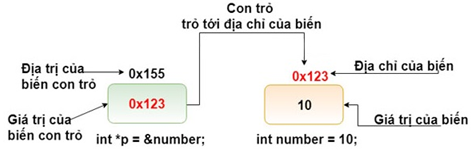
# CON TRỎ

## GIỚI THIỆU CON TRỎ

### Giới thiệu con trỏ

#### Con trỏ là gì

Con trỏ trong C là một loại biến đặc biệt dùng để lưu trữ địa chỉ của một biến khác trong bộ nhớ, mà giá trị của nó là địa chỉ của 1 biến khác.



Một số khái niệm về con trỏ:

* Giá trị của con trỏ: **địa chỉ** mà con trỏ đang trỏ đến.
* Địa chỉ của con trỏ: địa chỉ của bản thân biến con trỏ đó.
* Giá trị của biến: nơi con trỏ đang trỏ tới.
* Địa chỉ của biến nơi con trỏ đang trỏ tới = giá trị của con trỏ.

Chính vì con trỏ mang địa chỉ, nó là một biến đặc biệt có thêm những quyền năng mà biến bình thường không có. Nhờ việc nó mang địa chỉ, nó có thể trỏ lung tung trong bộ nhớ.

#### Khai báo con trỏ

Con trỏ trong C cũng có thể khai báo giống như biến bình thường, tên biến là một định danh hợp lệ.

|  |
| --- |
| **Cú pháp:** |
| <Kiểu dữ liệu> \*<Tên biến>; |

Trong đó:

* Kiểu dữ liệu có thể là: void, int, float, double,…
* Dấu **\*** trước tên biến là ký hiệu báo cho trình biên dịch biết ta đang khai báo con trỏ.

**VD 1:**

|  |
| --- |
| int \*ptr, a, b, c;      // Con trỏ trỏ đến kiểu int  int\* ptr, a, b, c;      // Con trỏ trỏ đến kiểu int  float \*fptr;   // Con trỏ trỏ đến kiểu float  char \*cptr;    // Con trỏ trỏ đến kiểu char |

**Chú ý:**

Kiểu dữ liệu của con trỏ phải trùng với kiểu dữ liệu của biến mà con trỏ sẽ trỏ đến.

#### Gán giá trị cho con trỏ

Sau khi khai báo con trỏ, bạn cần khởi tạo giá trị cho nó. Nếu con trỏ được sử dụng mà không được khởi tạo, giá trị của nó sẽ là giá trị rác, điều này sẽ làm chương trình của bạn chạy không đúng, thậm chí là nguy hiểm nếu giá trị rác đó chẳng may lại chính là địa chỉ của 1 biến nào đó bạn đang dùng.

**VD:** Gán giá trị cho con trỏ

|  |
| --- |
| int \*p, value;  value = 5;  p = &value; // khởi tạo giá trị cho con trỏ p là địa chỉ của value |

Hoặc bạn có thể khai báo và khởi tạo đồng thời:

|  |
| --- |
| int value = 5;  int \*p = &value; // khai báo con trỏ p và khởi tạo giá trị cho con trỏ là địa chỉ của value |

**Chú ý:**

Khi khai báo con trỏ mà hiện chưa sử dụng đến thì chúng ta nên gán giá trị cho nó là **NULL** (phải được viết hoa). Khi đó con trỏ NULL luôn có giá trị 0.

Việc gán giá trị NULL cho một biến con trỏ luôn là một thực tiễn tốt trong trường hợp bạn không có địa chỉ chính xác để gán. Điều này được thực hiện tại thời điểm khai báo biến. Một con trỏ được gán NULL được gọi là con trỏ null .

