

# DANH SÁCH LIÊN KẾT

## với ngôn ngữ C++, C



[2017 – 02 – 11]

Biên soạn bởi: Nguyễn Trung Thành  
<https://www.facebook.com/abcxyztcit>

“ Học từ cái đáy của thế giới đi lên là cách duy nhất bạn trở thành master. ”

Nguyễn Trung Thành

## Mục lục

A.	Giới thiệu .....	1
1.	Điều kiện để học được tài liệu này .....	1
2.	Giới thiệu danh sách liên kết (DSLK).....	1
B.	Từ trừu tượng đến thực tế.....	2
1.	Node .....	2
2.	Thế nào gọi là “nối với nhau” ? .....	3
3.	Demo code .....	6
4.	Một số nhận xét.....	10
C.	Các thao tác cơ bản với DSLK .....	11
1.	Duyệt DSLK .....	11
2.	Create node .....	12
3.	Giải phóng bộ nhớ.....	13
4.	Is empty.....	14
5.	Insert head .....	15
6.	Remove head.....	19
D.	Kỹ thuật nâng cao.....	21
	Duyệt DSLK bằng 2 con trỏ liên tiếp .....	21
E.	Một số thao tác mở rộng .....	24
1.	Insert tail .....	24
2.	Remove tail.....	28
3.	Thao tác với 1 node nằm ở giữa DSLK (chèn, xóa).....	29
F.	Mở rộng DSLK với tail .....	29
G.	Các loại DSLK.....	31
H.	Bài tập .....	32
I.	Tổng kết.....	34



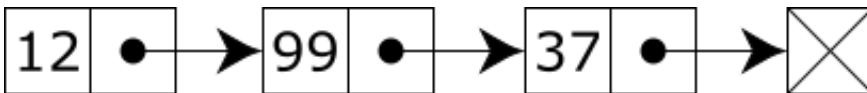
## A. Giới thiệu

### 1. Điều kiện để học được tài liệu này


- Ngôn ngữ C và C++ (trong tài liệu sử dụng C++ cho tiện lợi).
- Kiến thức vững chắc về con trỏ trong C/C++.
- Đã học qua lập trình cơ bản.

### 2. Giới thiệu danh sách liên kết (DSLK)

Danh sách liên kết (DSLK) trong tiếng Anh gọi là “linked list”, là 1 trong những cấu trúc dữ liệu thuộc dạng cơ bản. Hình ảnh của nó là như sau:



Danh sách liên kết chứa các nút (tiếng Anh gọi là “node”).

- Ở hình trên, ta thấy **có 3 node** mang giá trị 12, 99 và 37.
- Kí hiệu  không phải là 1 node mà là kí hiệu của “kết thúc danh sách liên kết”.
  - Kí hiệu này gọi là NIL hoặc NULL. Trong tài liệu sử dụng “NULL”.

Nhìn vào hình trên ta dễ dàng thấy được các node trỏ lần lượt từ trái qua phải:

- Node giá trị 12 trỏ vào node giá trị 99.
- Node giá trị 99 trỏ vào node giá trị 37.
- Node giá trị 37 trỏ vào NULL.

Vâng, DSLK là vậy đấy. DSLK khiến ta liên tưởng đến hình ảnh 1 đoàn tàu hỏa: toa phía trước kéo toa phía sau, toa 1 kéo toa 2, toa 2 kéo toa 3,....

## B. Từ trừu tượng đến thực tế

Đọc phần giới thiệu DSLK, ta thấy nó có nét gì đó khá trừu tượng, mà trừu tượng thì làm sao mà lập trình được ? Bây giờ ta sẽ làm rõ hơn nhé.

Danh sách liên kết là các node nối với nhau

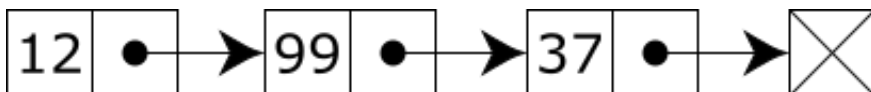
Ta sẽ làm rõ 2 ý “node” và “nối với nhau”, làm rõ được là lập trình được liền hehe.

### 1. Node

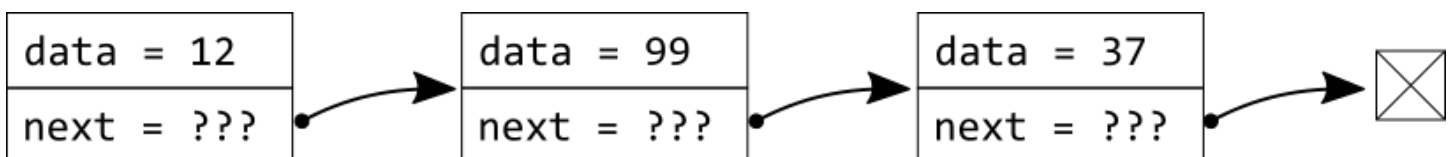
1 node gồm 2 thành phần:

- Dữ liệu (data)
- Con trỏ next

Lấy lại hình ảnh ở phần giới thiệu



Chi tiết hóa lên thì hình ảnh DSLK sẽ như sau:



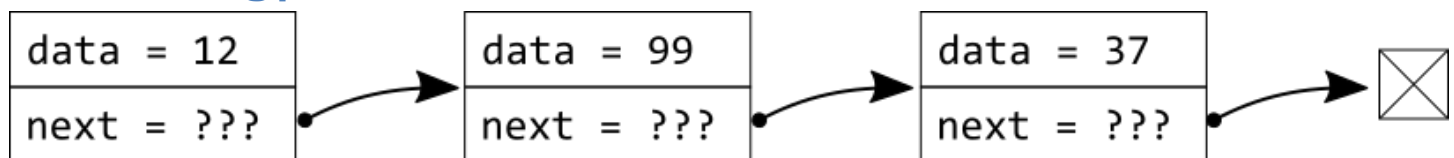
Tạm thời ta chưa cần biết “next” là cái quái gì cả.

Trong lập trình, ta sẽ cấu trúc Node như sau:

```
struct Node
{
    int data;
    Node *next;
};
```

Nhìn vào struct trên, bạn đã bắt đầu hình dung ra được next là cái gì rồi chứ ?

## 2. Thế nào gọi là “nối với nhau” ?



Dễ dàng thấy được node 1 nối với node 2, node 2 nối với node 3, node 3 nối với NULL.

Xét node 1 (data = 12) và node 2 (data = 99):

- Làm sao để từ node 1 ta “đi đến” được node 2 ? Nhờ vào việc “nối với nhau”.
- **Bản chất “nối với nhau” là việc tạo ra liên kết.**

➔ **next của node 1 lưu địa chỉ của node 2.**

```
int main()
{
    Node node1;
    Node node2;

    node1.data = 12;
    node2.data = 99;

    // CHÚ Ý: node.next lưu địa chỉ của node 2
    node1.next = &node2;

    return 0;
}
```

Nhờ vào việc lưu địa chỉ mà ta có thể từ node1 đi đến node2 được rồi, thích quá !!!

Nào, hãy xem đây

```
#include <stdio.h>

struct Node
{
    int data;
    Node *next;
};

int main()
{
    Node node1;
    Node node2;

    node1.data = 12;
    node2.data = 99;

    node1.next = &node2;

    printf("%d \n", node1.data);
    printf("%d \n", node1.next->data);

    return 0;
}
```

Chạy chương trình:

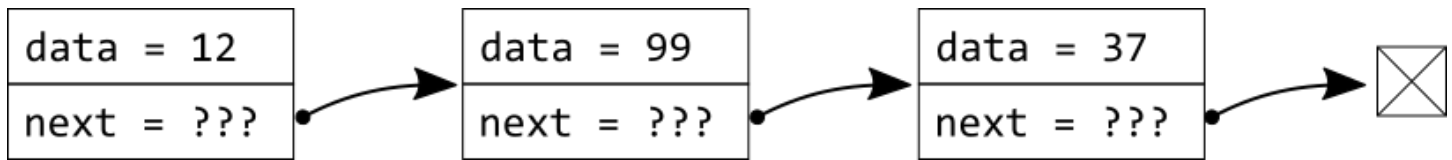
```
12
99
```

Nhờ vào `node1.next->data` mà ta đã gián tiếp lấy được giá trị `node2.data`

Như vậy là từ node 1 ta đã đi đến được node 2 và lấy giá trị data của node 2 rồi !!!

