Con trỏ, mảng và quản lý bộ nhớ

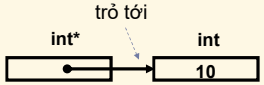
1

Con trỏ

2

Khái niệm "Con trỏ"

- **Là** biến dùng chứa địa chỉ của một vùng trong bộ nhớ và có kiểu xác định
- Kích thước **tương đương** kích thước của kiểu *int*,
- **Không chứa** thông tin về kích thước của vùng nhớ được trỏ tới (kích thước của vùng nhớ được trỏ tới là không xác định)
- **Khai báo:** thêm dấu * trước tên biến khi khai báo
  - `int *pInt;`
  - `char *pChar;`
  - `struct SinhVien *pSV;`

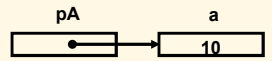


- **Truy xuất** giá trị của vùng nhớ được trỏ tới: dùng toán tử *
  - `int aInt = *pInt;` (*pInt được hiểu là biến int mà pInt trỏ tới)
  - `printf("Gia tri: %d", *pInt);`
  - `*pChar = 'A';`

3

Thay đổi địa chỉ trỏ tới

- **Giá trị** của con trỏ là địa chỉ => khi thay đổi giá trị đó, biến con trỏ sẽ trỏ tới một vùng nhớ khác
- **Gán** giá trị/địa chỉ mới cho con trỏ: toán từ gán =
  - `int a = 1, b = 2;`
    `int *pInt1 = &a, *pInt2 = &b;`
    `pInt1 = pInt2;`
- **Lấy địa chỉ** của một biến: toán tử địa chỉ `&`
  - `int a = 10, b = 20;`
    `int *pA = &a; /* pA trỏ tới a *`
    `int *pB;`
    `int *pB = &b;`



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

4

Minh hoạ

```
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 hoạ. 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
```


5

Con trỏ void*

- Con trỏ void* được dùng để làm việc với bộ nhớ thuần tuý hoặc để thao tác với những biến chưa xác định kiểu
  - `memcpy(void* dest, const void* src, int size);`
- 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 được toán tử * với con trỏ void*
  - `*pVoid  /* lỗi */`

6

Con trỏ NULL

- Con trỏ NULL dùng để **xác định tính hợp lệ** của một biến con trỏ
- **Là** một hằng con trỏ chứa giá trị 0, kiểu (void*), mang ý nghĩa đặc biệt là không trỏ tới địa chỉ nào
- Bản chất là một macro được khai báo
  - `#define NULL  ((void*)0)`
- **Không truy xuất được** giá trị của vùng nhớ mà con trỏ NULL trỏ tới ⇔ không dùng được toán tử * với con trỏ NULL

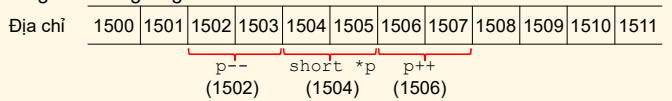
```
int *pInt = NULL
*pInt = 100;    /* lỗi */
```

- Cần phân biệt con trỏ NULL (chứa giá trị 0) và con trỏ chưa được khởi tạo (chứa giá trị ngẫu nhiên / trỏ đến địa chỉ ngẫu nhiên)
- Để tránh lỗi, **cần luôn gán** con trỏ bằng NULL khi chưa dùng hoặc tạm thời không dùng
- So sánh một con trỏ với NULL cũng có thể được bỏ qua trong các biểu thức logic: `if (p != NULL) …` → `if (p) …`

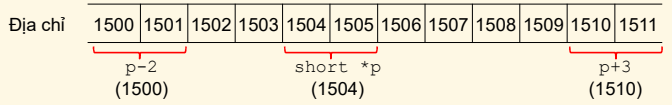
7

Các phép toán với con trỏ

- **Tăng giảm**: được dùng để thay đổi giá trị con trỏ ⇔ con trỏ sẽ trỏ tới vị trí tiếp theo (tăng) hay trỏ tới vị trí trước đó (giảm). Giá trị tăng / giảm tương ứng với **kích thước kiểu** con trỏ trỏ tới



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



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

8

Con trỏ và mảng

9

Con trỏ và mảng

- Mảng là một **con trỏ tĩnh** (không thể thay đổi giá trị (nội dung con trỏ))
  - `int x;`
    `int arr[] = {1, 2, 3, 4, 5};`
    `arr = &x;  /* lỗi */`
- Khi dùng như con trỏ, tên mảng tương ứng với tên con trỏ trỏ phần tử đầu tiên của mảng, kiểu của con trỏ này là kiểu của phần tử của mảng
  => Có thể thao tác với biến kiểu mảng như thao tác với con trỏ
  - `int arr[] = {1, 2, 3, 4, 5};`
    `int x;`
    `*arr = 10;  /* như: arr[0] = 10; */`
    `printf("%d", *(arr+2));  /* arr[2] */`
- Con trỏ cũng có thể được thao tác như 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, tuỳ trường hợp mà dùng cái nào cho thuận tiện

10

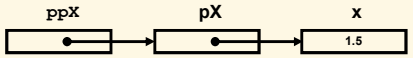
Con trỏ và mảng *(khác nhau)*

- **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**
  - `sizeof()` với mảng trả về **kích thước thực** của mảng (tổng các phần tử), `sizeof()` với con trỏ trả về kích thước của bản thân con trỏ (kích thước kiểu số nguyên)
    - `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
  - Dùng được **chỉ số âm** trong trường hợp con trỏ được dùng như mảng:
    - `int arr[] = {1, 2, 3, 4, 5};`
      `int *p = arr + 2;`
      `p[-1] = 10;  /* như: arr[1] = 10; */`