Con trỏ NULL là một hằng số có giá trị bằng 0 được xác định trong một số thư viện chuẩn. Hãy xem xét chương trình sau:

**VD:** Gán giá trị của con trỏ là NULL

|  |
| --- |
| #include <stdio.h>  int main () {      int  \*p;      p = NULL; // Khi gán cho NULL thì không có dấu &      printf("\nDia chi con tro p: %d - Gia tri cua con tro p: %d", &p, p);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Dia chi con tro p: 6422300 - Gia tri cua con tro p: 0 |

**Mở rộng:**

Trong hầu hết các hệ điều hành, các chương trình không được phép truy cập bộ nhớ ở địa chỉ 0 vì bộ nhớ đó được hệ điều hành dành riêng. Tuy nhiên, địa chỉ bộ nhớ 0 có ý nghĩa đặc biệt; nó báo hiệu rằng con trỏ không nhằm mục đích trỏ đến một vị trí bộ nhớ có thể truy cập được. Nhưng theo quy ước, nếu một con trỏ chứa giá trị NULL (\0), nó được coi là không trỏ đến.

### Các loại con trỏ phổ biến trong C

#### Con trỏ NULL

**Mô tả:**

* Con trỏ NULL là một con trỏ **không trỏ đến bất kỳ địa chỉ hợp lệ nào trong bộ nhớ**.
* Con trỏ NULL thường được sử dụng để khởi tạo con trỏ khi chưa biết nó sẽ trỏ đến đâu hoặc để kiểm tra xem một con trỏ đã được gán địa chỉ hay chưa.

**Cú pháp:**

|  |
| --- |
| int \*ptr = NULL; |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int \*ptr = NULL;  // Khởi tạo con trỏ NULL      if (ptr == NULL) {          printf("Con trỏ chưa trỏ đến địa chỉ hợp lệ.\n");      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| Con trỏ chưa trỏ đến địa chỉ hợp lệ. |

**Ứng dụng:**

* Tránh truy cập vào vùng nhớ không hợp lệ.
* Dùng trong việc kiểm tra con trỏ trước khi sử dụng.

#### Con trỏ void

**Mô tả:**

* Con trỏ void là con trỏ **không có kiểu dữ liệu cụ thể**. Nó có thể trỏ đến bất kỳ kiểu dữ liệu nào. Nhưng không thể truy xuất giá trị trực tiếp, bạn cần **ép kiểu** con trỏ void về kiểu cụ thể.

**Cú pháp:**

|  |
| --- |
| void \*ptr; // void\* ptr, a, b, c; |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10;      float b = 3.14;      void \*ptr;  // Khai báo con trỏ void      ptr = &a;   // Con trỏ void trỏ đến biến int      printf("Giá trị của a: %d\n", \*(int \*)ptr);  // Ép kiểu về int để truy cập giá trị      ptr = &b;   // Con trỏ void trỏ đến biến float      printf("Giá trị của b: %.2f\n", \*(float \*)ptr);  // Ép kiểu về float để truy cập giá trị      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị của a: 10  Giá trị của b: 3.14 |

#### Con trỏ cấp phát động -

**Mô tả**:

* Con trỏ cấp phát động được sử dụng để quản lý vùng nhớ được cấp phát động bởi các hàm như malloc, calloc hoặc realloc.

**Ví dụ:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10, b = 20;      const int \*ptr = &a;  // Hằng con trỏ trỏ tới a      // \*ptr = 15;         // Lỗi: Không thể thay đổi giá trị của a thông qua ptr      ptr = &b;             // Hợp lệ: Có thể thay đổi địa chỉ mà ptr trỏ tới      printf("Giá trị của b: %d\n", \*ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị trong vùng nhớ cấp phát động: 42 |

#### Con trỏ hàm

**Mô tả:**

* Con trỏ hàm là một **con trỏ** dùng để **trỏ đến một hàm**.
* Nó lưu trữ địa chỉ của hàm, cho phép bạn gọi hàm thông qua con trỏ thay vì gọi trực tiếp Tên hàm.
* Con trỏ hàm giúp chuyển hàm như một tham số, gọi hàm gián tiếp, và hỗ trợ các kỹ thuật như callback function.

**Cú pháp:**

|  |
| --- |
| return\_type (\*pointer\_name)(parameter\_list); |

Trong đó:

* return\_type: Kiểu trả về của hàm mà con trỏ trỏ tới.
* pointer\_name: Tên của con trỏ hàm.
* parameter\_list: Danh sách các tham số của hàm.