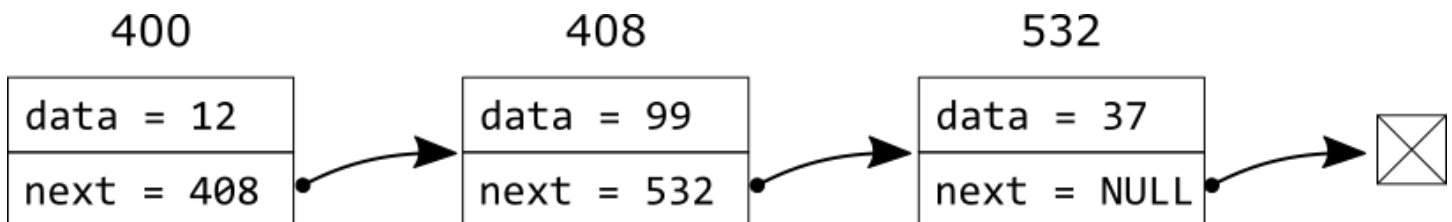
➔ Nhờ vào sự liên kết bằng con trỏ. Con trỏ next đã tạo liên kết giữa các node với nhau.

Lấy lại hình minh họa. Với node1 (data = 12), node 2 (data = 99) và node 3 (data = 37).



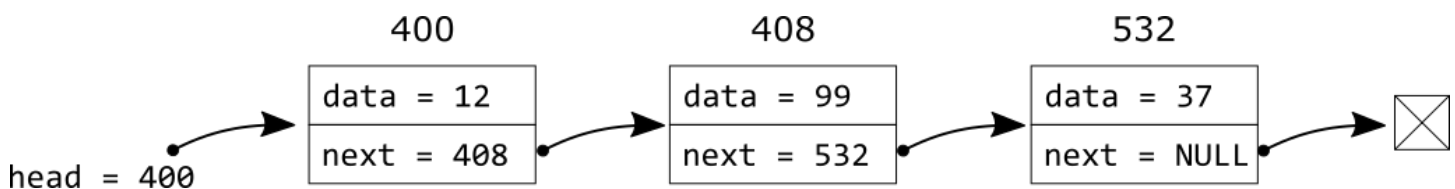
Nhờ vào sự thí nghiệm ở trên, ta có thể rút ra:

- node1.next lưu địa chỉ của node2
- node2.next lưu địa chỉ của node3
- node3.next có giá trị = NULL



(Giả sử node1 có địa chỉ là 400, node2 có địa chỉ là 408, node3 có địa chỉ là 532)

Tuy nhiên trong thực tế, ta sẽ cần thêm con trỏ head như sau:



Hoàn thành danh sách liên kết cơ bản, hahaha=))) Đó, danh sách liên kết là vậy đó, có gì đâu mà khó? Hiểu được thì bạn sẽ lập trình được.



### 3. Demo code

Đây là code hoàn chỉnh minh họa 1 DSLK đơn giản nhất, dựa vào tất cả những gì bạn đã học trước đó. Ta sẽ tạo 1 DSLK gồm 3 node và in nội dung DSLK ra màn hình.

```
#include <stdio.h>

struct Node
{
    int data;
    Node *next;
};

int main()
{
    Node node1, node2, node3;
    Node *head = NULL;

    node1.data = 12;
    node2.data = 99;
    node3.data = 37;

    // TẠO DANH SÁCH LIÊN KẾT
    head = &node1;
    node1.next = &node2;
    node2.next = &node3;
    node3.next = NULL;

    // DUYỆT DSLK
    Node *p = head;

    while (p)
    {
        printf("%d ", p->data);
        p = p->next;
    }

    return 0;
}
```

Ghi chú: while (p) tức là while (p != NULL). Bây giờ chạy chương trình xem:

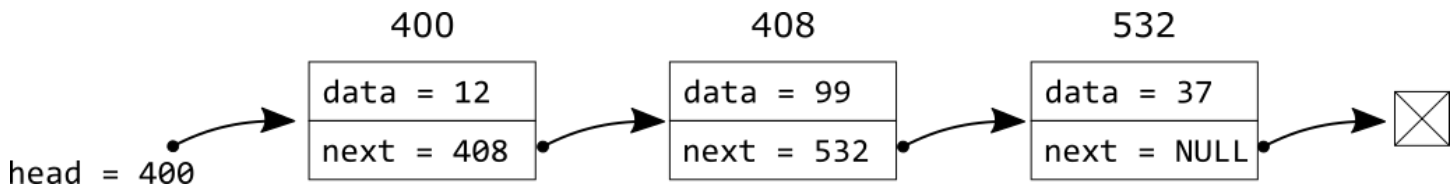
```
12  99  37
```

Đơn giản, dễ hiểu phải không nào ???

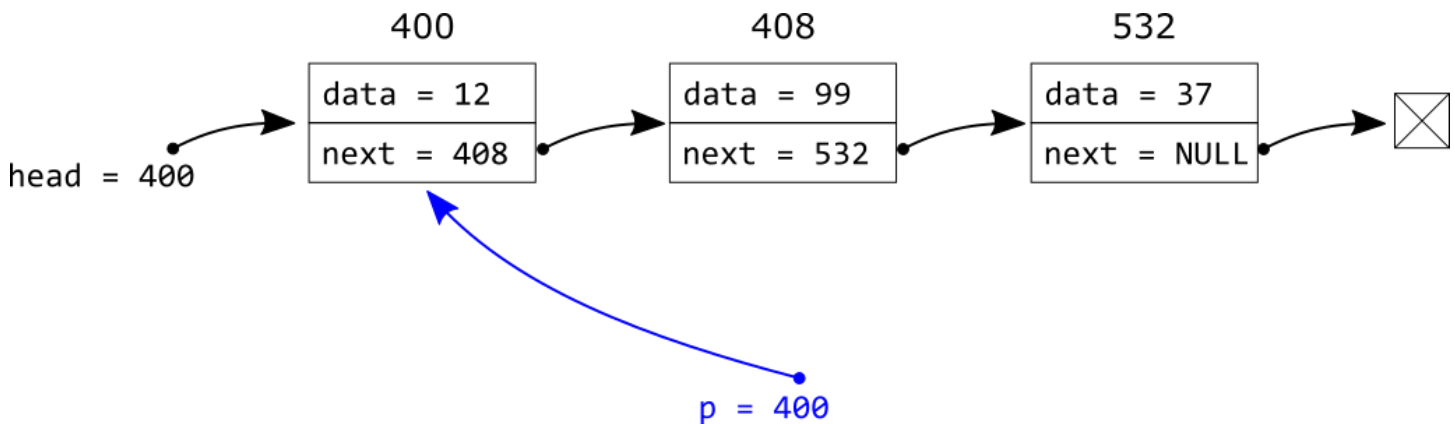


Minh họa, giải thích chi tiết đoạn code trên đến mức “không thể nào chi tiết hơn được nữa”.

Ta sẽ minh họa việc duyệt DSLK và in ra giá trị mỗi node.



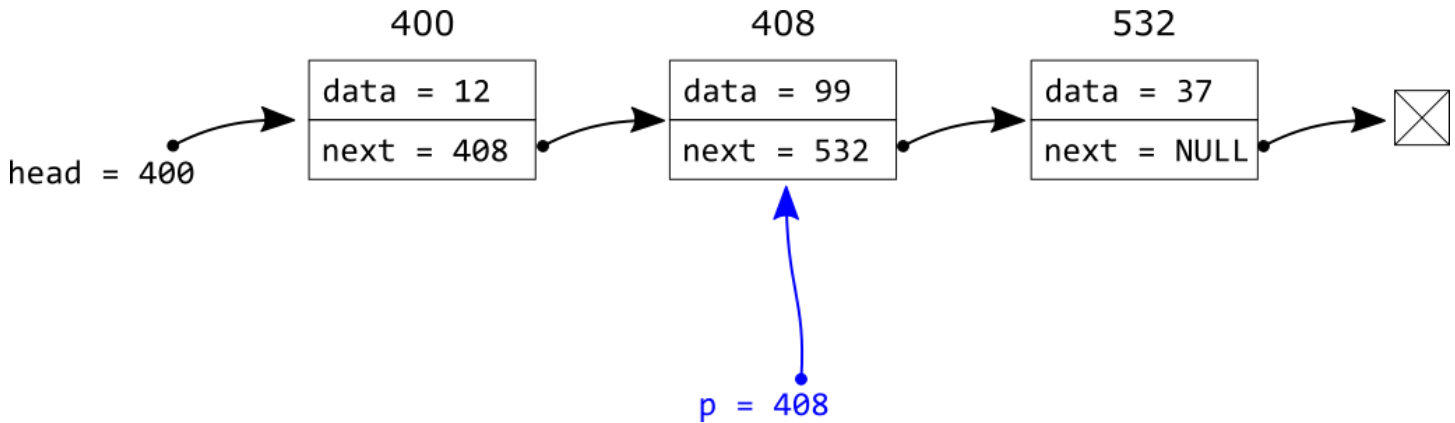
Ở bước đầu việc duyệt DSLK, ta có con trỏ p và gán  $p = \text{head} = 400$ .



