Cấu trúc dữ liệu và giải thuật

CÁC CẤU TRÚC DỮ LIỆU CƠ BẢN

Giảng viên:

Văn Chí Nam – Nguyễn Thị Hồng Nhung – Đặng Nguyễn Đức Tiến – Vũ Thanh Hưng

Nội dung trình bày

- Danh sách liên kết
- Ngăn xếp
- Hàng đợi

Danh sách liên kết

Nội dung

- Giới thiệu
- Các thao tác trên danh sách liên kết
- Các loại danh sách liên kết
- So sánh danh sách liên kết và mảng
- Ứng dụng

Giới thiệu

- Mảng: cấu trúc dữ liệu quen thuộc
  - Tập có thứ tự
  - Số lượng phần tử cố định (tĩnh)
  - Cấp phát vùng nhớ liên tục
  - Truy xuất phần tử thông qua chỉ số

Giới thiệu

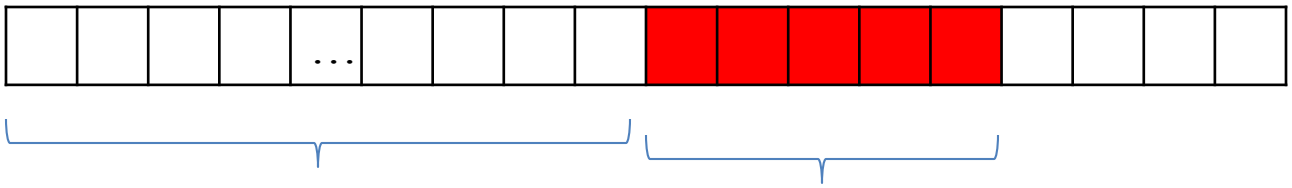
- Đánh giá thao tác trên mảng:
  - Truy xuất phần tử?
  - Cập nhật?
  - Chèn phần tử?
  - Xoá phần tử?

Vấn đề


```
struct HOC_SINH
{
        char[100] Ho_ten;
        char[10] MS;
        int Nam_sinh;
}
....
void main()
{
        HOC_SINH [40] Lop;                      //cấp phát tỉnh
        HOC_SINH* pLop = new HOC_SINH[n];       //hoặc cấp phát động

}
```

Lớp có thêm 3 học sinh mới chuyển vào ????



Giới thiệu

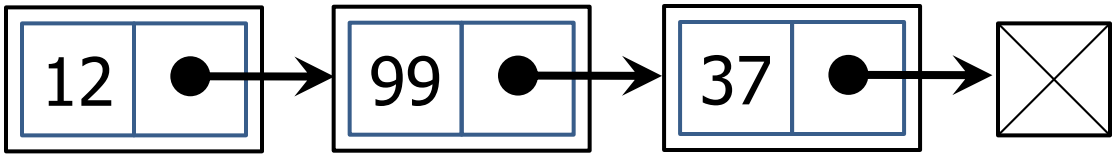
- Thực tế:
  - Không xác định được chính xác số lượng phần tử
    - Danh sách bệnh nhân: tăng/giảm.
    - Danh sách sinh viên: tăng/giảm.
  - Vùng nhớ thay đổi trong quá trình sử dụng

    => Không đủ vùng nhớ cấp phát liên tục.

=> Cần cấu trúc dữ liệu động để đáp ứng nhu cầu trên

Danh sách liên kết đơn

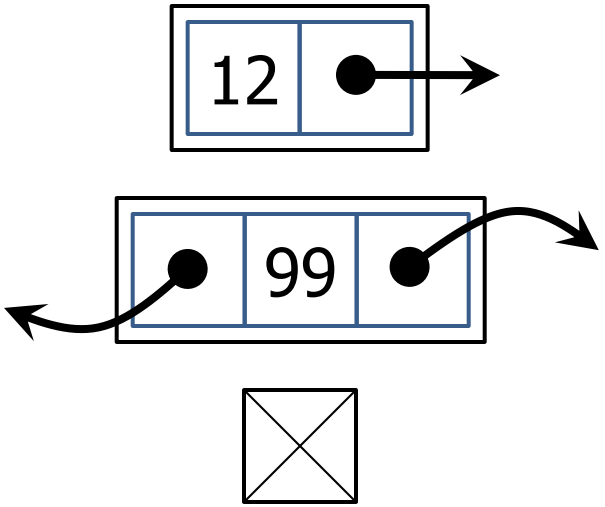
- Mỗi phần tử có MỘT liên kết đến phần tử phía sau nó.



