

KỸ THUẬT LẬP TRÌNH C/C++

Mảng và con trỏ

Thi-Lan Le

Thi-Lan.Le@mica.edu.vn; lan.lethi1@hust.edu.vn

[Webpage: http://www.mica.edu.vn/perso/Le-Thi-Lan](http://www.mica.edu.vn/perso/Le-Thi-Lan)

Con trỏ

- ◆ Biến con trỏ là biến có chứa địa chỉ của một vùng trong bộ nhớ và có kiểu xác định
- ◆ Kích thước của con trỏ tương đương của int, tuy nhiên kích thước của vùng nhớ được trỏ tới là không xác định (con trỏ không chứa thông tin về kích thước)
- ◆ Khai báo bằng cách thêm dấu * ở trước tên biến:

- `int *pInt;`
- `char *pChar;`
- `struct SinhVien *pSV;`



- ◆ Truy xuất giá trị thông qua con trỏ dùng toán tử *:
 - `int aInt = *pInt;` (*pInt được hiểu là biến int mà pInt trỏ tới)
 - `*pChar = 'A';`
 - `printf("Gia tri: %d", *pInt);`

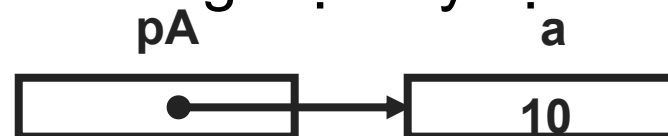
Thay đổi địa chỉ trỏ tới

- ◆ Vì giá trị của con trỏ là địa chỉ, nên khi thay đổi giá trị đó, biến con trỏ sẽ trỏ tới một vùng nhớ khác
- ◆ Gán địa chỉ mới cho con trỏ bằng phép gán như thông thường

```
int *pInt2;  
pInt2 = pInt;
```

- ◆ Toán tử địa chỉ &: tạo ra một con trỏ bằng việc lấy địa chỉ của một biến

```
int a;  
int* pA = &a; /* pA trỏ tới a */
```



- & là toán tử ngược với *, với một biến a bất kỳ thì *&a tương đương với a, và nếu p là một con trỏ thì &*p cũng tương đương với p

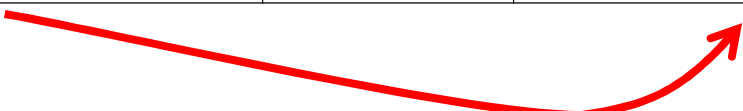
Minh họa

```
◆ char c = 'A';  
  int *pInt;  
  short s = 50;  
  int a = 10;  
  
  pInt = &a;  
  *pInt = 100;
```

Địa chỉ các biến trong bộ nhớ theo thứ tự tăng dần ở đây chỉ có tính chất minh họa. Trong thực tế, stack được cấp phát từ cao xuống thấp → biến khai báo sau sẽ có địa chỉ nhỏ hơn.

Địa chỉ	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511
Biến	char c	int* pInt				short s		int a				...
Giá trị	'A'	1507				50		100				...

```
pInt: 1507  
*pInt: 100  
&a: 1507  
a: 100
```



Con trỏ void*

- ◆ Là con trỏ nhưng không mang thông tin về kiểu
- ◆ Có thể được chuyển kiểu ngầm định sang bất kỳ kiểu con trỏ nào khác, và ngược lại (nhưng trong C++ thì không)

```
□ void* pVoid; int *pInt; char *pChar;
```

```
pInt = pVoid; /* OK */
```

```
pChar = pVoid; /* OK */
```

```
pVoid = pInt; /* OK */
```

```
pVoid = pChar; /* OK */
```

```
pChar = pInt; /* lỗi */
```

```
pChar = (char*)pInt; /* OK */
```

- ◆ Không dùng toán tử * được với con trỏ void*

```
□ *pVoid /* lỗi */
```

- ◆ Con trỏ void* được dùng để làm việc với bộ nhớ thuần túy hoặc để thao tác với những biến chưa xác định kiểu

```
□ memcpy(void* dest, const void* src, int size);
```