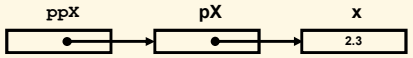
11

Con trỏ tới con trỏ

- Con trỏ có thể **trỏ tới một con trỏ khác**
  - `float x = 1.5;`
    `float *pX = &x;  /* pX trỏ đến x */`
    `float **ppX = &pX;  /* ppX trỏ đến 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)

12

Kiểu chuỗi ký tự

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

```
1. char ten[10] = "Tung";   (khởi tạo mảng bằng con trỏ)
2. char ten[10] = {'T', 'u', 'n', 'g', '\0' }; (bằng mảng)
   /* sizeof(ten) == 10 */
3. char *ten = "VietTung";      (khởi tạo con trỏ bằng con trỏ)
   /* sizeof(ten) == 4, trên hệ 32 bit */
4. char *ten = {'T', 'u', 'n', 'g', '\0' };        /* sai */
```

- Tính độ dài của chuỗi:
  - `for (n=0; *s!=0; 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, trả về con trỏ đến dst
  - `char *strcat(s1, s2)` → nối chuỗi s2 vào chuỗi s1, trả về con trỏ đến s1
  - `int strcmp(s1, s2)` → so sánh chuỗi s1 với s2 (s1-s2), kết quả: >0, =0, <0
  - `char *strstr(s1, s2)` → tìm vị trí chuỗi s2 trong s1

13

Xử lý dòng lệnh

- Tham số có thể được truyền cho chương trình từ dòng lệnh
  - `C:\>sum.exe 1 2`
  - `C:\>movefile.exe abc.txt Documents`
- Khai báo hàm main()
  - `int main(int argc, char* argv[]) { … }`
    - argc: số tham số từ dòng lệnh (argc ≥ 1)
    - argv: mảng các tham số dưới dạng chuỗi ký tự
  - Đường dẫn và tên chương trình luôn là tham số đầu tiên
  - Đổi chuỗi thành số nguyên:
    - `char *s = "3"; int n = atoi(s);`
- Trong ví dụ trên:
  - argc: 3
  - argv: ["movefile", "abc.txt", "Documents"]

14

Cấp phát bộ nhớ động

- Thông thường, các biến được cấp phát bộ nhớ khi tạo ra / khi khai báo ->cấp phát tĩnh (lưu trong stack)
- Cấp phát bộ nhớ cho biến theo nhu cầu sử dụng (chỉ biết khi chạy chương trình) → cấp phát động (lưu trong heap)
  - `#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);`

15

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; */
```


16

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 huỷ bỏ nó để có thể sử dụng lại

17

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

18

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. Không sử dụng thư viện string.h
   1. Viết chương trình nhập chuỗi s1, sau đó copy s1 vào chuỗi s2
   2. Viết chương trình nhập 2 chuỗi s1 và s2, sau đó so sánh xem s1 và s2 có giống nhau không
4. Viết chương trình nhận một chuỗi từ tham số dòng lệnh, sau đó:
   1. Viết chương trình tách từ đầu tiên ra khỏi chuỗi
   2. Viết chương trình tách từ thứ hai ra khỏi chuỗi
   3. Viết chương trình tách thành một mảng các từ tương ứng
5. 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

19

Kiến thức cần nắm
Con trỏ

- Khái niệm – lý thuyết
  - Phân biệt sự giống và khác nhau giữa biến con trỏ và biến thường
  - Phân biệt được ý nghĩa dấu * trong câu lệnh khai báo và trong sử dụng con trỏ
  - Phân biệt được con trỏ void và con trỏ có xác định kiểu
  - Phân biệt con trỏ NULL và con trỏ chưa được khởi tạo
- Sử dụng
  - Biết khai báo, khởi tạo, thay đổi giá trị (cái mà con trỏ trỏ đến)
  - Biết và sử dụng đúng các phép toán trên con trỏ: tăng/giảm, thêm bớt, so sánh, trừ

20

Kiến thức cần nắm

- Phân biệt con trỏ và mảng
  - Khác nhau giữa con trỏ và mảng
  - Dùng con trỏ như tên mảng
  - Dùng tên mảng như con trỏ
  - Dùng con trỏ trong scanf
  - Chú ý khi sử dụng con trỏ như tên mảng
- Con trỏ đến struct và union
  - Cách khai báo, truy cập và sử dụng
- Con trỏ đến con trỏ
  - Tại sao lại cần
  - Nắm được mô hình
    - Mảng 2 chiều
    - Mảng của mảng
    - Mảng các con trỏ
    - Con trỏ đến mảng

21

Kiến thức cần nắm

- Chuỗi kí tự
  - Cách biểu diễn chuỗi kí tự trong C
  - Hiểu kí tự kết thúc chuỗi và các biểu diễn '\0'
  - Các cách khai báo chuỗi kí tự
  - Các cách khởi tạo chuỗi kí tự
  - Phân biệt khai báo chuỗi kí tự như mảng và như con trỏ
  - Các hàm làm việc với con trỏ của thư viện <string.h>

22