Bắt đầu nhảy vào vòng lặp while.

- Vì  $p = 400 \rightarrow p$  đang trỏ đến node đầu tiên, suy ra  $p \rightarrow \text{data} = 12$ . Ta in số 12 ra màn hình.
- Ta cũng có  $p \rightarrow \text{next} = 408$ . Suy ra  $p = p \rightarrow \text{next}$  tức là gán  $p = 408$ .

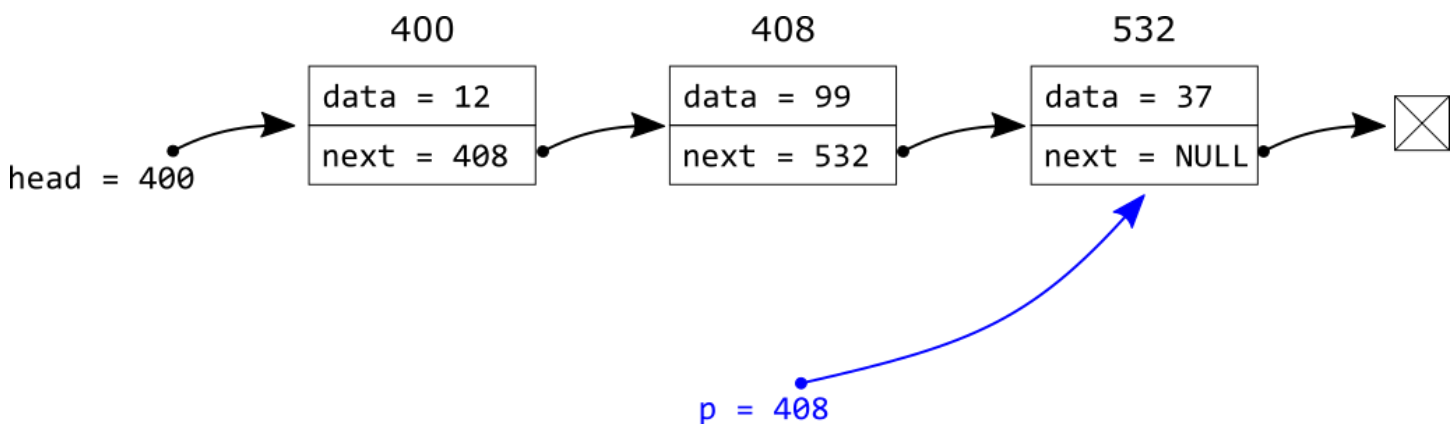
Lúc này  $p = 408$  tức là  $p$  đang trỏ đến node thứ 2 như sau:



Tiếp tục vòng lặp while:

- In số 99 ra màn hình.
- Vì  $p$  trỏ đến node thứ 2 nên  $p \rightarrow \text{next} = 532$ . Suy ra  $p = p \rightarrow \text{next}$  tức là gán  $p = 532$ .

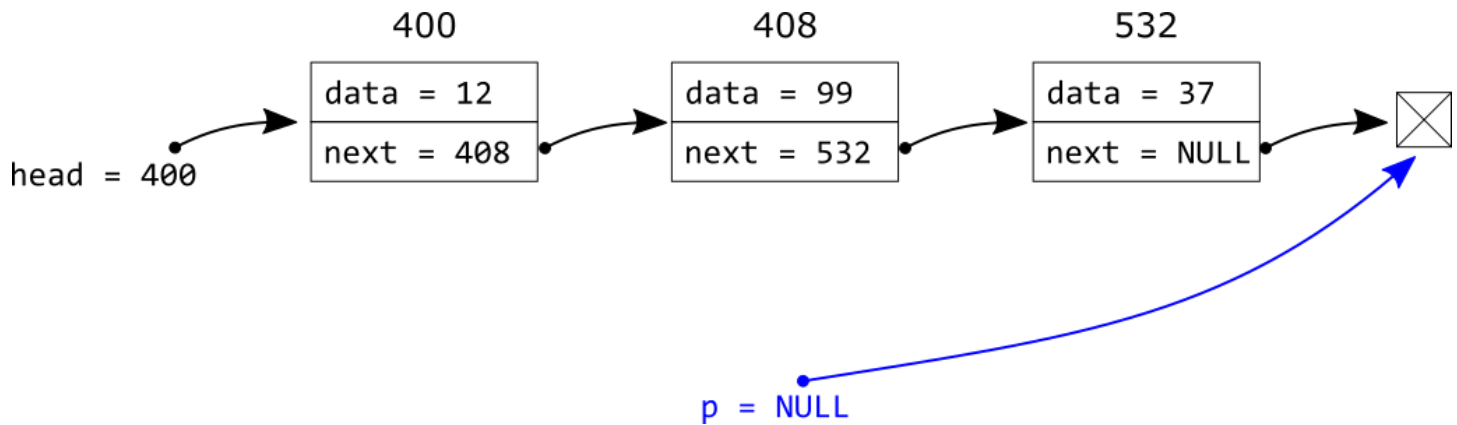
Lúc này  $p = 532$  tức là  $p$  đang trỏ đến node thứ 3 như sau:



Tiếp tục vòng lặp while:

- In số 37 ra màn hình.
- Vì  $p$  trỏ đến node thứ 3 nên  $p \rightarrow \text{next} = \text{NULL}$ . Suy ra  $p = p \rightarrow \text{next}$  tức là gán  $p = \text{NULL}$ .

Lúc này  $p = \text{NULL}$ .



Vì  $p = \text{NULL}$  nên không thỏa điều kiện của vòng lặp `while` nữa  $\rightarrow$  dừng vòng lặp `while`.

Kết luận: ta đã duyệt DSLK và in ra 3 giá trị của 3 node: 12, 99, 37.

## 4. Một số nhận xét

Dựa vào việc minh họa duyệt DSLK ở trên ta có nhận xét:

- Muốn lấy giá trị data = 12 của node 1, ta có thể lấy trực tiếp từ head.
- Muốn lấy giá trị data = 99 của node 2, ta phải đi theo thứ tự node1 → node2.
- Muốn lấy giá trị data = 37 của node 3, ta phải đi theo thứ tự node1 → node2 → node3.

Với mảng thì sao ?

```
int a[] = { 12, 99, 37 };
```

Ta có thể trực tiếp lấy ra các giá trị của mảng, không cần phải đi theo thứ tự gì cả.

Vậy, sinh ra danh sách liên kết để làm gì trong khi nó dài dòng, phức tạp, gò bó hơn ?

Qua các phần sau bạn sẽ dần dần hiểu ra.

Ta đi đến 2 kết luận quan trọng sau:

1. Chỉ cần nắm được head tức là nắm được toàn bộ DSLK
2. head = NULL tức là DSLK rỗng (không có phần tử nào)

## C. Các thao tác cơ bản với DSLK

Để làm việc với DSLK một cách chuyên nghiệp và đúng đắn, ta cần thiết lập các hàm chuẩn.

### 1. Duyệt DSLK

Duyệt DSLK thì ta dùng 1 con trỏ p lần lượt đi qua các node (dựa theo liên kết next).

Phong cách 1:

```
Node *p = head;

while (p)
{
    // xử lý với p->data, ví dụ như in ra màn hình
    printf("%d \n", p->data);
    p = p->next;
}
```

Phong cách 2:

```
Node *p = head;

for (; p; p = p->next)
{
    // xử lý với p->data, ví dụ như in ra màn hình
    printf("%d \n", p->data);
}
```

Hoặc thậm chí ngắn gọn hơn với C++:

```
for (Node *p = head; p; p = p->next)
{
    // xử lý với p->data, ví dụ như in ra màn hình
    printf("%d \n", p->data);
}
```

Thật ra vòng lặp for bản chất cũng là vòng lặp while thôi. Vấn đề là khi bạn code theo phong cách 2, bạn sẽ khó quan sát được và debug được luồng chạy. Vì vậy lời khuyên là bạn sử dụng phong cách 1.

## 2. Create node

“Create node” là việc tạo ra 1 node. Trong thực tế, để tạo 1 node ta cần cấp phát động bộ nhớ chứ không phải tạo bình thường.

