

Tin học cho lớp 11 chuyên Tin
Năm học 2009-2010
Phần II. NHẬP MÔN LẬP TRÌNH THEO NGÔN NGỮ C

Một số chú ý trước khi vào bài 4:

1. Hàm chuyển đổi kiểu

Có hiệu lực khi có bao hàm: `#include <ctype.h>` và `#include <stdlib.h>`

`atoi(<xâu kí tự số nguyên>)`

biến <xâu kí tự số nguyên> đó thành số kiểu nguyên (2B),

ví dụ: "123" -> 123

`atol(<xâu kí tự số nguyên dài>)`

biến <xâu kí tự số nguyên dài> thành số kiểu nguyên dài (4B),

ví dụ: "123456789" -> 123456789

`atof(<xâu kí tự số thực>)`

biến xâu kí tự số thành số kiểu thực độ chính xác cao **double** (8B),

ví dụ: "1.23" -> 1.23

`toupper(<kí tự>)`

biến chữ cái in thường thành in hoa, các kí tự khác không đổi.

`tolower(<kí tự>)`

biến chữ cái in hoa thành in thường, các kí tự khác không đổi.

Nhớ rằng **float** (4B) là kiểu thực độ chính xác không cao, giá trị tuyệt đối của nó thuộc khoảng $3.4 \cdot 10^{-38}$ đến $3.4 \cdot 10^{38}$ mà thôi, những số dưới khoảng đó được cho là bằng 0.0; còn kiểu **double** (8B) thì từ $1.7 \cdot 10^{-308}$ đến $1.7 \cdot 10^{308}$, nên gọi là số thực độ chính xác kép

Ví dụ: Dùng hàm chuyển đổi kiểu để nhập dữ liệu.

```
int n; float x; char chon, s[10];
printf("Nhap n = "); gets(s); n=atoi(s);
printf("  Nhap x = "); gets(s); x=atof(s);
printf("Ban co muon tiep tuc?-(C/K) ");
chon=toupper(getch());
if chon='K' exit;
else <công việc khác>;
```

2. Hàm toán học

a - Các hàm sau đây cần phải có bao hàm: `#include <stdlib.h>`

`abs(<số nguyên>)` cho giá trị tuyệt đối của <số nguyên> đó.

`randomize()` khởi động bộ sinh số ngẫu nhiên.

`random(<số nguyên>)` cho số ngẫu nhiên trong khoảng từ 0 đến <số nguyên đó>-1.

b - Các hàm sau đây cần phải có bao hàm: `#include <math.h>`, giả sử x, y là các số thực độ chính xác cao **double**:

`sin(x)`, `cos(x)`, `tan(x)`, ở đây x đo bằng radian.

`abs(x)` cho giá trị tuyệt đối của x.

`exp(x)` cho e^x

`log(x)` cho logarit cơ số $e = 2.71828...$ tức là bằng $\ln(x)$.

`pow(x,y)` cho x^y .

`sqrt(x)` cho căn bậc hai của x, tức là bằng \sqrt{x}

`floor(x)` cho số nguyên lớn nhất nhỏ hơn hoặc bằng x

`ceil(x)` cho số nguyên nhỏ nhất lớn hơn hoặc bằng x.

3. Sơ lược về Bộ nhớ

Bộ nhớ là dãy các ô nhớ cỡ 1 Byte, tức là 8 bits, đánh số từ 0 đến hết... Đó là cách đánh **địa chỉ vật lý**. Cách đánh địa chỉ như vậy rất khó quản lý (truy nhập), giống như số nhà của một thành phố mà đánh số như vậy thì khó tìm đến được... Do vậy, người ta dùng cách đánh **địa chỉ logic**, quy ước như sau: cá ô nhớ từ 0 đến 65535 gọi là đoạn 0, từ ô nhớ 16 đến 65535+16 gọi là đoạn 1,..., từ ô nhớ 16k đến ô nhớ 65535+16k gọi là 65535+16k gọi là đoạn k. Trong tin học đoạn còn gọi là **segment**, trong mỗi đoạn, thì các ô nhớ đánh từ 0 đến 65535, còn gọi là **offset**, là độ lệch so với đầu đoạn. Cặp segment và offset là địa chỉ logic của một ô nhớ cụ thể. Dễ thấy, các đoạn không rời nhau, mà kể từ đầu, cứ sau 16 ô nhớ, thì lại bắt đầu một đoạn mới. Mỗi đoạn có đúng 65536 Byte.

