

Cấu trúc dữ liệu và thuật toán

PGS. TS. Phạm Tuấn Minh

Trường Công nghệ Thông tin, Đại học Phenikaa

minh.phamtuan@phenikaa-uni.edu.vn

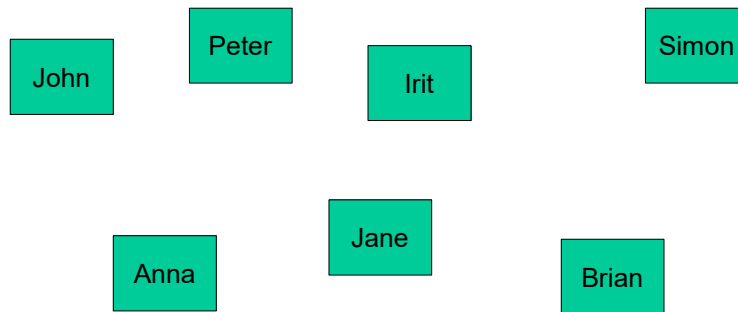
<https://sites.google.com/site/phamtuanminh/>

Chương 2: Mảng và danh sách liên kết

- ❑ Cấu trúc lưu trữ mảng
- ❑ **Danh sách liên kết**
 - **Giới thiệu danh sách liên kết**
 - Cài đặt danh sách liên kết
 - Các thao tác trên danh sách liên kết
- ❑ Ngăn xếp
- ❑ Hàng đợi

Cấu trúc dữ liệu tuyến tính

- Giả sử có một tập các tên



1-3

Cấu trúc dữ liệu tuyến tính

- Nếu các tên sắp thành một danh sách xếp hàng



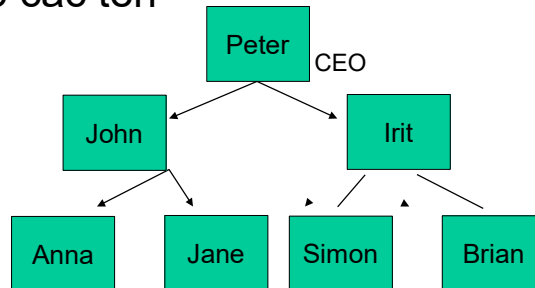
Danh sách

- Quản lý danh sách dữ liệu trên như thế nào ?

1-4

Cấu trúc dữ liệu không tuyến tính

- Tập các tên



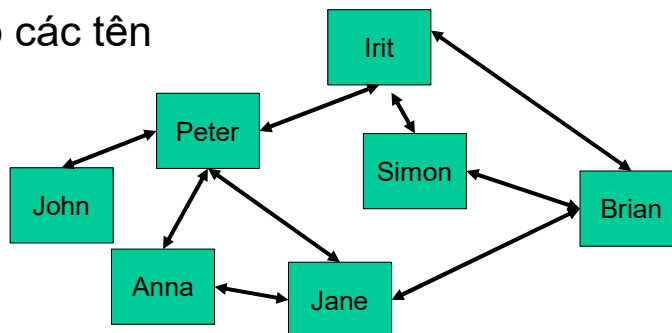
Cây

- Tổ chức công ty

1-5

Cấu trúc dữ liệu không tuyến tính

- Tập các tên



Đồ thị

- Mạng bạn bè

1-6

Cấu trúc dữ liệu đơn giản nhất

- Danh sách



- Dữ liệu tuần tự
 - Thứ tự giữa các phần tử (No.1, No.2., No.3, ...)
 - Mỗi phần tử có một vị trí trong chuỗi
 - Mỗi phần tử đến sau phần tử khác
- Lưu trữ danh sách các phần tử
 - Danh sách tên, danh sách số, ...
 - Dùng mảng để lưu trữ danh sách: Hạn chế?
 - Dùng danh sách liên kết để lưu trữ danh sách

1-7

Danh sách liên kết

- Chúng ta muốn
 - Dễ dàng thêm một phần tử mới vào bất kì vị trí nào trong danh sách
 - Dễ dàng xóa một phần tử trong danh sách
 - Dễ dàng di chuyển vị trí của một phần tử trong danh sách
- Mảng không hỗ trợ các yêu cầu này
 - Hỗ trợ truy cập ngẫu nhiên nhanh: Vị trí trong danh sách = vị trí trong mảng

arr[0] arr[1] arr[2] arr[3] arr[4]

