

TRƯỜNG ĐÀO TẠO LẬP TRÌNH VIÊN VÀ QUẢN TRỊ MẠNG QUỐC TẾ BACHKHOA-APTECH

BÀI 8: CON TRỎ

MỤC TIÊU BÀI HỌC

Tìm hiểu về con trỏ và khi nào thì sử dụng con trỏ

Cách sử dụng biến con trỏ và các toán tử con trỏ

Gán giá trị cho con trỏ

Phép toán trên con trỏ

So sánh con trỏ

Con trỏ và mảng một chiều

Con trỏ và mảng nhiều chiều

Tìm hiểu cách cấp phát bộ nhớ

CON TRỎ LÀ GÌ?

Con trỏ là một biến, nó chứa địa chỉ ô nhớ của một biến khác

Nếu một biến chứa địa chỉ của một biến khác, thì biến này được gọi là con trỏ trỏ đến biến thứ hai

Con trỏ cung cấp phương thức truy xuất gián tiếp đến giá trị của một phần tử dữ liệu

Các con trỏ có thể trỏ đến các biến có kiểu dữ liệu cơ bản như int, char, double, hay dữ liệu tập hợp như mảng hoặc cấu trúc.

CON TRỎ ĐƯỢC SỬ DỤNG ĐỂ LÀM GÌ?

Các tình huống con trỏ có thể được sử dụng:

- Để trả về nhiều hơn một giá trị từ một hàm
- Để truyền mảng và chuỗi từ một hàm đến một hàm khác thuận tiện hơn
- Để làm việc với các phần tử của mảng thay vì truy xuất trực tiếp vào các phần tử này
- Để cấp phát bộ nhớ và truy xuất bộ nhớ (Cấp phát bộ nhớ trực tiếp)

BIẾN CON TRỞ

- Khai báo con trỏ: chỉ ra một kiểu cơ sở và một tên biến được đặt trước bởi dấu *

Cú pháp khai báo tổng quát:

```
type *name;
```

Ví dụ:

```
int *var2;
```

CÁC TOÁN TỬ CON TRỎ

- Hai toán tử đặc biệt được sử dụng với con trỏ:

& và *

- &** là toán tử một ngôi và nó trả về địa chỉ ô nhớ của toán hạng

var2 = &var1;

- Toán tử ***** là phần bổ xung của toán tử **&**. Đây là toán tử một ngôi và nó trả về giá trị chứa trong vùng nhớ được trỏ đến bởi biến con trỏ

temp = *var2;

GÁN TRỊ ĐỐI VỚI CON TRỎ

Các giá trị có thể được gán cho con trỏ thông qua toán tử &.

- `ptr_var = &var;`

Ở đây địa chỉ của var được lưu vào biến ptr_var.

Cũng có thể gán giá trị cho con trỏ thông qua một biến con trỏ khác trỏ có cùng kiểu.

- `ptr_var = &var;`
- `ptr_var2 = ptr_var;`

Có thể gán giá trị cho các biến thông qua con trỏ

- `*ptr_var = 10;`

Câu lệnh trên gán giá trị 10 cho biến var nếu ptr_var đang trỏ đến var

PHÉP TOÁN CON TRỎ

Chỉ có thể thực hiện phép toán cộng và trừ trên con trỏ

- `int var, * ptr_var;`
- `ptr_var = & var;`
- `var = 500;`
- `ptr_var ++;`

Giả sử biến **var** được lưu trữ tại địa chỉ **1000**

`ptr_var` lưu giá trị 1000. Vì số nguyên có kích thước là 2 bytes, nên sau biểu thức “`ptr_var++;`” `ptr_var` sẽ có giá trị là 1002 mà không là 1001

PHÉP TOÁN CON TRỎ

| | |
|---|---|
| <code>++ptr_var or ptr_var++</code> | Trở đến số nguyên kế tiếp đứng sau var |
| <code>--ptr_var or ptr_var--</code> | Trở đến số nguyên đứng trước var |
| <code>ptr_var + i</code> | Trở đến số nguyên thứ i sau var |
| <code>ptr_var - i</code> | Trở đến số nguyên thứ i trước var |
| <code>++*ptr_var or (*ptr_var)++</code> | Sẽ tăng trị var bởi 1 |
| <code>*ptr_var++</code> | Sẽ tác động đến giá trị của số nguyên kế tiếp sau var |

Mỗi lần con trỏ được tăng trị, nó trở đến ô nhớ của phần tử kế tiếp

Mỗi lần con trỏ được giảm trị, nó trở đến ô nhớ của phần tử đứng trước nó

Tất cả con trỏ sẽ tăng hoặc giảm trị theo kích thước của kiểu dữ liệu mà chúng đang trở đến