**VD:**

|  |
| --- |
| #include <stdio.h>  // Hàm cộng hai số  int add(int *a*, int *b*) {      return *a* + *b*;  }  int main() {      int \*func\_ptr(int, int) ;  // Khai báo con trỏ hàm      func\_ptr = add;             // Gán con trỏ hàm trỏ tới hàm add      printf("Kết quả của add(3, 5): %d\n", func\_ptr(3, 5));  // Gọi hàm thông qua con trỏ      return 0;  } |

**Kết quả:**

|  |
| --- |
| Kết quả của add(3, 5): 8 |

**Ứng dụng:**

* Gọi hàm gián tiếp.
* Truyền hàm như tham số (callback function).
* Thực thi các hàm khác nhau tại runtime.

#### Hàm con trỏ (Function Returning a Pointer)

**Mô tả:**

* Hàm con trỏ là một **hàm** mà **giá trị trả về là một con trỏ**.
* Hàm này trả về địa chỉ của một biến, một vùng nhớ động, hoặc một phần tử mảng.

**Cú pháp:**

|  |
| --- |
| return\_type\* function\_name(parameter\_list); |

Trong đó:

* return\_type\*: Con trỏ tới kiểu dữ liệu được trả về.
* function\_name: Tên của hàm.
* parameter\_list: Danh sách các tham số của hàm.

**VD**: Hàm con trỏ

|  |
| --- |
| #include <stdio.h>  // Hàm trả về con trỏ tới một số nguyên  int\* getPointer() {      static int value = 42;  // Biến tĩnh      return &value;          // Trả về địa chỉ của biến tĩnh  }  int main() {      // Gọi hàm và nhận con trỏ trả về      int \*ptr = getPointer();      // Truy cập giá trị mà con trỏ trỏ tới      printf("Giá trị trả về từ hàm: %d\n", \*ptr);      return 0;  } |

**Kết quả**

|  |
| --- |
| Giá trị trả về từ hàm: 42 |

**Ứng dụng:**

1. Hàm con trỏ trả về một con trỏ (địa chỉ).
2. Giá trị trả về của hàm có thể được sử dụng để trỏ tới một vùng nhớ.
3. Hữu ích khi làm việc với các biến tĩnh, vùng nhớ động, hoặc khi cần trả về địa chỉ của dữ liệu.

##### Trả về con trỏ tĩnh – Con trỏ là biến static

Một biến tĩnh được khai báo bên trong hàm sẽ **không bị hủy** sau khi hàm kết thúc. Điều này cho phép bạn trả về địa chỉ của biến tĩnh từ hàm.

**VD**: về sử dụng cấp phát động

|  |
| --- |
| #include <stdio.h>  int\* getStaticPointer() {      static int value = 42;  // Biến tĩnh (không bị hủy sau khi hàm kết thúc)      return &value;          // Trả về địa chỉ của biến tĩnh  }  int main() {      int \*ptr = getStaticPointer();      printf("Giá trị trả về từ hàm: %d\n", \*ptr);      return 0;  } |

**Kết quả**

|  |
| --- |
| Giá trị trả về từ hàm: 42 |

**Giải thích:**

* Biến value là biến tĩnh (static), vì vậy nó vẫn tồn tại sau khi hàm getStaticPointer kết thúc.
* Con trỏ ptr nhận địa chỉ của value và có thể truy cập giá trị.

##### Trả về địa chỉ của vùng nhớ được cấp phát động

Khi sử dụng các hàm cấp phát bộ nhớ động như malloc, calloc, vùng nhớ được cấp phát nằm trên **heap** và tồn tại cho đến khi bạn giải phóng nó (bằng free). Do đó, bạn có thể trả về địa chỉ của vùng nhớ đó từ hàm.

**VD**: về sử dụng cấp phát động

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int\* allocateMemory(int *size*) {      int \*ptr = (int \*)malloc(*size* \* sizeof(int));  // Cấp phát bộ nhớ động      if (ptr == NULL) {          printf("Cấp phát bộ nhớ thất bại!\n");          exit(1);      }      // Gán giá trị cho mảng      for (int i = 0; i < *size*; i++) {          ptr[i] = i + 1;      }      return ptr;  // Trả về địa chỉ của vùng nhớ được cấp phát  }  int main() {      int size = 5;      int \*arr = allocateMemory(size);  // Nhận con trỏ trả về từ hàm      printf("Các phần tử của mảng:\n");      for (int i = 0; i < size; i++) {          printf("%d ", arr[i]);      }      printf("\n");      free(arr);  // Giải phóng bộ nhớ      return 0;  } |

**Kết quả**

|  |
| --- |
| Các phần tử của mảng:  1 2 3 4 5 |

**Giải thích:**

* Hàm allocateMemory trả về địa chỉ của vùng nhớ được cấp phát động.
* Con trỏ arr nhận địa chỉ này và sử dụng để thao tác với mảng.