C++	C
<pre>Node* CreateNode(int data) {     Node* node = new Node;     node-&gt;data = data;     node-&gt;next = 0;     // hoặc node-&gt;next = NULL      return node; }</pre>	<pre>Node* CreateNode(int data) {     Node* node = (Node*)malloc(sizeof(Node));     node-&gt;data = data;     node-&gt;next = 0;     // hoặc node-&gt;next = NULL      return node; }</pre>

Minh họa lại việc tạo 1 DSLK đơn giản sau đó duyệt DSLK in ra màn hình bằng C++:

<pre>#include &lt;stdio.h&gt;  struct Node {     int data;     Node *next; };  Node* CreateNode(int data) {     Node* node = new Node;     node-&gt;data = data;     node-&gt;next = 0;      return node; }  // trả về head Node* TaoDSLK() {     Node *node1 = CreateNode(12);     Node *node2 = CreateNode(99);     Node *node3 = CreateNode(37);      node1-&gt;next = node2;     node2-&gt;next = node3;      return node1; }</pre>	<pre>int main() {     Node *head = TaoDSLK();      Node *p = head;     while (p)     {         printf("%d ", p-&gt;data);         p = p-&gt;pNext;     }      // tạm thời quên đi việc giải phóng bộ nhớ      return 0; }</pre>
---	---

### 3. Giải phóng bộ nhớ

Với ngôn ngữ lập trình bậc cao có quản lý bộ nhớ như C# và Java thì ta không cần quan tâm đến việc giải phóng bộ nhớ.

Tuy nhiên với C/C++ thì đây là 1 thao tác hết sức quan trọng.

Giải phóng bộ nhớ đơn giản là ta duyệt DSLK rồi giải phóng bộ nhớ cho từng node.

C++	C
<pre> void FreeMemory(Node *&amp;head) {     Node *p = head;     Node *next = 0;      while (p)     {         next = p-&gt;next;         delete p;         p = next;     }      head = 0; }  int main() {     Node *head = TaoDSLK();     FreeMemory(head);     return 0; } </pre>	<pre> void FreeMemory(Node **p2head) {     Node *p = *p2head;     Node *next = 0;      while (p)     {         next = p-&gt;next;         free(p);         p = next;     }      *p2head = 0; }  int main() {     Node *head = TaoDSLK();     FreeMemory(&amp;head);     return 0; } </pre>

Nếu đang dùng thuần ngôn ngữ C thì ta nên truyền vào head dưới dạng con trỏ cấp 2 thì mới gán = 0 được.



## 4. Is empty

“Is empty” là thao tác kiểm tra xem DSLK có bị rỗng hay không. DSLK rỗng là DSLK không có phần tử nào.



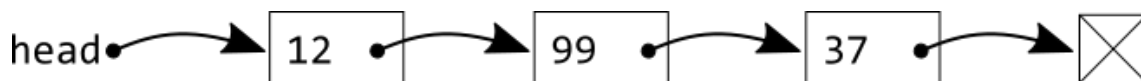
Rất dễ dàng suy luận ra DSLK rỗng tức là `head = NULL`.

```
int isEmpty(Node *head)
{
    return (head == 0);
}
```

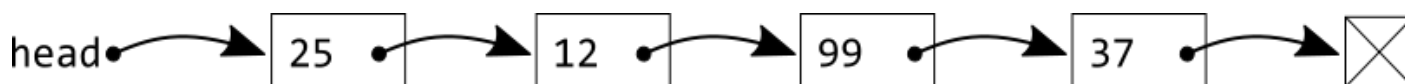
## 5. Insert head

“Insert head” là thao tác chèn thêm 1 node vào đầu DSLK. Cũng có nơi gọi là “add head”.

DSLK ban đầu:

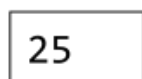
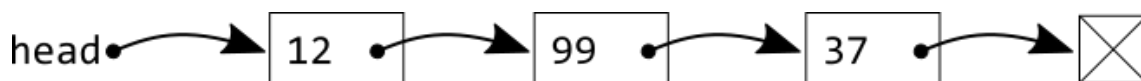


Sau khi chèn 1 node có data = 25 vào đầu DSLK:

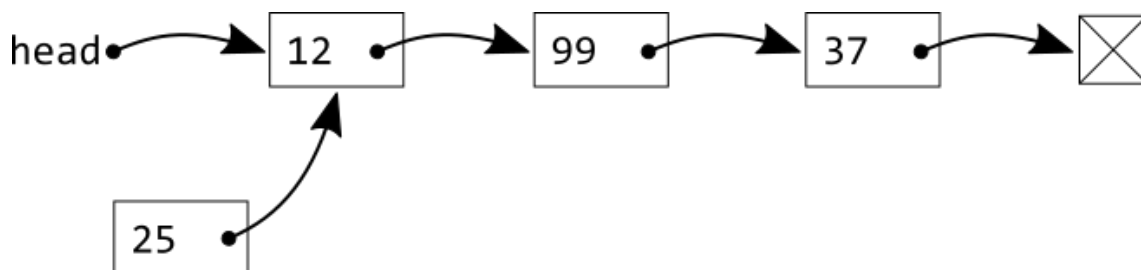


**Làm sao bây giờ ?**

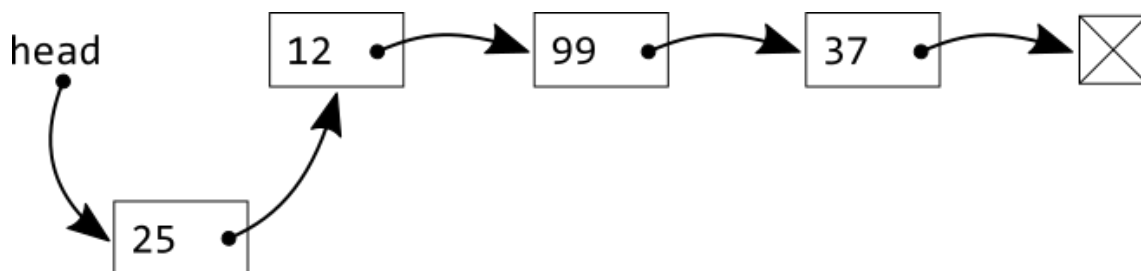
Gọi newNode là node cần chèn vào đầu DSLK. Trong ví dụ này ta có newNode (data = 25).



Bước 1. Cho newNode liên kết trở vào node đầu tiên trong DSLK.



Bước 2. Cho head là newNode.



Nhìn có vẻ không được thẳng hàng đẹp mắt nhưng ta đã làm xong rồi đó !!!

Code minh họa:

C++	C
<pre> void InsertHead(Node *&amp;head, Node *newNode) {     // BƯỚC 1     newNode-&gt;next = head;      // BƯỚC 2     head = newNode; }  int main() {     Node *head = TaoDSLK();     Node *newNode = CreateNode(25);      InsertHead(head, newNode);      FreeMemory(head);     return 0; } </pre>	<pre> void InsertHead(Node **p2head, Node *newNode) {     // BƯỚC 1     newNode-&gt;next = (*p2head);      // BƯỚC 2     (*p2head) = newNode; }  int main() {     Node *head = TaoDSLK();     Node *newNode = CreateNode(25);      InsertHead(&amp;head, newNode);      FreeMemory(&amp;head);     return 0; } </pre>

Ghi chú với bước 1: ta có head lưu địa chỉ của node (tức là head trỏ vào node đầu tiên). Như vậy nếu muốn newNode liên kết trỏ vào node đầu tiên thì đơn giản ta gán `newNode->next = head` = địa chỉ của node đầu tiên trong DSLK.

Tuy nhiên ta cần làm đầy đủ trường hợp hơn. Khi `head = NULL` (DSLK rỗng chưa có phần tử) thì thao tác `InsertHead` là việc gán `head = newNode`.

Code đầy đủ:

C++	C
<pre>void InsertHead(Node *&amp;head, Node *newNode) {     if (isEmpty(head))     {         head = newNode;         return;     }      newNode-&gt;next = head;     head = newNode; }</pre>	<pre>void InsertHead(Node **p2head, Node *newNode) {     if (isEmpty(*p2head))     {         (*p2head) = newNode;         return;     }      newNode-&gt;next = (*p2head);     (*p2head) = newNode; }</pre>

Ghi chú với kĩ thuật code ở trên: ta nên xử lý triệt để các trường hợp đặc biệt, các điều kiện lề vì vậy nên nếu DSLK rỗng thì xử lý trọn vẹn rồi return luôn. Ta thấy có vẻ code bị lặp lại lệnh `head = newNode` không hay nhưng thực tế đây là phong cách tốt.

Sẽ có bạn xử lý nếu DSLK không rỗng `if (!isEmpty(head))` thì `newNode->next = head`. Vậy cũng được không sao.

Trong thực tế khi truyền `newNode` vào khá nguy hiểm (có thể `newNode` đã có trong DSLK rồi), vì vậy ta nên truyền trực tiếp data vào.

