HUST

ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

ONE LOVE. ONE FUTURE.

# CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ĐẠI HỌC
BÁCH KHOA HÀ NỘI
HANOI UNIVERSITY
OF SCIENCE AND TECHNOLOGY

# CẤU TRÚC DỮ LIỆU VÀ THUẬT TOÁN

## TUẦN 4 : DANH SÁCH LIÊN KẾT

ONE LOVE. ONE FUTURE.

# NỘI DUNG

1. Kiến thức cơ sở
2. Danh sách liên kết đơn
3. Thao tác trên danh sách liên kết

# MỤC TIÊU

***Sau bài học này, người học có thể:***

1. Hiểu về cấu trúc dữ liệu danh sách liên kết đơn;

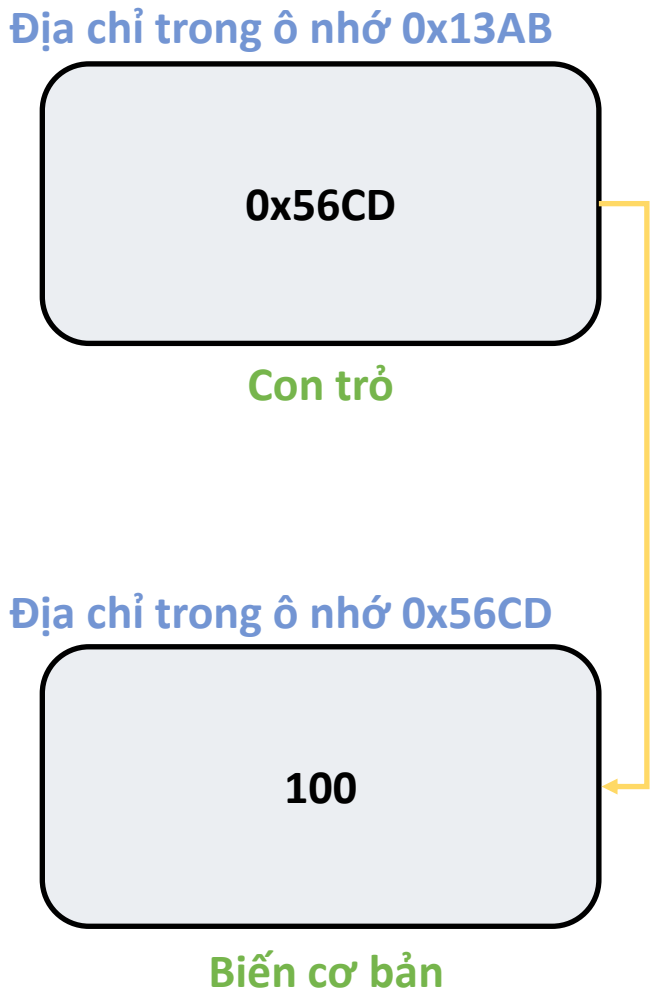
2. Xây dựng hai thao tác cơ bản trên cấu trúc dữ liệu danh sách liên kết đơn: Duyệt và Tìm kiếm

# NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở
   1.1. Con trỏ
   1.2. Cấu trúc
2. Danh sách liên kết đơn
3. Thao tác trên danh sách liên kết
   3.1 Duyệt danh sách
   3.2 Tìm kiếm

