Tập các bước có thể tính toán được để đạt được kết quả mong muốn. Khi đã có mô hình thích hợp cho một bài toán ta cần cố gắng tìm cách giải quyết bài toán trong mô hình đó. Khởi đầu là tìm một giải thuật, đó là một chuỗi hữu hạn các chỉ thị (instruction) mà mỗi chỉ thị có một ý nghĩa rõ ràng và thực hiện được trong một lượng thời gian hữu hạn.

Knuth (1973) định nghĩa giải thuật là một chuỗi hữu hạn các thao tác để giải một bài toán nào đó. Các tính chất quan trọng của giải thuật là:

- *Hữu hạn* (finiteness): giải thuật phải luôn luôn kết thúc sau một số hữu hạn bước.

- *Xác định* (definiteness): mỗi bước của giải thuật phải được xác định rõ ràng và phải được thực hiện chính xác, nhất quán.

- *Hiệu quả* (effectiveness): các thao tác trong giải thuật phải được thực hiện trong một lượng thời gian hữu hạn.

Mỗi thuật toán có một dữ liệu vào (Input) và một dữ liệu ra (Output);

Nói tóm lại, một giải thuật phải giải quyết xong công việc khi ta cho dữ liệu vào. Có nhiều cách để thể hiện giải thuật: dùng lời, dùng lưu đồ, ... Và một lối dùng rất phổ biến là dùng ngôn ngữ giả, đó là sự kết hợp của ngôn ngữ tự nhiên và các cấu trúc của ngôn ngữ lập trình

Ví dụ: Tính tổng các số nguyên lẻ từ 1→n

B1: S=0

B2: i=1

B3: Nếu i>n thì sang B7, ngược lại sang B4

B4: S=S+i

B5: i=i+2

B6: Quay lại B3

B7: Tổng cần tìm là S

1.2. Đặc trưng của thuật giải

- Tính đúng đắn: Thuật toán cần phải đảm bảo cho một kết quả đúng sau khi thực hiện đối với các bộ dữ liệu đầu vào. Đây có thể nói là đặc trưng quan trọng nhất đối với một thuật toán: Thuật toán cần phải đảm bảo sẽ dừng sau một số hữu hạn bước.

- Tính xác định: Các bước của thuật toán phải được phát biểu rõ ràng, cụ thể, tránh gây nhập nhằng hoặc nhầm lẫn đối với người đọc và hiểu, cài đặt thuật toán.

- Tính hiệu quả: Thuật toán được xem là hiệu quả nếu như nó có khả năng giải quyết hiệu quả bài toán đặt ra trong thời gian hoặc các điều kiện cho phép trên thực tế đáp ứng được yêu cầu của người dùng.

- Tính phổ quát: Thuật toán được gọi là có tính phổ quát (phổ biến) nếu nó có thể giải quyết được một lớp các bài toán tương tự.

1.3. Biểu diễn thuật giải

- Cách 1: Mô tả các bước thực hiện của thuật giải
- Cách 2: Sử dụng sơ đồ giải thuật

Ví dụ: mô tả thuật toán giải phương trình bậc nhất ax+b=0 (a,b là các số thực)

Cách 1: Mô tả các bước thực hiện của thuật toán

Input: Hai số thực a, b

Output: nghiệm phương trình bậc nhất ax+b=0

Thuật toán:

Bước 1: Nhập 2 số thực a và b

Bước 2: Nếu a = 0 thì

2.1. Nếu b = 0 thì: Phương trình vô số nghiệm; kết thúc thuật toán;

2.2. Ngược lại: Phương trình vô nghiệm; kết thúc thuật toán;

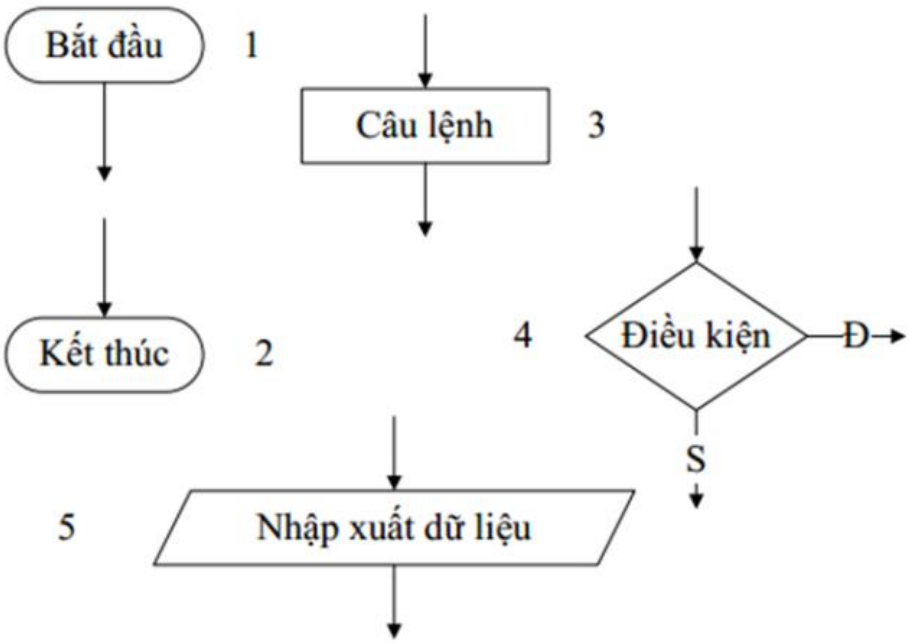
Ngược lại: Phương trình có nghiệm;