**Lưu ý quan trọng:**

* Bạn phải giải phóng bộ nhớ được cấp phát động sau khi sử dụng (free()), nếu không sẽ dẫn đến rò rỉ bộ nhớ (memory leak).

##### Không được trả về con trỏ đến biến cục bộ

Trả về con trỏ đến một biến cục bộ (khai báo trong hàm) là **không hợp lệ**, vì biến cục bộ sẽ bị hủy ngay sau khi hàm kết thúc. Điều này dẫn đến **lỗi truy cập vùng nhớ không hợp lệ (undefined behavior)**.

**VD**: Sai

|  |
| --- |
| #include <stdio.h>  int\* invalidPointer() {      int value = 42;  // Biến cục bộ      return &value;   // Trả về địa chỉ của biến cục bộ  }  int main() {      int \*ptr = invalidPointer();      printf("Giá trị trả về từ hàm: %d\n", \*ptr);  // Truy cập vào vùng nhớ không hợp lệ      return 0;  } |

**Kết quả**

|  |
| --- |
| Kết quả có thể không xác định (undefined behavior). Chương trình có thể in ra giá trị không mong muốn hoặc gây lỗi. |

**Giải thích:**

* Biến value trong hàm invalidPointer là biến cục bộ.
* Sau khi hàm kết thúc, biến value bị hủy, nên con trỏ ptr trỏ đến vùng nhớ không hợp lệ.

**Cách sửa:**

* Sử dụng biến tĩnh hoặc cấp phát bộ nhớ động thay vì biến cục bộ.

##### Trả về phần tử trong mảng

Nếu một mảng được khai báo trong hàm gọi hoặc được cấp phát động, bạn có thể trả về địa chỉ của một phần tử trong mảng.

**VD**: về sử dụng cấp phát động

|  |
| --- |
| #include <stdio.h>  int\* getElement(int \**arr*, int *index*) {      return &*arr*[*index*];  // Trả về địa chỉ của phần tử mảng  }  int main() {      int arr[] = {10, 20, 30, 40, 50};      int index = 2;      int \*ptr = getElement(arr, index);  // Nhận con trỏ trả về từ hàm      printf("Phần tử tại chỉ số %d: %d\n", index, \*ptr);      return 0;  } |

**Kết quả**

|  |
| --- |
| Phần tử tại chỉ số 2: 30 |

**Giải thích:**

* Mảng arr được khai báo trong main, và tồn tại trong toàn bộ thời gian chạy của chương trình.
* Hàm getElement trả về địa chỉ của phần tử thứ index trong mảng.

##### Các lưu ý quan trọng khi trả về con trỏ từ hàm

1. **Không trả về con trỏ đến biến cục bộ:**
   * Biến cục bộ bị hủy sau khi hàm kết thúc, dẫn đến lỗi vùng nhớ.
2. **Cẩn thận với vùng nhớ được cấp phát động:**
   * Vùng nhớ được cấp phát động cần được giải phóng sau khi sử dụng bằng free(), nếu không sẽ gây rò rỉ bộ nhớ.
3. **Sử dụng biến tĩnh khi cần:**
   * Biến tĩnh là cách an toàn để lưu trữ dữ liệu bên trong hàm và trả về địa chỉ của nó.
4. **Quản lý phạm vi vùng nhớ:**
   * Khi trả về con trỏ, đảm bảo rằng vùng nhớ được trỏ tới vẫn tồn tại khi sử dụng.

#### Con trỏ tới con trỏ (Pointer to Pointer)

**Mô tả:**

* Con trỏ tới con trỏ là một con trỏ lưu trữ địa chỉ của một con trỏ khác.
* Nó được sử dụng khi bạn cần quản lý hoặc thay đổi giá trị của con trỏ gốc thông qua một hàm.