SO SÁNH CON TRỎ

- ✓ Hai con trỏ có thể được so sánh trong một biểu thức quan hệ nếu chúng trỏ đến các biến có cùng kiểu dữ liệu
- ✓ Giả sử ptr_a và ptr_b là hai biến con trỏ trỏ đến các phần tử dữ liệu a và b. Trong trường hợp này, các phép so sánh sau là có

| | |
|--------------------------------|---|
| <code>ptr_a < ptr_b</code> | Trả về giá trị true nếu a được lưu trữ ở vị trí trước b |
| <code>ptr_a > ptr_b</code> | Trả về giá trị true nếu a được lưu trữ ở vị trí sau b |
| <code>ptr_a <= ptr_b</code> | Trả về giá trị true nếu a được lưu trữ ở vị trí trước b hoặc ptr_a và ptr_b trỏ đến cùng một vị trí |
| <code>ptr_a >= ptr_b</code> | Trả về giá trị true nếu a được lưu trữ ở vị trí sau b hoặc ptr_a và ptr_b trỏ đến cùng một vị trí |
| <code>ptr_a == ptr_b</code> | Trả về giá trị true nếu cả hai con trỏ ptr_a và ptr_b trỏ đến cùng một phần tử dữ liệu. |
| <code>ptr_a != ptr_b</code> | Trả về giá trị true nếu cả hai con trỏ ptr_a và ptr_b trỏ đến các phần tử dữ liệu khác nhau nhưng có cùng kiểu dữ liệu. |
| <code>ptr_a == NULL</code> | Trả về giá trị true nếu ptr_a được gán giá trị NULL (0) |

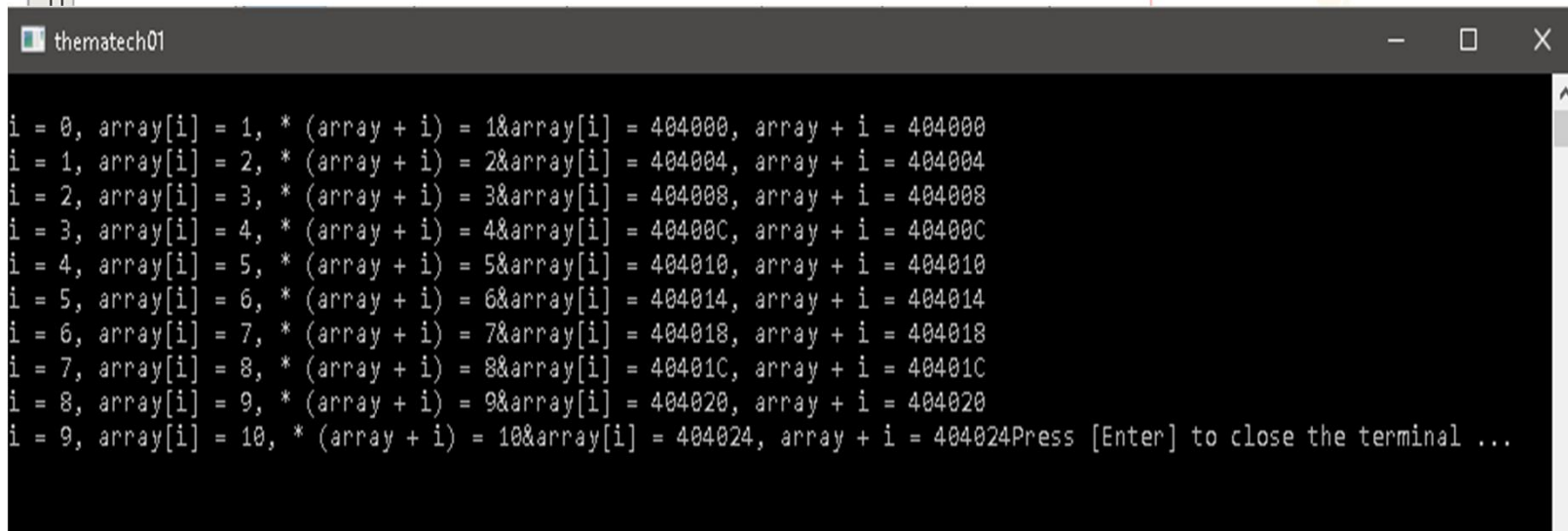
CON TRỎ VÀ MẢNG MỘT CHIỀU

Địa chỉ của một phần tử mảng có thể được biểu diễn theo hai cách:

Sử dụng ký hiệu & trước một phần tử mảng.