- Mảng mặc định phần tử ngay sau là phần tử kế tiếp.

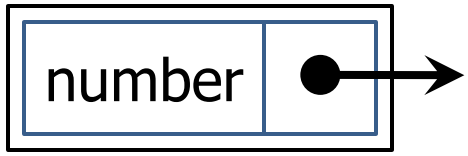
Phần tử trên danh sách liên kết

- Phần tử (Node, Element)
  - **Phần tử = Dữ liệu + Liên kết**
  - Ví dụ:
    - Phần tử có 1 liên kết
    - Phần tử có 2 liên kết
    - Phần tử rỗng

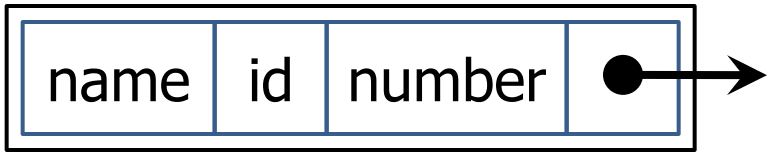


Ví dụ

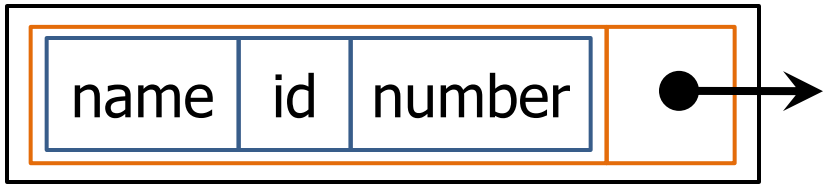
- Ví dụ:
  - Phần tử có dữ liệu gồm 1 thành phần



