

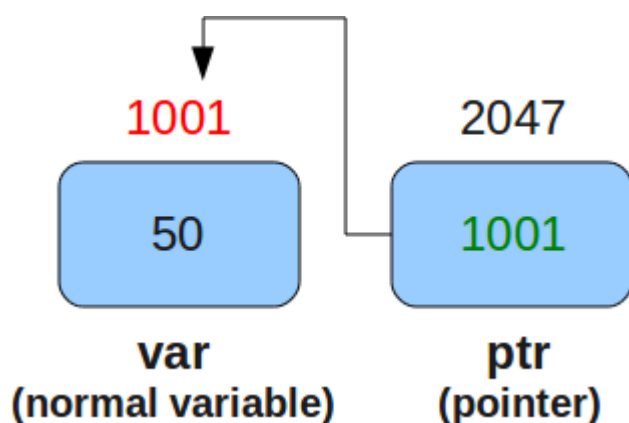
# Mảng và danh sách liên kết

## Mảng và danh sách liên kết

### Biến, con trỏ

Cấu trúc dữ liệu đơn giản nhất chính là các biến (variable). Chúng nắm giữ duy nhất một giá trị, hơn nữa, phạm vi sử dụng có giới hạn. Khi nhiều giá trị cần lưu trữ, **mảng** (arrays) được sử dụng.

Một khái niệm hơi khó hơn mặc dù không kém phần căn bản là con trỏ (pointer). Con trỏ thay vì giữ một giá trị, nó lại giữ một địa chỉ vùng nhớ:



Trong hình trên:

- **var** là một biến thông thường, có địa chỉ **1001** và giá trị **50**
- **ptr** là một biến kiểu con trỏ, có địa chỉ **2047** và giá trị **1001**. Giá trị của **ptr** là địa chỉ của biến **var**.

### Mảng (Arrays)

Mảng là một cấu trúc dữ liệu cực kỳ đơn giản và có thể xem như một danh sách với chiều dài cố định. Mảng là một cấu trúc dữ liệu “đẹp” bởi tính đơn giản của nó. Mảng đặc biệt thích hợp cho các tình huống mà ta biết trước được số lượng phần tử hoặc có thể xác định được khi chạy chương trình.

Giả sử bạn cần một đoạn mã để tính giá trị trung bình của một vài con số. Mảng là sự lựa chọn tuyệt vời để giữ các giá trị bởi yêu cầu bài toán không đòi hỏi một thứ tự lưu trữ cụ thể và các phép tính toán cũng không đòi hỏi gì khác ngoài việc duyệt qua toàn bộ các phần tử.

Một trong những sức mạnh khác của mảng chính là ta có thể truy cập các phần tử của mảng một cách ngẫu nhiên bằng chỉ số. Ví dụ như, bạn có một danh sách gồm tên của các học sinh trong lớp học. Mỗi học sinh

đánh số từ 1 đến  $n$ . Khi đó để đọc hoặc lưu tên học sinh thứ  $i$  bạn chỉ cần gọi tới `studentName[i]`. Do các phần tử của mảng ở vị trí liên tiếp nhau trong bộ nhớ máy tính, nên thao tác này chỉ mất độ phức tạp  $\mathcal{O}(1)$ . Tuy nhiên cũng vì lý do này, nên việc tăng kích thước mảng hay thêm / xóa 1 phần tử vào vị trí bất kỳ của mảng có độ phức tạp  $\mathcal{O}(N)$ .

Mảng có số lượng phần tử cố định, mỗi phần tử giữ của mảng một thông tin và ở một vị trí không đổi đã được định nghĩa trước đó.

## Tổng kết

- Bộ nhớ cố định, cần biết trước số phần tử
- Truy cập một vị trí bất kỳ:  $\mathcal{O}(1)$ .
- Thêm / xóa một phần tử:  $\mathcal{O}(N)$ .

## Mở rộng

Trong C++ STL, có CTDL `vector`, được gọi là mảng động. CTDL này cho phép thực hiện các thao tác:

- Truy cập một vị trí bất kỳ:  $\mathcal{O}(1)$ .
- Thêm / Xóa 1 phần tử vào cuối mảng: độ phức tạp trung bình:  $\mathcal{O}(1)$ .
- Thêm / xóa một phần tử bất kỳ:  $\mathcal{O}(N)$ .