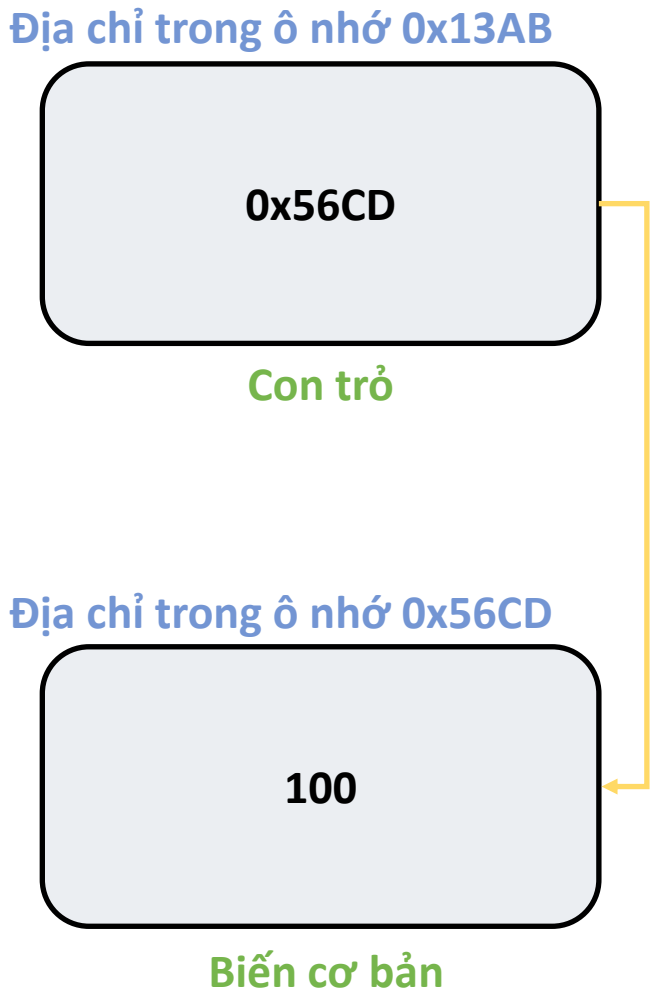
## 1.1. Con trỏ

- Con trỏ (Pointer) là khái niệm cơ bản trong ngôn ngữ lập trình C, dùng để làm việc với địa chỉ bộ nhớ.
- Biến con trỏ cũng là một biến, cũng cần khai báo, khởi tạo và dùng để lưu trữ dữ liệu.
- Biến con trỏ có địa chỉ riêng.
- Biến con trỏ không lưu giá trị như biến cơ bản, nó trỏ tới một địa chỉ khác, tức mang giá trị là một địa chỉ trong RAM.



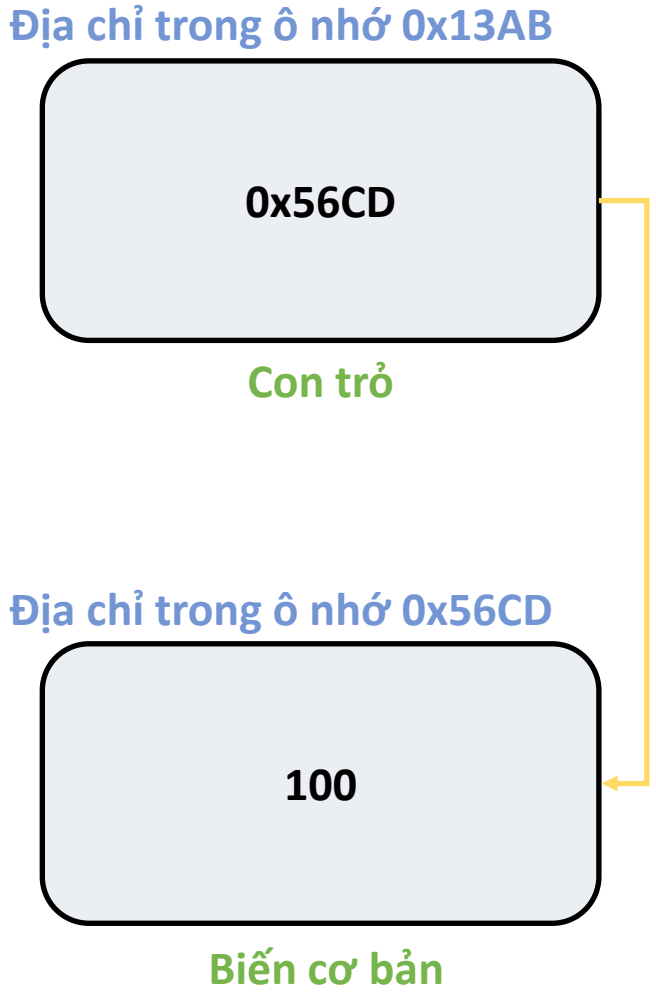
## 1.1. Con trỏ

- Con trỏ (Pointer) là khái niệm cơ bản trong ngôn ngữ lập trình C, dùng để làm việc với địa chỉ bộ nhớ.
- Kiểu dữ liệu của con trỏ trùng với kiểu dữ liệu tại vùng nhớ mà nó trỏ đến.
- Giá trị của con trỏ chứa địa chỉ vùng nhớ mà con trỏ trỏ đến.
- Địa chỉ của con trỏ là địa chỉ của bản thân biến con trỏ đó trong RAM.



## 1.1. Con trỏ

- Khai báo: **`int *p;`**
  - Khai báo con trỏ để trỏ tới biến kiểu nguyên
  - Giá trị của **`p`** xác định địa chỉ của biến
- Gán giá trị: **`int a; int* p = &a;`**
  - **`p`** trỏ vào biến **`a`**



## 1.1. Con trỏ

- Ví dụ

```c
#include <stdio.h>
int main() {
    int* pc, c;

    c = 22;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);

    pc = &c;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);

    c = 11;
    printf("Address of pointer pc: %p\n", pc);
    printf("Content of pointer pc: %d\n\n", *pc);

    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c);
    return 0;
}
```