  - Phần tử có dữ liệu gồm 3 thành phần



  - Phần tử có dữ liệu gồm 1 cấu trúc

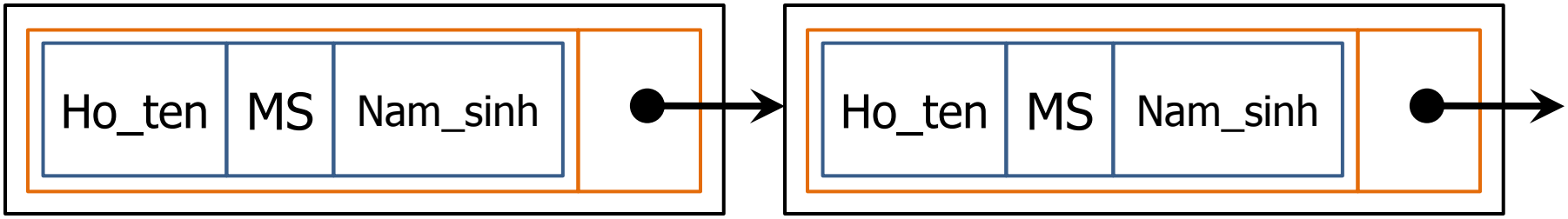


Cài đặt

- Sinh viên tự viết phần cài đặt cho các ví dụ HOC_SINH

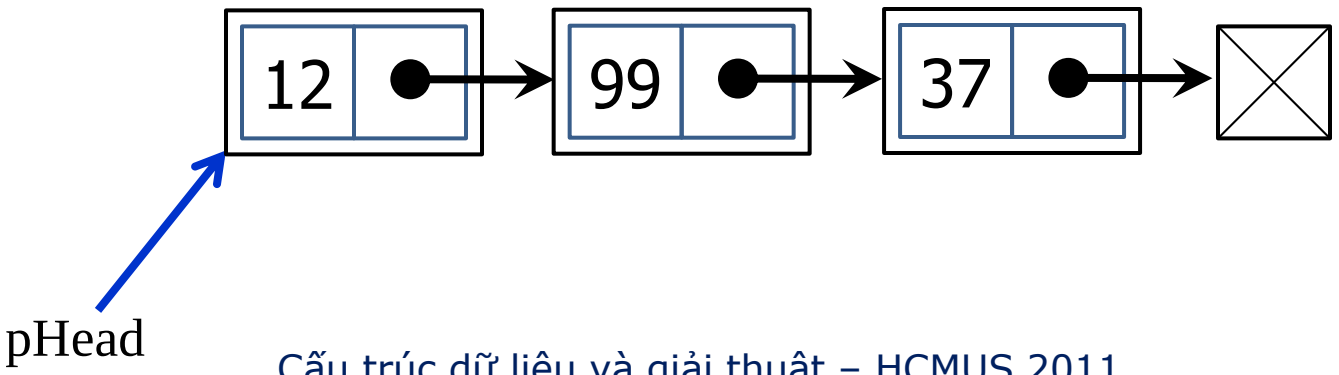
Cài đặt Node (phần tử) cho HOC_SINH


```
struct NODE
{
    HOC_SINH hs;
    NODE* pNext;
}
```



Danh sách liên kết

- Mỗi bạn là một node
  - Giá trị là tên bạn đó
  - Liên kết: là cánh tay trỏ đến bạn tiếp theo
- Bạn cuối cùng sẽ không liên kết với ai cả.
- Cánh tay pHead sẽ nhiệm vụ ghi nhớ bạn đầu tiên trong danh sách liên kết



Các thao tác trên danh sách liên kết

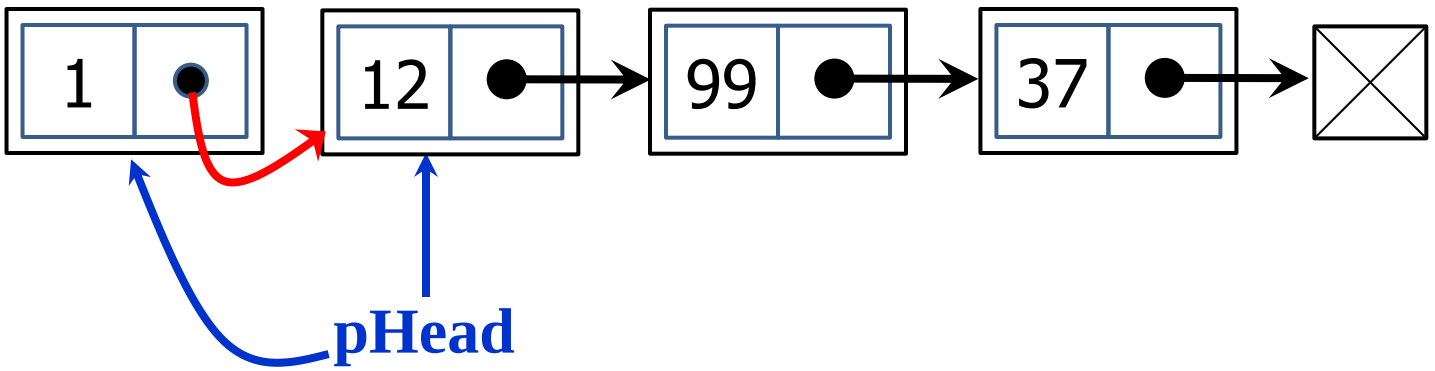
- Thêm phần tử
- Duyệt danh sách
- Xoá phần tử
- Truy xuất phần tử
- Xoá danh sách

