

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

TRƯỜNG CÔNG NGHỆ THÔNG TIN
VÀ TRUYỀN THÔNG

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

TUẦN 7: Danh sách liên kết

ONE LOVE. ONE FUTURE.

NỘI DUNG

1 Kiến thức cơ sở

- Con trỏ
- Cấu trúc

2 Danh sách liên kết đơn

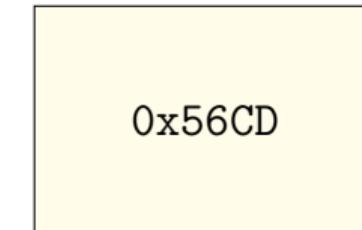
3 Thao tác trên danh sách liên kết

- Duyệt danh sách
- Tìm kiếm
- Chèn một phần tử vào đầu danh sách
- Chèn một phần tử vào cuối danh sách
- Chèn một phần tử vào trước một phần tử của danh sách
- Xóa một phần tử của danh sách
- Đảo ngược thứ tự các phần tử của danh sách

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.

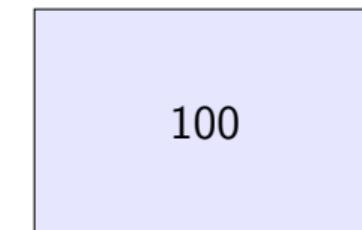
Địa chỉ trong ô nhớ: 0x13AB



Con trỏ p

trỏ tới

Địa chỉ trong ô nhớ: 0x56CD

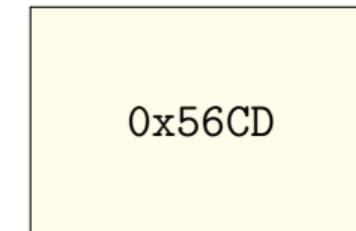


Biến a

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.

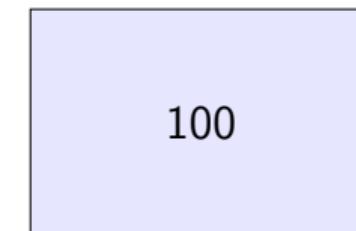
Địa chỉ trong ô nhớ: 0x13AB



Con trỏ p

trỏ tới

Địa chỉ trong ô nhớ: 0x56CD

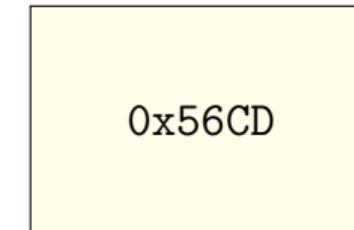


Biến a

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**

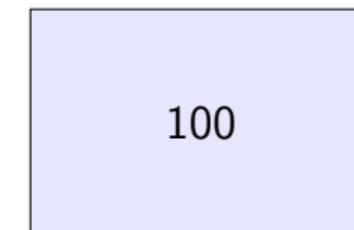
Địa chỉ trong ô nhớ: 0x13AB



Con trỏ p

trỏ tới

Địa chỉ trong ô nhớ: 0x56CD



Biến a

1.1. Con trỏ

- Ví dụ

In ra cùng một địa chỉ bộ nhớ

```
1 #include <stdio.h>
2 int main() {
3     int* pc, c;
4
5     c = 22;
6     printf("Address of c: %p\n", &c);
7     printf("Value of c: %d\n\n", c); ← In ra giá trị: 22
8
9     pc = &c;
10    printf("Address of pointer pc: %p\n", pc);
11    printf("Content of pointer pc: %d\n\n", *pc); ← In ra giá trị: 22
12
13    c = 11;
14    printf("Address of pointer pc: %p\n", pc);
15    printf("Content of pointer pc: %d\n\n", *pc); ← In ra giá trị: 11
16
17    *pc = 2;
18    printf("Address of c: %p\n", &c);
19    printf("Value of c: %d\n\n", c); ← In ra giá trị: 2
20
21 }
```