C++	C
<pre>void InsertHead(Node *&amp;head, int data) {     Node *newNode = CreateNode(data);      if (isEmpty(head))     {         head = newNode;         return;     }      newNode-&gt;next = head;     head = newNode; }</pre>	<pre>void InsertHead(Node **p2head, int data) {     Node *newNode = CreateNode(data);      if (isEmpty(*p2head))     {         *p2head = newNode;         return;     }      newNode-&gt;next = (*p2head);     (*p2head) = newNode; }</pre>

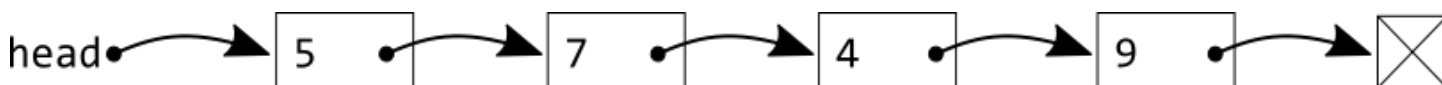
## THỰC HÀNH MỞ RỘNG

Giả sử ta có node1 (data = 9), node2 (data = 4), node3 (data = 7), node4 (data = 5).

Với các thao tác sau:

- 1. InsertHead(head, node1)
- 2. InsertHead(head, node2)
- 3. InsertHead(head, node3)
- 4. InsertHead(head, node4)

Hỏi: DSLK sẽ như thế nào ?



InsertHead thì giá trị theo theo thứ tự: 9, 4, 7, 5.

Nhưng DSLK kết quả thì lại là: 5, 7, 4, 9.

➔ Có vẻ như DSLK bị ngược so với ý muốn của ta.

Vậy có thể nào khắc phục được tình trạng này không ? Chắc chắn là có.

Thay vì chèn vào đầu DSLK (InsertHead) thì ta chèn vào cuối DSLK (InsertTail). Tuy nhiên InsertTail là vấn đề khó hơn 1 chút xíu, ta sẽ tạm gác lại và gặp nó vào 1 ngày không xa.

## 6. Remove head

“Remove head” là thao tác xóa node đầu tiên của DSLK.

Thật ra từ đầu bài học đến giờ nếu nhìn nhận một cách tổng quát thì DSLK không khác gì mảng 1 chiều cho lắm: cũng đều là lưu dãy các phần tử cùng dữ liệu.

**Vậy DSLK có gì hay hơn mảng không ?**

Trong các chương trình cần quan trọng hiệu năng, tốc độ thì điều này khá quan trọng. Xét mảng 1 triệu phần tử, ta cần xóa phần tử đầu tiên trong mảng.

Rất dễ dàng với đoạn code sau:

```
n--;  
  
for (i = 0; i < n; i++)  
    a[i] = a[i + 1];
```

Vấn đề là, với 1 triệu phần tử, ta phải cho vòng lặp chạy hết mảng mất rất nhiều chi phí bộ nhớ, vòng lặp for lặp lại khoảng 1 triệu lần. Chưa kể nếu ta có mảng các nhân viên, mảng các sinh viên thì lại càng tốn chi phí cho việc gán giá trị (struct chứa nhiều thành phần bên trong nên tốn nhiều bộ nhớ cho việc lưu trữ và xử lý).

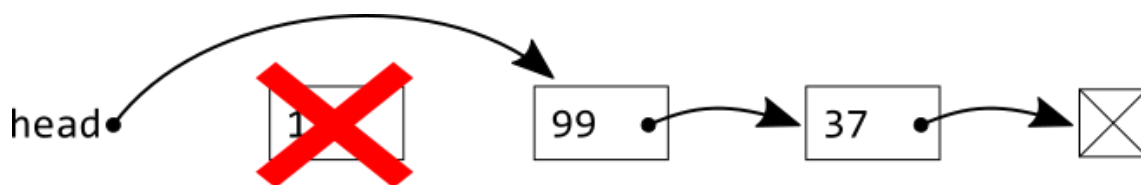
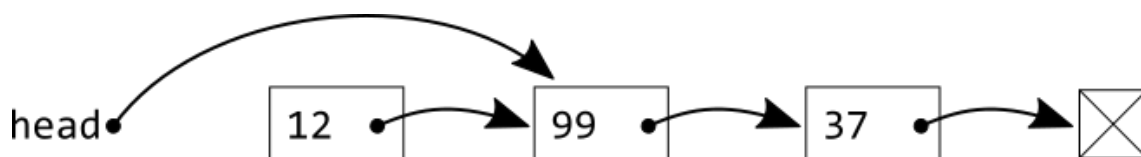
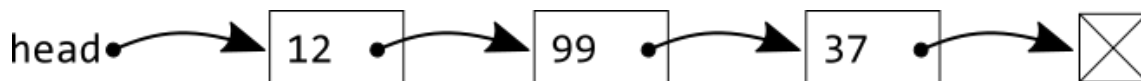
→ Kết luận: xử lý khá lâu, tốn nhiều thời gian và chi phí.

Bây giờ chuyển qua DSLK xem, xét DSLK có 1 triệu phần tử, ta muốn xóa phần tử đầu tiên (tức là remove head đó), vậy thì sao ?

Chỉ với 1 lệnh duy nhất ta giải xong bài toán luôn:

```
head = head->next;
```

Và đây là hình minh họa:



Cái hay của DSLK là với vài thao tác đặc biệt (như xóa 1 phần tử) thì DSLK xử lý cực nhanh.



WOW !!!

Tất nhiên DSLK còn có vài cái hay khác, bạn hãy tự khám phá nhé.

## D. Kĩ thuật nâng cao

### Duyệt DSLK bằng 2 con trỏ liên tiếp

Đây là 1 kĩ thuật rất quan trọng khi làm việc với DSLK mà ít người nhắc đến.

Khi duyệt DSLK, ta thường sử dụng con trỏ p. Bây giờ ta có thêm 1 con trỏ mới là "pPrev" có tác dụng **luôn luôn bám theo sau con trỏ p**.

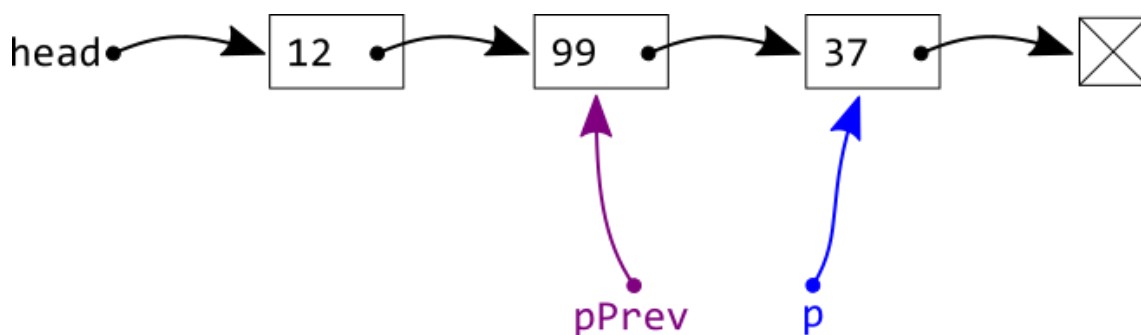
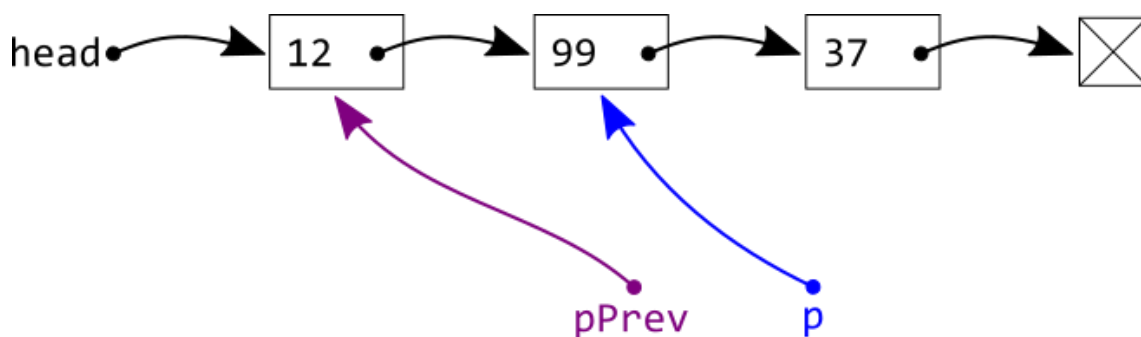
Cụ thể hơn: ta có  $pPrev \rightarrow next = p$

Ví dụ DSLK có 4 node theo thứ tự node1  $\rightarrow$  node2  $\rightarrow$  node3  $\rightarrow$  node4.

