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

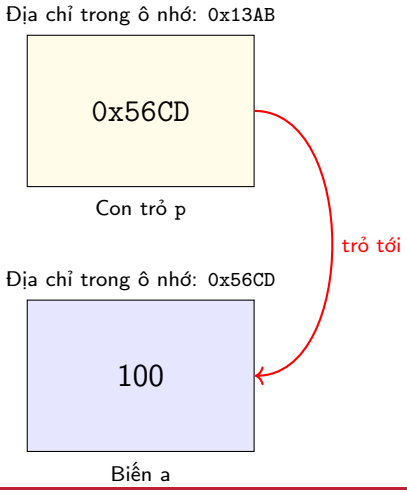
ONE LOVE. ONE FUTURE.

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

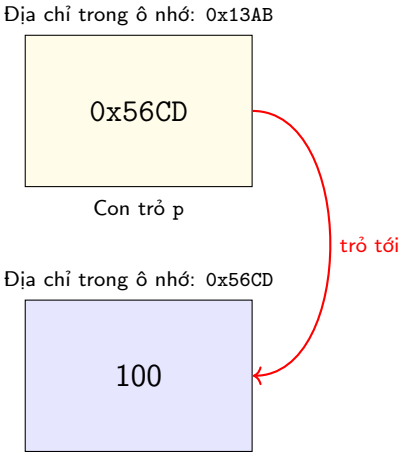
1.1 Con trỏ

- **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.



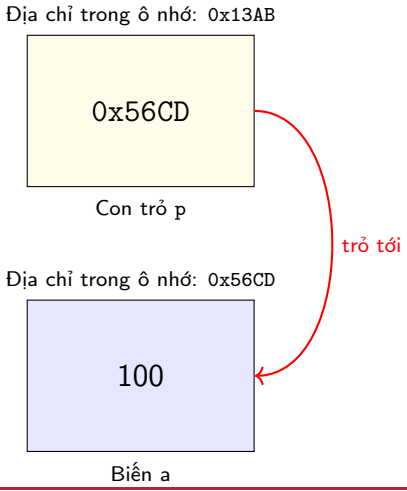
1.1 Con trỏ

- **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

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

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

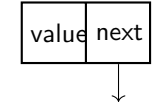
- **TNode* q:** **q** là con trỏ trỏ đến 1 biến có kiểu **TNode**
- **Q→a:** truy nhập đến thành phần a của kiểu cấu trúc
- `q = (TNode*)malloc(sizeof(TNode))`: cấp phát bộ nhớ cho 1 kiểu **TNode** và **q** trỏ vào vùng nhớ được cấp phát

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

Cấu trúc 1 Node



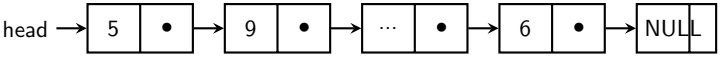
2.1. Giới thiệu

- **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.

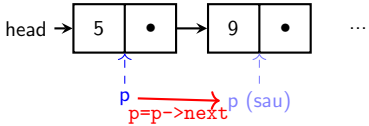



```
1  struct Node{
2      int value;
3      Node* next;
4  };
5  Node* head;
```

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

```c
#include<stdio.h>
#include <stdlib.h> //thư viện để dùng malloc
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;
}
```

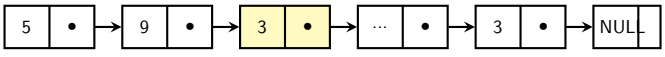
```c
//Print a list
void printList(Node* h){
    Node* p = h;
    while(p != NULL){
        printf("%d",p->value);
        p = p->next;
    }
}

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 (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**:



Found!

Thao tác trên danh sách liên kết đơn

```c
#include <stdio.h>
#include <stdlib.h> //thư viện để dùng malloc
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;
}
```

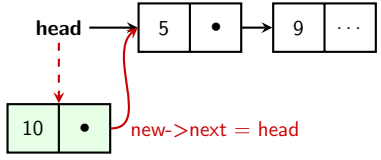
```c
int main() {
    Node* head, *node1, *node2;
    head = makeNode(10);
    node1 = makeNode(20);
    node2 = makeNode(30);

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

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

    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 trỏ về phần tử mới: `head = new_node`.