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

```
1 struct structureName {  
2     dataType member1;  
3     dataType member2;  
4     ...  
5 };
```

```
1 typedef struct Node {  
2     int value;           // Dữ liệu  
3     struct Node* next; // Liên kết  
4 } 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
 - Cấu trúc là một tập hợp các biến, những biến này có thể có kiểu dữ liệu khác nhau
-
- **TNode* q:** q là con trỏ trả về 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 trả vào vùng nhớ được cấp phát

```
1     typedef struct Node {  
2         int value;           // Dữ liệu  
3         struct Node* next; // Liên  
4         ket  
5     } Node;
```

2.1. Giới thiệu

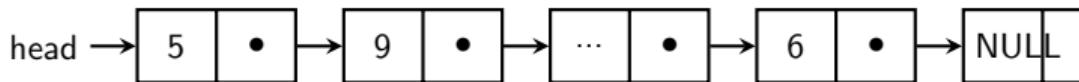
- **Định nghĩa:** Danh sách liên kết đơn (Singly linked list) là một danh sách có thứ tự các phần tử; các phần tử được kết nối với nhau thông qua một liên kết (link)
- **Danh sách liên kết và Mảng**

Đặc điểm	Danh sách liên kết	Mảng (Array)
Kiểu dữ liệu	Không cần đồng nhất	Đồng nhất
Bộ nhớ	Phân tán (Cấp phát động)	Liên tục, cạnh nhau
Kích thước	Linh hoạt	Cố định

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.

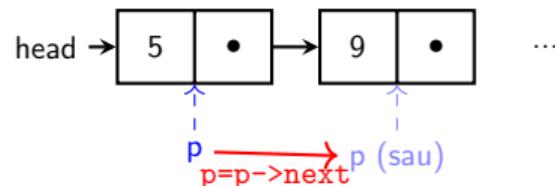


```
1 struct Node{  
2     int value;  
3     Node* next;  
4 };  
5 Node* head;
```

3.1. Duyệt danh sách

- **Nhiệm vụ:** Thăm mỗi phần tử đúng 1 lần.
- **Ý tưởng:** Dùng con trỏ **next** để truy cập đến phần tử tiếp theo.

```
1 void printList(Node* h) {  
2     Node* p = h;  
3     while (p != NULL) {  
4         printf("%d ", p->value);  
5         p = p->next; // Nhảy sang node sau  
6     }  
7 }
```



3.1. Duyệt danh sách

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

```
1 #include<stdio.h>
2 #include <stdlib.h> //thư viện để dùng
3     malloc
4
5     typedef struct Node{
6         int value;
7         struct Node* next;
8     }Node;
9
10    //Create a physical node
11    Node*makeNode(int v){
12        Node* p = (Node*)malloc(sizeof(Node));
13        p->value = v;
14        p->next = NULL;
15        return p;
16    }
```

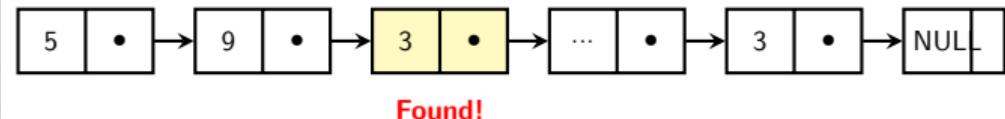
```
1 //Print a list
2 void printList(Node* h){
3     Node* p = h;
4     while(p != NULL){
5         printf("%d",p->value);
6         p = p->next;
7     }
8
9
10    int main(){
11        Node* head, *node1, *node2;
12        head = makeNode(10);
13        node1 = makeNode(20);
14        node2 = makeNode(30);
15
16        head->next = node1;
17        node1->next = node2;
18
19        printList(head);
20
21    }
```

3.2 Tìm kiếm (Search)

- **Nhiệm vụ:** Tìm Node đầu tiên có giá trị bằng giá trị đầu vào.
- **Ý tưởng:** Dùng con trỏ **next** để truy cập đến phần tử tiếp theo