In ra cùng một địa chỉ bộ nhớ

In ra giá trị: 22

In ra giá trị: 22

In ra giá trị: 11

In ra giá trị: 2

*Nguồn: https://www.programiz.com/c-programming*

## 1.2. Cấu trúc

- Cấu trúc (struct) là một kiểu dữ liệu người dùng tự định nghĩa (user defined datatype)
- Cấu trúc là một tập hợp các biến, những biến này có thể có kiểu dữ liệu khác nhau

```
struct structureName {
   dataType member1;
   dataType member2;
   ...
};
```

```
typedef struct  TNode{
          int a;
          double b;
          char* s;
      }TNode;
```

## 1.2. Cấu trúc

- Cấu trúc (struct) là một kiểu dữ liệu người dùng tự định nghĩa, bao gồm một tập hợp các biến, những biến này có thể có kiểu dữ liệu khác nhau

```
typedef struct  TNode{
          int a;
          double b;
          char* s;
      }TNode;
```

- **TNode* q:** **q** là con trỏ trỏ đến 1 biến có kiểu **TNode**
- **Q→a:** truy nhập đến thành phần a của kiểu cấu trúc
- **q = (TNode*)malloc(sizeof(TNode)):** cấp phát bộ nhớ cho 1 kiểu **TNode** và **q** trỏ vào vùng nhớ được cấp phát

# NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở

   1.1. Con trỏ

   1.2. Cấu trúc

**2. Danh sách liên kết đơn**

3. Thao tác trên danh sách liên kết

   3.1 Duyệt danh sách

   3.2 Tìm kiếm

# 2. DANH SÁCH LIÊN KẾT ĐƠN

## 2.1. Giới thiệu

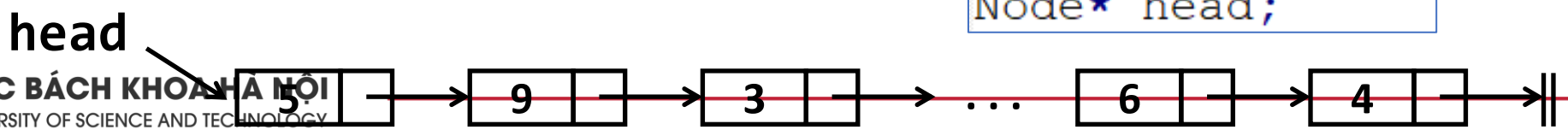
- Danh sách liên kết đơn (Singly linked list) là một danh sách có thứ tự các phần tử; các phần tử được kết nối với nhau thông qua một liên kết (link)
- ***Danh sách liên kết và Mảng***:

| | Danh sách liên kết | Mảng |
|---|---|---|
| **Kiểu dữ liệu** | Không cần đồng nhất | Đồng nhất |
| **Cấp phát bộ nhớ** | Phân tán | Liên tục, cạnh nhau |

## 2.1. Giới thiệu

- **Đặc điểm**:
  - Mỗi phần tử của danh sách gồm 2 phần: Dữ liệu và Con trỏ lưu trữ địa chỉ của phần tử kế tiếp trong danh sách;
  - Trong danh sách liên kết đơn, mỗi phần tử chỉ trỏ tới một phần tử kế tiếp;
  - Một danh sách liên kết có một phần tử đầu tiên – head và phần tử cuối, phần con trỏ của phần tử cuối cùng luôn  null.

```
struct Node{
    int value;
    Node* next;
};
Node* head;
```



# NỘI DUNG TIẾP THEO

1. Kiến thức cơ sở

   1.1. Con trỏ

   1.2. Cấu trúc

2. Danh sách liên kết đơn

**3. Thao tác trên danh sách liên kết**

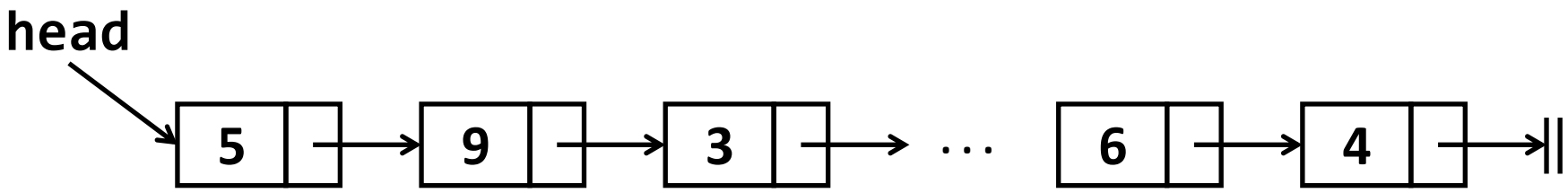
   3.1 Duyệt danh sách

   3.2 Tìm kiếm

# 3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

## 3.1. Duyệt danh sách

- **Nhiệm vụ**: Thăm mỗi phần tử của danh sách đúng một lần
- **Ý tưởng**: Dùng con trỏ **next** để truy cập đến phần tử tiếp theo