**Cú pháp:**

|  |
| --- |
| data\_type \*\*ptr; |

**Sơ đồ**

A black rectangle with red text

Description automatically generated

Theo hình trên, con trỏ đầu tiên sẽ chứa địa chỉ của con trỏ thứ 2, và con trỏ thứ 2 lại chứa địa chỉ của biến Variable.

A picture containing text, diagram, screenshot, line

Description automatically generated

Theo hình trên, con trỏ ptr2 sẽ chứa địa chỉ của con trỏ ptr1, và con trỏ ptr1 sẽ chứa địa chỉ của biến num. Ta gọi ptr2 là con trỏ trỏ tới con trỏ (pointer to pointer). Đây là cách hiểu đơn giản của pointer to pointer.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10;      int \*p = &a;      // Con trỏ p trỏ tới a      int \*\*pp = &p;    // Con trỏ pp trỏ tới p      printf("Giá trị của a: %d\n", \*\*pp);  // Truy cập giá trị của a qua pp      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị của a: 10 |

**Ứng dụng:**

* Quản lý con trỏ động trong các cấu trúc dữ liệu phức tạp.
* Thay đổi giá trị của con trỏ trong các hàm.

#### Con trỏ hằng (Constant Pointer)

**Mô tả:**

* Con trỏ hằng là con trỏ mà **địa chỉ mà nó trỏ tới không thể thay đổi** sau khi được gán.
* Giá trị của vùng nhớ mà nó trỏ tới có thể thay đổi.

**Cú pháp:**

|  |
| --- |
| data\_type \*const ptr = &variable; |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10, b = 20;      int \*const ptr = &a;  // Con trỏ hằng trỏ tới a      \*ptr = 15;            // Hợp lệ: Thay đổi giá trị của a      // ptr = &b;          // Lỗi: Không thể thay đổi địa chỉ của ptr      printf("Giá trị của a: %d\n", \*ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị của a: 15 |

#### ****Hằng con trỏ (Pointer to Constant)****

**Mô tả**:

* Hằng con trỏ là con trỏ mà ***giá trị mà nó trỏ tới không thể thay đổi***, nhưng địa chỉ của con trỏ có thể thay đổi.

**Cú pháp:**

|  |
| --- |
| const data\_type \*ptr; |

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10, b = 20;      const int \*ptr = &a;  // Hằng con trỏ trỏ tới a      // \*ptr = 15;         // Lỗi: Không thể thay đổi giá trị của a thông qua ptr      ptr = &b;             // Hợp lệ: Có thể thay đổi địa chỉ mà ptr trỏ tới      printf("Giá trị của b: %d\n", \*ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị của b: 20 |

### Các lỗi phổ biến khi sử dụng con trỏ

* **Truy cập con trỏ NULL**: Sử dụng con trỏ chưa được gán giá trị hoặc đã gán NULL.
* **Giải phóng bộ nhớ 2 lần**: Gọi free() trên cùng một con trỏ nhiều lần.
* **Truy cập ngoài phạm vi**: Sử dụng con trỏ để truy cập bộ nhớ mà không hợp lệ.
* **Quên giải phóng bộ nhớ**: Không giải phóng bộ nhớ được cấp phát động, gây rò rỉ bộ nhớ (memory leak).**Các loại con trỏ**

### Các phép toán cơ bản trên con trỏ

Một con trỏ trong C là một địa chỉ, là một giá trị số. Do đó, bạn có thể thực hiện các phép toán số học trên một con trỏ giống như bạn có thể thực hiện trên một giá trị số. Có bốn toán tử số học có thể được sử dụng trên con trỏ: ++, --, +, –. Tuy nhiên khi sử dụng toán tử tăng/giảm trên biến con trỏ, nó sẽ nhảy sang phần tử liền kề chứ không phải tăng/giảm giá trị lên 1 đơn vị.