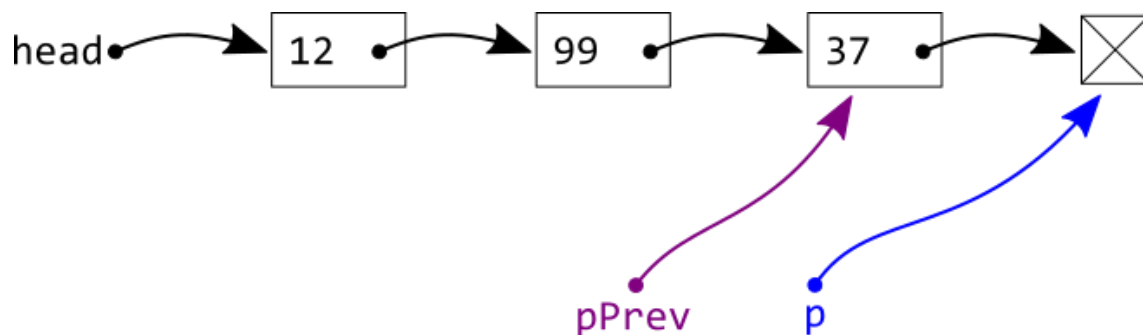
- p trỏ đến node2 thì pPrev trỏ đến node1.
- p trỏ đến node3 thì pPrev trỏ đến node2.
- p trỏ đến node4 thì pPrev trỏ đến node3.
- p trỏ đến NULL (cuối DSLK) thì pPrev trỏ đến node4.

Trường hợp đặc biệt: khi p trỏ đến node1 ta cần xét riêng.

Minh họa việc duyệt DSLK bằng 2 con trỏ p và pPrev







Code minh họa:

```

void DuyệtDSLK_2p(Node *head)
{
    Node *p = head;
    Node *pPrev = 0;

    // KHÔNG LÀM VIỆC VỚI DSLK RỖNG
    if (isEmpty(head))
        return;

    // XÉT RIÊNG VỚI p = head
    printf("p: %d \n", p->data);
    //-----

    pPrev = head;
    p = pPrev->next;

    while (p)
    {
        printf("p: %d, pPrev: %d \n", p->data, pPrev->data);

        pPrev = p;
        p = p->next;
    }
}
  
```

Chạy chương trình:

```

p: 12
p: 99, pPrev: 12
p: 37, pPrev: 99
  
```

Kĩ thuật duyệt DSLK bằng 2 con trỏ rất quan trọng vì nó sẽ giúp ta làm những bài toán khó hơn, hãy chờ xem !!!

## E. Một số thao tác mở rộng

### 1. Insert tail

“Insert tail” là thao tác chèn thêm 1 node vào cuối DSLK. Cũng có nơi gọi là “add tail”.

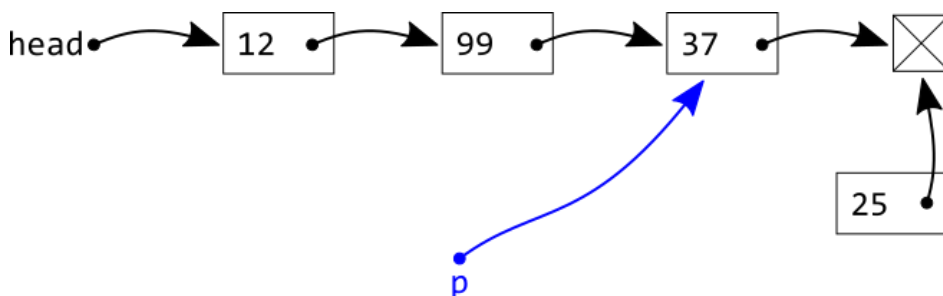
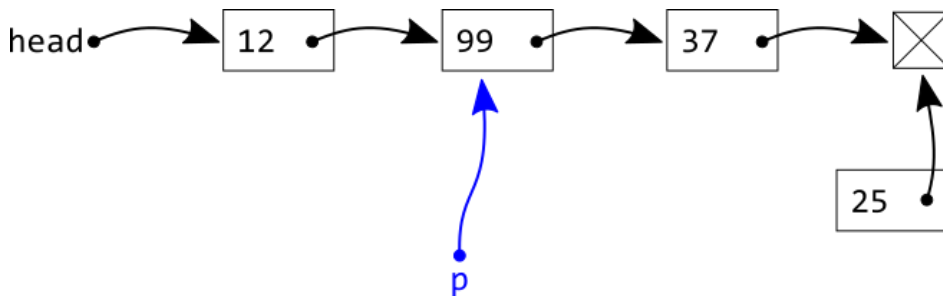
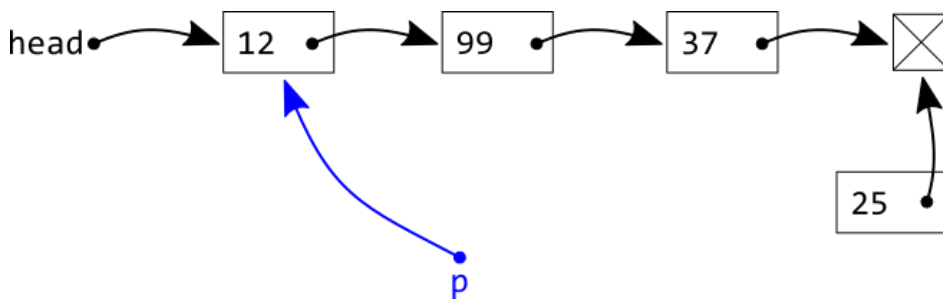
Có thể bạn sẽ nghĩ như vậy: gọi p là con trỏ duyệt lần lượt các node của DSLK.

Bước 1. Duyệt DSLK sao cho p đi đến node cuối cùng thì ngừng lại.

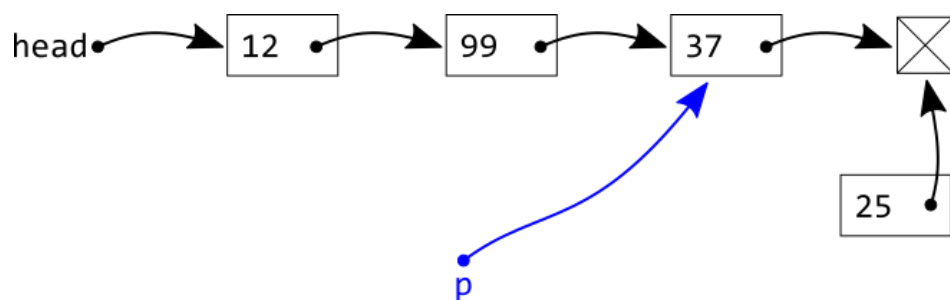
Bước 2. Cho  $p \rightarrow \text{next} = \text{newNode}$ .

Minh họa:

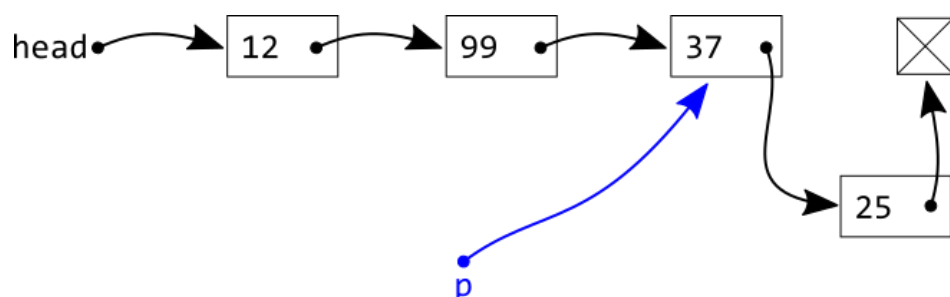
Bước 1:



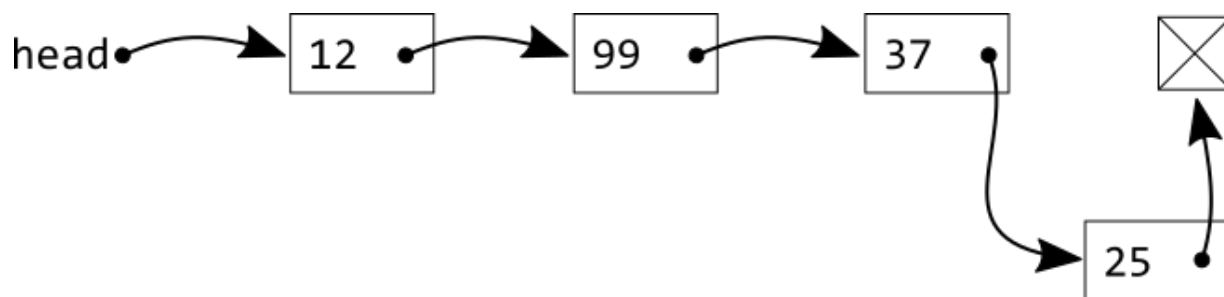
Bước 2: bây giờ p đang trở đến node cuối cùng, như vậy ta có thể điều khiển và thay đổi được giá trị của node cuối cùng.



