

Tổng hợp về mạng và con trỏ



- Định nghĩa về mảng trong C: là tập hợp các phần tử có cùng kiểu dữ liệu, được lưu trữ liên tiếp trong bộ nhớ
- Cú pháp khai báo mảng
- Truy cập Phần Tử Trong Mảng: Sử dụng chỉ số (index) để truy cập từng phần tử: `arr[0]`, `arr[1]`, ...
- Vòng lặp giúp duyệt toàn bộ mảng
- Mảng Hai Chiều

- Giới thiệu về Con Trỏ: Con trỏ (Pointer) là biến lưu địa chỉ của biến khác trong bộ nhớ.
- Hệ điều hành chia bộ nhớ thành các ô nhớ và gán địa chỉ lên các ô nhớ đó để truy cập
- Con trỏ chứa địa chỉ ô nhớ của 1 biến → chương trình sử dụng để truy cập được đến ô nhớ đó
- Khai báo và Khởi tạo Con Trỏ:
 - Khai báo: `int *p;` (Con trỏ p lưu địa chỉ biến kiểu int).
 - `&` lấy địa chỉ, `*` truy cập giá trị tại địa chỉ.
- Truy xuất Dữ liệu Qua Con Trỏ:
 - `printf("%d", *p);` // In giá trị của x (10)
 - `*p = 20;` // x thay đổi thành 20

- Mảng là tập hợp các phần tử cùng kiểu dữ liệu, được lưu liên tiếp trong bộ nhớ → Nếu dùng 1 con trỏ có thể trỏ đến phần tử đầu tiên của mảng, ta có thể truy xuất đến các phần tử đằng sau của mảng
- Khi khai báo mảng `int arr[5];`, tên mảng chính là con trỏ trỏ tới phần tử đầu tiên.
- Truy cập phần tử bằng con trỏ:
 - `int *p = arr;`
 - `printf("%d", *(p + 2));`
- Thay vì sử dụng chỉ số (index), ta có thể duyệt bằng con trỏ:
 - `for (int i = 0; i < 3; i++) {`
 - `printf("%d ", *(arr + i)); // In ra 10 20 30`
 - `}`

Con trỏ và mảng hai chiều / nhiều chiều

- Mảng hai chiều là tập hợp các mảng một chiều, được tổ chức theo dạng bảng (hàng và cột)
- Mảng hai chiều thực chất là mảng của con trỏ
 - Ví dụ `int matrix[2][3]` là 1 mảng chứa 3 mảng độ dài 2, tương đương 1 mảng chứa 3 con trỏ, mỗi con trỏ là địa chỉ đầu tiên của 1 mảng có 2 phần tử `int`
- Cấp Phát Động Mảng Hai Chiều: Sử dụng con trỏ cấp hai để cấp phát động:
 - `int **matrix;`
 - `matrix = (int **)malloc(2 * sizeof(int *));`
 - `for (int i = 0; i < 2; i++) {`
 - `matrix[i] = (int *)malloc(2 * sizeof(int));`
 - `}`
- Giải phóng bộ nhớ:
 - `for (int i = 0; i < 2; i++) {`
 - `free(matrix[i]);`
 - `}`
 - `free(matrix);`

- Trong RAM, 1 ô nhớ không chỉ lưu giá trị mà có thể lưu một hàm. Vì vậy 1 con trỏ có thể sử dụng để trỏ tới một hàm
- Con trỏ hàm lưu địa chỉ của một hàm, giúp gọi hàm linh hoạt
 - `void sayHello() {`
 - `printf("Hello, world!");`
 - `}`
 - `void (*funcPtr)() = sayHello; // Con trỏ trỏ đến hàm`
 - `funcPtr(); // Gọi hàm thông qua con trỏ`
- Khai báo con trỏ hàm
- Sử dụng con trỏ hàm trong các hàm khác → linh hoạt logic trong hàm, tạo callback function, tối ưu code trong chương trình

- Cấp phát động (Dynamic Memory Allocation) cho phép chương trình cấp phát và giải phóng bộ nhớ trong thời gian chạy
- Ưu điểm:
 - Tiết kiệm bộ nhớ: Chỉ cấp phát khi cần.
 - Linh hoạt: Có thể thay đổi kích thước vùng nhớ.
 - Hỗ trợ cấu trúc dữ liệu động như danh sách liên kết, cây nhị phân
- Các Hàm Cấp Phát Động
- Các Lỗi Thường Gặp:
 - Rò rỉ bộ nhớ: Không giải phóng vùng nhớ sau khi sử dụng
 - Sử dụng phải con trỏ rác (Giải phóng vùng nhớ nhưng không gán lại con trỏ về NULL → có thể tiếp tục truy cập vào vùng nhớ không được cấp phát)
- Các biện pháp phòng tránh lỗi (sử dụng thư viện, đưa các con trỏ vào struct để dễ quản lý)

Viết 1 chương trình chứa các hàm sau (Mỗi yêu cầu sẽ viết ít nhất 1 hàm và các hàm liên quan, hàm main gọi 1 trong số các hàm đã viết):

- Hàm in ra 1 mảng bình thường
- Hàm in ra 1 mảng sử dụng con trỏ
- Hàm sử dụng con trỏ cấp 2 để khởi tạo, nhập dữ liệu và in ra mảng 2 chiều
- Hàm sắp xếp mảng sử dụng con trỏ hàm để có thể tùy chỉnh sắp xếp tăng dần/ giảm dần

Lưu chương trình với tên baitap1.cpp (hoặc .c) nộp vào folder có tên là account của mình tại (tạo folder khi nộp):

<https://drive.google.com/drive/folders/1k3Tfk0qD9GW0TE5Sw6Hk88BCtrMjBcCb>

Thời hạn: đến hết ngày 18/04/2025