Sử dụng một biểu thức trong đó chỉ số của phần tử được cộng vào tên của mảng.

```
1  #include <stdio.h>
2
3  void main(){
4      static int array[10] = {1,2,3,4,5,6,7,8,9,10};
5      int i;
6      for(i = 0; i < 10; i++){
7          printf("\ni = %d, array[i] = %d, * (array + i) = %d", i, array[i], *(array + i));
8          printf("&array[i] = %X, array + i = %X", &array[i], array + i);
9      }
10 }
11
```



```
thematech01
i = 0, array[i] = 1, * (array + i) = 1&array[i] = 404000, array + i = 404000
i = 1, array[i] = 2, * (array + i) = 2&array[i] = 404004, array + i = 404004
i = 2, array[i] = 3, * (array + i) = 3&array[i] = 404008, array + i = 404008
i = 3, array[i] = 4, * (array + i) = 4&array[i] = 40400C, array + i = 40400C
i = 4, array[i] = 5, * (array + i) = 5&array[i] = 404010, array + i = 404010
i = 5, array[i] = 6, * (array + i) = 6&array[i] = 404014, array + i = 404014
i = 6, array[i] = 7, * (array + i) = 7&array[i] = 404018, array + i = 404018
i = 7, array[i] = 8, * (array + i) = 8&array[i] = 40401C, array + i = 40401C
i = 8, array[i] = 9, * (array + i) = 9&array[i] = 404020, array + i = 404020
i = 9, array[i] = 10, * (array + i) = 10&array[i] = 404024, array + i = 404024Press [Enter] to close the terminal ...
```

CON TRỎ VÀ MẢNG HAI CHIỀU

Mảng hai chiều có thể được định nghĩa như là một con trỏ trỏ tới một nhóm các mảng một chiều liên tiếp nhau

Khai báo một mảng hai chiều có thể như sau:

- **data_type (*ptr_var) [expr 2];**
- thay vì
- **data_type (*ptr_var) [expr1] [expr 2];**

CON TRỎ VÀ CHUỖI

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <conio.h>
4
5 int main() {
6     int x, *ptrx;
7     ptrx = &x;
8     x = 42;
9
10    printf("Vi tri cua bien x la %p \n", &x);
11    printf("Noi dung cua bien x la %d \n", x);
12    printf("Vi tri cua bien x la %p \n", ptrx);
13    printf("Noi dung cua bien x la %d \n", *ptrx);
14
15    *ptrx = 7826;
16    printf("\n ----- \n\n");
17
18    printf("Noi dung cua bien x la %d \n", x);
19    printf("Noi dung cua bien x la %d \n", *ptrx);
20
21    return 0;
22 }
```

```
Vi tri cua bien x la 0061FED8
Noi dung cua bien x la 42
Vi tri cua bien x la 0061FED8
Noi dung cua bien x la 42
-----
```

HÀM malloc()

- Hàm malloc() là một trong các hàm được sử dụng thường xuyên nhất để thực hiện việc cấp phát bộ nhớ từ vùng nhớ còn tự do.
- Tham số của hàm malloc() là một số nguyên xác định số bytes cần cấp phát.

HÀM free()

- Hàm free() được sử dụng để giải phóng bộ nhớ khi nó không cần dùng nữa.
- Cú pháp: `void free(void*ptr);`
- Hàm này giải phóng không gian được trả bởi ptr, để dùng cho tương lai.
- ptr phải được dùng trước đó với lời gọi hàm malloc(), calloc(), hoặc realloc().

CẤP PHÁT BỘ NHỚ

```
1 #include <stdio.h>
2 #include <malloc.h>
3
4 int main(){
5     int e = 0;
6     int i;
7     int *point = NULL;
8     printf("Nhap vao so phan tu: ");
9     scanf("%d", &e);
10
11     // cap phat bo nho
12     point = (int *) malloc(e * sizeof(int));
13     if(!point){
14         printf("Co loi xay ra!");
15         exit(1);
16     }
17     for(i = 0; i < e; i++){
18         printf("Nhap vao phan tu %d: ", i + 1);
19         scanf("%d", point + i);
20     }
21     printf("Mang da nhap: [");
22     for(i = 0; i < e; i++){
23         printf((i == e - 1) ? "%d]\n" : "%d ", *(point + i));
24     }
25     free(point);
26     return 0;
27 }
```

```
Nhap vao so phan tu: 4
Nhap vao phan tu 1: 6
Nhap vao phan tu 2: 2
Nhap vao phan tu 3: 0
Nhap vao phan tu 4: 3
Mang da nhap: [6 2 0 3]
```


HÀM calloc()

- calloc tương tự như malloc, nhưng điểm khác biệt chính là mặc nhiên giá trị 0 được lưu vào không gian bộ nhớ vừa cấp phát
- calloc yêu cầu hai tham số
 - Tham số thứ nhất là số lượng các biến cần cấp phát bộ nhớ
 - Tham số thứ hai là kích thước của mỗi biến
- Cú pháp:
 - **void *calloc(size_t num, size_t size);**

CẤP PHÁT BỘ NHỚ

