Con trỏ (P.2)

Kỹ thuật Lập trình (CO1027)

Ngày 11 tháng 8 năm 2022

ThS. Trần Ngọc Bảo Duy

Khoa Khoa học và Kỹ thuật Máy tính

Trường Đại học Bách Khoa, ĐHQG-HCM

Tổng quan

ThS.
Trần Ngọc Bảo Duy

TRUYỀN CON TRỎ NHƯ THAM SỐ

ThS.
Trần Ngọc Bảo Duy

Truyền con trỏ như tham số

Trong C, cách **truyền tham số bằng tham chiếu** (pass by reference) không tồn tại. Vì vậy, các hàm muốn thay đổi giá trị của đối số, người ta phải sử dụng **con trỏ**.

ThS.
Trần Ngọc Bảo Duy

Truyền con trỏ như tham số

Trong C, cách **truyền tham số bằng tham chiếu** (pass by reference) không tồn tại. Vì vậy, các hàm muốn thay đổi giá trị của đối số, người ta phải sử dụng **con trỏ**.

```
1  #include <iostream>
2  using namespace std;
3
4  void inc(int* p2) {
5      *p2 = *p2 + 1;
6  }
7
8  int main(){
9      int a = 5;
10     int *p1 = &a;
11     inc(p1);
12     cout << a;
13 }
```


ThS.
Trần Ngọc Bảo Duy

Truyền con trỏ như tham số

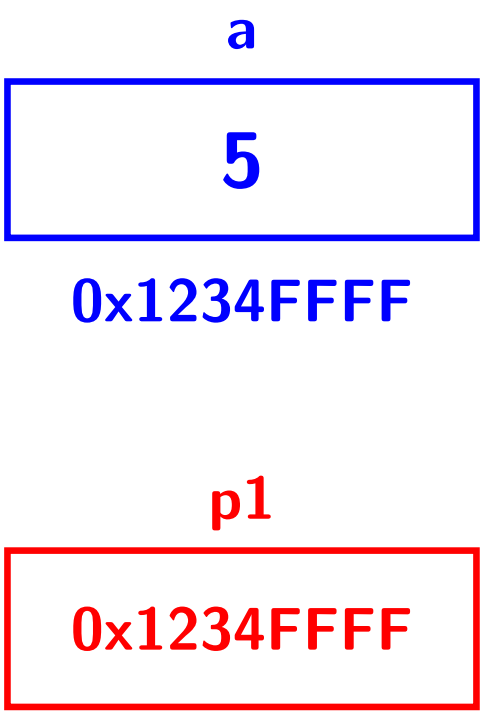
Trong C, cách **truyền tham số bằng tham chiếu** (pass by reference) không tồn tại. Vì vậy, các hàm muốn thay đổi giá trị của đối số, người ta phải sử dụng **con trỏ**.

```
1  #include <iostream>
2  using namespace std;
3
4  void inc(int* p2) {
5      *p2 = *p2 + 1;
6  }
7
8  int main(){
9      int a = 5;
10     int *p1 = &a;
11     inc(p1);
12     cout << a;
13 }
```

Chương trình trên in ra giá trị: 6

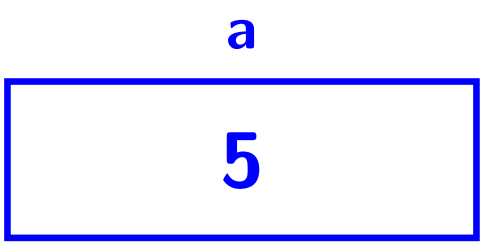
ThS.
Trần Ngọc Bảo Duy

Bản chất của việc truyền con trỏ như tham số

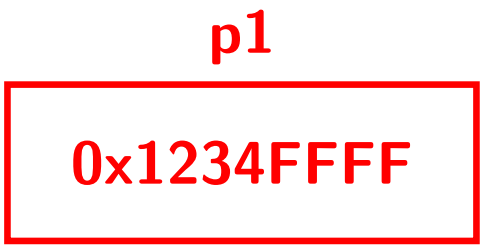


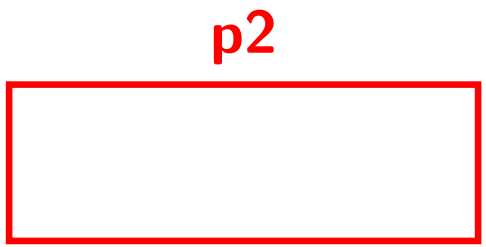
ThS.
Trần Ngọc Bảo Duy

Bản chất của việc truyền con trỏ như tham số



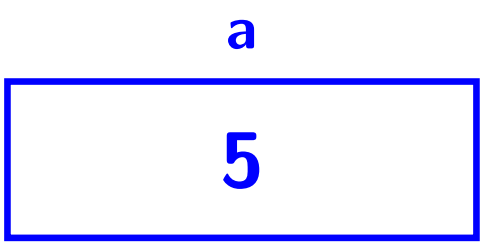
Khi gọi inc(p1):



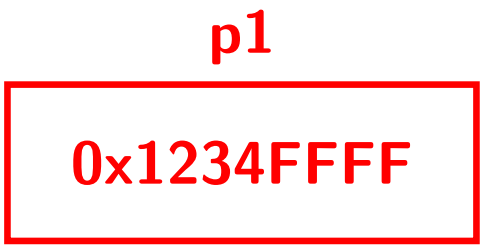


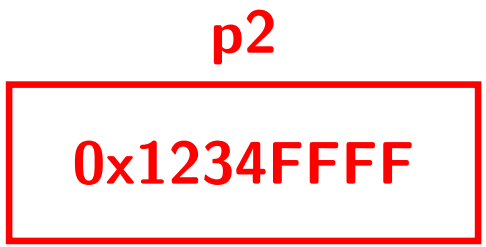
ThS.
Trần Ngọc Bảo Duy

Bản chất của việc truyền con trỏ như tham số



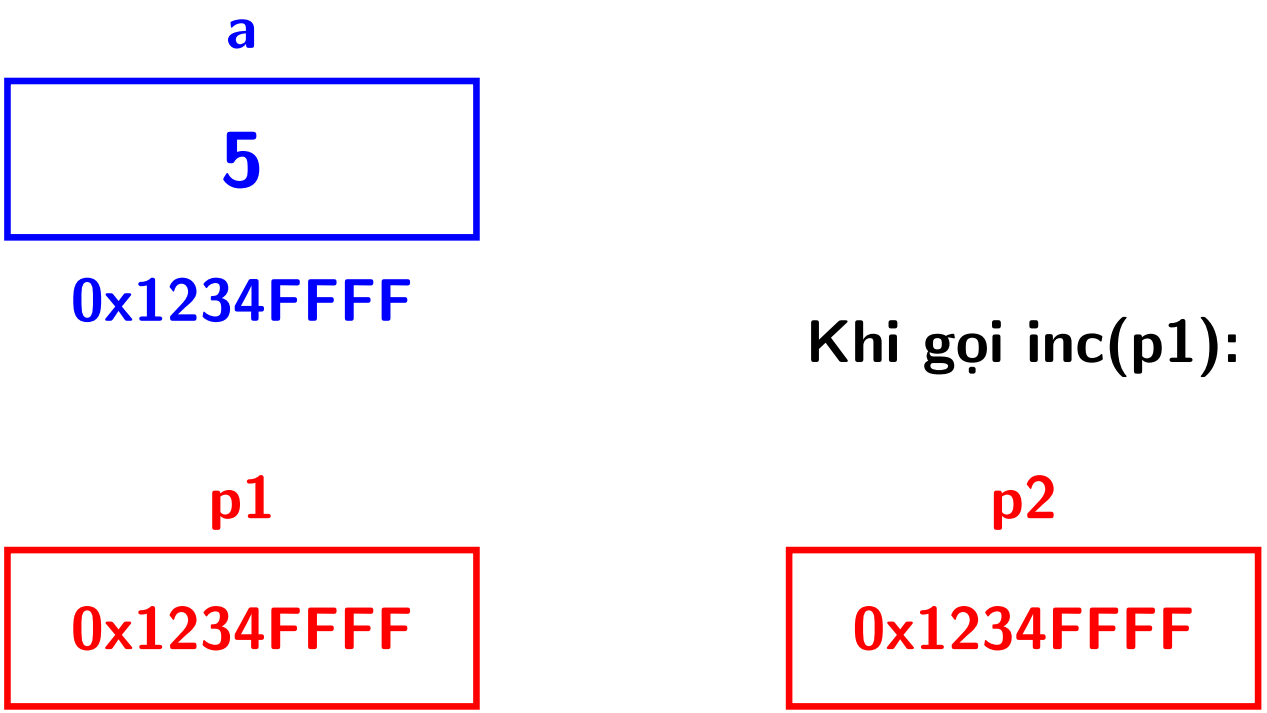
Khi gọi inc(p1):





ThS.
Trần Ngọc Bảo Duy

Bản chất của việc truyền con trỏ như tham số



Việc sử dụng địa chỉ mà p2 chứa rồi lấy tham khảo dẫn đến việc thay đổi giá trị mà a chứa.

ThS.
Trần Ngọc Bảo Duy

Truyền tham chiếu một con trỏ

```
1  #include <iostream>
2  using namespace std;
3
4  int gV = 10;
5
6  void foo(int* p) {
7      p = &gV;
8  }
9
10 int main(){
11     int a = 5;
12     int *p1 = &a;
13     foo(p1);
14     cout << *p1;
15 }
```

Chương trình trên in ra giá trị: 5

ThS.
Trần Ngọc Bảo Duy

Truyền tham chiếu một con trỏ

```
1  #include <iostream>
2  using namespace std;
3
4  int gV = 10;
5
6  void foo(int*& p) {
7      p = &gV;
8  }
9
10 int main(){
11     int a = 5;
12     int *p1 = &a;
13     foo(p1);
14     cout << *p1;
15 }
```

Chương trình trên in ra giá trị: 10

ThS.
Trần Ngọc Bảo Duy

Truyền mảng bằng con trỏ

NHẮC LẠI: Con trỏ và mảng đều giữ địa chỉ của ô nhớ.

- Con trỏ: giữa địa chỉ của ô nhớ nào đó (của biến khác, của bộ nhớ động).
- Mảng: giữ địa chỉ của phần tử đầu tiên.

ThS.
Trần Ngọc Bảo Duy

Truyền mảng bằng con trỏ

NHẮC LẠI: Con trỏ và mảng đều giữ địa chỉ của ô nhớ.

- Con trỏ: giữa địa chỉ của ô nhớ nào đó (của biến khác, của bộ nhớ động).
- Mảng: giữ địa chỉ của phần tử đầu tiên.

Các prototype sau là tương đương:

```
int foo(int arr[], int n);
int foo(int* arr, int n);
```


ThS.
Trần Ngọc Bảo Duy

Truyền mảng bằng con trỏ

NHẮC LẠI: Con trỏ và mảng đều giữ địa chỉ của ô nhớ.

- Con trỏ: giữa địa chỉ của ô nhớ nào đó (của biến khác, của bộ nhớ động).
- Mảng: giữ địa chỉ của phần tử đầu tiên.

Các prototype sau là tương đương:

```
int foo(int arr[], int n);
int foo(int* arr, int n);
```

Vì vậy, bản chất của việc truyền mảng là truyền địa chỉ của phần tử đầu tiên, chứ **KHÔNG** truyền thông tin đó là mảng, trong thân hàm ta không thể lấy được kích thước của mảng truyền vào.

ThS.
Trần Ngọc Bảo Duy

HIỆN THỰC DANH SÁCH BẰNG MẢNG

ThS.
Trần Ngọc Bảo Duy

Sự khác biệt giữa mảng và danh sách

- Trong C/C++, **mảng** (array) là một **vùng nhớ liên tục** có kích thước cố định bao gồm nhiều phần tử cùng kiểu. Trong vùng nhớ đó, mỗi ô nhớ chứa một phần tử của mảng.

ThS.
Trần Ngọc Bảo Duy

Sự khác biệt giữa mảng và danh sách

- Trong C/C++, **mảng** (array) là một **vùng nhớ liên tục** có kích thước cố định bao gồm nhiều phần tử cùng kiểu. Trong vùng nhớ đó, mỗi ô nhớ chứa một phần tử của mảng.

- **Danh sách**, nói chung, là một kiểu dữ liệu có thể chứa nhiều phần tử nhưng **kích thước không cố định**. Trong các kiểu dữ liệu cơ bản (primitive type) của C/C++, không có kiểu dữ liệu cho danh sách. Vì vậy, ta phải tự hiện thực nó hoặc sử dụng các thư viện có sẵn trong C/C++.