Thêm phần tử

- Vào đầu danh sách
- Sau một phần tử
- Vào cuối danh sách

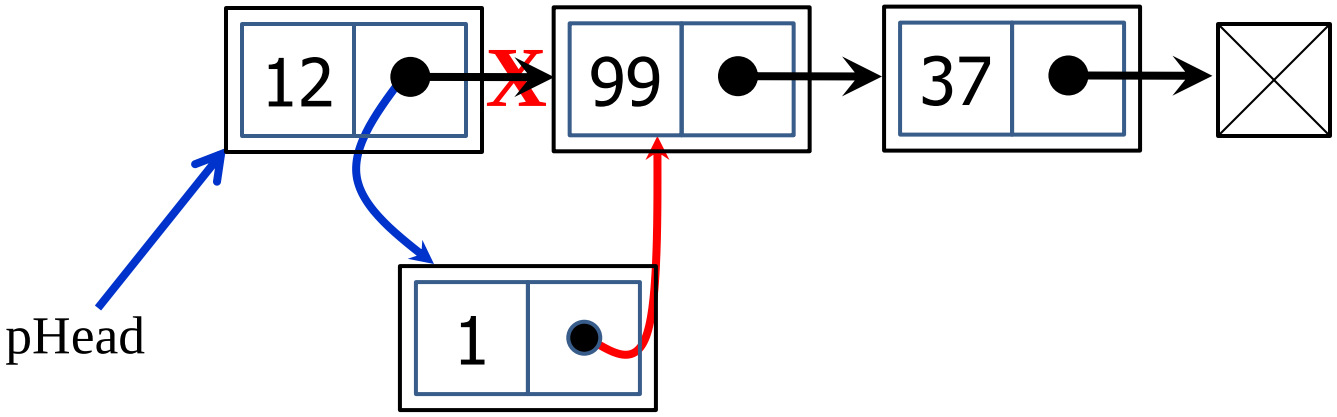
Thêm phần tử

- Vào đầu danh sách:
  - Nếu danh sách rỗng
    - Phần tử vừa thêm là phần tử đầu danh sách
  - Ngược lại,



Thêm phần tử

- Sau một phần tử (pNode):
  - Nếu danh sách rỗng?
  - Nếu danh sách khác rỗng?
    - Tạo node mới có dữ liệu là Data.
    - Cập nhật lại liên kết của pHead và node vừa tạo.



Duyệt danh sách

- Đảm bảo việc truy xuất đến tất cả các phần tử trên danh sách

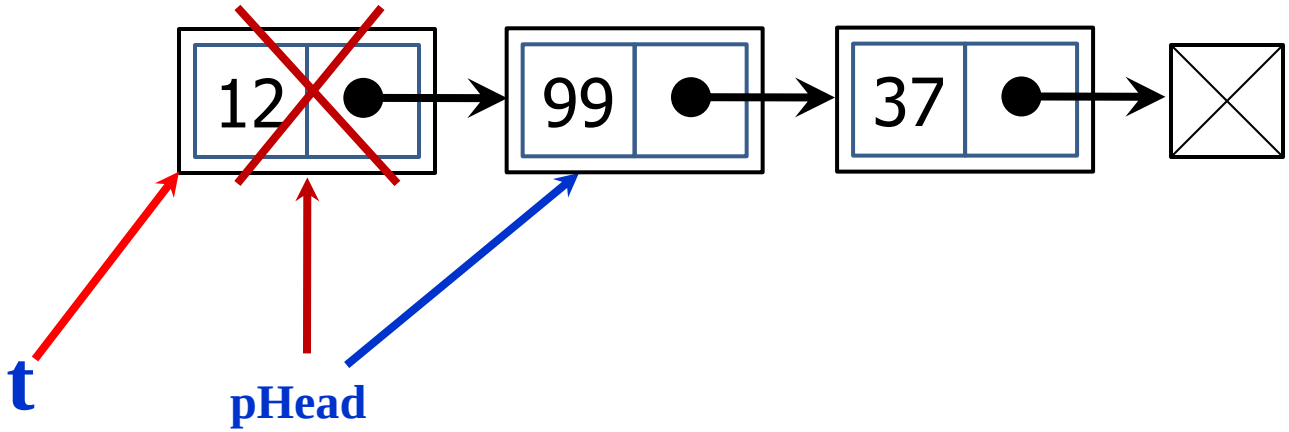
- Thuật toán:
  - Bắt đầu từ phần tử đầu tiên
  - Trong khi chưa hết danh sách
    - Xử lý phần tử hiện hành
    - Di chuyển đến phần tử kế tiếp