```
1 Node* findFirst(Node* head, int val) {  
2     Node* p = head;  
3     while (p != NULL) {  
4         if (p->value == val) {  
5             return p; // Tìm thấy  
6         }  
7         p = p->next;  
8     }  
9     return NULL; // Không thấy  
10 }
```

Ví dụ: Tìm giá trị 3:



Thao tác trên danh sách liên kết đơn

```
1 #include <stdio.h>
2 #include <stdlib.h> // thư viện để dùng malloc
3 typedef struct Node{
4     int value;
5     struct Node* next;
6 } Node;
7 // Create a physical node
8 Node* makeNode(int v){
9     Node* p = (Node*)malloc(sizeof(Node));
10    p->value = v;
11    p->next = NULL;
12    return p;
13 }
14 // Find a node with given value
15 Node * findFirst(Node * head, int val){
16     Node* p = head;
17     while(p != NULL){
18         if(p->value == val)
19             return p;
20         p = p->next;
21     }
22     return NULL;
```

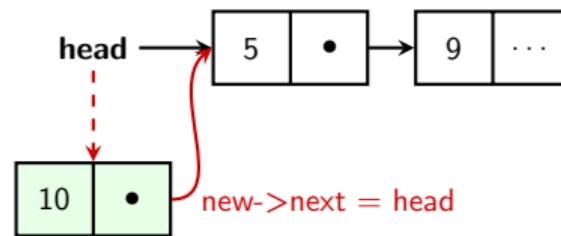
```
1 int main() {
2     Node* head, *node1, *node2;
3     head = makeNode(10);
4     node1 = makeNode(20);
5     node2 = makeNode(30);
6
7     head->next = node1;
8     node1->next = node2;
9
10    // Phần tìm kiếm
11    Node * res = findFirst(head,
12                           20);
13    if(res != NULL){
14        printf("Found");
15    }else{
16        printf("Not Found");
17    }
18
19    return 0;
}
```

Chèn phần tử vào ĐẦU danh sách

Ý tưởng:

- ① Tạo node mới: `Node* newNode = makeNode(v)`
- ② Cập nhật `next` của phần tử của phần tử mới về đầu danh sách cũ, để biến phần tử mới thành phần tử đầu danh sách: `new_node->next = head`.
- ③ Cập nhật `head` trả về phần tử mới: `head = new_node`.

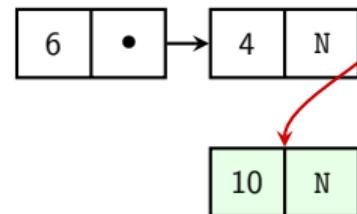
```
1 Node* insertFirst(Node* head, int v) {  
2     Node* newNode = makeNode(v);  
3     if (head == NULL) return newNode;  
4  
5     newNode->next = head;  
6     head = newNode;  
7     return head;  
8 }
```



Chèn phần tử vào CUỐI danh sách

Yêu cầu: Duyệt đến cuối (`p->next == NULL`), sau đó trỏ `p->next` vào node mới.

```
1 Node * findLastNode (Node * head){  
2     Node* p = head;  
3     while(p != NULL){  
4         if(p->next == NULL) return p;  
5         p = p->next;  
6     }  
7     return NULL  
8 }
```



Chèn phần tử vào CUỐI danh sách

Ý tưởng:

- ① Tạo phần tử mới:

```
Node* newNode = makeNode(v);
```

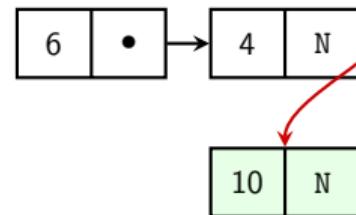
- ② Tìm phần tử cuối cùng của danh sách:

```
Node* last = findLastNode(head);
```

- ③ Cập nhật next của phần tử cuối cùng trả tối
phần tử mới:

```
last->next = newNode;
```

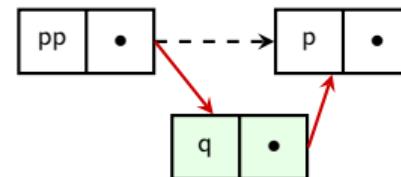
```
1 Node* insertLast(Node* head, int v) {  
2     Node* newNode = makeNode(v);  
3     if (head == NULL) return newNode;  
4  
5     // Tìm node cuối  
6     Node* last = findLastNode(head);  
7     last->next = newNode;  
8  
9 }
```



Chèn vào TRƯỚC một phần tử

Yêu cầu: Cần tìm node pp đứng ngay trước node mục tiêu p.