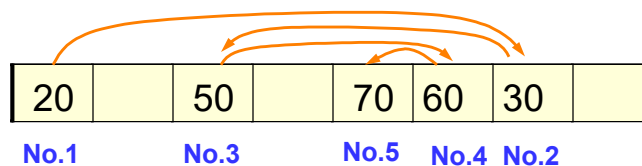
20	30	50	60	70			
----	----	----	----	----	--	--	--

No.1 No.2 No.3 No.4 No.5

1-8

Danh sách liên kết

- Danh sách liên kết
 - Vị trí trong danh sách khác vị trí trong bộ nhớ
 - Phần tử có thể lưu trữ ở bất kì vị trí nào
 - Cần thêm dữ liệu để chỉ ra vị trí trong danh sách
 - Liên kết (Link): Con trỏ tới phần tử tiếp theo
 - Danh sách liên kết (Linked list): Các nút với các liên kết



1-9

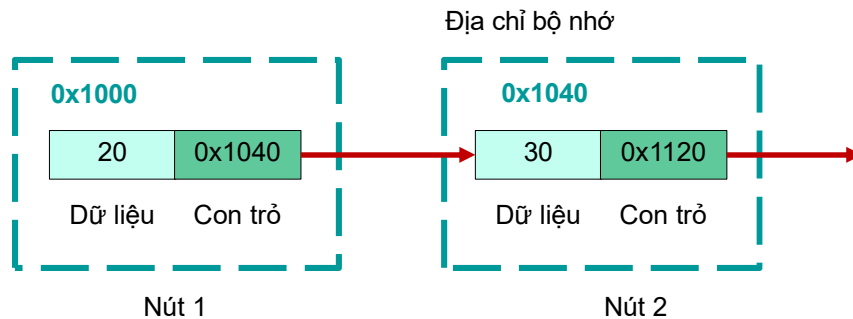
Nút trong danh sách liên kết

- Mỗi nút có cấu trúc ListNode
- Một nút cơ bản có hai thành phần
 - Dữ liệu lưu trữ bởi nút: integer, char, ...
 - Liên kết: Con trỏ tham chiếu tới nút tiếp theo trong danh sách

```
typedef struct _listnode{  
    int num;  
    struct _listnode *next;  
}ListNode;
```

1-10

Nút trong danh sách liên kết



```
typedef struct _listnode{  
    int num;  
    struct _listnode *next;  
} ListNode;
```

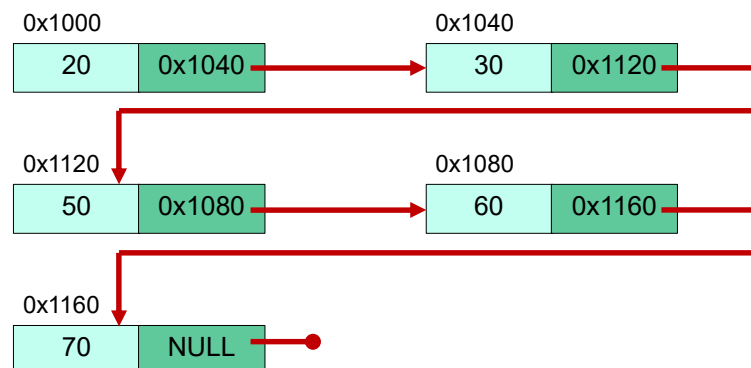
Mỗi structure có hai phần

- Dữ liệu
- Liên kết

1-11

Nút trong danh sách liên kết

- Mỗi nút chứa một phần tử
- Con trỏ của mỗi nút chỉ tới nút tiếp theo
- Mỗi nút có thể được lưu trữ ở bất kì vị trí nào (không cần liên tục) trong bộ nhớ
- Các nút có thể tạo động bằng malloc()



1-12

Truy cập tới phần tử trong danh sách

- Đối với danh sách lưu trữ trong mảng:
 - Dễ dàng truy cập phần tử thứ i trong danh sách: $\text{arr}[i-1]$
 - Phần tử tiếp theo của $\text{arr}[i]$ trong danh sách được lưu trữ trong $\text{arr}[i+1]$
 - Phần tử trước phần tử chứa trong $\text{arr}[i]$ trong danh sách được lưu trữ trong $\text{arr}[i-1]$