Xoá phần tử

- Đầu danh sách
- Cuối danh sách
- Sau một phần tử
- Theo khóa k

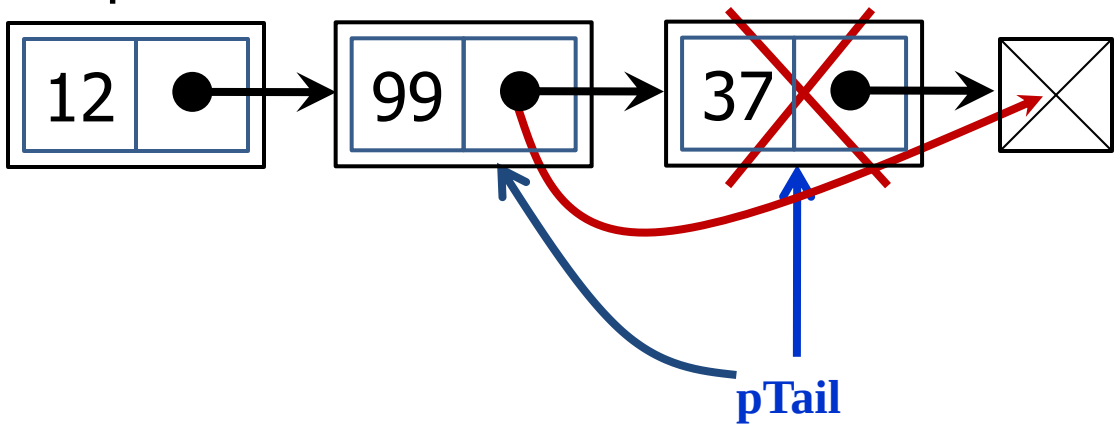
Xoá phần tử

- Đầu danh sách
  - Nếu danh sách rỗng:
  - Nếu danh sách khác rỗng:
    - Cập nhật lại pHead
    - Xóa con trỏ pHead cũ



Xoá phần tử

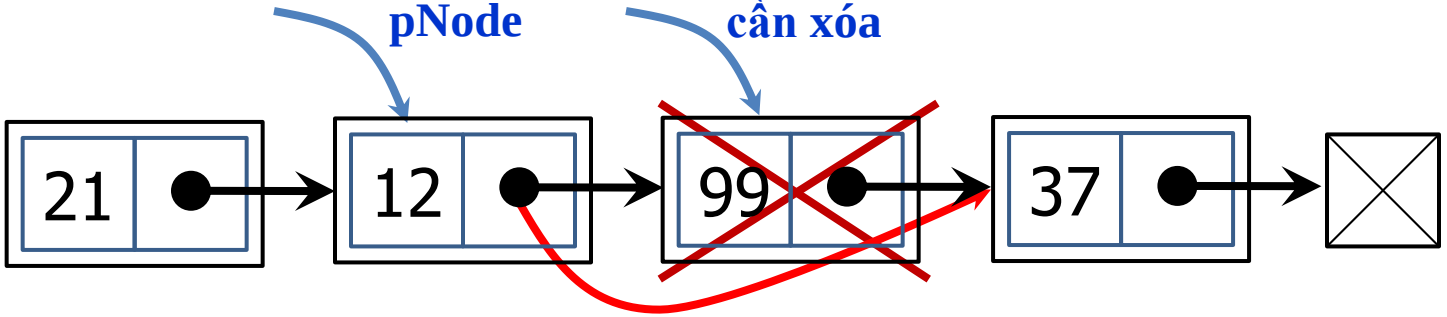
- Cuối danh sách:
  - Danh sách rỗng?
  - Danh sách khác rỗng:
    - tìm con trỏ cuối danh sách pTail
    - Cập nhật lại pTail
    - Xóa pTail cũ