Ta cho  $p \rightarrow \text{next} = \text{newNode}$ . Kết quả sẽ là:



Vậy là ta đã chèn thêm newNode (data = 25) vào cuối DSLK.



Wow, nhìn có vẻ dễ ghê, chỉ cần bạn giải tưởng tượng giống như khi bạn xem phim thì học DSLK không còn khó khăn nữa.

**HỎI:** làm sao biết được khi nào p trở đến node cuối cùng ? Khi  $p \rightarrow \text{next} = \text{NULL}$ .

Và đây là code:

C++

```
void InsertTail(Node *&head, Node *newNode)
{
    Node *p = head;

    // DSLK rỗng là trường hợp đặc biệt
    if (isEmpty(head))
    {
        head = newNode;
        return;
    }

    while (p)
    {
        // nếu p là node cuối cùng
        if (p->next == 0)
        {
            p->next = newNode;
            break;
        }

        p = p->next;
    }
}
```

Hoặc sửa lại ngắn gọn hơn:

C++

```
void InsertTail(Node *&head, Node *newNode)
{
    Node *p = head;

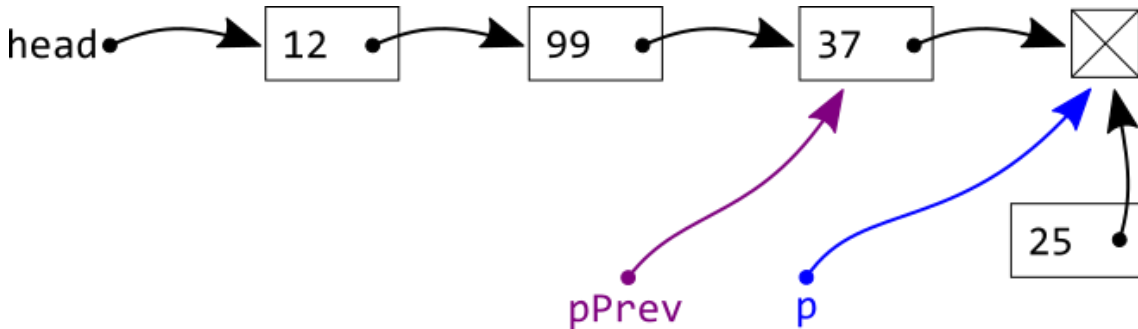
    // DSLK rỗng là trường hợp đặc biệt
    if (isEmpty(head))
    {
        head = newNode;
        return;
    }

    while (p->next)
    {
        p = p->next;
    }

    p->next = newNode;
}
```

Tuy nhiên ta còn 1 cách khác, chính là việc áp dụng chiêu thức “duyệt DSLK bằng 2 con trỏ”.

Hãy xem đây:



Khi  $p = \text{NULL}$  tức là  $p\text{Prev}$  trở đến node cuối cùng  $\rightarrow$  ta sẽ nhờ vả  $p\text{Prev}$  xử lý giùm.

Hết phim !!! Quá dễ !!!

C++

```

void InsertTail(Node *&head, Node *newNode)
{
    Node *p = head;
    Node *pPrev = 0;

    // DSLK rỗng là trường hợp đặc biệt
    if (isEmpty(head))
    {
        head = newNode;
        return;
    }

    pPrev = head;
    p = pPrev->next;

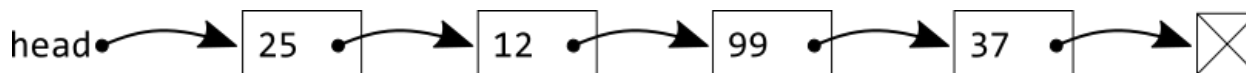
    while (p)
    {
        pPrev = p;
        p = p->next;
    }

    // KẾT THÚC VÒNG LẶP WHILE tức là p = NULL và pPrev trở đến node cuối cùng
    pPrev->next = newNode;
}
  
```

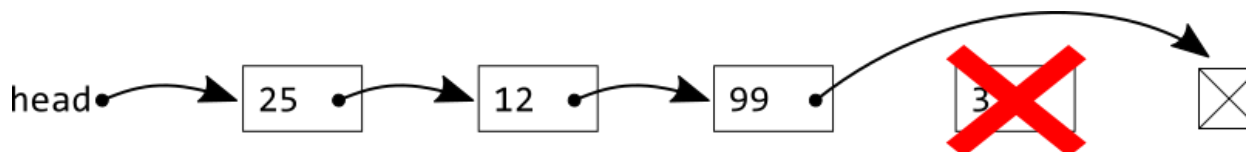
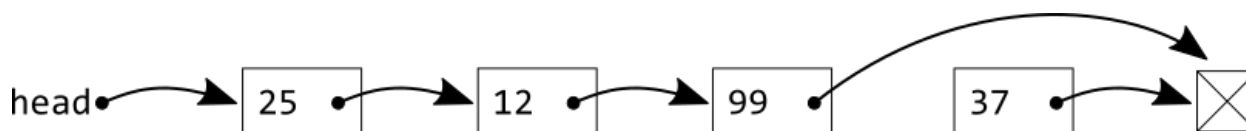
## 2. Remove tail

Nếu bạn đã học vững tất cả phần trước thì phần này chỉ là chuyện nhỏ.

Nếu InsertTail ta có 2 cách thì RemoveTail ta chỉ có 1 cách duy nhất là dùng kĩ thuật duyệt DSLK bằng 2 con trỏ. Lý do là vì ta cần tác động đến node “áp cuối” nằm phía trước node cuối.



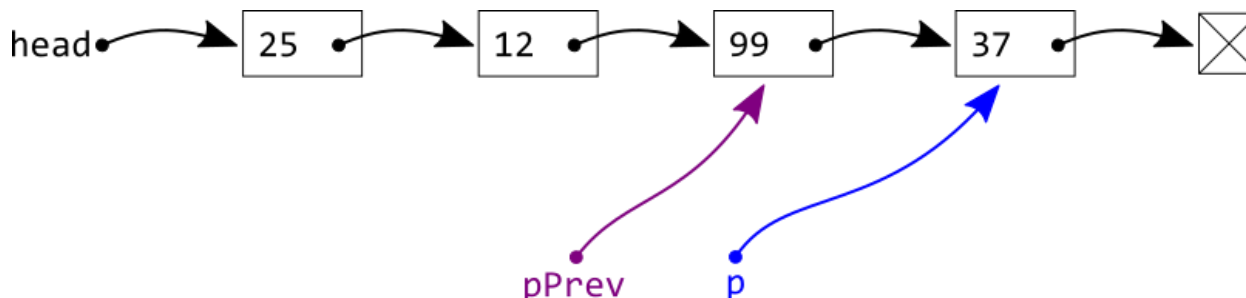
Điều mà ta muốn đó là node “áp cuối” (có data = 99) trở về NULL.



Vậy là ta đã hoàn thành việc “remove tail”.

Vấn đề là làm sao xác định được node “áp cuối” → chắc chắn ta sẽ cần thông qua node cuối cùng vì node cuối cùng ta có thể xác định được.

→ Áp dụng chiêu thức duyệt = 2 con trỏ, với pPrev = node áp cuối và p = node cuối.



### 3. Thao tác với 1 node nằm ở giữa DSLK (chèn, xóa)

Vẫn là áp dụng kỹ thuật duyệt DSLK bằng 2 con trỏ.

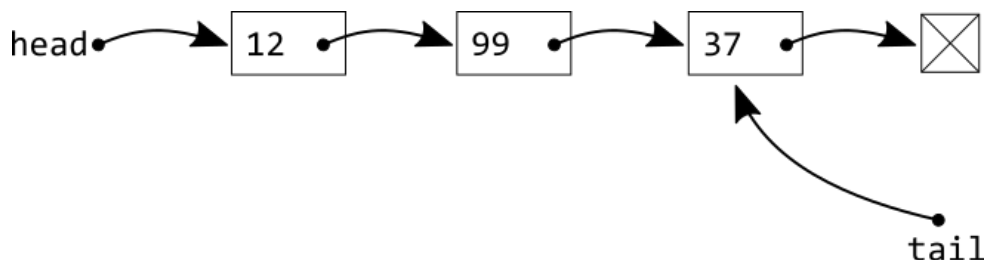
## F. Mở rộng DSLK với tail

Bây giờ ta sẽ làm rõ hơn các khái niệm cơ bản với DSLK

- Head: node đầu tiên.
- Tail: node cuối cùng.