```
1 #include <stdio.h>
2 #include <malloc.h>
3
4 int main(){
5     int e = 0;
6     int i;
7     int *point = NULL;
8     printf("Nhap vao so phan tu: ");
9     scanf("%d", &e);
10
11     // cap phat bo nho
12     point = (int *) calloc(e, sizeof(int));
13     if(!point){
14         printf("Co loi xay ra!");
15         exit(1);
16     }
17     for(i = 0; i < e; i++){
18         printf("Nhap vao phan tu %d: ", i + 1);
19         scanf("%d", point + i);
20     }
21     printf("Mang da nhap: [");
22     for(i = 0; i < e; i++){
23         printf((i == e - 1) ? "%d\\n" : "%d ", *(point + i));
24     }
25     free(point);
26     return 0;
27 }
```

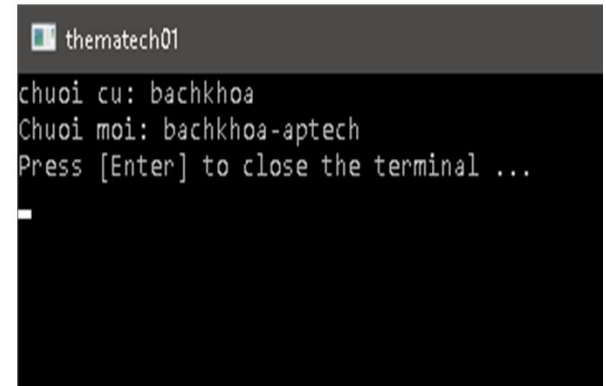
```
Nhap vao so phan tu: 6
Nhap vao phan tu 1: 5
Nhap vao phan tu 2: 1
Nhap vao phan tu 3: 0
Nhap vao phan tu 4: 2
Nhap vao phan tu 5: 4
Nhap vao phan tu 6: 9
Mang da nhap: [5 1 0 2 4 9]
```

HÀM `realloc()`

- Có thể cấp phát lại cho một vùng đã được cấp (thêm/bớt số bytes) bằng cách sử dụng hàm **`realloc`**, mà không làm mất dữ liệu.
- `realloc` nhận hai tham số
 - Tham số thứ nhất là con trỏ tham chiếu đến bộ nhớ
 - Tham số thứ hai là tổng số byte muốn cấp phát
- Cú pháp:
 - **`void *realloc(void *ptr, size_t size);`**

CẤP PHÁT BỘ NHỚ

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <stdlib.h>
4
5 int main(){
6     char *str;
7     // cấp phát bộ nhớ
8     if(!(str = (char *) calloc(10, sizeof(char)))){
9         printf("Co loi xay ra trong qua trinh cap phat bo nho!\n");
10        exit(1);
11    }
12    strcpy(str, "bachkhoa");
13    printf("chuoi cu: %s\n", str);
14
15    // tái cấp phát bộ nhớ
16    if(!(str = (char *) realloc(str, 15))){
17        printf("Co loi xay ra trong qua trinh cap phat bo nho!\n");
18        exit(1);
19    }
20    strcat(str, "-aptech");
21    printf("Chuoi moi: %s\n", str);
22    free(str);
23    return 0;
24 }
```



```
thematech01
chuoi cu: bachkhoa
Chuoi moi: bachkhoa-aptech
Press [Enter] to close the terminal ...
```

TÓM TẮT BÀI HỌC

- ✓ Một con trỏ cung cấp một phương thức truy xuất một biến mà không cần tham chiếu trực tiếp đến biến
- ✓ Một con trỏ là một biến, chứa địa chỉ vùng nhớ của một biến khác
- ✓ Sự khai báo con trỏ bao gồm một kiểu dữ liệu cơ sở, một dấu *, và một tên biến
- ✓ Có hai toán tử đặc biệt được dùng với con trỏ: * và &
- ✓ Chỉ có phép cộng và phép trừ là có thể được thực thi với con trỏ
- ✓ Các con trỏ được truyền tới hàm như các đối số
- ✓ Một tên mảng thật ra là một con trỏ trỏ đến phần tử đầu tiên của mảng
- ✓ Một hằng con trỏ là một địa chỉ; một biến con trỏ là một nơi để lưu địa chỉ
- ✓ cấp phát bộ nhớ động(`malloc()`,`calloc()`,`realloc()`)

TRƯỜNG ĐÀO TẠO LẬP TRÌNH VIÊN VÀ QUẢN TRỊ MẠNG QUỐC TẾ BACHKHOA-APTECH

THANK FOR WATCH!