Giá trị của nghiệm đó là x = -b/a;

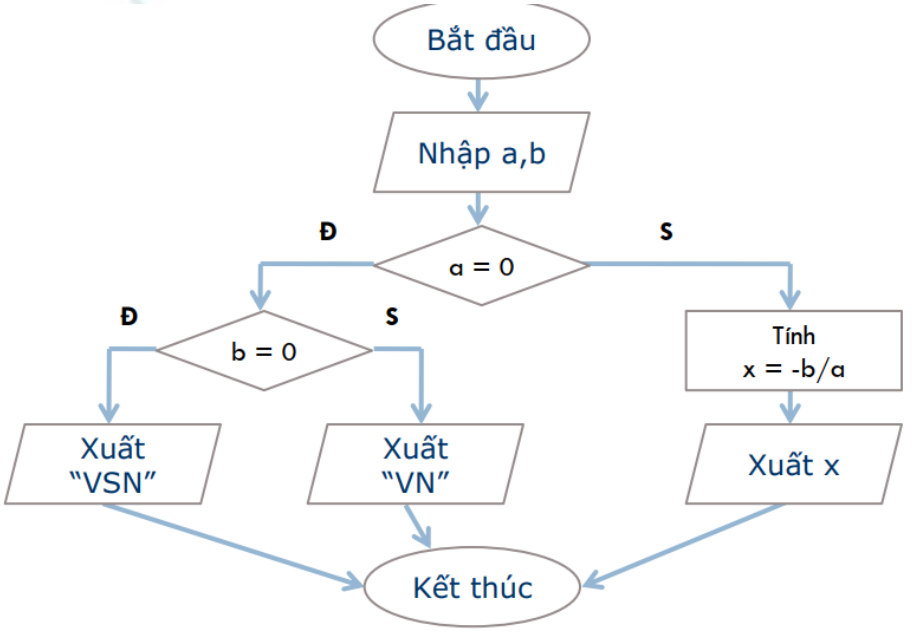
Kết thúc thuật toán;

Cách 2: Sử dụng sơ đồ giải thuật

Sử dụng các ký hiệu hình khối cơ bản để tạo thành một mô tả mang tính hình thức



Hình 3.1. ký hiệu hình khối cơ bản



Hình 3.2. Sơ đồ giải thuật

2. Đánh giá độ phức tạp của giải thuật

2.1. Các tiêu chuẩn đánh giá cấu trúc dữ liệu

Để đánh giá một cấu trúc dữ liệu chúng ta thường dựa vào một số tiêu chí sau:

- Cấu trúc dữ liệu phải tiết kiệm tài nguyên (bộ nhớ trong),
- Cấu trúc dữ liệu phải phản ảnh đúng thực tế của bài toán,

- Cấu trúc dữ liệu phải dễ dàng trong việc thao tác dữ liệu.

2.2. Đánh giá độ phức tạp của thuật giải

Việc đánh giá độ phức tạp của một thuật toán quả không dễ dàng chút nào. Ở dây, chúng ta chỉ muốn ước lượng thời gian thực hiện thuận toán T(n) để có thể có sự so sánh tương đối giữa các thuật toán với nhau. Trong thực tế, thời gian thực hiện một thuật toán còn phụ thuộc rất nhiều vào các điều kiện khác như cấu tạo của máy tính, dữ liệu đưa vào, …, ở đây chúng ta chỉ xem xét trên mức độ của lượng dữ liệu đưa vào ban đầu cho thuật toán thực hiện.