Chiêu thức mới: DSLK có 2 con trỏ head và tail (thay vì chỉ có head như trước kia). Head luôn luôn là node đầu tiên, và tail luôn luôn là node cuối cùng.



Nhằm thuận tiện và chuyên môn hóa, ta sẽ tạo ra riêng struct DSLK.

Với cách làm trước kia:

```
struct DSLK
{
    Node *head;
};
```

Với cách làm cải tiến – DSLK có 2 con trỏ head và tail:

```
struct DSLK
{
    Node *head;
    Node *tail;
};
```



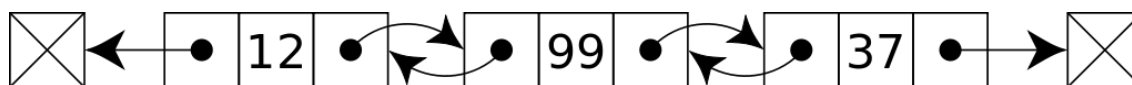
Một số lưu ý với các thao tác cơ bản với DSLK có 2 con trỏ head và tail:

- InsertHead
  - Khi DSLK rỗng thì phải gán cả  $head = tail = newNode$ .
- RemoveHead
  - Cần thận khi DSLK đang có 1 node, khi RemoveHead thì sẽ thành DSLK rỗng nên cần gán  $head = tail = NULL$ .
- InsertTail
  - Khi DSLK rỗng thì phải gán cả  $head = tail = newNode$ .
  - Đâu cần duyệt DSLK đâu khổ nữa, chúng ta đã có con trỏ tail luôn luôn trỏ đến node cuối.  $tail->next = newNode$ . 1 lệnh duy nhất là xong !!!
- RemoveTail
  - Cần thận khi DSLK đang có 1 node.
  - Thao tác này cần tác động đến node “áp cuối” nên vẫn cần 2 con trỏ p và pPrev.

## G. Các loại DSLK

Loại đầu tiên, dễ nhất đó chính là DSLK đơn bình thường, là DSLK mà mình đã trình bày hướng dẫn xuyên suốt tài liệu này. Tên tiếng Anh của nó là “**singly linked list**”. Nếu ta nói “linked list” thì mặc định hiểu là DSLK đơn singly linked list.

Loại tiếp theo là **doubly linked list** (DSLK đôi).



1 node sẽ có 2 liên kết:

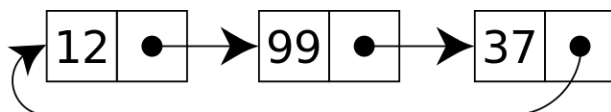
- next: trỏ đến node phía sau
- prev: trỏ đến node phía trước

Nhờ liên kết này mà ta có thể di chuyển dễ dàng hơn, từ trái sang phải hoặc từ phải sang trái. Head cũng có thể là tail mà tail cũng có thể xem là head.

### Mutiply linked list (DSLK đa liên kết)

1 node chứa nhiều liên kết (từ 2 trở lên). Mỗi liên kết sẽ có 1 ý nghĩa riêng tùy vào người lập trình quy định. Loại này được sử dụng rất ít.

### Circular linked list (DSLK vòng)



Dễ dàng nhận thấy node cuối cùng lại trỏ đến node đầu tiên.

## H. Bài tập

**Bài 1.** Tạo ra DSLK lưu các số nguyên (chỉ được phép có head).

Người dùng sẽ nhập vào số lượng phần tử của DSLK, sau đó nhập vào từng phần tử.

Xuất danh sách liên kết đã tạo.

**Bài 2.** Tạo ra DSLK lưu các chuỗi (có head và tail).

Chương trình cho phép người dùng nhập vô hạn các chuỗi. Khi nào người dùng nhập chuỗi rỗng thì xem như kết thúc việc nhập DSLK.

- **Yêu cầu 1.** Xuất ra các chuỗi đã nhập, mỗi chuỗi nằm trên 1 dòng.
- **Yêu cầu 2.** Cho biết DSLK có bao nhiêu chuỗi.

**Bài 3.** Tạo ra DSLK lưu các số nguyên nguyên (bạn tùy ý quyết định có tail hay không).

Người dùng sẽ nhập vô hạn các phần tử. Quá trình nhập sẽ dừng lại khi người dùng nhập số 0.

- **Yêu cầu 1.** Xuất ra DSLK.
- **Yêu cầu 2.** Gọi  $n$  là số lượng phần tử, cho biết  $n =$  bao nhiêu.
- **Yêu cầu 3.** Người dùng nhập vào vị trí xóa ( $0 \leq vt < n$ ). Hãy xóa 1 phần tử tại vị trí nhập vào. Xuất ra lại DSLK.

**Bài 4.** Cho struct như sau:

```
struct TestData
{
    char dump[10000];
};

struct Node
{
    TestData data;
    Node *next;
};
```

Cho  $n = 10000$  (mười nghìn) là số lượng phần tử.

- **Yêu cầu 1.** Tạo ra 2 mảng a và b với mỗi phần tử có kiểu TestData. Mảng a và b đều có n phần tử bằng cách cấp phát động, tuy nhiên với mảng a bạn cấp phát động tối đa  $2n$  phần tử. Bạn không cần gán dữ liệu cho mảng a và b (để dữ liệu rác cũng được). Bạn hãy ghép n phần tử của mảng b vào sau n phần tử của mảng a (bằng 1 vòng lặp for). Kết quả mong đợi là mảng a có  $2n$  phần tử.
- **Yêu cầu 2.** Tạo ra 2 DSLK ds1 và ds2 (có head và tail), mỗi DSLK đều có n node (không gán dữ liệu cho data). Bạn hãy ghép ds2 vào sau ds1 **sao cho hiệu quả**. Kết quả mong đợi là ds1 có  $2n$  node.

Với mỗi yêu cầu, bạn hãy đo thời gian chạy của việc xử lý gộp dữ liệu (độ chính xác mili-giây). Hãy cho nhận xét và giải thích, so sánh giữa việc sử dụng mảng và DSLK trong bài toán này.

Ghi chú: nếu cấp phát động dữ liệu báo lỗi (do vượt quá khả năng đáp ứng của bộ nhớ RAM) thì bạn được quyền giảm giá trị n đi sao cho phù hợp.

# I. Tổng kết

Một số lời khuyên dành cho bạn khi làm việc với DSLK:

- ❖ Luôn nghĩ đến các trường hợp đặc biệt như DSLK rỗng, DSLK chỉ có 1 phần tử.
- ❖ Lỗi thường gặp của các bạn beginner là: newNode xuất hiện 2 lần trong DSLK, không xét các trường hợp đặc biệt.
- ❖ Sử dụng IDE hỗ trợ debug tốt ta có thể xem được toàn bộ DSLK (ví dụ Visual Studio).

OK, học này giờ quá trời, vậy cuối cùng DSLK có ý nghĩa gì, có ứng dụng gì không ?

- ❖ Mảng cần bộ nhớ liên tục (ví dụ mảng a có 500 phần tử kiểu int, như vậy mảng a cần 2000 bytes trống liên tục). Trong khi đó DSLK có thể áp dụng với bộ nhớ rời rạc (mỗi node có thể nằm ở các vị trí bộ nhớ tùy ý → linh động).  
→ Nếu RAM có 1999 bytes trống ở khu vực A và 10 bytes trống ở khu vực B → sử dụng mảng là bó tay.  
→ Như vậy trong trường hợp bộ nhớ bị phân mảnh hoặc ít vùng nhớ trống liên tục thì mảng sẽ chịu thua và DSLK lên ngôi.
- ❖ Xử lý cho một số bài toán đặc thù (ví dụ như xóa phần tử đầu tiên của mảng).
- ❖ Là nền tảng cho các cấu trúc dữ liệu khác (thường là phức tạp hơn).  
Ví dụ: tree, stack, queue.
- ❖ Ứng dụng cho một số thuật toán sort, quản lý stream. Ví dụ: thư viện enet xử lý UDP network stream có sử dụng DSLK với mỗi node là 1 packet thông tin trên đường truyền mạng máy tính.



Hi vọng tài liệu này sẽ giúp ích cho bạn. Cảm ơn bạn đã xem.

Tài liệu này thuộc môn Kỹ thuật lập trình – chương D – bài 1.

Tác giả: Nguyễn Trung Thành

Facebook: <https://www.facebook.com/abcxyztcit>