# 3. THAO TÁC TRÊN DANH SÁCH LIÊN KẾT ĐƠN

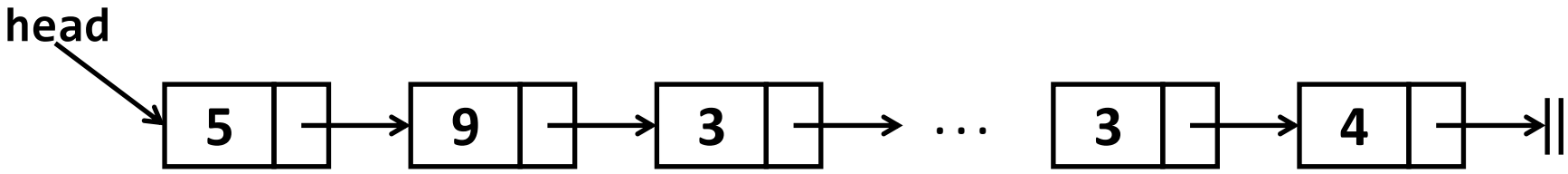
## 3.1. Duyệt danh sách

- **Nhiệm vụ**: Thăm mỗi phần tử của danh sách đúng một lần

```c
#include <stdio.h>

typedef struct Node{
    int value;
    struct Node* next;
}Node;

//Create a physical node
Node*makeNode(int v){
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}

//Print a list
void printList(Node* h){
  Node* p = h;
  while(p != NULL){
    printf("%d ",p->value);
    p = p->next;
  }
}

int main() {
    Node* head, *node1, *node2;
    head = makeNode(10);
    node1 = makeNode(20);
    node2 = makeNode(30);

    head->next = node1;
    node1->next = node2;

    printList(head);
    return 0;
}
```

## 3.2. Tìm kiếm

- **Nhiệm vụ**: Tìm phần tử đầu tiên của danh sách có giá trị bằng giá trị đầu vào
- **Ý tưởng**: Dùng con trỏ **next** để truy cập đến phần tử tiếp theo
  - Ví dụ tìm phần tử đầu tiên của danh sách có giá trị 3



## 3.1. Duyệt danh sách

- **Nhiệm vụ**: Thăm mỗi phần tử của danh sách đúng một lần

```c
#include <stdio.h>

typedef struct Node{
    int value;
    struct Node* next;
}Node;

//Create a physical node
Node*makeNode(int v){
    Node* p = (Node*)malloc(sizeof(Node));
    p->value = v;
    p->next = NULL;
    return p;
}

//Find a node with given value
Node * findFirst(Node * head, int val){
    Node* p = head;
    while(p != NULL){
        if(p->value == val)
            return p;
        p = p->next;
    }
    return NULL;
}
```

```c
int main() {
    Node* head, *node1, *node2;
    head = makeNode(10);
    node1 = makeNode(20);
    node2 = makeNode(30);

    head->next = node1;
    node1->next = node2;

    Node * res = findFirst(head, 20);
    if(res != NULL){
        printf("Found");
    }else{
        printf("Not Found");
    }

    return 0;
}
```

# TỔNG KẾT VÀ GỢI MỞ

## 1. Tổng kết:

Bài học đã giới thiệu **danh sách liên kết đơn** và 2 thao tác cơ bản trên danh sách đơn là **duyệt và tìm kiếm**

## 2. Gợi mở:

Thiết kế và cài đặt các thao tác khác trên danh sách

# NỘI DUNG

1. Chèn một phần tử vào đầu danh sách
2. Chèn một phần tử vào cuối danh sách
3. Chèn một phần tử vào trước một phần tử của danh sách