Để làm được điều này, `vector` sử dụng kĩ thuật "nhân đôi mảng":

- Mảng của ta sẽ có thể cấp phát bộ nhớ tối đa là 2 lần chiều dài thật (số phần tử) của mảng.
- Nếu mảng vẫn chưa đầy (bộ nhớ cấp phát lớn hơn chiều dài thật của mảng), ta chỉ cần đơn giản là thêm phần tử mới vào cuối mảng.
- Nếu mảng đầy, ta làm như sau:
  - Cấp phát bộ nhớ mới bằng 2 lần chiều dài mảng
  - Copy mảng cũ qua bộ nhớ mới cấp phát. Như vậy ta có mảng mới với dữ liệu giống với mảng cũ, và còn dư ra chỗ trống ở cuối mảng mới.
  - Thêm phần tử mới vào cuối mảng mới.

Nếu ta thêm  $N$  phần tử vào cuối mảng, thì độ phức tạp là:

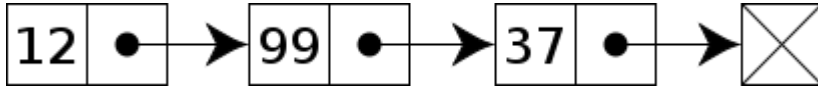
- $\mathcal{O}(1 + 2 + 4 + 8 + 16 + \dots)$  cho các thao tác cấp phát bộ nhớ
- $\mathcal{O}(1)$  cho việc ghi phần tử mới vào phần bộ nhớ trống.

Do đó độ phức tạp tổng không quá  $\mathcal{O}(N)$ , và độ phức tạp trung bình của 1 thao tác là  $\mathcal{O}(1)$ .

## Danh sách liên kết (Linked Lists)

Danh sách liên kết là một cấu trúc dữ liệu có thể giữ một số lượng phần tử tùy ý và dễ dàng thay đổi kích thước, cũng như dễ dàng bỏ đi các phần tử bên trong nó.

Danh sách liên kết, hiểu theo cách đơn giản nhất là một con trỏ trỏ tới một nút dữ liệu. Mỗi nút dữ liệu bao gồm dữ liệu cần chứa và một con trỏ trỏ tới nút tiếp theo. Tại điểm cuối cùng, con trỏ trỏ tới giá trị NULL.



Với thiết kế như ban đầu, một danh sách liên kết thích hợp để lưu trữ dữ liệu khi chưa biết trước được số lượng các phần tử hoặc các phần tử thường xuyên thay đổi. Tuy vậy, chúng ta không thể truy cập một cách ngẫu nhiên các phần tử của danh sách liên kết. Để tìm kiếm một giá trị, ta phải bắt đầu tại phần tử đầu tiên và duyệt tuần tự qua các phần tử cho tới khi bắt gặp được giá trị mà mình cần tìm kiếm. Để chèn một nút vào danh sách liên kết, bạn cũng phải thực hiện tương tự. Độ phức tạp của cả 2 thao tác này là  $\mathcal{O}(N)$ . Tuy nhiên, nếu ta biết được con trỏ trỏ đến phần tử cần xóa, thì độ phức tạp chỉ là  $\mathcal{O}(1)$ . Dễ dàng nhận thấy, thao tác tìm kiếm và chèn trong danh sách liên kết không thật sự hiệu quả.

Sau đây là cài đặt kiểu dữ liệu danh sách liên kết thông thường:

```

1 struct ListNode {
2     int data; // dữ liệu được lưu ở nút của linked list
3     ListNode* nextNode; // con trỏ trỏ tới phần tử tiếp theo của linked list.
4 };
5 ListNode* firstNode;
  
```

Bạn có thể chèn một nút mới vào bằng cách chèn chúng vào đầu danh sách. Thao tác này có độ phức tạp là  $\mathcal{O}(1)$ .

```

1 ListNode* newNode = new ListNode();
2 newNode->nextNode = firstNode;
3 firstNode = newNode;
  
```

Duyệt qua toàn bộ danh sách liên kết rất đơn giản như sau:

```



1 ListNode* curNode = firstNode;
2 while (curNode != NULL) {
3     cout << curNode->data << endl;
4     curNode = curNode->nextNode;
5 }
  
```

Ngoài ra, ta cũng có thể lưu thêm con trỏ trỏ vào phần tử cuối của danh sách. Khi đó độ phức tạp để thêm 1 phần tử vào cuối danh sách là  $\mathcal{O}(1)$ .

## Tổng kết

- Thêm / xóa 1 phần tử mới vào đầu / cuối:  $\mathcal{O}(1)$
- Truy cập 1 phần tử ở vị trí bất kỳ:  $\mathcal{O}(N)$ .

# Tài liệu tham khảo

- [Topcoder](#) 
- [Wikipedia](#) 

Được cung cấp bởi [Wiki.js](#)