ThS.
Trần Ngọc Bảo Duy

Sự khác biệt giữa mảng và danh sách

- Trong C/C++, **mảng** (array) là một **vùng nhớ liên tục** có kích thước cố định bao gồm nhiều phần tử cùng kiểu. Trong vùng nhớ đó, mỗi ô nhớ chứa một phần tử của mảng.

- **Danh sách**, nói chung, là một kiểu dữ liệu có thể chứa nhiều phần tử nhưng **kích thước không cố định**. Trong các kiểu dữ liệu cơ bản (primitive type) của C/C++, không có kiểu dữ liệu cho danh sách. Vì vậy, ta phải tự hiện thực nó hoặc sử dụng các thư viện có sẵn trong C/C++.

  Một số thư viện có sẵn: `<vector>`, `<list>`.

ThS.
Trần Ngọc Bảo Duy

Hiện thực danh sách bằng mảng

Ý tưởng

1. Cấp phát một mảng có kích thước lớn (sức chứa của danh sách).

ThS.
Trần Ngọc Bảo Duy

Hiện thực danh sách bằng mảng

Ý tưởng

1. Cấp phát một mảng có kích thước lớn (sức chứa của danh sách).
2. Sử dụng mảng trên cho đến khi sức chứa của danh sách không đáp ứng được số lượng phần tử của danh sách.

ThS.
Trần Ngọc Bảo Duy

Hiện thực danh sách bằng mảng

Ý tưởng

1. Cấp phát một mảng có kích thước lớn (sức chứa của danh sách).
2. Sử dụng mảng trên cho đến khi sức chứa của danh sách không đáp ứng được số lượng phần tử của danh sách.
3. Cấp phát một vùng mảng mới có kích thước lớn hơn và chép các phần tử cũ qua.

ThS.
Trần Ngọc Bảo Duy

Hiện thực danh sách bằng mảng

Ý tưởng

1. Cấp phát một mảng có kích thước lớn (sức chứa của danh sách).
2. Sử dụng mảng trên cho đến khi sức chứa của danh sách không đáp ứng được số lượng phần tử của danh sách.
3. Cấp phát một vùng mảng mới có kích thước lớn hơn và chép các phần tử cũ qua.

$\Rightarrow$ Muốn vậy, mảng trên **PHẢI** được cấp phát động.

ThS.
Trần Ngọc Bảo Duy

Hiện thực danh sách bằng mảng

Ý tưởng

1. Cấp phát một mảng có kích thước lớn (sức chứa của danh sách).
2. Sử dụng mảng trên cho đến khi sức chứa của danh sách không đáp ứng được số lượng phần tử của danh sách.
3. Cấp phát một vùng mảng mới có kích thước lớn hơn và chép các phần tử cũ qua.

$\Rightarrow$ Muốn vậy, mảng trên **PHẢI** được cấp phát động.

0	1	2	3	4	5	6	7	8	9
13	6	42	43	30	9				

ThS.
Trần Ngọc Bảo Duy

Tạo ra danh sách

```
1 void createList(int*& arr, int cap = 10) {
2     arr = new int[cap];
3 }
```


ThS.
Trần Ngọc Bảo Duy

Tạo ra danh sách

```
1 void createList(int*& arr, int cap = 10) {
2     arr = new int[cap];
3 }
```

Hàm `createList` có hai tham số:

1. `arr`: con trỏ dùng để nhận lại địa chỉ của mảng sau khi cấp phát.
2. `cap`: sức chứa của danh sách, hay số lượng phần tử tối đa của mảng.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	42	43	30	9				

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	42	43	30	9	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	42	43	30	30	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	42	43	43	30	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	42	42	43	30	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

0	1	2	3	4	5	6	7	8	9
13	6	15	42	43	30	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.
2. Gán giá trị **15** vào vị trí **2**.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách

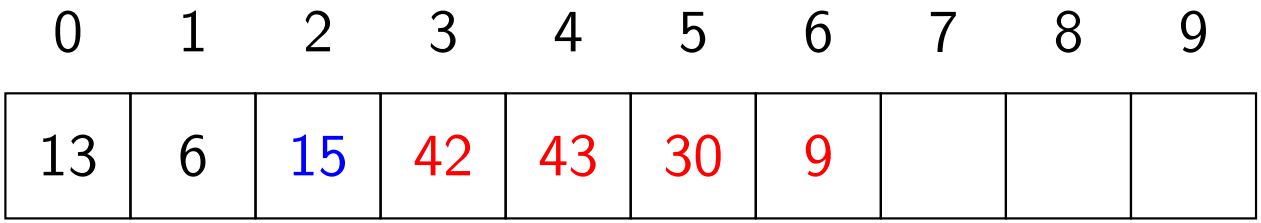
0	1	2	3	4	5	6	7	8	9
13	6	15	42	43	30	9			

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.
2. Gán giá trị **15** vào vị trí **2**.
3. Tăng kích thước của danh sách lên 1.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào danh sách



Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang phải **1** đơn vị.
2. Gán giá trị **15** vào vị trí **2**.
3. Tăng kích thước của danh sách lên 1.

Prototype:

```
void add(int*& arr, int& len, int& cap, int ele, int pos = 0);
```


ThS.
Trần Ngọc Bảo Duy

Xóa một phần tử ra khỏi danh sách

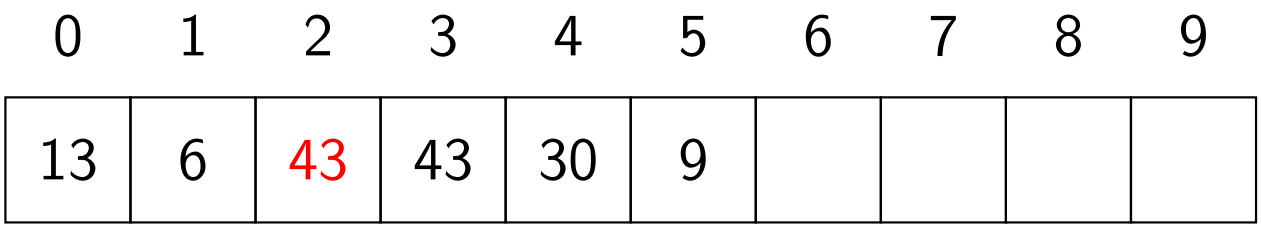
0	1	2	3	4	5	6	7	8	9
13	6	42	43	30	9				

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang trái **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Xóa một phần tử ra khỏi danh sách



Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang trái **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Xóa một phần tử ra khỏi danh sách

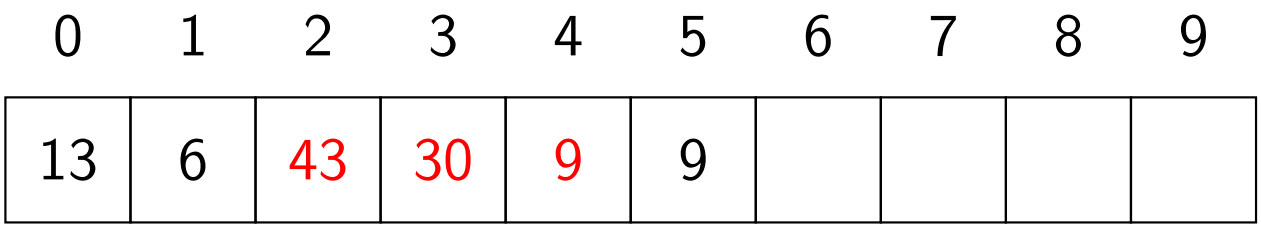
0	1	2	3	4	5	6	7	8	9
13	6	43	30	30	9				

Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang trái **1** đơn vị.

ThS.
Trần Ngọc Bảo Duy

Xóa một phần tử ra khỏi danh sách

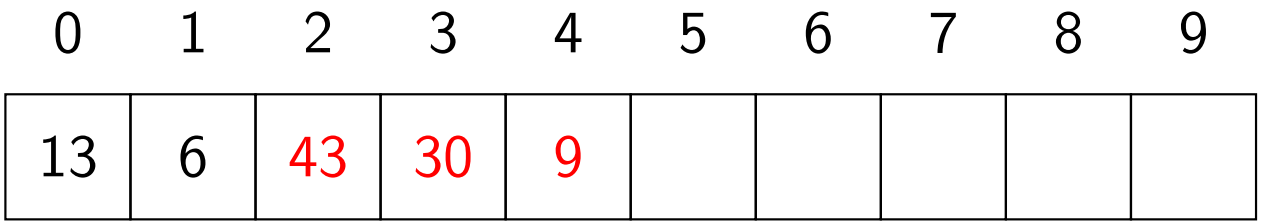


Chèn giá trị **15** tại vị trí **2**:

1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang trái **1** đơn vị.
2. Giảm kích thước của danh sách xuống 1.

ThS.
Trần Ngọc Bảo Duy

Xóa một phần tử ra khỏi danh sách



Chèn giá trị **15** tại vị trí **2**:

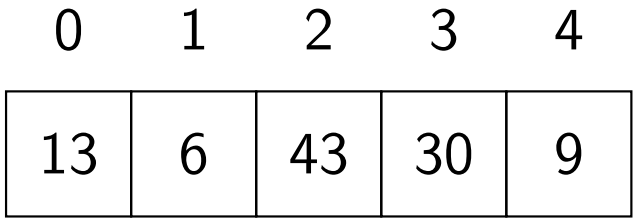
1. Dịch chuyển toàn bộ các phần tử từ vị trí **2** sang trái **1** đơn vị.
2. Giảm kích thước của danh sách xuống 1.

Prototype:

```
int remove(int*& arr, int& len, int& cap);
```


ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa

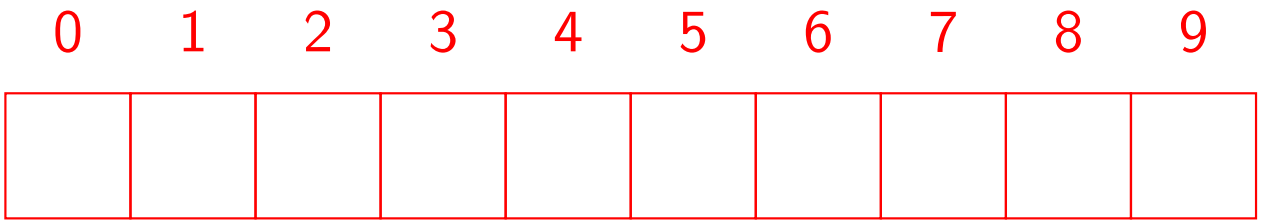


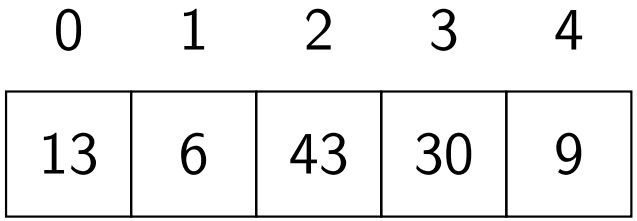
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



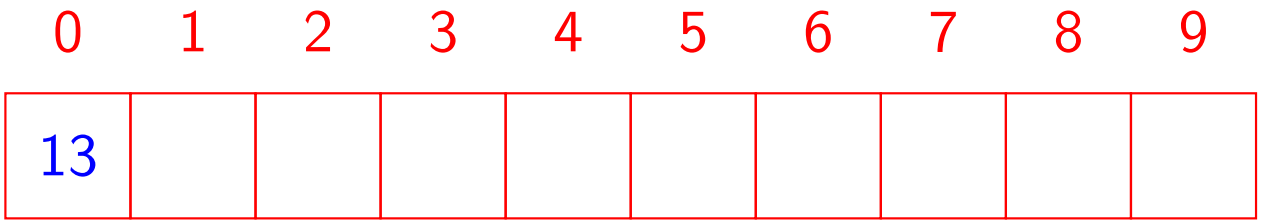


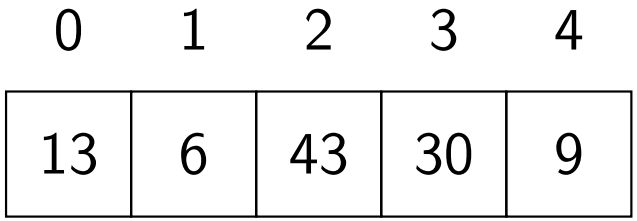
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