Để ước lượng thời gian thực hiện thuật toán chúng ta có thể xem xét thời gian thực hiện thuật toán trong hai trường hợp:

- Trong trường hợp tốt nhất: T_{min}
- Trong trường hợp xấu nhất: T_{max}

Từ đó chúng ta có thể ước lượng thời gian thực hiện trung bình của thuật toán: T_a

Để đánh giá hiệu quả của một thuật toán, có thể tính số lượng các phép tính phải thực hiện của thuật toán này:

- Phép so sánh
- Phép gán

- Thông thường số các phép tính được thực hiện phụ thuộc vào cỡ của bài toán,tức là độ lớn của đầu vào

- Vì thế độ phức tạp thuật toán là một hàm phụ thuộc đầu vào

- Tuy nhiên, không cần biết chính xác hàm này mà chỉ cần biết một ước lượng đủ tốt của chúng

- Để ước lượng độ phức tạp của một thuật toán ta thường dùng khái niệm Big-O

Bước 1. Gán Tổng = 0. Gán i = 0.

Bước 2.

- Tăng i thêm 1 đơn vị.
- Gán Tổng = Tổng + i

Bước 3. So sánh i với n

- Nếu i < n, quay lại bước 2.
- Ngược lại, dừng thuật toán.

```
int tong=0, i=0;
do{
        i++;
        tong+=i;
}while (i<n);
```

Số phép gán của thuật toán là bao nhiêu?

• Số phép so sánh là bao nhiêu?

- Gán: f(2n + 2), So sánh: f(n) → Độ phức tạp: O(n)

Ví dụ: Thuật toán tính tổng các số từ 1 đến n

s=0;

```
for(i= 1; i<=n; i++)
s=s+i;
```

Sử dụng quy tắc cộng: O(1)+O(n) → Độ phức tạp: O(n)

Ví dụ: Xác định độ phức tạp của thuật toán sau:

```
for (i= 1;i<=n;i++)
// lệnh1
for (j= 1;j<=m;j++)
// lệnh 2
```

→ Độ phức tạp: O(max(n,m))

Ví dụ: Xác định độ phức tạp của thuật toán sau:

```
for(i=1; i<=n; i++)
for (j=1; j<=n; j++)
//Lệnh thực hiện
```

Sử dụng quy tắc nhân: O(n)*O(n) → Độ phức tạp: O(n2)

2.3. Các độ phức tạp thường gặp

Độ phức tạp hằng số: O(1) – thời gian chạy không phụ thuộc vào độ lớn đầu vào

- Độ phức tạp tuyến tính: O(n) – thời gian chạy tỉ lệ thuận với độ lớn đầu vào
- Độ phức tạp logarit: O(logn)
- Độ phức tạp đa thức: O(P(n)), với P là đa thức có bậc từ 2 trở lên
- Độ phức tạp hàm mũ: O(2n)

Độ phức tạp	Thuật ngữ/tên phân lớp
O(1)	Độ phức tạp hằng số
O(logn)	Độ phức tạp logarit
O(n)	Độ phức tạp tuyến tính
O(nlogn)	Độ phức tạp nlogn
$O(n^a)$	Độ phức tạp đa thức
$O(a^n)$, a > 1	Độ phức tạp hàm mũ
O(n!)	Độ phức tạp giai thừa

$O(2^n)$, $n!$ } độ phức tạp cao $\Rightarrow$ khó chấp nhận

II. CÁC KIỂU DỮ LIỆU CƠ BẢN

1. Định nghĩa kiểu dữ liệu.

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

V (Value): tập các giá trị hợp lệ mà một đối tượng kiểu T có thể lưu trữ

O (Operation): tập các thao tác xử lý có thể thi hành trên đối tượng kiểu T.

Ví du: Giả sử có kiểu dữ liệu **mẫu tự** = < Vc , Oc > với

Vc = { a-z,A-Z}

Oc = { lấy mã ASCII của ký tự, biến đổi ký tự thường thành ký tự hoa…}

Giả sử có kiểu dữ liệu **số nguyên** = < Vi, Oi > với

Vi = { -32768..32767} Oi = { +, -, *, /, %}

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.

Các thuộc tính của 1 kiểu dữ liệu bao gồm:

* Tên kiểu dữ liệu
* Miền giá trị
* Kích thước lưu trữ
* Tập các toán tử tác động lên kiểu dữ liệu

2. Các kiểu dữ liệu cơ bản

Các loại dữ liệu cơ bản thường là các loại dữ liệu đơn giản, không có cấu trúc như số nguyên, số thực, các ký tự, các giá trị logic ... Các loại dữ liệu này, do tính thông dụng và đơn giản của mình, thường được các ngôn ngữ lập trình (NNLT) cấp cao xây dựng sẵn như một thành phần của ngôn ngữ để giảm nhẹ công việc cho người lập trình. Chính vì vậy đôi khi người ta còn gọi chúng là các kiểu dữ liệu định sẵn.

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

Tùy ngôn ngữ lập trình, các kiểu dữ liệu định nghĩa sẵn có thể khác nhau đôi chút. 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 C như là các giá trị nguyên khác zero và zero.