Đoạn thứ nhất DOS thường dùng 640KB cho một chương trình ứng dụng: Lấy làm đoạn chứa các mã nhị phân các câu lệnh, được gọi là **code segment**. Nếu đoạn này dùng không nhiều ô nhớ, thì kể từ ô nhớ 16k gần nhất còn rỗi sẽ bắt đầu đoạn dữ liệu (**data segment**), chứa các khai báo của chương trình chính. Rồi sau đó đến đoạn ngăn xếp để chứa các dữ liệu hay mã lệnh của các chương trình con (hàm hay thủ tục) khi được gọi tới, thực hiện xong thì giải phóng ngay: Cái nào gói sau thì giải phóng trước (LIFO), vì vậy gọi là đoạn ngăn xếp (**data segment**). Ba đoạn đó chiếm $\leq 3 \times 64\text{KB}$, còn lại hơn $7 \times 64\text{KB}$ sẽ giành cho vùng nhớ động (Heap). Tuy nhiên, sau này ta sẽ thấy C/C++ có nhiều cách tổ chức khác!

Trước mắt ta thấy rằng C/C++ đã sớm sử dụng bộ nhớ trong lập trình. Đó cũng là một lý do để nói rằng: ngôn ngữ C/C++ rất ưu việt vì tạo ra chương trình chạy khá nhanh, hơn hẳn Pascal.

Bài 4. Con trỏ và mảng

4. Con trỏ

Cú pháp:

<kiểu> *<tên con trỏ>;

Ví dụ:

float *p1, *p2, x;

Chú ý:

Có thể gán địa chỉ của một biến cho con trỏ, hoặc các con trỏ cho nhau. Biến con trỏ chiếm 4 Bytes. Nhưng khi viết giá trị của một con trỏ ra màn hình thì nó chỉ viết bằng 4 chữ số HEX. Đó là địa chỉ offset, mà không viết phần segment. Hôm nay, ta hãy chỉ làm quen với địa chỉ này!

Ví dụ:

p1=&x;

Có thể viết địa chỉ **p1** ra màn hình như ví dụ sau:

printf("Giá trị của p1=%p,", p1); sẽ cho 4 chữ số HEX in hoa.

printf("Giá trị của p1=%X,", p1); sẽ cho các chữ số HEX in hoa.

printf("Giá trị của p1=%x", p1); sẽ cho các chữ số HEX in thường.

Giả sử địa chỉ = \$1a3f:3ad thì nó sẽ viết ra địa chỉ offset.

Khi đó, kết quả 3 câu lệnh trên sẽ là: 03AD, 3AD, 3ad

Nhờ khai báo kiểu ở trên. TC cho phép cộng hay trừ các con trỏ với một số nguyên. Tuy nhiên **số nguyên này là số bước nhảy của con trỏ** chứ không phải là đơn vị thông thường.

float *p;

Giả sử **p=AB10**, thì **p+1** sẽ cho kết quả **AB14**, vì một bước nhảy của nó sẽ qua một số thực có độ dài là 4 byte. Tương tự **p-1** sẽ cho kết quả là **AB06**.

5. Mảng một chiều

Cú pháp: **<kiểu> <tên mảng>[số lượng max các phần tử];**

và chỉ số của mảng sẽ được tính từ 0 trở đi đến số lượng-1.

Tuy nhiên nó cũng có thể khai báo cùng với các giá trị khởi tạo:

```
float a[5]={1.2,2.3,3.4,4.5,5.6}    bình thường.  
float a[5]={1.2,2.3}                các phần tử đầu đã được xác định.  
float a[]={1.2,2.3,3.4}            cỡ của mảng đã được xác định, máy tự đếm.
```

Chú ý:

Tên mảng lại chính là một con trỏ giữ địa chỉ của mảng, và cũng bằng địa chỉ của phần tử đầu tiên của mảng. $a = &a[0]$. Nó có quan hệ bình đẳng với các con trỏ.

Nhờ khai báo kiểu hai địa chỉ của hai phần tử liên tiếp cách nhau một đoạn bằng cỡ của kiểu đã được khai báo, nên muốn truy nhập tới phần tử thứ i của mảng, ngoài cách thông thường như trong Pascal, ta còn có thể dùng con trỏ: $*(a+i)$. Tức là ta có đẳng thức: $a[i]=*(a+i)$. Cộng i ở đây không phải phép cộng thông thường, mà là con trỏ a nhảy đi i bước, nhảy qua i lần cỡ của phần tử của mảng!

Sau đây là một ví dụ đơn giản về một mảng 5 số nguyên:

```
int a[5];  
long int s;  
char i,n;  
printf("Nhap 5 phan tu cua mang:\n");  
for (i=0;i<5;i++)  
{  
    printf("a[%d]="); scanf("%d",&a[i]);  
}  
s=0;  
for (i=0;i<5;i++) s+=a[i];  
printf("Tong cac phan tu cua mang = %ld",s);  
getch();
```

6. Xâu kí tự

Xâu kí tự là mảng một chiều các kí tự (kiểu char) mà kết thúc là kí tự 0 (NULL).