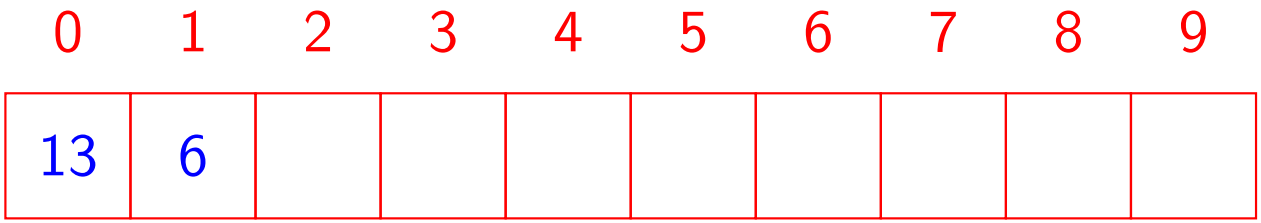


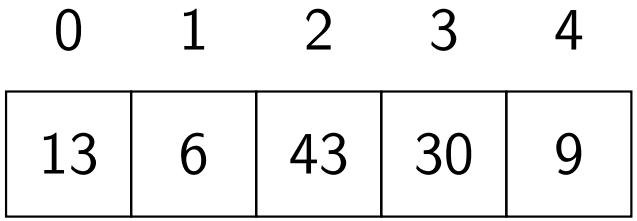
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



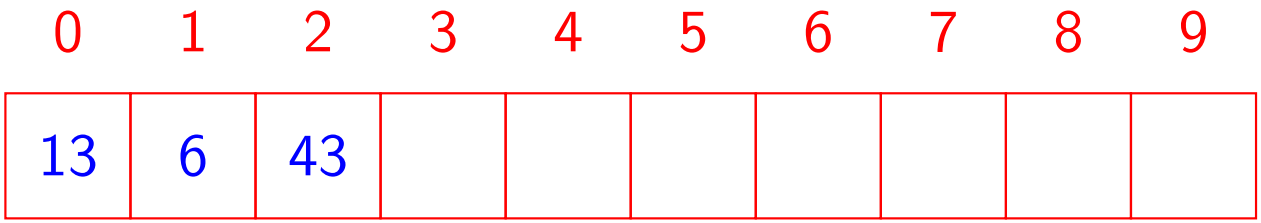


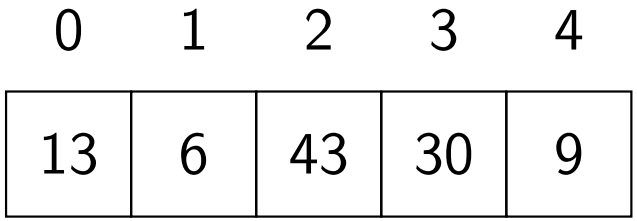
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



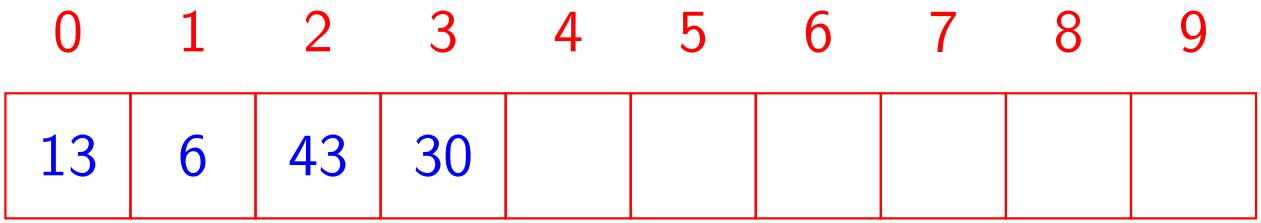


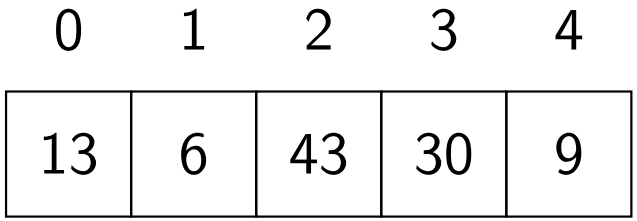
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



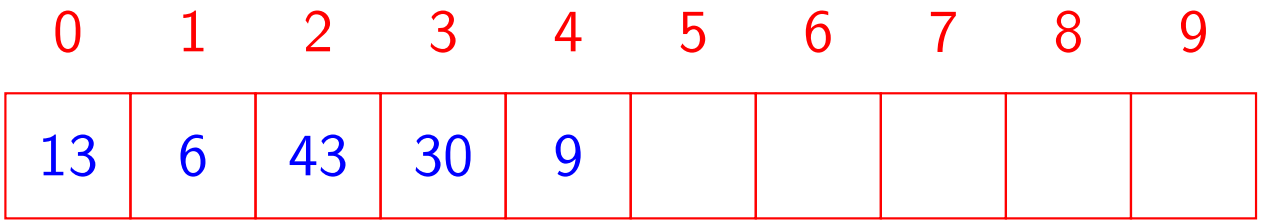


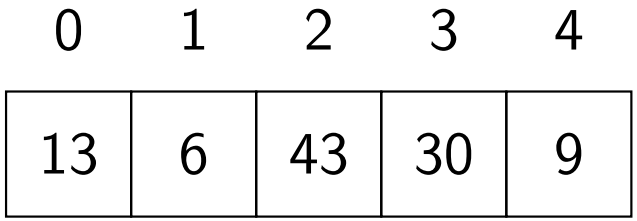
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



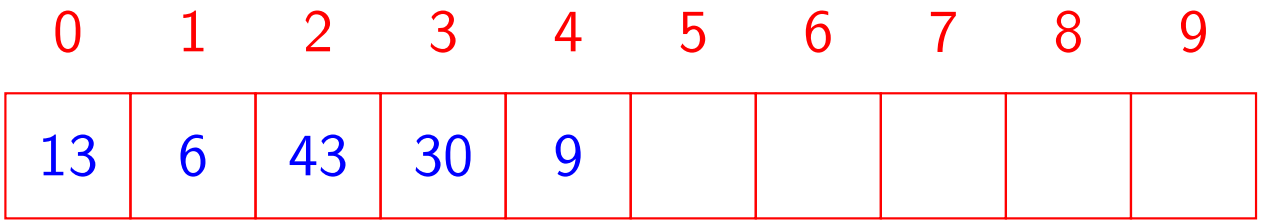


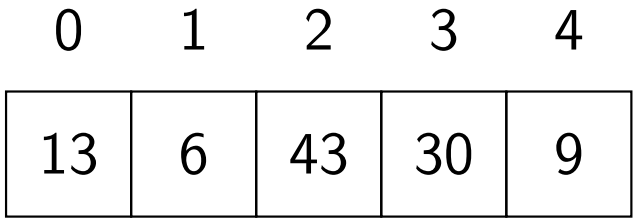
Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



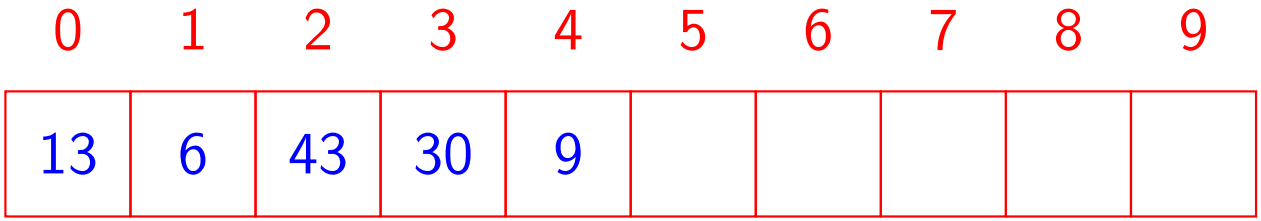


Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.
3. Chuyển con trỏ sang chỉ cho mảng mới, tăng sức chứa.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa

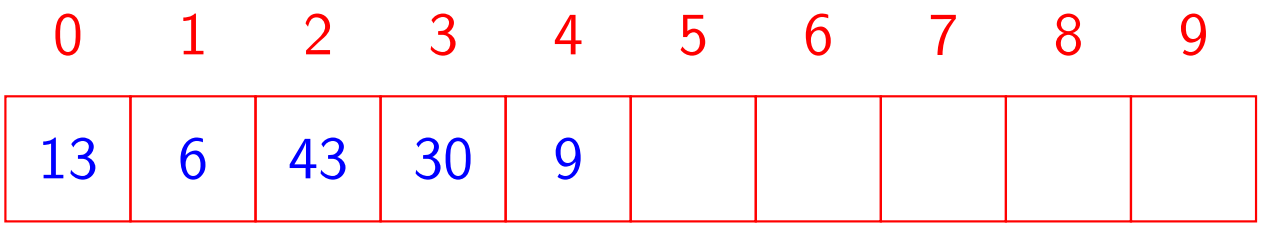


Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.
3. Chuyển con trỏ sang chỉ cho mảng mới, tăng sức chứa.
4. Thu hồi mảng cũ đã cấp phát.

ThS.
Trần Ngọc Bảo Duy

Xử lý khi danh sách vượt sức chứa



Khi danh sách vượt sức chứa:

1. Cấp phát một danh sách mới có sức chứa lớn hơn.
2. Chép từng phần tử trong danh sách cũ lên danh sách trên theo đúng vị trí.
3. Chuyển con trỏ sang chỉ cho mảng mới, tăng sức chứa.
4. Thu hồi mảng cũ đã cấp phát.

Prototype:

```
void ensureCap(int*& arr, int& cap, int newCap);
```


ThS.
Trần Ngọc Bảo Duy

Khai báo cấu trúc cho danh sách

```
1 struct ArrayList {
2     int* data;
3     int cap;
4     int len;
5 };
```


ThS.
Trần Ngọc Bảo Duy

Khai báo cấu trúc cho danh sách

```
1 struct ArrayList {
2     int* data;
3     int cap;
4     int len;
5 };
```

Prototype:

```
void add(ArrayList& list, int ele, int pos = 0);
int remove(ArrayList& list, int pos = 0);
void ensureCap(ArrayList& list, int newCap);
```


ThS.
Trần Ngọc Bảo Duy

DANH SÁCH LIÊN KẾT

ThS.
Trần Ngọc Bảo Duy

Nhược điểm của hiện thực danh sách bằng mảng

1. Mảng đòi hỏi phải cấp phát một vùng nhớ liên tục.

ThS.
Trần Ngọc Bảo Duy

Nhược điểm của hiện thực danh sách bằng mảng

1. Mảng đòi hỏi phải cấp phát một vùng nhớ liên tục.
2. Khi thêm hoặc xóa phần tử, cần phải dịch chuyển một khối lớn nhiều phần tử sang trái hoặc sang phải.

ThS.
Trần Ngọc Bảo Duy

Ý tưởng của danh sách liên kết

Ý tưởng chung

- Các phần tử nằm rời rạc trong vùng nhớ máy tính để khắc phục các nhược điểm nêu trên.
- Các phần tử phải liên kết với nhau bằng việc chứa thêm thông tin địa chỉ của các phần tử khác theo một thứ tự nào đó.

ThS.
Trần Ngọc Bảo Duy

Ý tưởng của danh sách liên kết

Ý tưởng chung

- Các phần tử nằm rời rạc trong vùng nhớ máy tính để khắc phục các nhược điểm nêu trên.
- Các phần tử phải liên kết với nhau bằng việc chứa thêm thông tin địa chỉ của các phần tử khác theo một thứ tự nào đó.

Danh sách liên kết (DSLK) đơn

DSLK đơn là danh sách mà ở đó mỗi phần tử (`node` - nút):

- Chứa phần dữ liệu của mỗi phần tử tương tự như mảng.
- Chứa địa chỉ của phần tử tiếp theo trong danh sách trong đó, phần tử cuối cùng không giữ địa chỉ của bất kỳ phần tử nào.

ThS.
Trần Ngọc Bảo Duy

Ý tưởng của danh sách liên kết

Ý tưởng chung

- Các phần tử nằm rời rạc trong vùng nhớ máy tính để khắc phục các nhược điểm nêu trên.
- Các phần tử phải liên kết với nhau bằng việc chứa thêm thông tin địa chỉ của các phần tử khác theo một thứ tự nào đó.