# MỤC TIÊU

***Sau bài học này, người học có thể:***

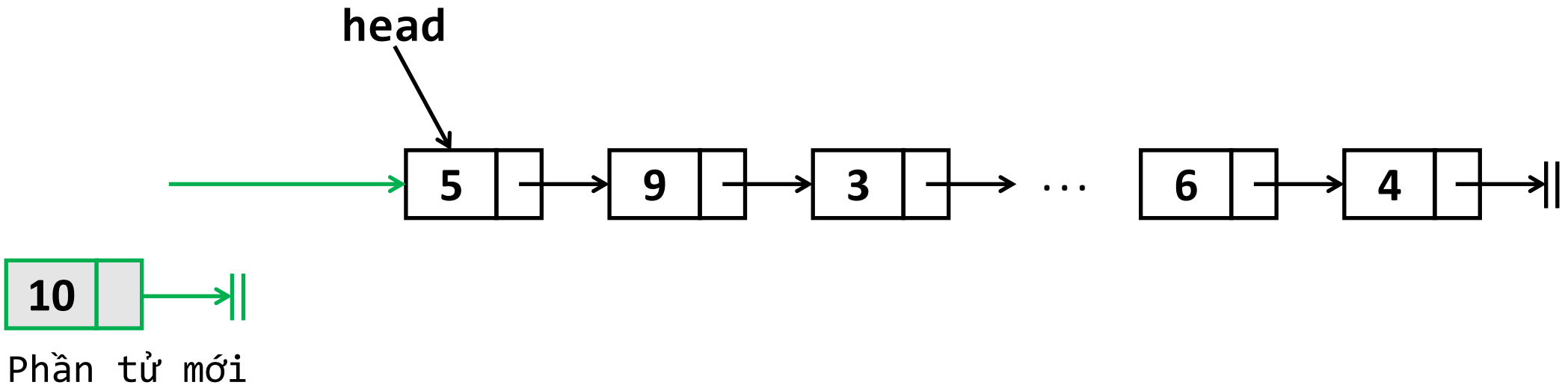
1. Hiểu thuật toán và cài đặt thành công ba thao tác cơ bản trên danh sách liên kết đơn: chèn một phần tử vào đầu, cuối và trước một phần tử trong danh sách liên kết đơn.

# NỘI DUNG TIẾP THEO

1. **Chèn một phần tử vào đầu danh sách**
2. Chèn một phần tử vào cuối danh sách
3. Chèn một phần tử vào trước một phần tử của danh sách

# 1. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

## 1.1. Ý tưởng




**Bước 1**: Tạo phần tử mới

**Bước 2**: Cập nhật head trỏ về phần tử mới

**Bước 3**: Cập nhật next của phần tử mới về đầu danh sách cũ, để biến phần tử mới thành phần tử đầu danh sách

# 1. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

## 1.2. Cài đặt

```
Node* insertFirst(Node * head, int v){
    Node * new_node = makeNode(v);
    if(head == NULL) return new_node;
    else{
        new_node->next = head;
        head = new_node;
        return head;
    }
}
```

Tạo phần tử mới

Nếu danh sách hiện tại rỗng, trả về phần tử mới

(1) Cập nhật head trỏ về phần tử mới
(2) Cập nhật next của phần tử mới về đầu danh sách cũ, để biến phần tử mới thành phần tử đầu danh sách

# NỘI DUNG TIẾP THEO

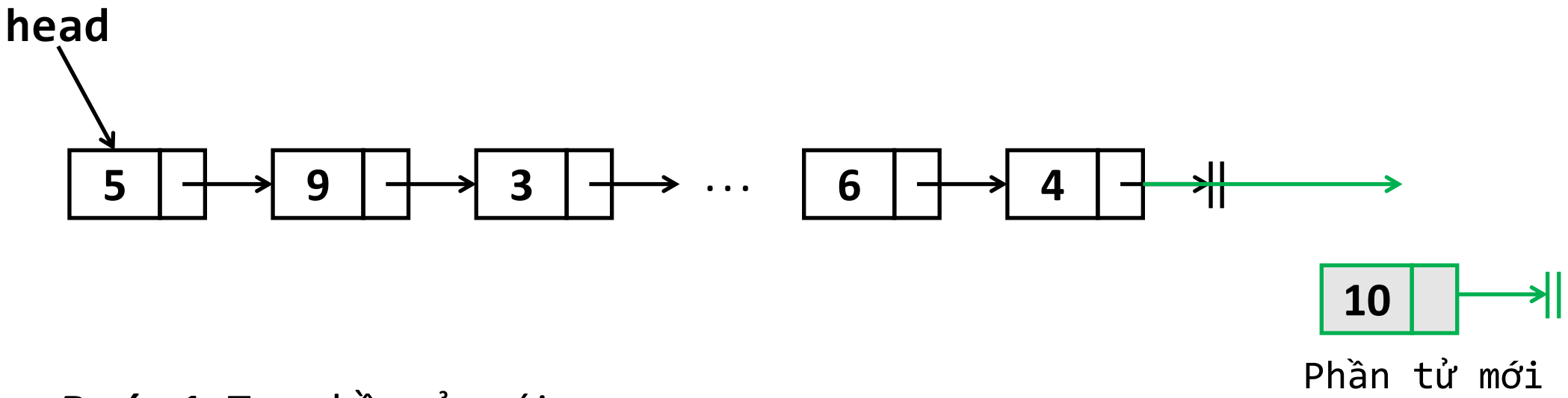
1. Chèn một phần tử vào đầu danh sách

**2. Chèn một phần tử vào cuối danh sách**

3. Chèn một phần tử vào trước một phần tử của danh sách

# 2. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

## 2.1. Ý tưởng




**Bước 1**: Tạo phần tử mới

**Bước 2**: Tìm phần tử cuối cùng của danh danh sách

**Bước 3**: Cập nhật next của phần tử cuối cùng trỏ tới phần tử mới

# 2. CHÈN MỘT PHẦN TỬ VÀO ĐẦU DANH SÁCH

## 2.2. Cài đặt

```
Node * findLastNode(Node * head){
    Node* p = head;
    while(p != NULL){
        if(p->next == NULL) return p;
        p = p->next;
    }
    return NULL;
}
```