Các kiểu dữ liệu định sẵn trong C gồm các kiểu sau:

Tên kiểu	Kích thước	Miền giá trị	*Ghi chú*

Char	01 byte	-128 đến 127	Có thể dùng như số nguyên 1 byte có dấu hoặc kiểu ký tự
Unsign char	01 byte	0 đến 255	Số nguyên 1 byte không dấu
Int	02 byte	-32738 đến 32767	Số nguyên 2 byte
Unsign int	02 byte	0 đến 65535	Có thể gọi tắt là unsign
Long	04 byte	-2^{32} đến 2^{31} -1	
Unsign long	04 byte	0 đến 2^{32}-1	
Float	04 byte	3.4E-38 đến 3.4E38	Giới hạn chỉ trị tuyệt đối.Các giá trị < 3.4E-38 được coi = 0. Tuy nhiên kiểu float chỉ có 7 chữ số có nghĩa.
Double	08 byte	1.7E-308 đến 1.7E308	
Long double	10 byte	3.4E-4932 đến 1.1E4932	

Như vậy, trong C xét cho cùng chỉ có 2 loại dữ liệu cơ bản là số nguyên và số thực. Tức là chỉ có dữ liệu số. Hơn nữa các số nguyên trong C có thể được thể hiện trong 3 hệ cơ số là hệ thập phân, hệ thập lục phân và hệ bát phân.

Các kiểu cơ sở rất đơn giản và không thể hiện rõ sự tổ chức dữ liệu trong một cấu trúc, thường chỉ được sử dụng làm nền để xây dựng các kiểu dữ liệu phức tạp khác.

III. CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG

1. Khái niệm trừu tượng hóa

Trong tin học, trừu tượng hóa nghĩa là đơn giản hóa, làm cho nó sáng sủa hơn và dễ hiểu hơn. Cụ thể trừu tượng hóa là che đi những chi tiết, làm nổi bật cái tổng thể. Trừu tượng hóa có thể thực hiện trên hai khía cạnh là trừu tượng hóa dữ liệu và trừu tượng hóa chương trình.

2. Trừu tượng hóa chương trình

Trừu tượng hóa chương trình là sự định nghĩa các chương trình con để tạo ra các phép toán trừu tượng (sự tổng quát hóa của các phép toán nguyên thủy). Chẳng hạn ta có thể tạo ra một chương trình con Matrix_Mult để thực hiện phép toán nhân hai ma trận. Sau khi Matrix_mult đã được tạo ra, ta có thể dùng nó như một phép toán nguyên thủy (chẳng hạn phép cộng hai số).

Trừu tượng hóa chương trình cho phép phân chia chương trình thành các chương trình con. Sự phân chia này sẽ che dấu tất cả các lệnh cài đặt chi tiết trong các chương trình con.

Ở cấp độ chương trình chính, ta chỉ thấy lời gọi các chương trình con và điều này được gọi là sự bao gói.

Ví dụ như một chương trình quản lý sinh viên được viết bằng trừu tượng hóa có thể là:

```
void Main() {
Nhap( Lop);
Xu_ly (Lop);
Xuat (Lop);
}
```

Trong chương trình trên, Nhap, Xu_ly, Xuat là các phép toán trừu tượng. Chúng che dấu bên trong rất nhiều lệnh phức tạp mà ở cấp độ chương trình chính ta không nhìn thấy được.

Còn Lop là một biến thuộc kiểu dữ liệu trừu tượng mà ta sẽ xét sau.

3. Trừu tượng hóa dữ liệu

Trừu tượng hóa dữ liệu là định nghĩa các kiểu dữ liệu trừu tượng.

Một kiểu dữ liệu trừu tượng là một mô hình toán học cùng với một tập hợp các phép toán (operator) trừu tượng được định nghĩa trên mô hình đó. Ví dụ tập hợp số nguyên cùng với các phép toán hợp, giao, hiệu là một kiểu dữ liệu trừu tượng.

Trong một kiểu dữ liệu trừu tượng các phép toán có thể thực hiện trên các đối tượng (toán hạng) không chỉ thuộc kiểu dữ liệu trừu tượng đó, cũng như kết quả không nhất thiết phải thuộc kiểu dữ liệu trừu tượng. Tuy nhiên phải có ít nhất một toán hạng hoặc kết quả phải thuộc kiểu dữ liệu trừu tượng đang xét.

Kiểu dữ liệu trừu tượng là sự tổng quát hoá của các kiểu dữ liệu nguyên thuỷ.

Để minh hoạ ta có thể xét bản phác thảo cuối cùng của thủ tục GREEDY. Ta đã dùng một danh sách (LIST) các số nguyên và các phép toán trên danh sách newclr là:

- Tạo một danh sách rỗng.
- Lấy phần tử đầu tiên trong danh sách và trả về giá trị null nếu danh sách rỗng.
- Lấy phần tử kế tiếp trong danh sách và trả về giá trị null nếu không còn phần tử kế tiếp.
- Thêm một số nguyên vào danh sách.