Con trỏ NULL

- ◆ Là một hằng con trỏ chứa địa chỉ 0, kiểu (void*), mang ý nghĩa đặc biệt là không trỏ tới địa chỉ nào trong bộ nhớ
- ◆ Bản chất là một macro được khai báo:
 - `#define NULL ((void*)0)`
- ◆ Không được gán giá trị cho con trỏ NULL
 - `int *pInt = NULL`
`*pInt = 100; /* lỗi */`
- ◆ Cần phân biệt con trỏ NULL và con trỏ chưa được khởi tạo (trỏ đến địa chỉ ngẫu nhiên)
- ◆ Con trỏ NULL thường được dùng để xác định tính hợp lệ của một biến con trỏ → để tránh lỗi, luôn gán con trỏ bằng NULL khi chưa hoặc tạm thời không được dùng tới
- ◆ Vì NULL có giá trị là 0 nên so sánh một con trỏ với NULL cũng có thể được bỏ qua trong các biểu thức logic:
 - `if (p != NULL) ...` → `if (p) ...`

Các phép toán với con trỏ

- ◆ Tăng giảm: để thay đổi con trỏ trỏ tới vị trí tiếp theo (tương ứng với kích thước kiểu nó trỏ tới)

Địa chỉ	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511
			p-- (1502)		short *p (1504)		p++ (1506)					

- ◆ Cộng địa chỉ: cũng tương ứng với kiểu nó trỏ tới

Địa chỉ	1500	1501	1502	1503	1504	1505	1506	1507	1508	1509	1510	1511
	p-2 (1500)				short *p (1504)						p+3 (1510)	

- ◆ So sánh: 2 con trỏ cùng kiểu có thể được so sánh địa chỉ với nhau như 2 số nguyên (lớn, nhỏ, bằng)
- ◆ Hai con trỏ cùng kiểu có thể trừ cho nhau để ra số phần tử sai khác

Con trỏ và mảng

- ◆ Mảng là một con trỏ tĩnh (không thể thay đổi địa chỉ), chứa địa chỉ (trỏ) tới phần tử đầu tiên của nó

- Có thể thao tác với biến kiểu mảng như thao tác với con trỏ, chỉ trừ việc gán địa chỉ mới cho nó

- ```
int arr[] = {1, 2, 3, 4, 5};
int x;
arr = 10; / như: arr[0] = 10; */
printf("%d", *(arr+2)); /* arr[2] */
arr = &x; /* lỗi */
```

- ◆ Con trỏ cũng có thể hiểu là mảng và có thể được thao tác như một mảng

- ```
int *p = arr;  
p[2] = 20; /* như: arr[2] = 20; */  
p = arr+2; /* như: p = &arr[2]; */  
p[0] = 30; /* như: arr[2] = 30; hoặc: *p = 30; */
```

- ◆ Kết luận: con trỏ và mảng có thể dùng thay thế cho nhau, tùy trường hợp mà dùng cái nào cho thuận tiện

Con trỏ và mảng (tiếp)

◆ Khác biệt:

- ❑ Không gán được địa chỉ mới cho biến kiểu mảng
- ❑ Biến kiểu mảng được cấp phát bộ nhớ cho các phần tử (trong stack) ngay từ khi khai báo
- ❑ Toán tử sizeof() với mảng cho biết kích thước thực của mảng (tổng các phần tử), trong khi dùng với con trỏ thì cho biết kích thước của bản thân nó (chứa địa chỉ)

❑ `float arr[5];` → sizeof(arr) trả về 20 (5*4)
❑ `float* p = arr;` → sizeof(p) trả về 4 với hệ thống 32 bit
❑ `sizeof(arr)/sizeof(arr[0])` → số phần tử của mảng

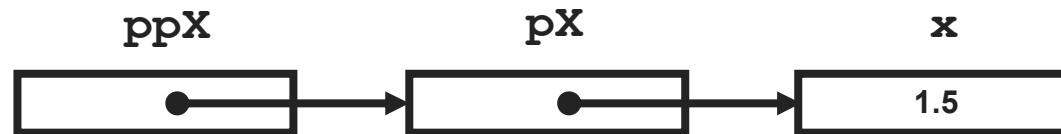
◆ Với bản chất con trỏ, có thể dùng được chỉ số âm với mảng:

```
❑ int arr[] = {1, 2, 3, 4, 5};  
  int *p = arr + 2;  
  p[-1] = 10; /* như: arr[1] = 10; */
```