Dùng vòng lặp để tìm phần tử cuối cùng của Danh sách

```
Node * insertLast(Node* head, int v){
    Node * new_node = makeNode(v);
    if(head == NULL) return new_node;
    else{
        Node * lastNode = findLastNode(head);
        lastNode->next = new_node;
        return head;
    }
}
```

(1) Tạo phần tử mới

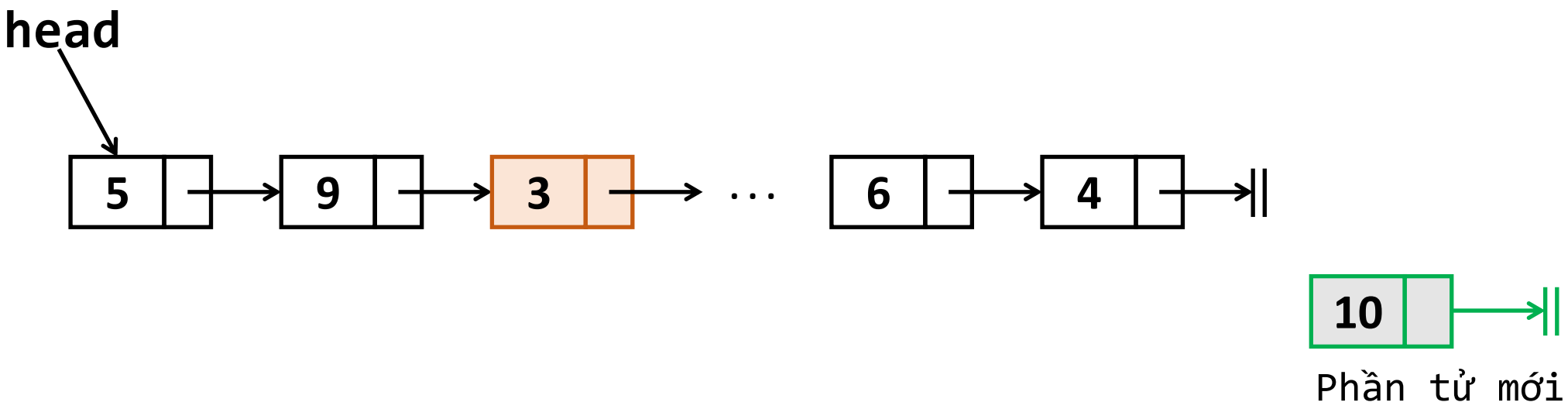
(2) Chèn phẩn tử mới vào sau phần tử cuối cùng của danh sách

# NỘI DUNG TIẾP THEO

1. Chèn một phần tử vào đầu danh sách
2. Chèn một phần tử vào cuối danh sách
3. **Chèn một phần tử vào trước một phần tử của danh sách**

# 3. CHÈN MỘT PHẦN TỬ VÀO TRƯỚC MỘT PHẦN TỬ

## 3.1. Ý tưởng




**Bước 1**: Tạo phần tử mới

**Bước 2**: Cập nhật next của phần từ mới là phần tử bị chèn trước

**Bước 3**: Cập nhật next của phần tử ngay trước phần tử bị chèn là phần tử mới

# 3. CHÈN MỘT PHẦN TỬ VÀO TRƯỚC MỘT PHẦN TỬ

## 3.1. Ý tưởng

```
Node* prevNode(Node* head, Node* p){
  Node* q = head;
  while(q != NULL){
    if(q->next == p) return q;
    q = q->next;
  }
  return NULL;
}
```

Tìm phần tử đứng trước một node cho trước

```
Node * insertBeforeNode(Node* head, Node* p, int v){
    Node* pp = prevNode(head, p);
    if(pp == NULL && p != NULL)  return head;
    Node* q = makeNode(v);
    if(pp == NULL){
        if(head == NULL)
            return q;
        q->next = head;
        return q;
    }
    q->next = p;
    pp->next = q;
    return head;
}
```

Thực hiện chèn

# TỔNG KẾT VÀ GỢI MỞ

1. **Tổng kết:**

   Cài đặt 3 thao tác chèn một phần tử mới vào một danh sách liên kết đơn: **chèn một phần tử vào đầu, cuối và trước một phần tử** của danh sách.