```
1 Node* prevNode(Node* head, Node* p) {  
2     Node* q = head;  
3     while(q != NULL) {  
4         if(q->next == p) return q;  
5         q = q->next;  
6     }  
7     return NULL;  
8 }
```

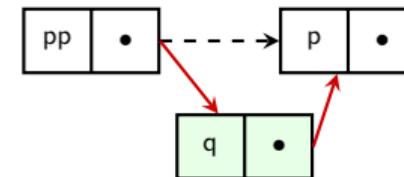


Chèn vào TRƯỚC một phần tử

Ý tưởng:

- ① Tạo phần tử mới: `Node* q = makeNode(v)`
- ② Cập nhật next của phần tử mới là phần tử bị chèn trước: `q->next = p;`
- ③ Cập nhật next của phần tử ngay trước phần tử bị chèn là phần tử mới: `pp->next = q;`

```
1 Node* insertBeforeNode(Node* head, Node* p, int v){  
2     if (p == NULL) return head;  
3     // Tìm prev của p  
4     Node* pp = prevNode(head, p);  
5  
6     if (pp == NULL && head == p)  
7         return insertFirst(head, v);  
8     if (pp == NULL) return head; // ko tìm thấy  
9  
10    Node* q = makeNode(v);  
11    q->next = p;  
12    pp->next = q;  
13    return head;  
14}
```



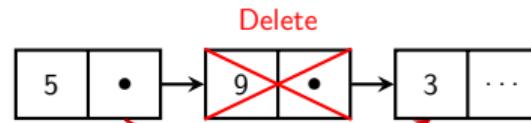
Xóa một phần tử (Recursive)

Ý tưởng:

- Kiểm tra danh sách và phần tử cần xóa p có NULL không?:
`if (head == NULL p == NULL)`
- Nếu phần tử cần xóa là phần tử đầu danh sách (`if (head == p)`), đổi phần tử đầu (`head = head->next;`) và xóa phần tử cần xóa(`free(p)`)

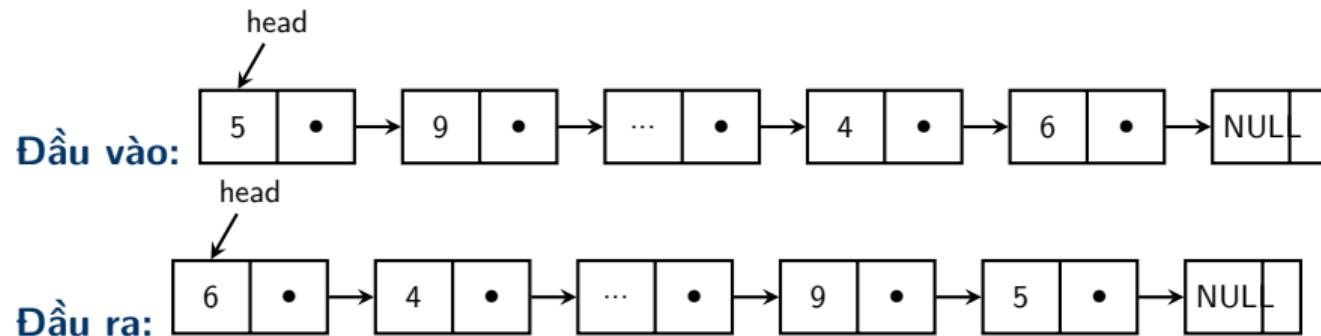
```
1 Node* removeNode(Node* head, Node* p) {  
2     if (head == NULL p == NULL)  
3         return head;  
4     if (head == p) {  
5         head = head->next;  
6         free(p);  