Khai báo:

```
char <tên_xâu>[Số_lượng_max_các_kí_tự];
```

Ví dụ:

```
char s[50];
```

Với cách làm này, một lần nữa ta thấy TC hơn hẳn TP về độ dài chuỗi kí tự: Trong TP chuỗi dài nhất là 255 kí tự, còn trong TC chuỗi dài tùy ý vì chuỗi chỉ kết thúc khi gặp kí tự 0, hơn cả cỡ của `long int` ấy chứ!

Tuy nhiên, vì chuỗi kí tự là mảng một chiều, nên tên chuỗi là tên mảng và là một con trỏ. Do vậy ta không thể gán một hằng chuỗi cho biến chuỗi `s="HA NOI"`; vì `s` là một địa chỉ!

Chú ý:

Xin nhắc lại là hằng chuỗi kí tự được xác định nhờ cặp dấu nháy kép "...", còn hằng kí tự xác định bởi cặp dấu nháy đơn '.'. Như vậy cần phân biệt 'A' và "A". Chúng là kí tự A và chuỗi kí tự A, cái sau cỡ lớn hơn vì phải kết thúc bằng kí tự 0. Có thể có chuỗi kí tự rỗng (empty), chứ không thể có kí tự rỗng...

Cũng như mảng một chiều, ta cũng có thể khởi tạo một chuỗi kí tự như sau:

```
char s[]="Bui The Tam";    không định kích thước của chuỗi, máy tự thêm kí tự 0.  
char s[40]="Ha Noi";      có định rõ kích thước của chuỗi, máy tự cho kí tự 0.
```

```
char s[25]={'H','A','N','O','I','\0'};
```

Giả sử khai báo:

```
char X[50],Y[30],K; int v,n;
```

Để nhập xâu kí tự ta thường dùng lệnh:

```
gets(X); hoặc scanf("%s",X); không phải viết & vì bản thân X là địa chỉ rồi.
```

Để xuất xâu kí tự ta thường dùng lệnh:

```
puts(X); hoặc printf("%s",X); (Mặc dù X là con trỏ!)
```

Để xuất kí tự ta thường dùng lệnh:

```
printf("%c",K);
```

Khi khai báo như trên thì ta có thể khai thác các hàm sau:

Có hiệu lực khi có bao hàm: `#include <string.h>`.

<code>strlen(X);</code>	độ dài xâu kí tự
<code>strupr(X);</code>	đổi chữ thường sang hoa.
<code>strlwr(X);</code>	đổi chữ hoa sang thường.
<code>strcat(X,Y);</code>	như $X:=X+Y$ của TP
<code>strcmp(X,Y);</code>	so sánh hai xâu cho kết quả âm, 0 hoặc dương và có phân biệt chữ hoa/thường, ví dụ: <code>strcmp("Anh","anh")</code> cho kết quả là một số =0.
<code>strncmpi(X,Y);</code>	so sánh hai xâu cho kết quả âm, 0 hoặc dương, mà không phân biệt chữ hoa/thường. ví dụ: <code>strncmpi("Anh","anh")</code> cho kết quả là một số <0.
<code>strcpy(X,Y);</code>	như $X:=Y$; của TP.
<code>strncpy(X,Y,n);</code>	như $X:=copy(Y,1,n)$;
<code>strnset(X,K,n);</code>	như $X:=''$; for $i:=1$ to n $X:=X+K$; của TP.
<code>strstr(X,Y);</code>	như <code>pos(Y,X)</code> ; của TP;
<code>strrev(X);</code>	đảo ngược xâu X, hàm cho địa chỉ của xâu kết quả.

7. Các hàm kiểm tra kí tự

Có hiệu lực khi có bao hàm: `#include <ctype.h>`

<code>isalpha(K);</code>	
<code>isdigit(K);</code>	
<code>islower(K);</code>	
<code>isupper(K);</code>	
<code>isspace(K);</code>	xem có là dấu cách, xuống dòng hay tab không.

8. Bài tập thực hành:

Nhập dãy số gồm 10 phần tử, rồi in ra màn hình:

1. Trung bình cộng các phần tử của dãy.
2. Tổng các số hạng dương
3. Số hạng âm đầu tiên và chỉ số của nó.
4. Số hạng lớn nhất và chỉ số của nó.
5. Các số hạng lớn nhất và chỉ số của chúng.
6. Đảo ngược lại thứ tự của dãy số.
7. Xét xem dãy có phải là cấp số cộng không?
8. Tìm tích các số hạng âm của dãy.