2. **Gợi mở**

   Thiết kế và cài đặt các thao tác khác trên danh sách

# NỘI DUNG

1. Xóa một phần tử của danh sách
2. Đảo ngược thứ tự các phần tử của danh sách

# MỤC TIÊU

***Sau bài học này, người học có thể:***

Hiểu thuật toán và cài đặt thành công hai thao tác cơ bản trên danh sách liên kết đơn:

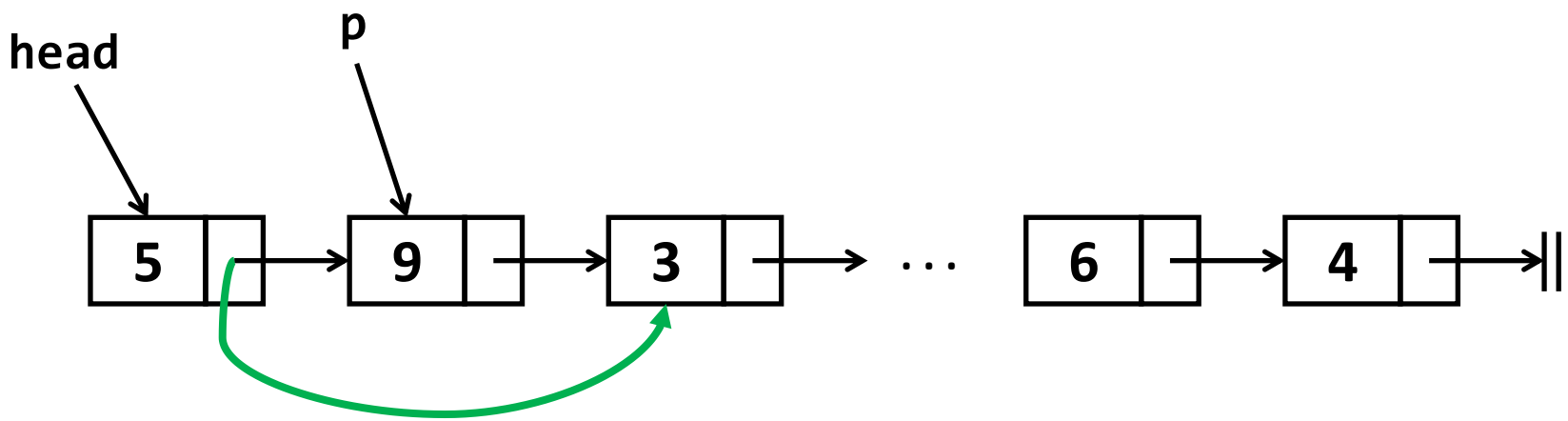
- xóa một phần tử khỏi danh sách
- đảo ngược thứ tự các phần tử trong danh sách

# NỘI DUNG TIẾP THEO

1. Xóa một phần tử của danh sách
2. Đảo ngược thứ tự các phần tử của danh sách

# 1. XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH

## 1.1. Ý tưởng




- Kiểm tra danh sách và phần tử cần xóa p có NULL không?
- Nếu phần tử cần xóa là đầu danh sách ➔ Đơn giản
- Dùng đệ quy để xóa

# 1. XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH

## 1.2. Cài đặt

```
//Remove
Node* removeNode(Node* head, Node * p){
    if(head == NULL || p == NULL) return head;
    if(head == p){
        head = head->next;
        free(p);
        return head;
    }else{
        head->next = removeNode(head->next, p);
        return head;
    }
}
```

Nếu danh sách hoặc phần tử cần xóa NULL, trả về đầu danh sách

Nếu phần tử cần xóa là phần tử đầu danh sách, đổi phần tử đầu và xóa phần tử cần xóa