7         return head;  
8     }  
9     // Để quy  
10    head->next = removeNode(head->next, p);  
11    return head;  
12 }
```

- Áp dụng kỹ thuật đệ quy để xóa:
`head->next = removeNode(head->next, p);`



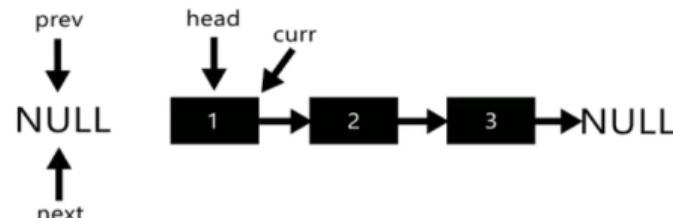
Đảo ngược danh sách (Reverse)

2.1. Bài toán



Đảo ngược danh sách (Reverse)

2.2. **Ý tưởng:** Dùng 3 con trỏ prev, cur, next để đảo chiều mũi tên.

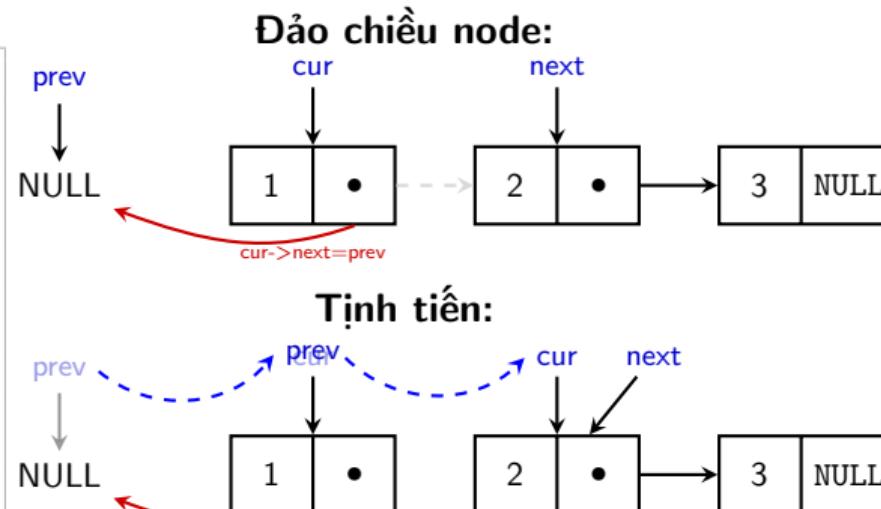


```
while (current != NULL)
{
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
}
*head_ref = prev;
```

Đảo ngược danh sách (Reverse)

2.3. Cài đặt:

```
1 Node* reverse(Node* head) {  
2     Node* cur = head;  
3     Node* prev = NULL;  
4     Node* next = NULL;  
5  
6     while (cur != NULL) {  
7         next = cur->next; // Lưu next  
8         cur->next = prev; // Đảo chiều  
9  
10        // Tịnh tiến  
11        prev = cur;  
12        cur = next;  
13    }  
14    return prev; // prev là head mới  
15 }
```



Tổng kết

Bài học đã giới thiệu về:

- Cấu trúc dữ liệu DSLK đơn.
- Các thao tác: Duyệt, Tìm kiếm, Chèn (Đầu, Cuối, Giữa), Xóa, Đảo ngược.

Gợi mở

Danh sách liên kết đơn chỉ có 1 chiều. Nếu có 2 liên kết (trước/sau) thì thao tác có dễ dàng hơn không? → **Danh sách liên kết đôi.**

A large, stylized graphic of the letters "HUST" is composed of numerous small red dots arranged in a circular, radiating pattern against a dark blue background.

HUST

THANK YOU!