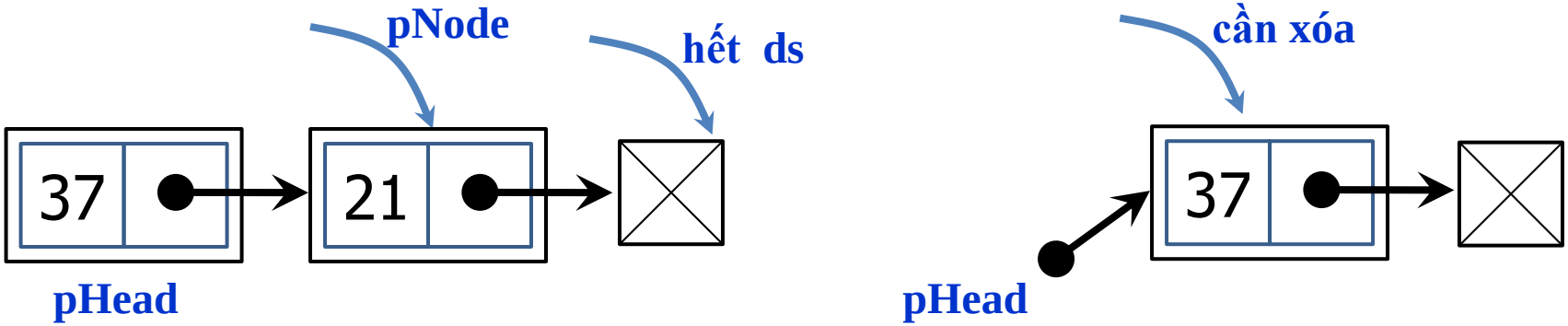
Xoá phần tử

- Xóa một phần tử giá trị là k

  - Trường hợp chung: Tìm node đứng trước pNode



  - Trường hợp đặc biệt:



Xoá danh sách

- Danh sách liên kết bao gồm các phần tử được cấp phát động.
- Phải xoá các phần tử trên danh sách sau khi đã sử dụng xong danh sách.
- Thuật toán:
  - Trong khi pHead != NULL
  - Xoá phần tử đầu danh sách

Tính chất của danh sách liên kết

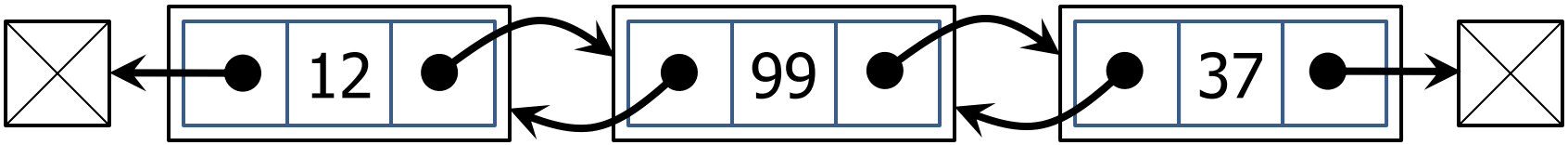
- Một dãy tuần tự các phần tử (node)
- Giữa hai phần tử có liên kết với nhau.
- Các phần tử không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
- Có thể mở rộng tuỳ ý (chỉ giới hạn bởi dung lượng bộ nhớ)
- Thao tác Chèn/Xóa không cần phải dịch chuyển phần tử
- Có thể truy xuất đến các phần tử khác thông qua các liên kết

Các loại danh sách liên kết

- Danh sách liên kết đơn
  - singly linked list
  - uni-directional linked list
- **Danh sách liên kết kép**
  - doubly linked list
  - bi-directional linked list
- **Danh sách liên kết vòng**
  - circularly linked list
  - ring list

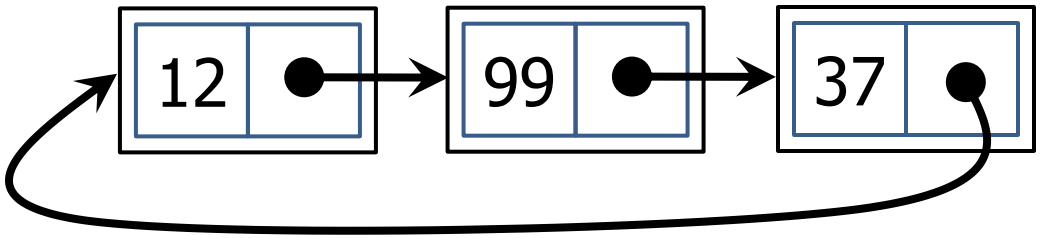
Danh sách liên kết kép

- Mỗi phần tử có HAI liên kết đến phần tử đứng sau và trước nó.