Danh sách liên kết (DSLK) đơn

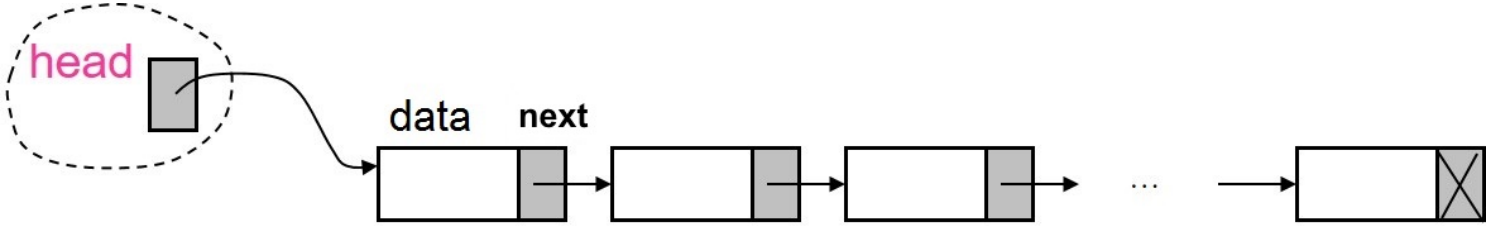
DSLK đơn là danh sách mà ở đó mỗi phần tử (`node` - nút):

- Chứa phần dữ liệu của mỗi phần tử tương tự như mảng.
- Chứa địa chỉ của phần tử tiếp theo trong danh sách trong đó, phần tử cuối cùng không giữ địa chỉ của bất kỳ phần tử nào.

NHƯ VẬY, DSLK đơn chỉ cần giữ địa chỉ của phần tử đầu tiên (`head`).

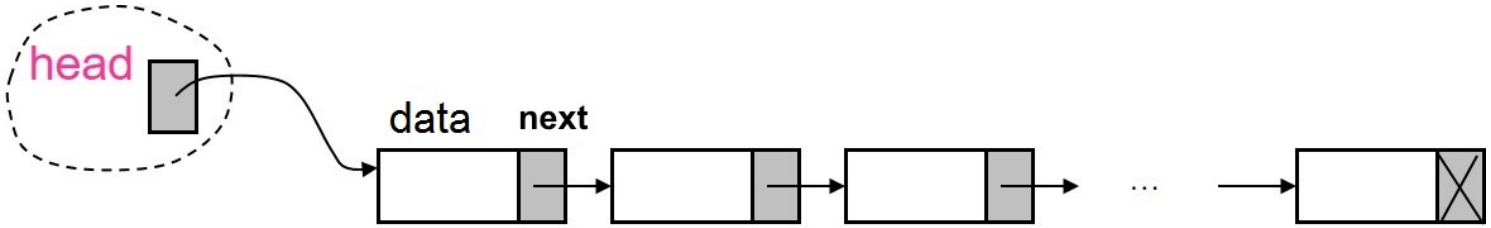
ThS.
Trần Ngọc Bảo Duy

Định nghĩa danh sách liên kết



ThS.
Trần Ngọc Bảo Duy

Định nghĩa danh sách liên kết

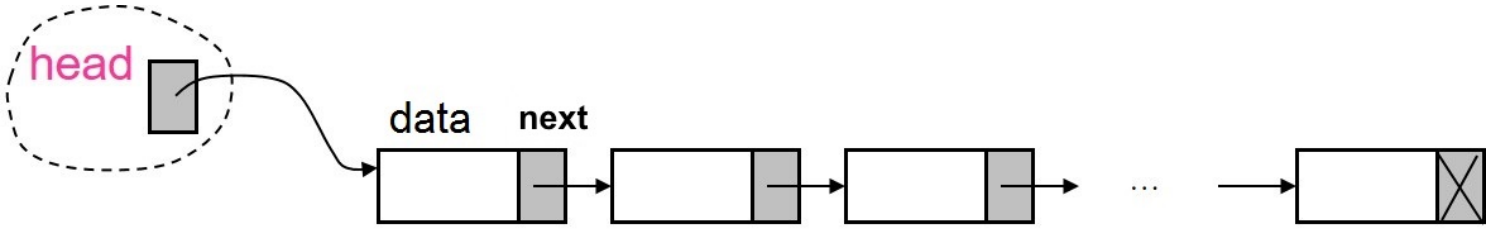


Để tạo ra danh sách liên kết mà mỗi nút chứa một dữ liệu kiểu T:

```
1 struct Node {
2     T data;
3     Node* next;
4 };
5
6 struct LList {
7     Node* head;
8 };
```


ThS.
Trần Ngọc Bảo Duy

Định nghĩa danh sách liên kết



Để tạo ra danh sách liên kết mà mỗi nút chứa một dữ liệu kiểu `T`:

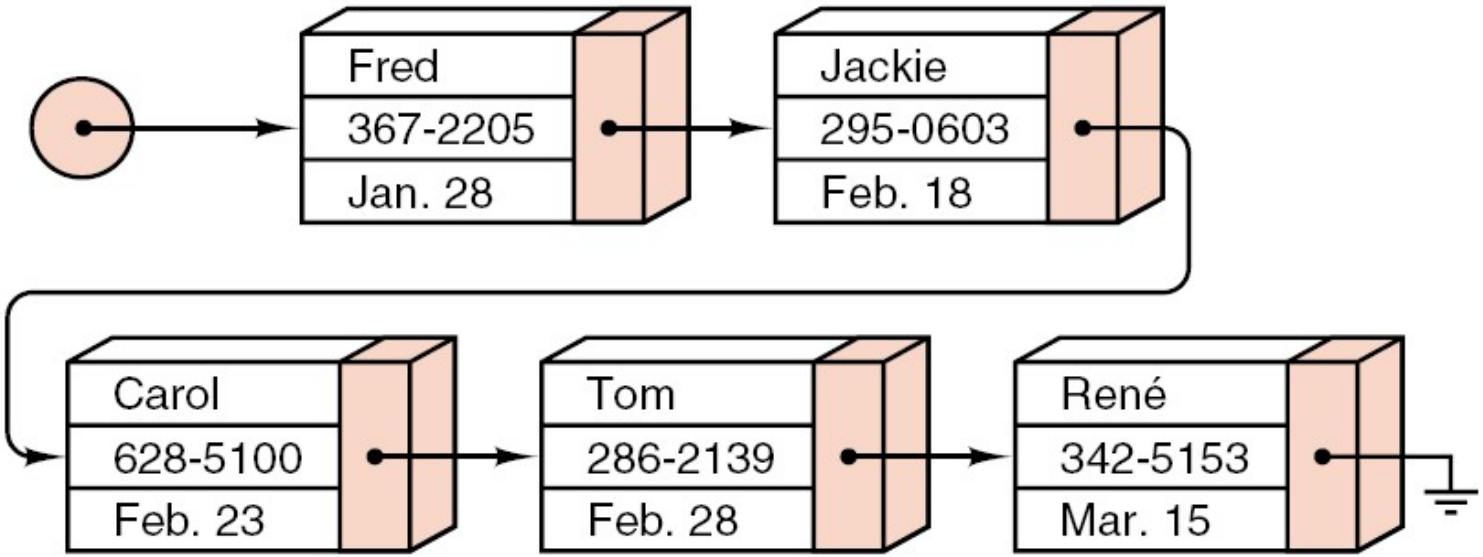
```
1 struct Node {
2     T data;
3     Node* next;
4 };
5
6 struct LList {
7     Node* head;
8 };
```

trong đó, cấu trúc `Node` dùng để chứa một nút trong danh sách liên kết, còn `LList` là cấu trúc đại diện của một danh sách liên kết.

Kiểu `T` trên có thể là kiểu số nguyên (`int`), kiểu số thực (`double`), kiểu Sinh viên tự định nghĩa (`Student`) ...

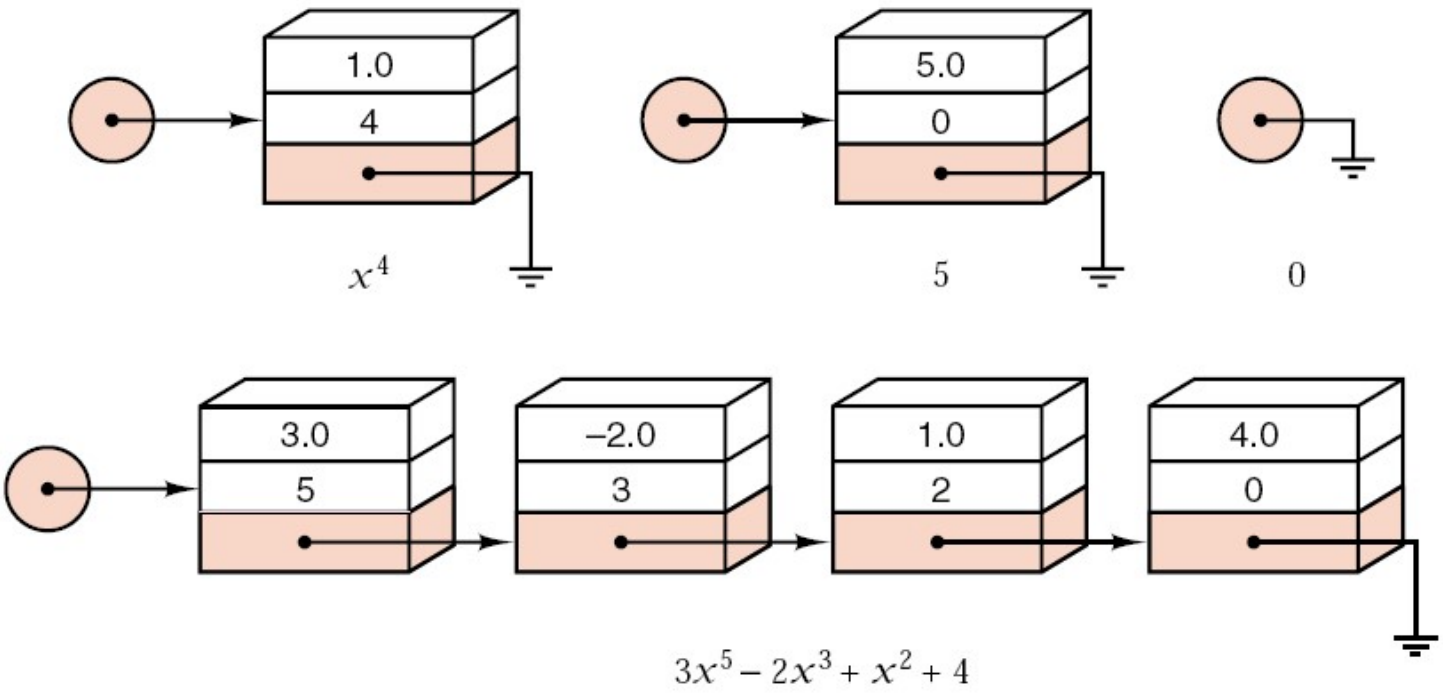
ThS.
Trần Ngọc Bảo Duy

Ví dụ về danh sách liên kết



ThS.
Trần Ngọc Bảo Duy

Ví dụ về danh sách liên kết



ThS.
Trần Ngọc Bảo Duy

Ví dụ về tạo danh sách liên kết

```
1  #include <iostream>
2  using namespace std;
3
4  struct Node { int data; Node* next; };
5  struct LList { Node* head; };
6
7  int main() {
8      LList l;
9      l.head = new Node(10, NULL);
10     l.head->next = new Node(20, NULL);
11     l.head->next->next = new Node(30, NULL);
12 }
```


ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào DSLK đơn

Khi thêm một phần tử vào DSLK đơn, ta cần quan tâm đến các trường hợp sau:

1. Trường hợp 1: DSLK rỗng.
2. Trường hợp 2: DSLK không rỗng.
   - Thêm vào đầu danh sách.
   - Thêm vào vị trí bất kỳ.

ThS.
Trần Ngọc Bảo Duy

Thêm một phần tử vào DSLK đơn

Khi thêm một phần tử vào DSLK đơn, ta cần quan tâm đến các trường hợp sau:

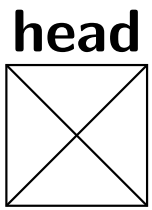
1. Trường hợp 1: DSLK rỗng.
2. Trường hợp 2: DSLK không rỗng.
   - Thêm vào đầu danh sách.
   - Thêm vào vị trí bất kỳ.