Nếu chúng ta viết các chương trình con thực hiện các phép toán này, thì ta dễ dàng thay các mệnh đề hình thức trong giải thuật bằng các câu lệnh đơn giản

Câu lệnh	Mệnh đề hình thức
MAKENULL(newclr)	newclr= ∅
w=FIRST(newclr)	w=phần tử đầu tiên trong newclr
w=NEXT(w,newclr)	w=phần tử kế tiếp trong newclr
INSERT(v,newclr)	Thêm v vào newclr

Điều này cho thấy sự thuận lợi của kiểu dữ liệu trừu tượng, đó là ta có thể định nghĩa một kiểu dữ liệu tuỳ ý cùng với các phép toán cần thiết trên nó rồi chúng ta dùng như là các đối tượng nguyên thuỷ. Hơn nữa chúng ta có thể cài đặt một kiểu dữ liệu trừu tượng bằng bất kỳ cách nào, chương trình dùng chúng cũng không thay đổi, chỉ có các chương trình con biểu diễn cho các phép toán của kiểu dữ liệu trừu tượng là thay đổi.

Cài đặt kiểu dữ liệu trừu tượng là sự thể hiện các phép toán mong muốn (các phép toán trừu tượng) thành các câu lệnh của ngôn ngữ lập trình, bao gồm các khai báo thích hợp và các thủ tục thực hiện các phép toán trừu tượng. Để cài đặt ta chọn một cấu trúc dữ liệu thích hợp có trong ngôn ngữ lập trình hoặc là một cấu trúc dữ liệu phức hợp được xây dựng lên từ các kiểu dữ liệu cơ bản của ngôn ngữ lập trình.

IV. CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

Tuy nhiên trong nhiều trường hợp, chỉ với các kiểu dữ liệu cơ sở không đủ để phản ánh tự nhiên và đầy đủ bản chất của sự vật thực tế, dẫn đến nhu cầu phải xây dựng các kiểu dữ liệu mới dựa trên việc tổ chức, liên kết các thành phần dữ liệu có kiểu dữ liệu đã được định nghĩa. Những kiểu dữ liệu được xây dựng như thế gọi là kiểu dữ liệu có cấu trúc.

Một số kiểu dữ liệu có cấu trúc cơ bản :

1. Kiểu chuỗi ký tự

Chuỗi ký tự là một trong các kiểu dữ liệu có cấu trúc đơn giản nhất và thường các ngôn ngữ lập trình đều định nghĩa nó như một kiểu cơ bản. Do tính thông dụng của kiểu chuỗi ký tự các ngôn ngữ lập trình luôn cung cấp sẵn một bộ các hàm thư viện các xử lý trên kiểu dữ liệu này. Các hàm này được đặt trong thư viện **string.lib** của C.

Chuỗi ký tự trong C được cấu trúc như một chuỗi liên tiếp các ký tự kết thúc bằng ký tự \0 có mã ASCII bằng 0 (NULL character). Như vậy, giới hạn chiều dài của một chuỗi ký tự trong C là 1 Segment (tối đa chứa 65535 ký tự), ký tự đầu tiên được đánh số là ký tự thứ 0.

Ta có thể khai báo một chuỗi ký tự theo một số cách sau đây:

```
char S[10]; //Khai báo một chuỗi ký tự S có chiều dài
// tối đa 10 (kể cả kí tự kết thúc)
char S[]="ABC"; // Khai báo một chuỗi ký tự S có chiều
// dài bằng chiều dài của chuỗi "ABC"
// và giá trị khởi đầu của S là "ABC"
char *S ="ABC"; //Giống cách khai báo trên.
```

Trong ví dụ trên ta cũng thấy được một hằng chuỗi ký tự được thể hiện bằng một chuỗi ký tự đặt trong cặp ngoặc kép “”.

Các thao tác trên chuỗi ký tự rất đa dạng. Sau đây là một số thao tác thông dụng:

- So sánh 2 chuỗi: **strcmp**
- Sao chép chuỗi: **strcpy**
- Độ dài chuỗi: **strlen**
- Kiểm tra 1 chuỗi nằm trong chuỗi kia: **strstr**
- Cắt 1 từ ra khỏi 1 chuỗi: **strtok**
- Đổi 1 số ra chuỗi: **itoa**
- Đổi 1 chuỗi ra số: **atoi, atof, ...**
- Nhập một chuỗi: **gets**
- Xuất một chuỗi: **puts**

2. Kiểu mảng

Kiểu dữ liệu mảng là kiểu dữ liệu trong đó mỗi phần tử của nó là một tập hợp có thứ tự các giá trị có cùng cấu trúc được lưu trữ liên tiếp nhau trong bộ nhớ. Mảng có thể một chiều hay nhiều chiều. Một dãy số chính là hình tượng của mảng 1 chiều, ma trận là hình tượng của mảng 2 chiều.