Danh sách liên kết vòng

- Có mối liên kết giữa phần tử cuối và phần tử đầu



So sánh danh sách liên kết và mảng

Danh sách liên kết đơn	Mảng
Số phần tử không cần xác định trước.	Cần xác định trước số phần tử.
Cấp phát vùng nhớ riêng lẻ cho từng phần tử.	Cần cấp phát vùng nhớ liên tục đủ lớn để lưu trữ mảng → lãng phí nếu không dùng hết.
Truy xuất tuần tự, danh sách liên kết đơn chỉ có thể duyệt 1 chiều.	Truy xuất ngẫu nhiên, đơn giản, nhanh chóng.
Cần nhiều bộ nhớ hơn để lưu trữ các liên kết	Không cần
Thêm/xóa phần tử đầu: O(1)	Thêm/xóa phần tử đầu: O(n)
Thêm/xóa phần tử cuối: O(n)	Thêm/xóa phần tử cuối: O(1)
Thêm/xóa phần tử giữa: O(n)	Thêm/xóa phần tử giữa: O(n)

Ứng dụng

- Là cấu trúc dữ liệu chính cho ngôn ngữ lập trình LISP (LIst Processing Language) – ngôn ngữ lập trình hàm.
- Giúp nâng cao hiệu quả của một số thuật toán sắp xếp: Quick Sort, Radix Sort

Bài tập 1

- Cho một DSLK đơn, mỗi node trong DSLK lưu thông tin là 1 số nguyên và con trỏ đến node kế. Tạo 2 DSLK đơn mới (không phá huỷ DSLK đã cho).
  - Một danh sách chứa các số lẻ của danh sách đã cho.
  - Một danh sách chứa các số chẵn của danh sách đã cho.

Bài tập 1

- In ra các *đường chạy* tự nhiên từ DSLK đã cho:

VÍ DỤ: DSLK ban đầu biểu diễn các số: 1 5 6 4 8 3 7

In ra các dãy số: 1 5 6

4 8

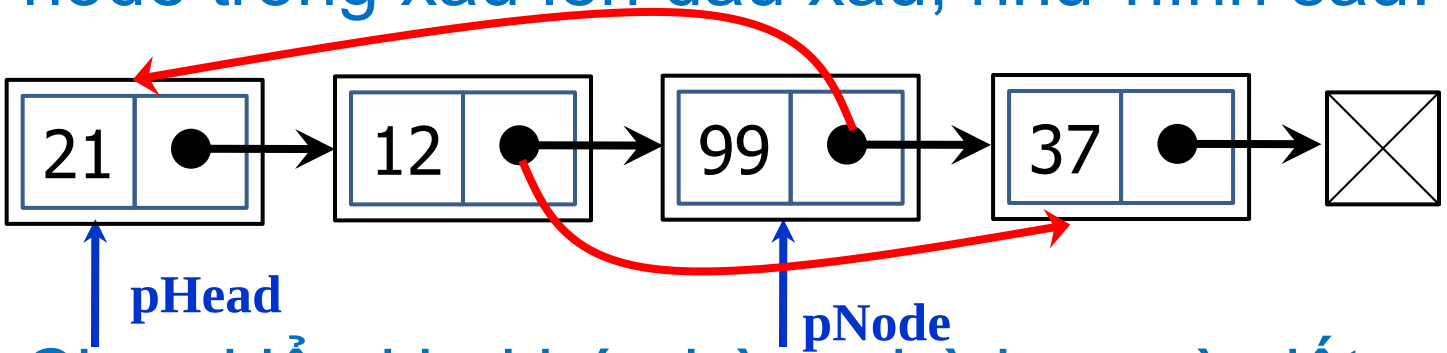
3 7

Bài tập 2

- Cho danh sách liên kết đơn L, lập giải thuật thực hiện các phép sau đây:
  - Tính số lượng các nút của danh sách.
  - Tìm tới nút thứ k trong danh sách, nếu có nút thứ k thì cho biết địa chỉ của nút đó, ngược lại trả về null.
  - Bổ sung một nút vào sau nút k.
  - Loại bỏ nút đứng trước nút k.
  - Đảo ngược danh sách đã cho.

Bài tập 3

- Hàm MoveToFront có tác dụng di chuyển 1 node trong xâu lên đầu xâu, như hình sau:



- Chọn kiểu khai báo hàm phù hợp và viết code

void MoveToFront(NODE □□ pHead, NODE □□ pTail, NODE □□ pNode)

Lưu ý: các kí hiệu □ có thể là *, & hoặc khoảng trắng

Hỏi và Đáp