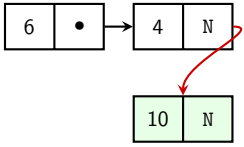
```
1  Node* insertFirst(Node* head, int v) {
2      Node* newNode = makeNode(v);
3      if (head == NULL) return newNode;
4
5      newNode->next = head;
6      head = newNode;
7      return head;
8  }
```



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  }
```

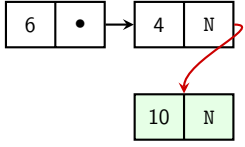


Chèn phần tử vào CUỐI danh sách

Ý tưởng:

1. Tạo phần tử mới:
   `Node* newNode = makeNode(v);`
2. Tìm phần tử cuối cùng của danh danh sách:
   `Node* last = findLastNode(head);`
3. Cập nhật next của phần tử cuối cùng trỏ tới phần tử mới:
   last->next = newNode;

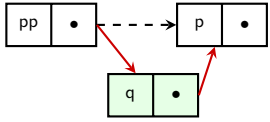
```
1  Node* insertLast(Node* head, int v) {
2      Node* newNode = makeNode(v);
3      if (head == NULL) return newNode;
4
5      // Tìm node cuối
6      Node* last = findLastNode(head);
7      last->next = newNode;
8      return head;
9  }
```



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  }
```

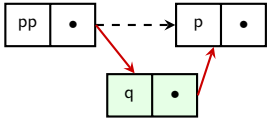


Chèn vào TRƯỚC một phần tử

Ý tưởng:

1. Tạo phần tử mới: `Node* q = makeNode(v)`
2. Cập nhật next của phần từ 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 }
```



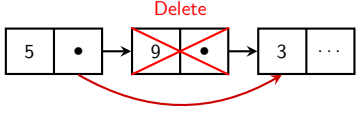
Xóa một phần tử (Recursive)

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))

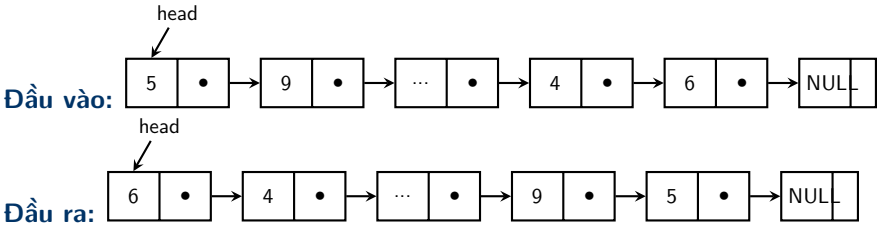
```
1  Node* removeNode(Node* head, Node* p) {
2      if (head == NULL  p == NULL)
3          return head;
4      if (head == p) {
5          head = head->next;
6          free(p);
7          return head;
8      }
9      // Đệ quy
10     head->next = removeNode(head->next, p);
11     return head;
12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

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



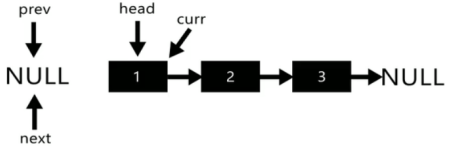
Đảo ngược danh sách (Reverse)

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.



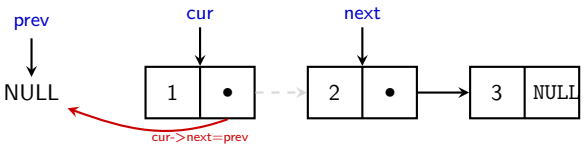

```
while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    *head_ref = prev;
```

Đảo ngược danh sách (Reverse)

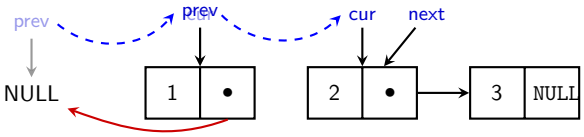
2.3.Cài đặt:

```
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 }
```

Đảo chiều node:



Tịnh tiến:



TỔNG KẾT VÀ GỢI MỞ

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? $\rightarrow$ **Danh sách liên kết đôi**.

THANK YOU!