$\text{arr}[0]$ $\text{arr}[1]$ $\text{arr}[2]$ $\text{arr}[3]$ $\text{arr}[4]$

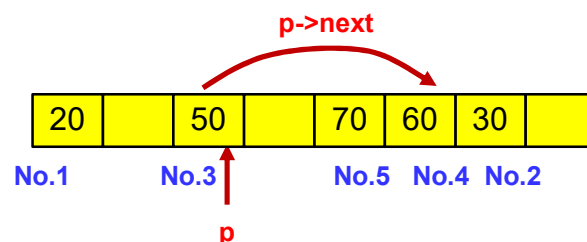
20	30	50	60	70			
----	----	----	----	----	--	--	--

No.1 No.2 No.3 No.4 No.5

1-13

Truy cập tới phần tử trong danh sách

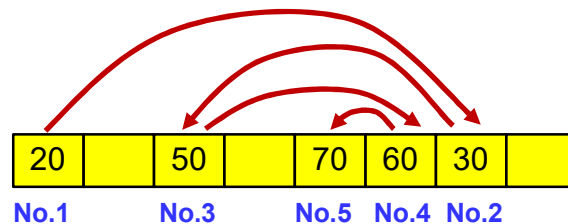
- Đối với danh sách lưu trữ trong mảng: Dễ thực hiện
- Đối với danh sách lưu trữ trong danh sách liên kết:
 - Mỗi nút theo dấu của nút tiếp theo sau nó
 - Nếu p chỉ tới phần tử thứ i trong danh sách, $p \rightarrow \text{next}$ chỉ tới phần tử thứ $(i+1)$



1-14

Theo vết các phần tử trong danh sách

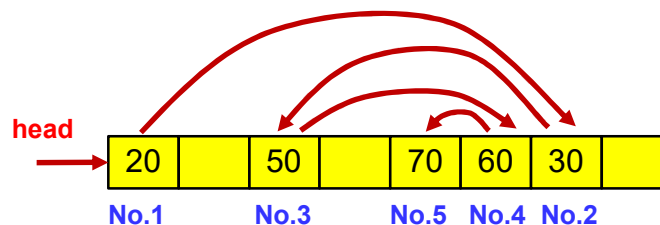
- Đối với danh sách lưu trữ trong mảng: dễ thực hiện
- Đối với danh sách lưu trữ trong danh sách liên kết:
 - Mỗi nút theo dấu của nút tiếp theo sau nó
 - Nếu **p** chỉ tới phần tử thứ *i* trong danh sách, **p->next** chỉ tới phần tử thứ (*i*+1)
 - Mọi nút trong danh sách đều truy cập bắt đầu từ nút đầu tiên trong danh sách



1-15

Theo vết các phần tử trong danh sách

- Không thể truy cập tới các nút trong danh sách liên kết nếu không có địa chỉ của nút đầu tiên
- Cần một biến con trỏ chỉ tới nút đầu tiên: **head**



1-16

So sánh lưu trữ danh sách bằng mảng và danh sách liên kết

- Truy cập ngẫu nhiên tới các phần tử trong danh sách
 - **Mảng: Dễ**
 - **Danh sách liên kết: Không dễ**
- Thay đổi danh sách: thêm, xóa các phần tử
 - **Mảng: Không dễ**
 - **Linked List: Dễ**

1-17

Chương 2: Mảng và danh sách liên kết

- Cấu trúc lưu trữ mảng
- Danh sách liên kết
 - Giới thiệu danh sách liên kết
 - **Cài đặt danh sách liên kết**
 - Các thao tác trên danh sách liên kết
- Ngăn xếp
- Hàng đợi

1-18

Nút trong danh sách liên kết

- Mỗi nút có cấu trúc ListNode
- Một nút cơ bản có hai thành phần
 - Dữ liệu lưu trữ bởi nút: integer, char, ...
 - Liên kết: Con trỏ tham chiếu tới nút tiếp theo trong danh sách

```
typedef struct _listnode{  
    int num;  
    struct _listnode *next;  
} ListNode;
```

```
typedef struct _listnode{  
    int num;  
    char name[20];  
    ...  
    struct _listnode *next;  
} ListNode;
```