Một điều đáng lưu ý là mảng 2 chiều có thể coi là mảng một chiều trong đó mỗi phần tử của nó là 1 mảng một chiều. Tương tự như vậy, một mảng n chiều có thể coi là mảng 1 chiều trong đó mỗi phần tử là 1 mảng n-1 chiều.

Mảng 1 chiều được khai báo như sau:

<Kiểu dữ liệu> <Tên biến>[<Số phần tử>];

Ví dụ: để khai báo một biến có tên a là một mảng nguyên 1 chiều có tối đa 100 phần tử ta phải khai báo như sau:

```
int a[100];
```

Ta cũng có thể vừa khai báo vừa gán giá trị khởi động cho một mảng như sau:

```
int a[5] = {1, 7, -3, 8, 19};
```

Trong trường hợp này C cho phép ta khai báo một cách tiện lợi hơn

```
int a[] = {1, 7, -3, 8, 19};
```

Như ta thấy, ta không cần chỉ ra số lượng phần tử cụ thể trong khai báo. Trình biên dịch của C sẽ tự động làm việc này cho chúng ta.

Tương tự ta có thể khai báo một mảng 2 chiều hay nhiều chiều theo cú pháp sau:

<Kiểu dữ liệu> <Tên biến>[<Số phần tử1>][<Số phần tử2>]...;

Ví dụ, ta có thể khai báo:

```
int a[100][150];
```

hay

```
int a[][]={{1, 7, -3, 8, 19},
           {4, 5, 2, 8, 9},
           {21, -7, 45, -3, 4}};
```

(mảng a sẽ có kích thước là 3x5).

3. Kiểu Union

Kiểu union là một dạng cấu trúc dữ liệu đặc biệt của ngôn ngữ C. Nó rất giống với kiểu struct. Chỉ khác một điều, trong kiểu union, các trường được phép dùng chung một vùng nhớ. Hay nói cách khác, cùng một vùng nhớ ta có thể truy xuất dưới các dạng khác nhau.

Khai báo tổng quát của kiểu union như sau:

```
typedef union <tên kiểu union>
{
    <KDL> <tên trường>;
    <KDL> <tên trường>;
    ……
}[<Name>];
```

Ví dụ, ta có thể định nghĩa kiểu số sau:

```
typedef union tagNumber
{
    int i;
    long l;
}Number;
```

Việc truy xuất đến một trường trong union được thực hiện hoàn toàn giống như trong struct. Giả sử có biến n kiểu Number. Khi đó, n.i cho ta một số kiểu int còn n.l cho ta một số kiểu long, nhưng cả hai đều dùng chung một vùng nhớ. Vì vậy, khi ta gán

```
n.l = 0xfd03;
```

thì giá trị của n.i cũng bị thay đổi (n.i sẽ bằng 3);

Việc dùng kiểu union rất có lợi khi cần khai báo các cấu trúc dữ liệu mà nội dung của nó thay đổi tùy trạng thái. Ví dụ để mô tả các thông tin về một con người ta có thể khai báo một kiểu dữ liệu như sau:

```
struct tag Nguoi
{
        char HoTen[35];
        int NamSinh;
        char NoiSinh[40];
        char GioiTinh; //0: Nữ, 1: Nam
        char DiaChi[50];
        char Ttgd; //0:Không có gia đình, 1: Có gia đình
        union {
                char tenVo[35];
                char tenChong[35];
                }
}Nguoi;
```

Tùy theo người mà ta đang xét là nam hay nữ ta sẽ truy xuất thông tin qua trường có tên tenVo hay tenChong.

4. Kiểu mẫu tin (cấu trúc)

Nếu kiểu dữ liệu mảng là kiểu dữ liệu trong đó mỗi phần tử của nó là một tập hợp có thứ tự các giá trị có cùng cấu trúc được lưu trữ liên tiếp nhau trong bộ nhớ thì mẫu tin là kiểu dữ liệu mà trong đó mỗi phần tử của nó là tập hợp các giá trị ***có thể khác cấu trúc***. Kiểu mẫu tin cho phép chúng ta mô tả các đối tượng có cấu trúc phức tạp.

Khai báo tổng quát của kiểu struct như sau:

```
typedef struct <tên kiểu struct>
{
        <KDL> <tên trường>;
        <KDL> <tên trường>;
        …
}[<Name>];
```

Ví dụ để mô tả các thông tin về một con người ta có thể khai báo một kiểu dữ liệu như sau:

```
struct tag Nguoi
{
```

```
        char HoTen[35];
        int NamSinh;
        char NoiSinh[40];
        char GioiTinh; //0: Nữ, 1: Nam
        char DiaChi[50];
        char Ttgd; //0: Không có gia đình, 1: Có gia đình
    }
    Nguoi;
```

Kiểu mẫu tin bổ sung những thiếu sót của kiểu mảng, giúp ta có khả năng thể hiện các đối tượng đa dạng của thể giới thực vào trong máy tính một cách dễ dàng và chính xác hơn.