Khi thực hiện phép toán tăng (++) hoặc giảm (--) trên con trỏ:

* Giá trị của con trỏ (địa chỉ mà nó trỏ tới) sẽ thay đổi theo kích thước của kiểu dữ liệu mà con trỏ trỏ đến.
* Kích thước này phụ thuộc vào kiểu dữ liệu, ví dụ:
  + int thường chiếm 4 byte (trên hệ thống 32-bit hoặc 64-bit).
  + char chiếm 1 byte.
  + float chiếm 4 byte.
  + double chiếm 8 byte.

**Chú ý quan trọng khi tăng/giảm con trỏ**

1. ***Không vượt quá phạm vi bộ nhớ:***
   * Khi tăng/giảm con trỏ, bạn phải đảm bảo rằng con trỏ không vượt ra ngoài phạm vi của mảng hoặc vùng bộ nhớ được cấp phát. Truy cập ngoài phạm vi có thể gây lỗi hoặc hành vi không xác định (undefined behavior).
2. ***Không áp dụng trên con trỏ NULL:***
   * Truy cập hoặc tăng/giảm con trỏ NULL sẽ dẫn đến lỗi runtime.
3. ***Kích thước của kiểu dữ liệu:***
   * Việc tăng/giảm con trỏ sẽ phụ thuộc vào kích thước của kiểu dữ liệu. Điều này phải được tính toán cẩn thận khi làm việc với nhiều kiểu con trỏ khác nhau.

#### Toán tử tăng (++)

* Dùng để tăng giá trị của con trỏ (di chuyển con trỏ tới vùng nhớ kế tiếp).
* Nếu con trỏ trỏ đến một kiểu dữ liệu có kích thước **n** byte, khi tăng, địa chỉ con trỏ sẽ tăng thêm **n** byte.

**VD:** Tăng biến con trỏ để truy cập từng phần tử kế tiếp của mảng với kiểu dữ liệu **int**.

|  |
| --- |
| #include <stdio.h>  int main() {      int arr[] = {10, 20, 30, 40};      int \*ptr = arr;  // Con trỏ trỏ đến phần tử đầu tiên của mảng      printf("Địa chỉ ban đầu của ptr: %p\n", ptr);      printf("Giá trị tại ptr: %d\n", \*ptr);      ptr++;  // Tăng con trỏ, trỏ đến phần tử tiếp theo      printf("Địa chỉ sau khi tăng ptr: %p\n", ptr);      printf("Giá trị tại ptr: %d\n", \*ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Địa chỉ ban đầu của ptr: 1000 (giả sử)  Giá trị tại ptr: 10  Địa chỉ sau khi tăng ptr: 1004  (tăng thêm 4 byte, vì kiểu dữ liệu “in” chiếm 4 byte)  Giá trị tại ptr: 20 |

#### Toán tử giảm (--)

* Dùng để giảm giá trị của con trỏ (di chuyển con trỏ tới vùng nhớ trước đó).
* Nếu con trỏ trỏ đến một kiểu dữ liệu có kích thước **n** byte, khi giảm, địa chỉ con trỏ sẽ giảm đi **n** byte.

**VD:** Giảm biến con trỏ để truy cập từng phần tử kế tiếp của mảng với kiểu dữ liệu **float**.

|  |
| --- |
| #include <stdio.h>  int main() {      float arr[] = {1.1, 2.2, 3.3, 4.4};      float \*ptr = &arr[3];  // Con trỏ trỏ đến phần tử cuối của mảng      printf("Địa chỉ ban đầu của ptr: %p\n", ptr);      printf("Giá trị tại ptr: %.1f\n", \*ptr);      ptr--;  // Giảm con trỏ, trỏ đến phần tử trước đó      printf("Địa chỉ sau khi giảm ptr: %p\n", ptr);      printf("Giá trị tại ptr: %.1f\n", \*ptr);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Địa chỉ ban đầu của ptr: 1016 (giả sử)  Giá trị tại ptr: 4.4  Địa chỉ sau khi giảm ptr: 1016 (giảm 4 byte, vì kiểu dữ liệu “float” chiếm 4 byte)  Giá trị tại ptr: 3.3 |

#### Toán tử cộng (+)

* Dùng để cộng thêm một số nguyên vào con trỏ.
* ptr + n di chuyển con trỏ tới vị trí thứ n trong vùng nhớ mà nó trỏ tới.