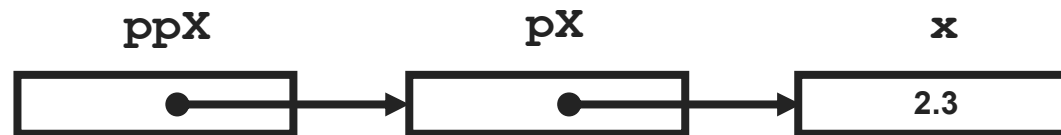
Con trỏ tới con trỏ

◆ Con trỏ có thể trỏ tới một con trỏ khác:

```
float x = 1.5;  
float *pX = &x;  
float **ppX = &pX;  
printf("%f", **ppX); /* in ra giá trị 1.5 */
```



```
**ppX = 2.3;
```



◆ Tương tự như mảng 2 chiều (hay mảng của mảng, con trỏ tới mảng, mảng các con trỏ)

Kiểu chuỗi ký tự

◆ Là mảng ký tự, kết thúc bằng ký tự '\0'

- ❑ `char ten[10] = "Tung";` (khởi tạo mảng bằng con trỏ)
- ❑ `char ten[10] = {'T', 'u', 'n', 'g', '\0'};` (bằng mảng)
- ❑ `char *ten = "Tung";` (khởi tạo con trỏ bằng con trỏ)
- ❑ `char *ten = {'T', 'u', 'n', 'g', '\0'}; /* sai */`

◆ Ví dụ tính độ dài của chuỗi:

- ❑ `for (n=0; *s; n++, s++) ;`

◆ Một số hàm xử lý chuỗi thông dụng

- ❑ `#include <string.h>`
- ❑ `int strlen(s)` → tính độ dài chuỗi s
- ❑ `char *strcpy(dst, src)` → copy chuỗi src sang chuỗi dst
- ❑ `char *strcat(dst, src)` → nối thêm chuỗi src vào chuỗi dst
- ❑ `int strcmp(str1, str2)` → so sánh 2 chuỗi, kết quả: 1, 0, -1
- ❑ `char *strstr(s1, s2)` → tìm vị trí chuỗi s2 trong s1

Xử lý dòng lệnh

- ◆ Tham số có thể được truyền cho chương trình từ dòng lệnh

- `C:\>movefile abc.txt Documents`

- ◆ Khai báo hàm `main()`

- `int main(int argc, char* argv[]) { ... }`

- `argc`: số tham số từ dòng lệnh ($\text{argc} \geq 1$)

- `argv`: mảng các tham số dưới dạng chuỗi ký tự

- Đường dẫn và tên chương trình luôn là tham số đầu tiên

- ◆ Trong ví dụ trên:

- `argc`: 3

- `argv`: ["movefile", "abc.txt", "Documents"]

Cấp phát bộ nhớ động

- ◆ Các biến khai báo được tạo ra và cấp phát bộ nhớ khi khai báo (trong stack)
- ◆ Có khi cần cấp phát theo nhu cầu sử dụng mà không biết từ khi viết chương trình → cấp phát động (trong heap)
- ◆ Cấp phát bộ nhớ:
 - ❑ `#include <stdlib.h>`
 - ❑ `void* malloc(int size) /* size: số byte cần cấp */`
 - ❑ `int *p = (int*)malloc(10*sizeof(int)); /*cấp 10 int*/`

 - ❑ `void* calloc(int num_elem, int elem_size)`
 - ❑ `void* realloc(void* ptr, int size)`
 - ❑ Việc cấp phát có thể không thành công và trả về NULL → cần kiểm tra
- ◆ Huỷ (trả lại) vùng nhớ đã được cấp phát:
 - ❑ `void free(void* p);`
 - ❑ `free(p);`

Con trỏ tới struct, union

- ◆ Với một con trỏ tới struct hoặc union, có thể dùng toán tử “->” để truy xuất các biến thành phần thay vì dùng “*” và “.”