4.1. Các thao tác trên biến cấu trúc:

- ***Truy xuất đến 1 thành phần của biến cấu trúc:***

Tên biến.tên thành phần

Trong C, địa chỉ của một thành phần trong biến cấu trúc được xác định bởi phép toán lấy địa chỉ: **&tên biến.tên thành phần**

- ***Gán 2 biến cấu trúc cho nhau***

4.2. Hàm và kiểu mẫu tin:

* Đối của hàm có thể là:

- Biến mẫu tin: khi đó tham số thực tương ứng là một giá trị mẫu tin
- Tham chiếu mẫu tin: khi đó tham số thực tương ứng là một giá trị mẫu tin
- Con trỏ mẫu tin: khi đó tham số thực là địa chỉ của biến cấu trúc.

* Hàm có thể trả về:

- Giá trị mẫu tin: nguoi tênhàm(...)
- Con trỏ mẫu tin: nguoi *tênhàm(....)

Ví dụ: Nhập và in danh sách thí sinh theo thứ tự tên và họ

```
# include <stdio.h>
# include <conio.h>
# include <string.h>
# include <stdlib.h>
typedef struct
    {   int ngay, thang, nam;
    } date;
typedef struct
    {   int sbd;
        char ho[25],ten[7];
        date ngaysinh;
```

```
            float toan,ly,hoa;
            int phongthi;
        }hoso;
hoso thisinh[100];
int n;
void NhapHoso (hoso ts[], int &n)
{       int i=0;
        hoso hs;
        printf ("\nNhap Ho so thi sinh \"Ho bang rong de ket thuc\"");
        do
        {
            hs.sbd = n+1;
            printf ("\nNhap ho so cho thi sinh: %3d",hs.sbd);
            printf ("\nHo : "); gets(hs.ho);
            if (hs.ho[0]=='\0') break;
            printf ("Ten: "); gets(hs.ten);
            printf ("Ngay sinh: ");
            scanf ("%d/%d/%d%*c",&hs.ngaysinh.ngay,&hs.ngaysinh.thang,
                    &hs.ngaysinh.nam);
            printf ("Diem Toan: "); scanf("%f%*c",&hs.toan);
            printf ("Diem Ly: "); scanf("%f%*c",&hs.ly);
            printf ("Diem Hoa: "); scanf("%f%*c",&hs.hoa);
            printf ("Phong thi: "); scanf("%d%*c",&hs.phongthi);
            ts[i] = hs;
            n++; i++;
        } while (i<100);
}
int sosanh (const void *p, const void *q)
{
        int kq;
        kq = strcmp(((hoso*)p)->ten,((hoso*)q)->ten);
        if (kq == 0) return strcmp(((hoso*)p)->ho,((hoso*)q)->ho);
        return kq;
}
void InKQ (hoso ts[], int n)
{
        int i;
```

```
        qsort (ts,n,sizeof(ts[0]),sosanh);
      printf ("\n%-4s %-25s %-10s %-4s %-4s %-4s %-5s","SBD", "Ho Ten","Ngay
                sinh","Toan","Ly","Hoa","Phong");
        for (i=0;i<n;i++)
      printf ("\n%4d %-18s %-7s %2d/%2d/%2d %4.1f %4.1f %4.1f %3d",
        ts[i].sbd,ts[i].ho,ts[i].ten,ts[i].ngaysinh.ngay,
                    ts[i].ngaysinh.thang,ts[i].ngaysinh.nam,ts[i].toan,
                    ts[i].ly, ts[i].hoa, ts[i].phongthi);
        getch ();
}
hoso Timhs(int sbd, hoso ts[], int n)
{
int i; hoso hs;
hs.sbd = hs.ngaysinh.ngay= hs.ngaysinh.thang= hs.ngaysinh.nam=0;
hs.ho[0]= hs.ten[0] = 0;
for (i=0; i<n; i++)
        if ( sbd == ts[i].sbd) return ts[i];
return hs;
}
hoso *pTimhs (int sbd, hoso ts[], int n)
{
int i;
for (i=0; i<n; i++)
        if (sbd==ts[i].sbd) return(&ts[i]);
return (NULL);
}
void main()
{       clrscr ();
        NhapHoso (thisinh,n);
        InKQ (thisinh,n);
}
```

5. Kiểu con trỏ

5.1. Giới thiệu

Các biến chúng ta đã biết và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là biến tĩnh. Khi khai báo biến tĩnh, một lượng ô nhớ cho các biến này sẽ được cấp phát mà không cần biết trong quá trình thực

thi chương trình có sử dụng hết lượng ô nhớ này hay không. Mặt khác, các biến tĩnh dạng này sẽ tồn tại trong suốt thời gian thực thi chương trình dù có những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ.