**VD:**

|  |
| --- |
| # #include <stdio.h>  int main() {      int arr[] = {10, 20, 30, 40};      int \*ptr = arr;  // Con trỏ trỏ tới phần tử đầu tiên của mảng      printf("Giá trị tại ptr + 2: %d\n", \*(ptr + 2));  // Truy cập phần tử thứ 3      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị tại ptr + 2: 30 |

#### Toán tử trừ (-)

* Dùng để trừ một số nguyên từ con trỏ.
* ptr - n di chuyển con trỏ ngược lại n vị trí trong vùng nhớ.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int arr[] = {10, 20, 30, 40};      int \*ptr = &arr[3];  // Con trỏ trỏ tới phần tử cuối cùng      printf("Giá trị tại ptr - 2: %d\n", \*(ptr - 2));  // Truy cập phần tử thứ 2      return 0;  } |

**Kết quả:**

|  |
| --- |
| Giá trị tại ptr - 2: 20 |

#### Trừ hai con trỏ (ptr2 - ptr1)

* Khi trừ hai con trỏ, bạn sẽ nhận được **khoảng cách giữa hai con trỏ tính bằng số phần tử**.
* Điều kiện: Cả hai con trỏ phải trỏ đến các phần tử của cùng một mảng.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int arr[] = {10, 20, 30, 40, 50};      int \*ptr1 = &arr[1];  // Trỏ tới phần tử thứ 2      int \*ptr2 = &arr[4];  // Trỏ tới phần tử thứ 5      printf("Khoảng cách giữa ptr2 và ptr1: %ld\n", ptr2 - ptr1);      return 0;  } |

**Kết quả:**

|  |
| --- |
| Khoảng cách giữa ptr2 và ptr1: 3 |

**Giải thích:**

* ptr2 và ptr1 trỏ tới các phần tử của cùng một mảng.
* ptr2 - ptr1 trả về khoảng cách giữa hai con trỏ theo số phần tử (không phải số byte).

#### So sánh hai con trỏ

* Bạn có thể sử dụng các toán tử so sánh (==, !=, <, >, <=, >=) trên các con trỏ.
* Điều kiện: Cả hai con trỏ phải trỏ tới cùng một mảng hoặc cùng một kiểu dữ liệu.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int arr[] = {10, 20, 30, 40};      int \*ptr1 = &arr[1];  // Trỏ tới phần tử thứ 2      int \*ptr2 = &arr[3];  // Trỏ tới phần tử thứ 4      if (ptr1 < ptr2) {          printf("ptr1 trỏ tới vị trí trước ptr2.\n");      }      return 0;  } |

**Kết quả:**

|  |
| --- |
| ptr1 trỏ tới vị trí trước ptr2. |

### Truy xuất giá trị mà con trỏ trỏ tới

Sử dụng toán tử \* (dereference operator) để truy xuất giá trị tại địa chỉ mà con trỏ trỏ tới.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10;      int \*ptr = &a;  // Con trỏ trỏ tới biến a      printf("Giá trị của a: %d\n", \*ptr);  // Truy cập giá trị của a qua con trỏ      return 0;  } |

**Kết quả**

|  |
| --- |
| Giá trị của a: 10 |

### ****Phép gán trên con trỏ****

* Bạn có thể gán một con trỏ cho một con trỏ khác nếu cả hai con trỏ có cùng kiểu dữ liệu.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10, b = 20;      int \*ptr1 = &a;      int \*ptr2 = &b;      ptr1 = ptr2;  // Gán con trỏ ptr2 cho ptr1      printf("Giá trị tại ptr1: %d\n", \*ptr1);      return 0;  } |

**Kết quả**

|  |
| --- |
| Giá trị tại ptr1: 20 |

### Phép gán giá trị cho vùng nhớ mà con trỏ trỏ tới

* Bạn có thể thay đổi giá trị tại địa chỉ mà con trỏ trỏ tới bằng toán tử \*.

**VD:**

|  |
| --- |
| #include <stdio.h>  int main() {      int a = 10;      int \*ptr = &a;      \*ptr = 20;  // Thay đổi giá trị của a thông qua con trỏ      printf("Giá trị của a: %d\n", a);      return 0;  } |

**Kết quả**

|  |
| --- |
| Giá trị của a: 20 |