- `p->member` tương đương với `(*p).member`

- ◆ Ví dụ:

- ```
typedef struct {
 int x, y;
} Point;

Point *pP = (Point*)malloc(sizeof(Point));
pP->x = 5; /* như: (*pP).x = 5; */
(*pP).y = 7; /* như: pP->y = 7; */
```

# Lỗi khi sử dụng con trỏ

- ◆ Trong các ứng dụng thông thường, chương trình không được truy xuất ngoài vùng nhớ được cấp cho nó → Phải kiểm soát địa chỉ mà con trỏ trỏ tới
- ◆ Hệ quả:
  - Không dùng con trỏ chưa được khởi tạo → nên có thói quen gán con trỏ bằng NULL khi chưa hoặc không dùng, để sau đó có thể kiểm tra nó đã được khởi tạo hay chưa
  - Chỉ gán địa chỉ các biến đã được tạo ra (biến tĩnh hoặc bộ nhớ cấp phát) cho con trỏ để đảm bảo con trỏ luôn trỏ tới vùng nhớ hợp lệ
  - Phải kiểm tra độ dài vùng nhớ mà con trỏ trỏ tới để không bị truy xuất vượt quá (lỗi buffer overflow)
  - Khi vùng nhớ đã cấp phát không còn dùng đến nữa, phải hủy bỏ nó để có thể sử dụng lại

# Kết luận

- ◆ Con trỏ là một đặc trưng quan trọng tạo nên sức mạnh của C so với các ngôn ngữ khác, nhưng là con dao hai lưỡi vì một khi sử dụng sai thì việc gỡ lỗi là rất khó khăn → cần nắm vững và sử dụng con trỏ một cách linh hoạt
  - ◆ Con trỏ còn được dùng rất nhiều trong các trường hợp sau:
    - Truyền giá trị từ hàm ra ngoài qua tham số
    - Con trỏ hàm
    - Cấu trúc dữ liệu: chuỗi liên kết, hàng đợi, mảng động,...
- sẽ còn trở lại trong các bài có liên quan



# Bài tập

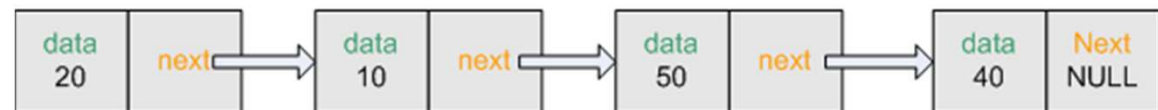
1. Viết chương trình:
  - Nhập một số nguyên N
  - Cấp phát một mảng N số nguyên và nhập dữ liệu cho nó
  - In ra màn hình mảng đó theo thứ tự ngược lại
2. Viết chương trình nhập một mảng số thực chưa biết trước số phần tử, và cũng không nhập số phần tử từ đầu (nhập đến đâu mở rộng mảng tới đó)
3. Viết chương trình nhập chuỗi s1, sau đó copy s1 vào chuỗi s2
4. Viết chương trình nhập 2 chuỗi s1 và s2, sau đó so sánh xem s1 và s2 có giống nhau không
5. Viết chương trình nhận một chuỗi từ tham số dòng lệnh, sau đó tách thành một mảng các từ tương ứng
6. Khai báo hai mảng float có giá trị tăng dần, viết chương trình trộn hai mảng đó thành mảng thứ 3 cũng theo thứ tự tăng dần

# Danh sách liên kết (Linked list)

## ◆ typedef struct node

- {
- int data; // will store information
- node \*next; // the reference to the next node
- };

- node \*temp1;
- temp1=(node\*)malloc(sizeof(node))
- temp1 = new node;



Linked list

# Bài tập

- ◆ Tạo một danh sách liên kết (linked list) và thực hiện các thao tác sau đây:
  - Chèn N phần tử (người dùng đưa vào) vào một danh sách. Với mỗi phần tử, luôn chèn vào đầu danh sách
  - Duyệt danh sách
  - Chèn một phần tử vào cuối danh sách
  - Chèn một phần tử vào một vị trí nhất định
  - Xóa phần tử đầu tiên
  - Xóa phần tử cuối cùng
  - Xóa một phần tử ở vị trí nhất định
  - Sắp xếp các phần tử trong danh sách theo thứ tự tăng hoặc giảm dần

# Xin cảm ơn!