Áp dụng kỹ thuật đệ quy để xóa

# NỘI DUNG TIẾP THEO

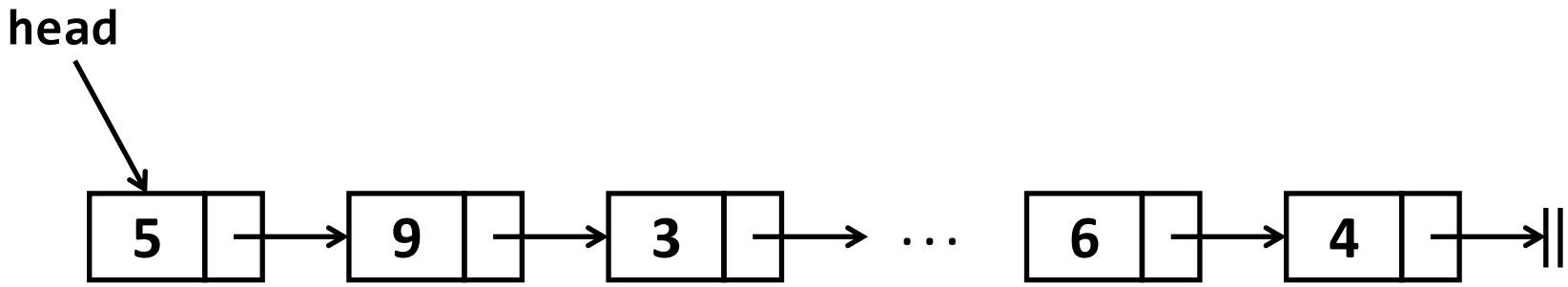
1. Xóa một phần tử của danh sách

2. Đảo ngược thứ tự các phần tử của danh sách

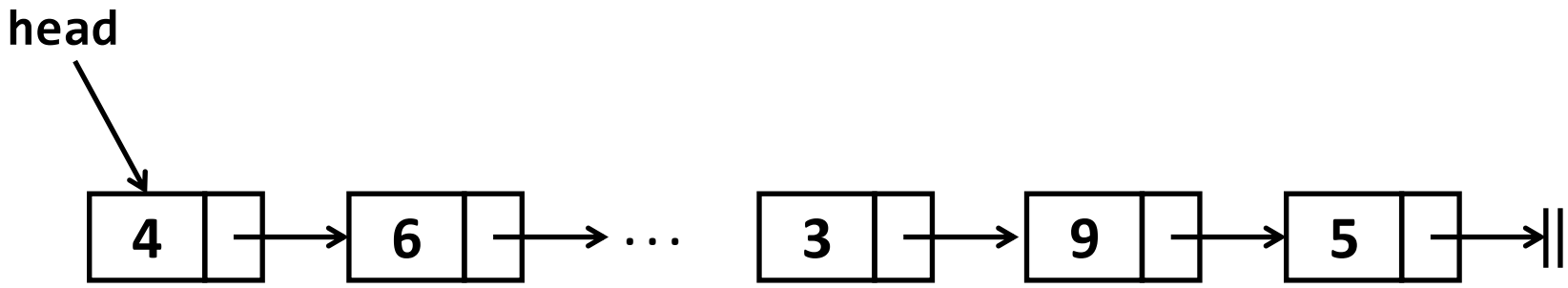
# 2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

## 2.1. Bài toán

Đầu vào:

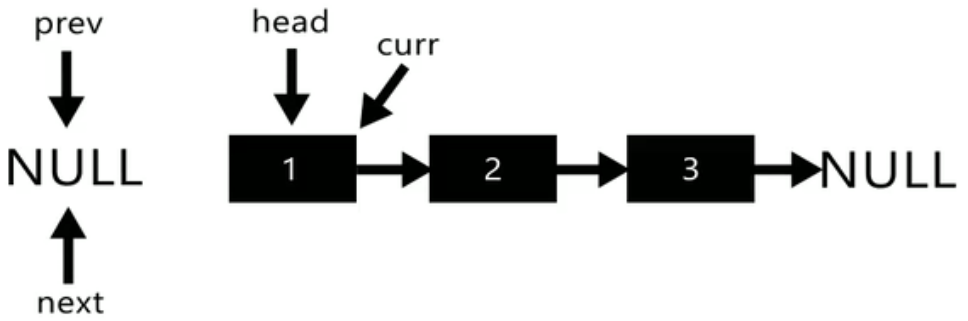



Đầu ra:



# 2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

## 2.2. Ý tưởng




```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Nguồn: https://www.geeksforgeeks.org/reverse-a-linked-list/

# 2. ĐẢO NGƯỢC THỨ TỰ CÁC PHẦN TỬ CỦA DANH SÁCH

## 2.3. Cài đặt

```
//Reverse
Node* reverse(Node* head){
    Node* cur = head;
    Node* next, *pre;
    pre = NULL;
    next = NULL;

    while(cur != NULL){
        //Get next
        next = cur->next;
        //Reverse
        cur->next = pre;
        //Move points ahead
        pre = cur;
        cur = next;
    }
    head = pre;
    return head;
}
```

# TỔNG KẾT VÀ GỢI MỞ

1. **Tổng kết:** Cài đặt 2 thao tác quan trọng trên danh sách liên kết đơn: **loại bỏ một phần tử ra khỏi danh sách và đảo ngược thứ tự các phần tử của danh sách.**

2. **Gợi mở:**

   - Danh sách liên kết đơn chỉ có có 1 liên kết giữa 2 phần tử liên tiếp trong danh sách, nếu có 2 liên kết thì thao tác trên danh sách có dễ dàng hơn không?

HUST

hust.edu.vn fb.com/dhbkhn

# THANK YOU !