Một số hạn chế có thể gặp phải khi sử dụng các biến tĩnh:

- Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
- Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

Để tránh những hạn chế trên, ngôn ngữ C cung cấp cho ta một loại biến đặc biệt gọi là biến động với các đặc điểm sau:

- Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
- Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
- Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên các biến động không có địa chỉ nhất định nên ta không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp cho ta một loại biến đặc biệt nữa để khắc phục tình trạng này, đó là biến con trỏ (pointer) với các đặc điểm:

- Biến con trỏ không chứa dữ liệu mà chỉ chứa địa chỉ của dữ liệu hay chứa địa chỉ của ô nhớ chứa dữ liệu.
- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

5.2. Khai báo và sử dụng biến con trỏ

5.1.1. Khai báo biến con trỏ

Cú pháp: <Kiểu> * <Tên con trỏ>

Ý nghĩa: Khai báo một biến có tên là ***Tên con trỏ*** dùng để chứa địa chỉ của các biến có kiểu ***Kiểu.***

Ví dụ 1: Khai báo 2 biến a,b có kiểu int và 2 biến pa, pb là 2 biến con trỏ kiểu int.

```
int a, b, *pa, *pb;
```

Ví dụ 2: Khai báo biến f kiểu float và biến pf là con trỏ float

```
float f, *pf;
```

Ghi chú: Nếu chưa muốn khai báo kiểu dữ liệu mà con trỏ ptr đang chỉ đến, ta sử dụng:

```
void *ptr;
```

Sau đó, nếu ta muốn con trỏ ptr chỉ đến kiểu dữ liệu gì cũng được. Tác dụng của khai báo này là chỉ dành ra 2 bytes trong bộ nhớ để cấp phát cho biến con trỏ ptr.

5.1.2. Các thao tác trên con trỏ

a/ Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

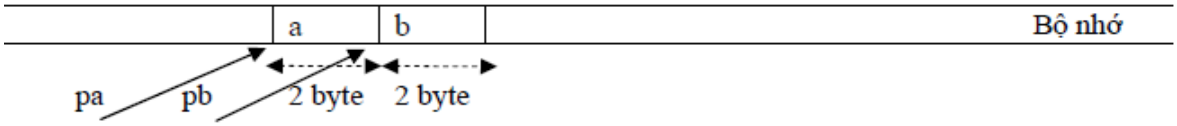
Cú pháp: <Tên biến con trỏ> = & <Tên biến>

Giải thích: Ta gán địa chỉ của biến ***Tên biến*** cho con trỏ ***Tên biến con trỏ***.

Ví dụ: Gán địa chỉ của biến a cho con trỏ pa, gán địa chỉ của biến b cho con trỏ pb.

pa=&a; pb=&b;

Lúc này, hình ảnh của các biến trong bộ nhớ được mô tả:



Lưu ý:

Khi gán địa chỉ của biến tĩnh cho con trỏ cần phải lưu ý kiểu dữ liệu của chúng. Ví dụ sau đây không đúng do không tương thích kiểu:

```
int Bien_Nguyen;
float *Con_Tro_Thuc;
...
Con_Tro_Thuc=&Bien_Nguyen;
```

Phép gán ở đây là sai vì Con_Tro_Thuc là một con trỏ kiểu float (nó chỉ có thể chứa được địa chỉ của biến kiểu float); trong khi đó, Bien_Nguyen có kiểu int.

b/ Nội dung của ô nhớ con trỏ chỉ tới

Để truy cập đến nội dung của ô nhớ mà con trỏ chỉ tới, ta sử dụng cú pháp:

***<Tên biến con trỏ>**

Với cách truy cập này thì ***<Tên biến con trỏ>** có thể coi là một biến có kiểu được mô tả trong phần khai báo biến con trỏ.

Ví dụ: Ví dụ sau đây cho phép khai báo, gán địa chỉ cũng như lấy nội dung vùng nhớ của biến con trỏ:

```
int x=100;
int *ptr;
ptr=&x;
int y= *ptr;
```

Lưu ý: Khi gán địa chỉ của một biến cho một biến con trỏ, mọi sự thay đổi trên nội dung ô nhớ con trỏ chỉ tới sẽ làm giá trị của biến thay đổi theo (thực chất nội dung ô nhớ và biến chỉ là một).

Ví dụ: Đoạn chương trình sau thấy rõ sự thay đổi này :

```
# include <stdio.h>
# include <conio.h>
```