Prototype:

```
bool insert(LList& list, int ele, int pos = 0);
```


ThS.
Trần Ngọc Bảo Duy

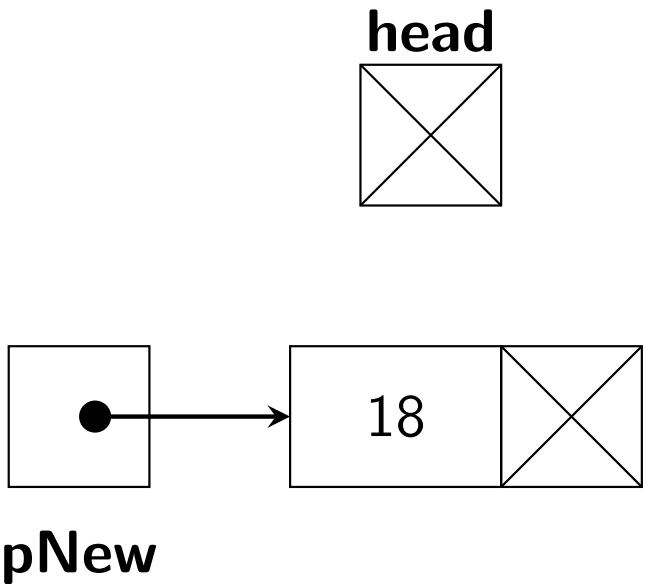
Thêm vào danh sách rỗng



Khi thêm phần tử 18 vào DSLK:

ThS.
Trần Ngọc Bảo Duy

Thêm vào danh sách rỗng

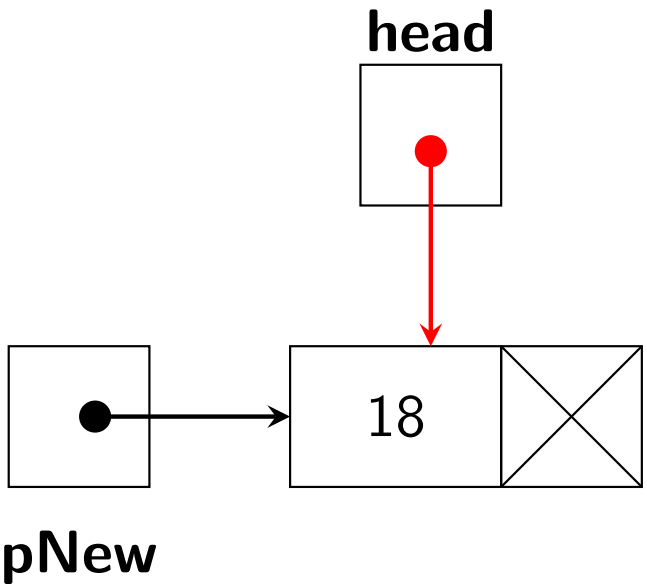


Khi thêm phần tử 18 vào DSLK:

1. Cấp phát ra một nút chứa số 18, và được giữ bởi con trỏ `pNew`.

ThS.
Trần Ngọc Bảo Duy

Thêm vào danh sách rỗng

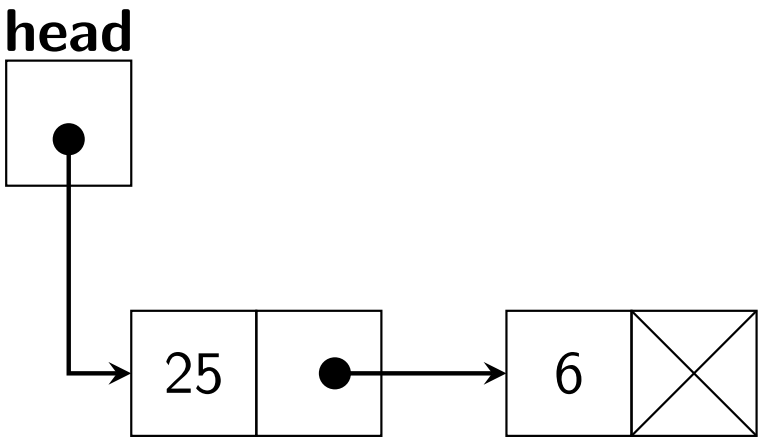


Khi thêm phần tử 18 vào DSLK:

1. Cấp phát ra một nút chứa số 18, và được giữ bởi con trỏ `pNew`.
2. Trỏ con trỏ `head` đến nút mới đó: head = pNew

ThS.
Trần Ngọc Bảo Duy

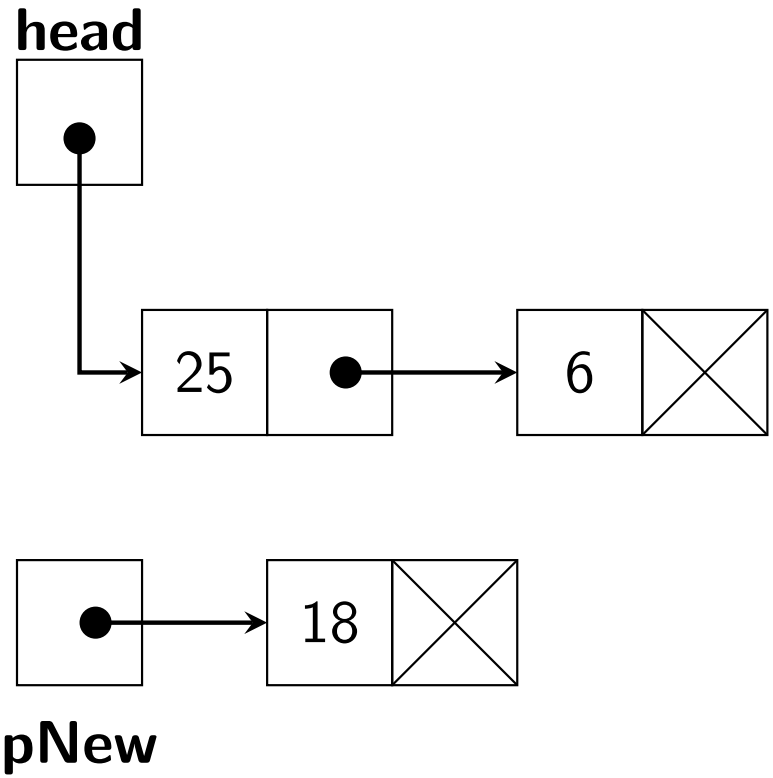
Thêm vào đầu danh sách không rỗng



Khi thêm phần tử số 18 vào đầu danh sách không rỗng:

ThS.
Trần Ngọc Bảo Duy

Thêm vào đầu danh sách không rỗng

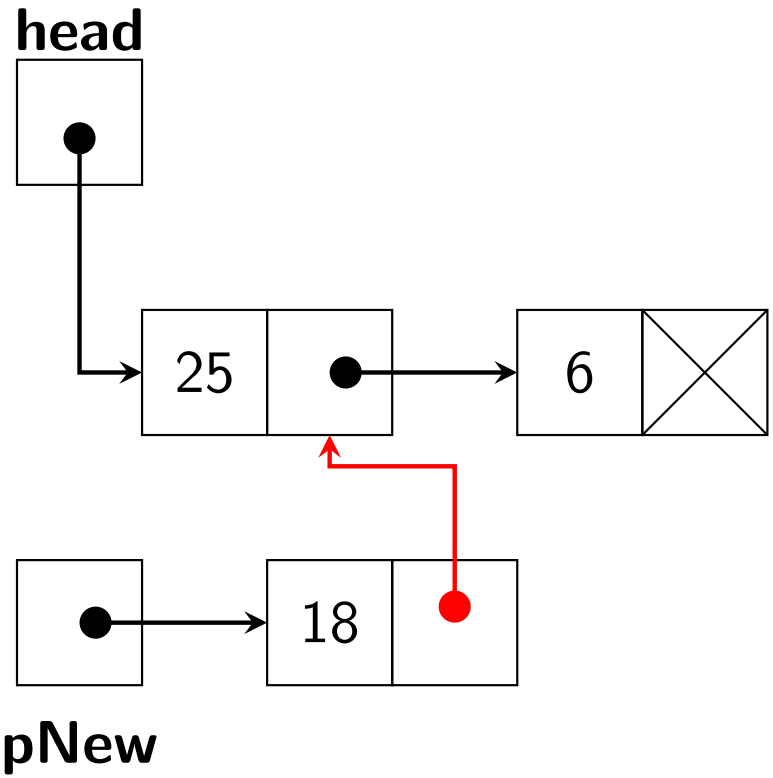


Khi thêm phần tử số 18 vào đầu danh sách không rỗng:

1. Cấp phát một nút mới chứa số 18 và được giữ bởi con trỏ `pNew`.

ThS.
Trần Ngọc Bảo Duy

Thêm vào đầu danh sách không rỗng

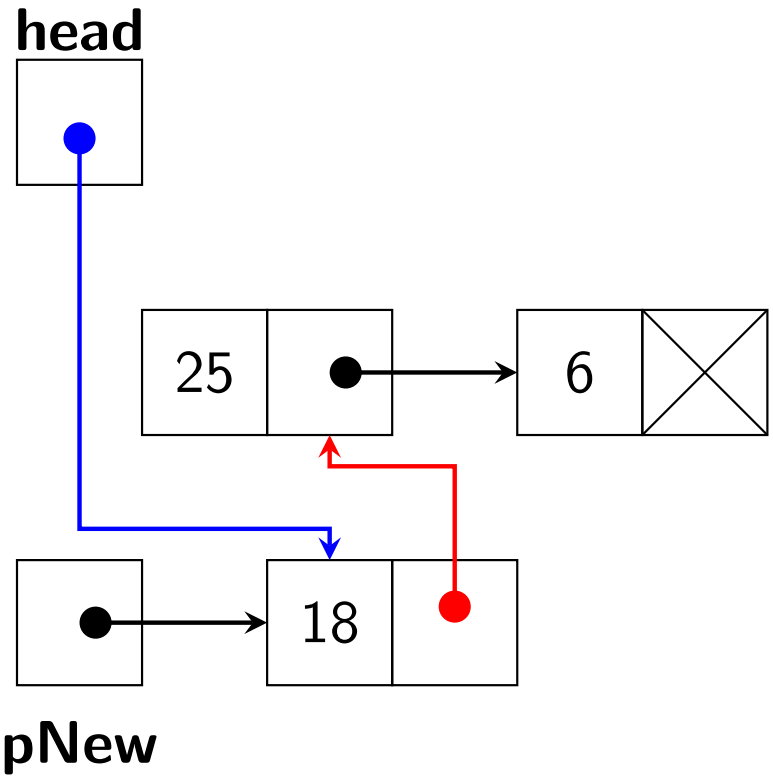


Khi thêm phần tử số 18 vào đầu danh sách không rỗng:

1. Cấp phát một nút mới chứa số 18 và được giữ bởi con trỏ `pNew`.
2. pNew->next = head

ThS.
Trần Ngọc Bảo Duy

Thêm vào đầu danh sách không rỗng

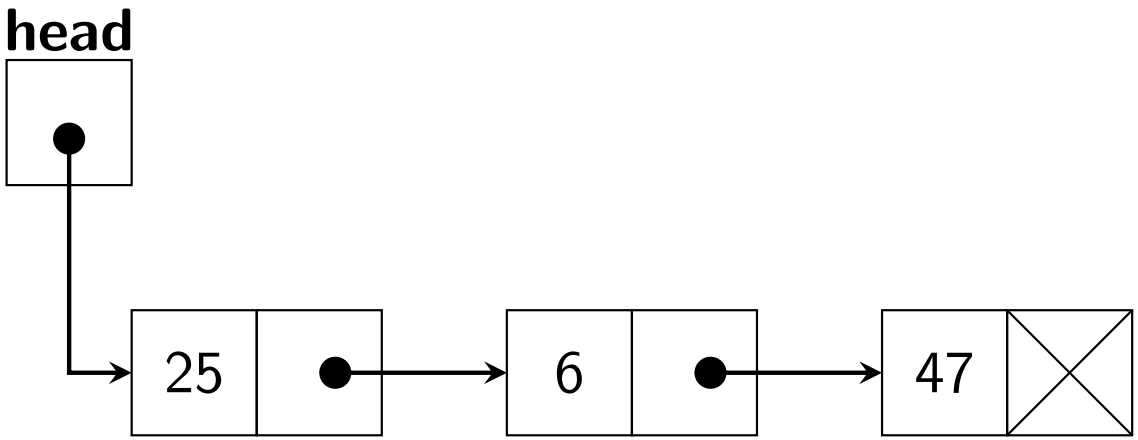


Khi thêm phần tử số 18 vào đầu danh sách không rỗng:

1. Cấp phát một nút mới chứa số 18 và được giữ bởi con trỏ `pNew`.
2. pNew->next = head
3. head = pNew

ThS.
Trần Ngọc Bảo Duy

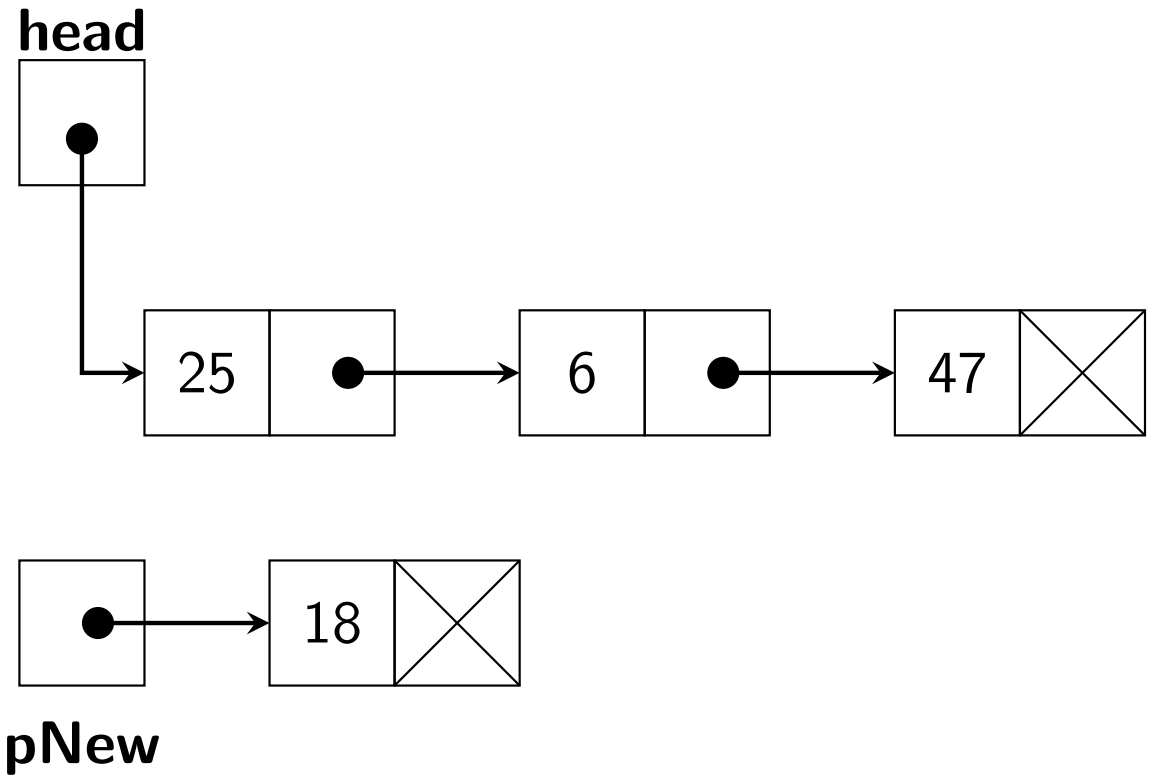
Thêm vào vị trí bất kỳ của danh sách không rỗng



Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

ThS.
Trần Ngọc Bảo Duy

Thêm vào vị trí bất kỳ của danh sách không rỗng

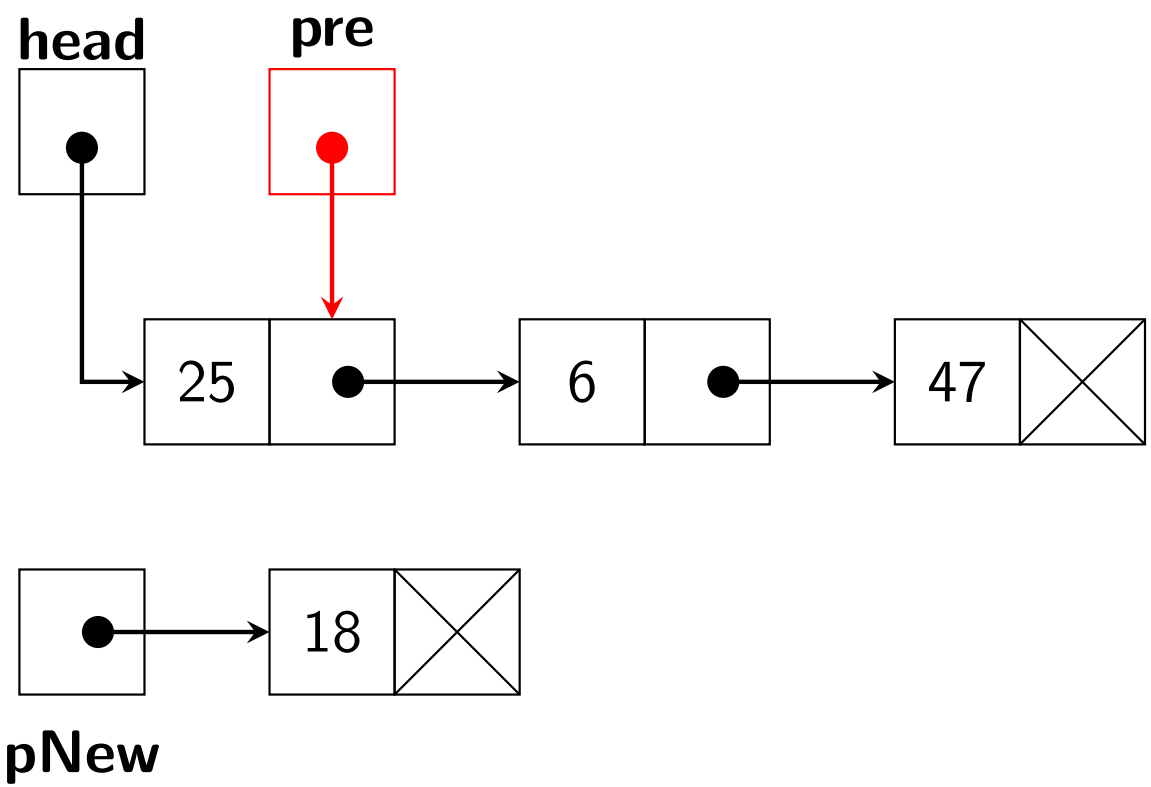


Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

1. Cấp phát một nút mới chứa số 18, giữ bởi con trỏ `pNew`.

ThS.
Trần Ngọc Bảo Duy

Thêm vào vị trí bất kỳ của danh sách không rỗng

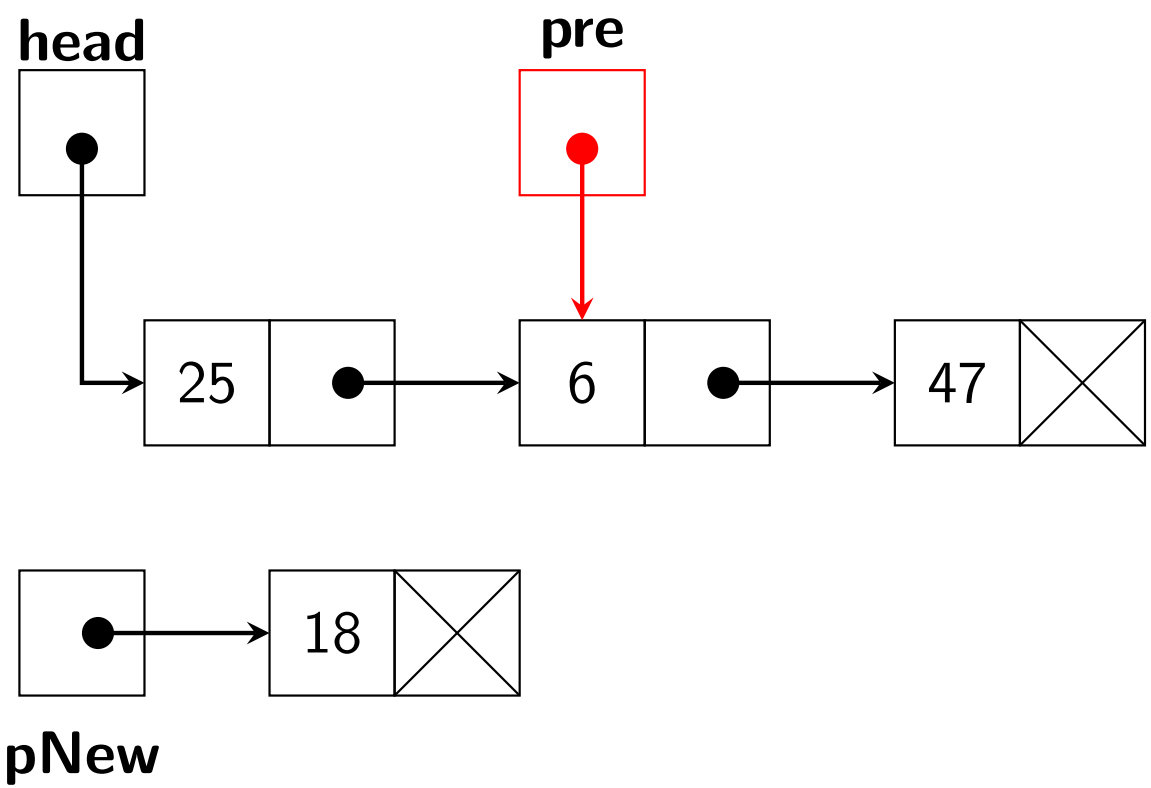


Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

1. Cấp phát một nút mới chứa số 18, giữ bởi con trỏ `pNew`.
2. Tìm ra địa chỉ (con trỏ) của nút tại vị trí **1**.

ThS.
Trần Ngọc Bảo Duy

Thêm vào vị trí bất kỳ của danh sách không rỗng

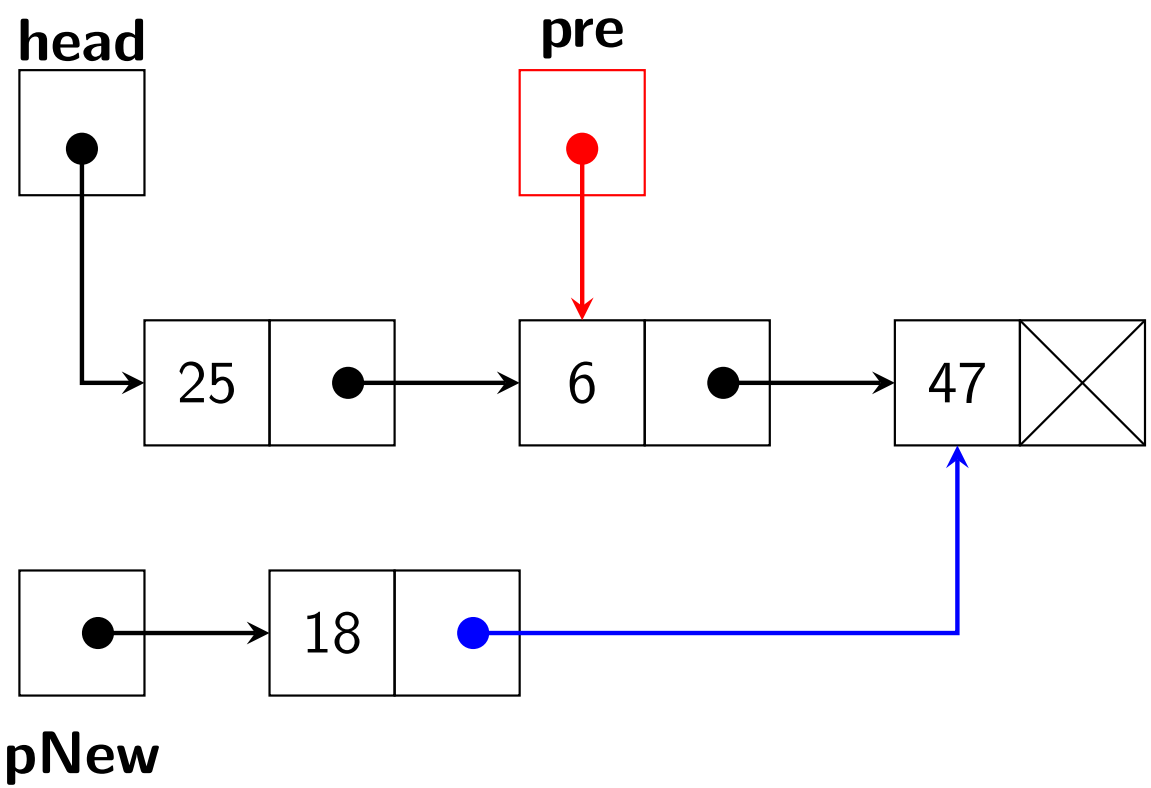


Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

1. Cấp phát một nút mới chứa số 18, giữ bởi con trỏ `pNew`.
2. Tìm ra địa chỉ (con trỏ) của nút tại vị trí **1**.

ThS.
Trần Ngọc Bảo Duy

Thêm vào vị trí bất kỳ của danh sách không rỗng

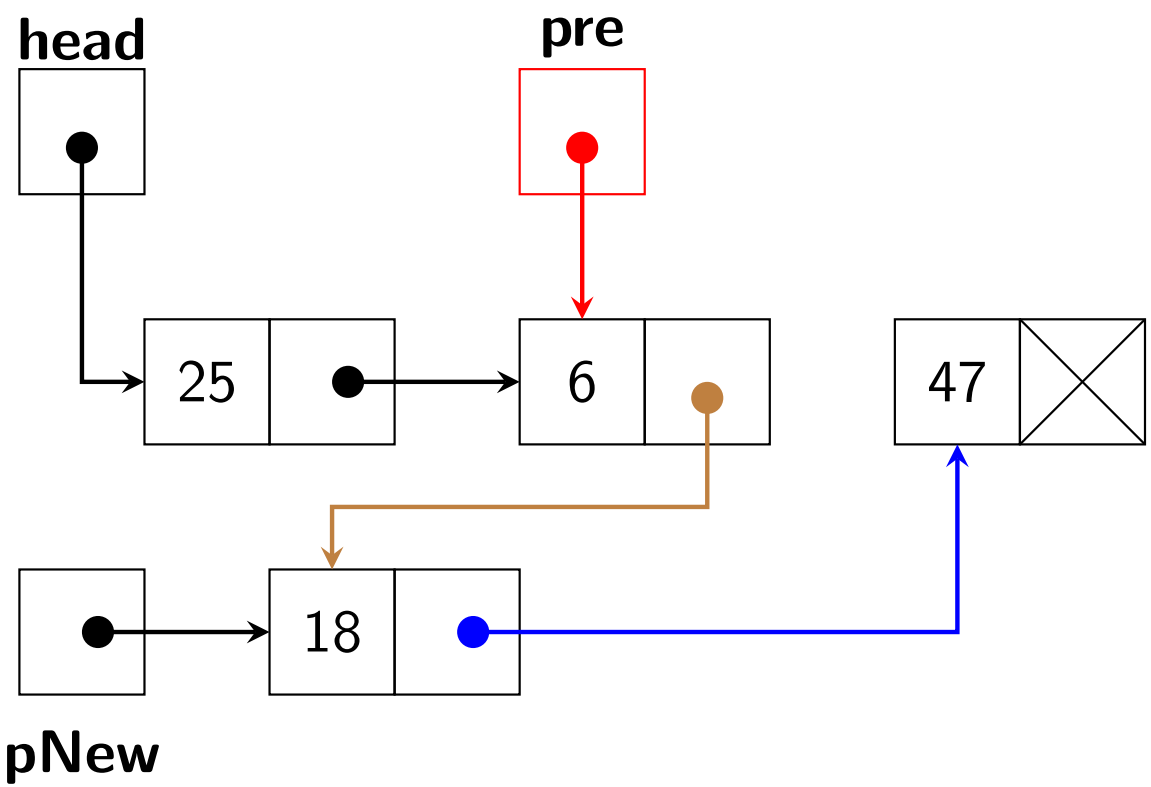


Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

1. Cấp phát một nút mới chứa số 18, giữ bởi con trỏ `pNew`.
2. Tìm ra địa chỉ (con trỏ) của nút tại vị trí **1**.
3. pNew->next = pre->next

ThS.
Trần Ngọc Bảo Duy

Thêm vào vị trí bất kỳ của danh sách không rỗng



Khi thêm phần tử số 18 vào vị trí **2** của DS không rỗng:

1. Cấp phát một nút mới chứa số 18, giữ bởi con trỏ `pNew`.
2. Tìm ra địa chỉ (con trỏ) của nút tại vị trí **1**.
3. pNew->next = pre->next
4. pre->next = pNew

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ra khỏi DSLK đơn

Khi xóa một phần tử ra khỏi DSLK đơn, ta cần quan tâm đến các trường hợp sau:

1. Xóa phần tử ở đầu danh sách.
2. Xóa phần tử ở vị trí bất kỳ trong danh sách.

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ra khỏi DSLK đơn

Khi xóa một phần tử ra khỏi DSLK đơn, ta cần quan tâm đến các trường hợp sau:

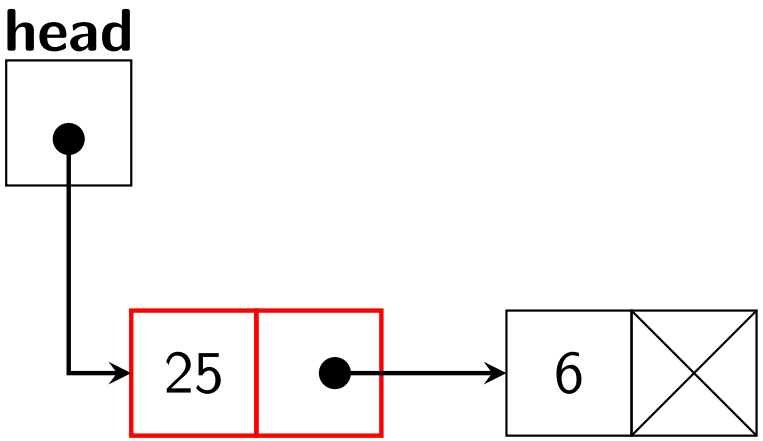
1. Xóa phần tử ở đầu danh sách.
2. Xóa phần tử ở vị trí bất kỳ trong danh sách.

Prototype:

```
bool remove(LList& list, int& rem, int pos = 0);
```


ThS.
Trần Ngọc Bảo Duy

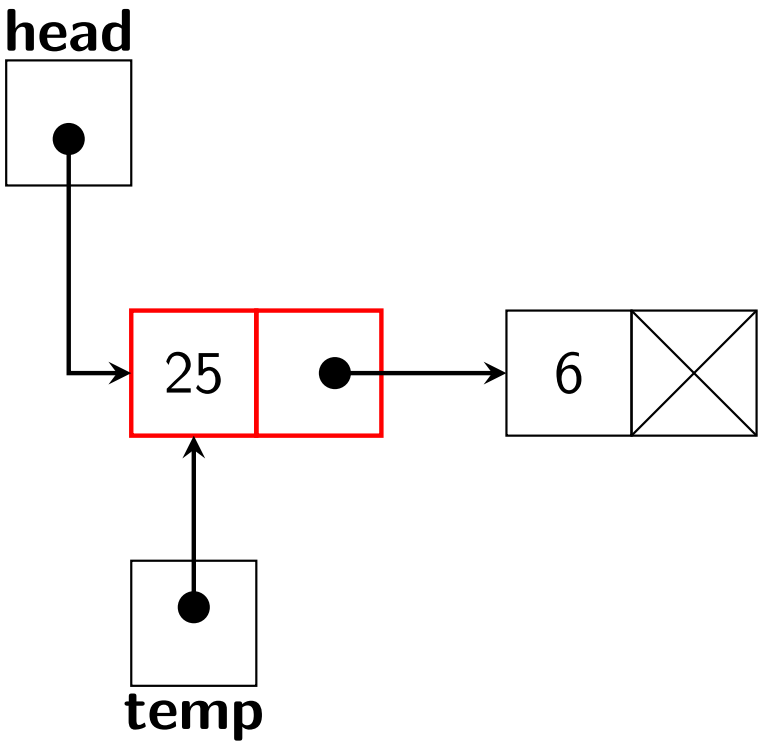
Xóa phần tử ở đầu danh sách



Xóa phần tử đầu tiên ra khỏi DS:

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ở đầu danh sách

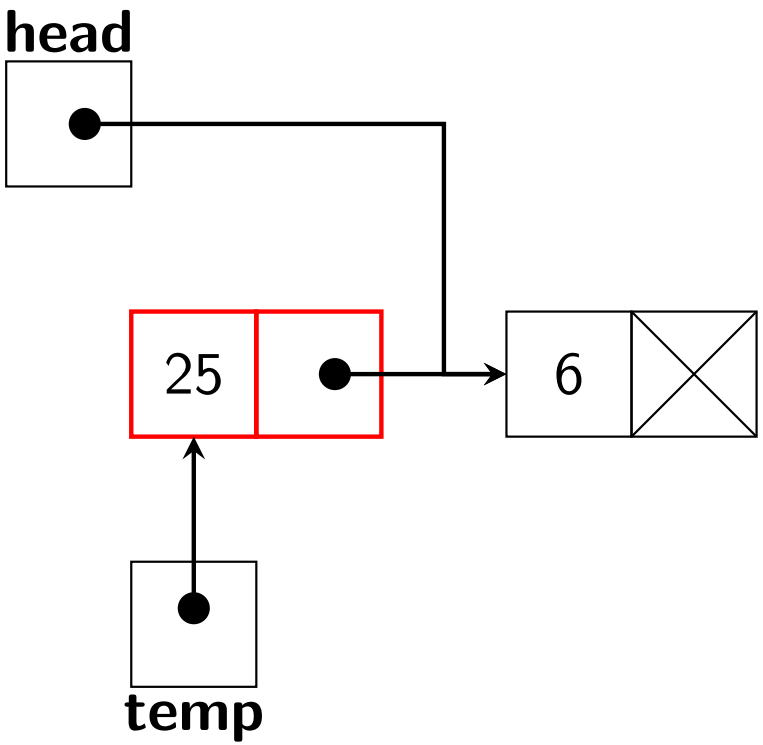


Xóa phần tử đầu tiên ra khỏi DS:

1. Đặt tên con trỏ tạm: `temp = head`

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ở đầu danh sách

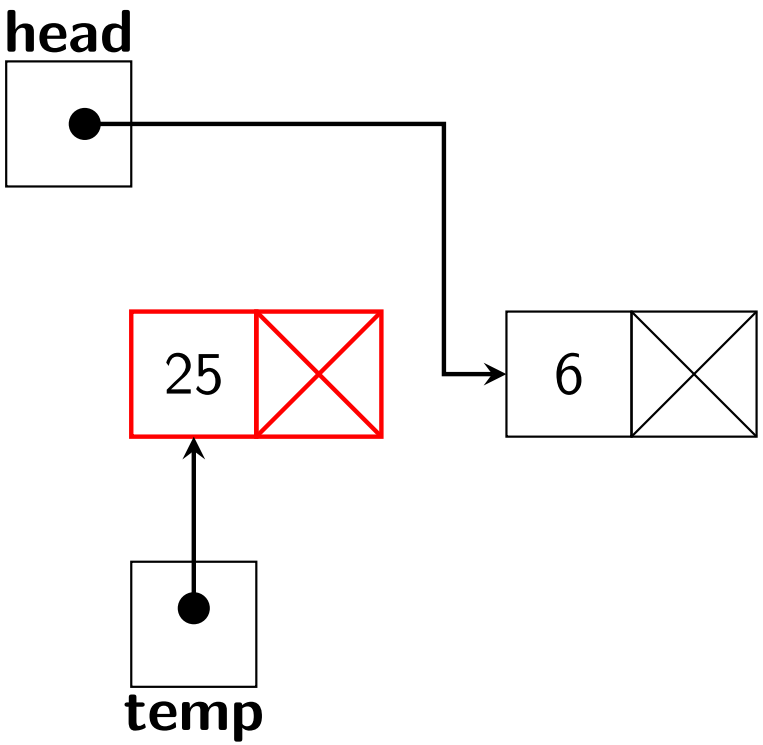


Xóa phần tử đầu tiên ra khỏi DS:

1. Đặt tên con trỏ tạm: `temp = head`
2. Chuyển địa chỉ mà head đang trỏ qua nút cạnh bên: `head = head->next`

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ở đầu danh sách

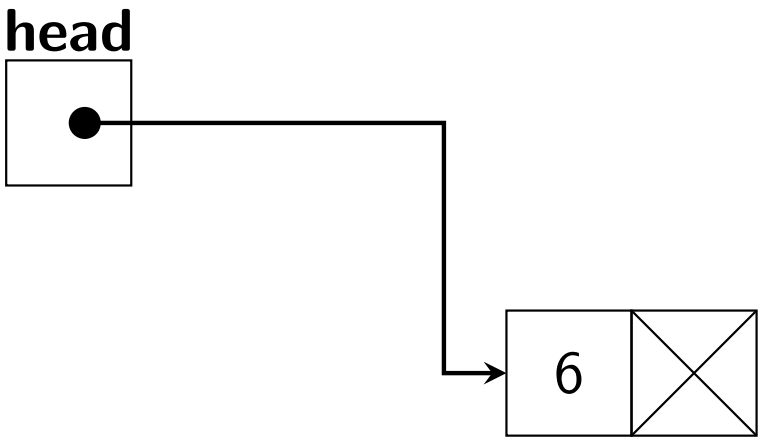


Xóa phần tử đầu tiên ra khỏi DS:

1. Đặt tên con trỏ tạm: `temp = head`
2. Chuyển địa chỉ mà head đang trỏ qua nút cạnh bên: `head = head->next`
3. Cô lập temp: `temp->next = NULL`

ThS.
Trần Ngọc Bảo Duy

Xóa phần tử ở đầu danh sách

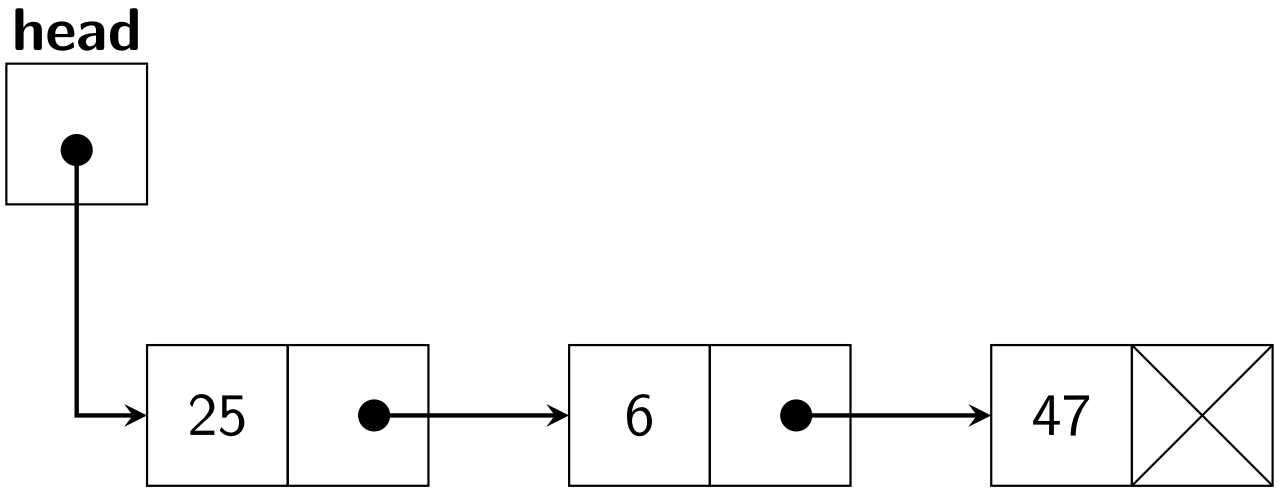


Xóa phần tử đầu tiên ra khỏi DS:

1. Đặt tên con trỏ tạm: `temp = head`
2. Chuyển địa chỉ mà head đang trỏ qua nút cạnh bên: `head = head->next`
3. Cô lập temp: `temp->next = NULL`
4. Xóa bỏ nút mà temp đang trỏ tới: `delete temp`

ThS.
Trần Ngọc Bảo Duy

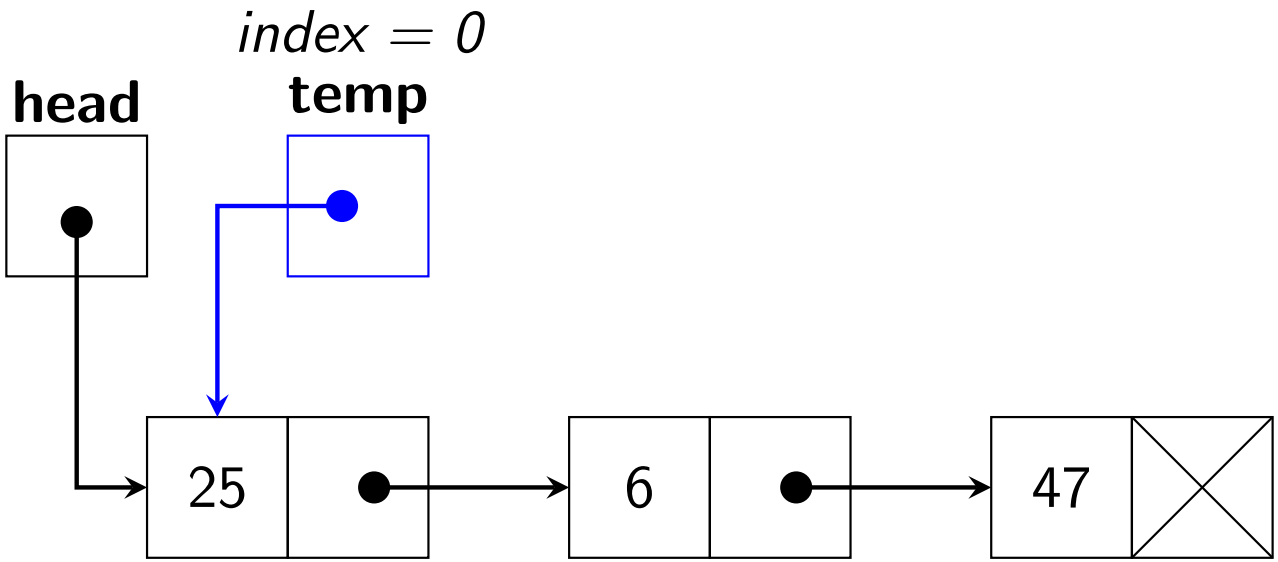
Tìm kiếm phần tử trong danh sách liên kết



Tìm kiếm phần tử tại vị trí `index = 1`:

ThS.
Trần Ngọc Bảo Duy

Tìm kiếm phần tử trong danh sách liên kết

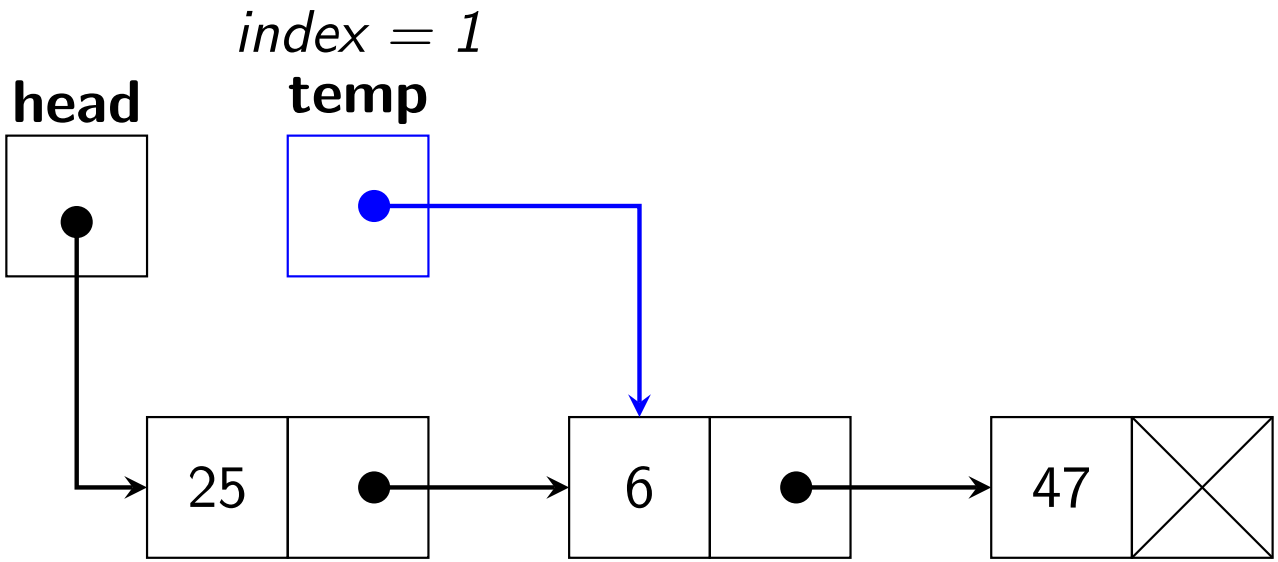


Tìm kiếm phần tử tại vị trí `index = 1`:

1. Đặt con trỏ tạm `temp = head` và biến đếm `index = 0` dùng để chạy.

ThS.
Trần Ngọc Bảo Duy

Tìm kiếm phần tử trong danh sách liên kết

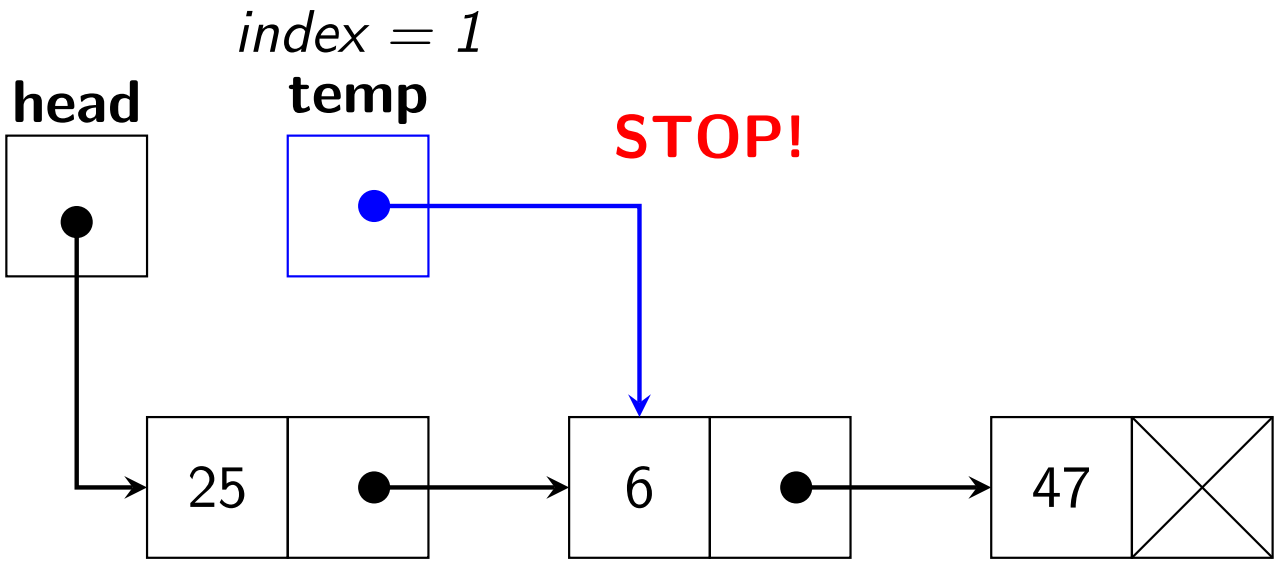


Tìm kiếm phần tử tại vị trí `index = 1`:

1. Đặt con trỏ tạm `temp = head` và biến đếm `index = 0` dùng để chạy.
2. Nếu `index` đang mang giá trị cần tìm thì dừng lại.
3. Nếu không, ta trỏ `temp = temp->next` và tăng biến đếm lên 1 đơn vị.

ThS.
Trần Ngọc Bảo Duy

Tìm kiếm phần tử trong danh sách liên kết



Tìm kiếm phần tử tại vị trí `index = 1`:

1. Đặt con trỏ tạm `temp = head` và biến đếm `index = 0` dùng để chạy.
2. Nếu `index` đang mang giá trị cần tìm thì dừng lại.
3. Nếu không, ta trỏ `temp = temp->next` và tăng biến đếm lên 1 đơn vị.
4. Quay lại bước 2.

ThS.
Trần Ngọc Bảo Duy

So sánh giữa hai phương pháp hiện thực danh sách

1. **Hiện thực bằng mảng được ưu tiên sử dụng khi**:
   - số lượng phần tử nhỏ;
   - kích thước danh sách nên được biết khi viết chương trình;
   - số lượng thao tác thêm và xóa ít được diễn ra;
   - thường xuyên truy cập ngẫu nhiên

ThS.
Trần Ngọc Bảo Duy

So sánh giữa hai phương pháp hiện thực danh sách

1. **Hiện thực bằng mảng được ưu tiên sử dụng khi**:
   - số lượng phần tử nhỏ;
   - kích thước danh sách nên được biết khi viết chương trình;
   - số lượng thao tác thêm và xóa ít được diễn ra;
   - thường xuyên truy cập ngẫu nhiên

2. **Danh sách liên kết sẽ được ưu tiên khi**:
   - số lượng phần tử khá lớn;
   - kích thước danh sách không được biết trước tại thời điểm viết;
   - thao tác thêm, xóa, tái sắp xếp danh sách thường xuyên diễn ra.

ThS.
Trần Ngọc Bảo Duy