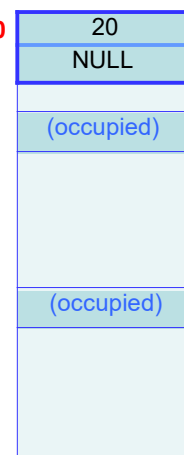
1-19

Tạo nút

```
ListNode *newNode;  
newNode=malloc(sizeof(ListNode));  
newNode->num=20;  
newNode->next=NULL;
```

newNode = 0x0100

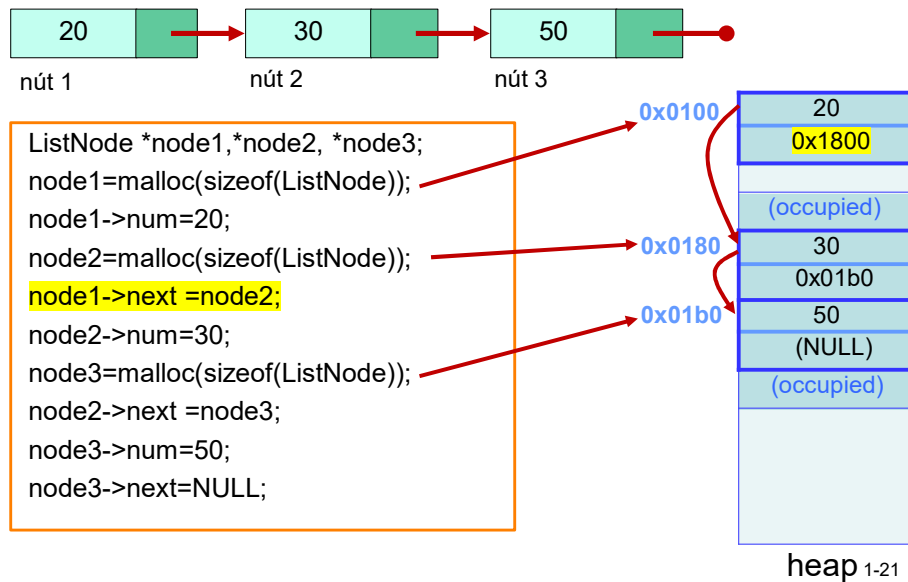
0x0100



heap 1-20

- Tạo một nút động
 - Trong thời gian chạy
 - Sử dụng malloc()

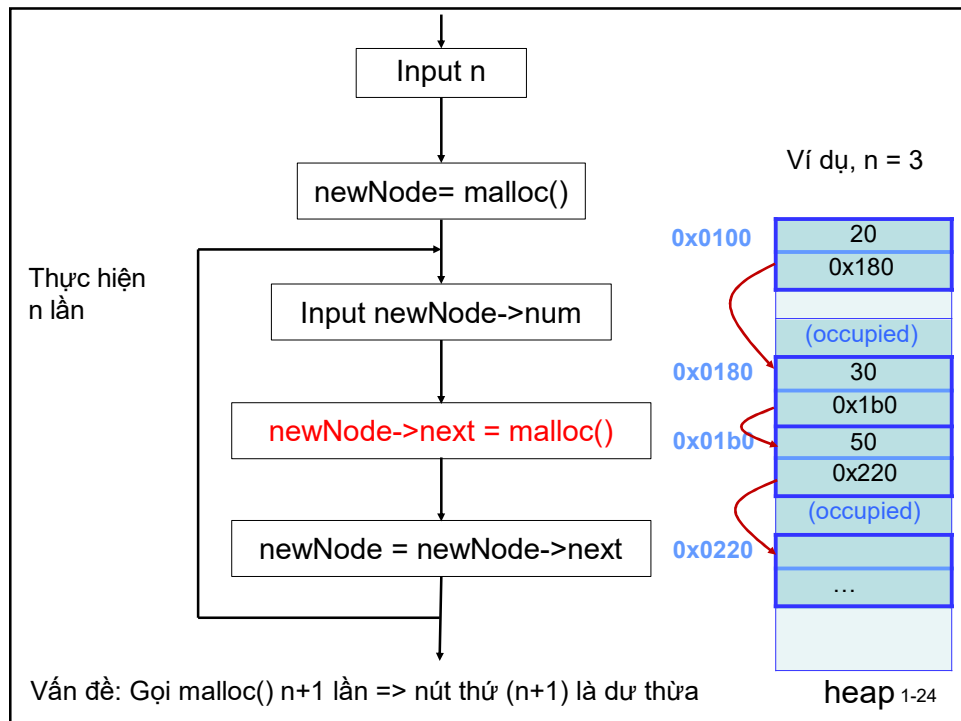
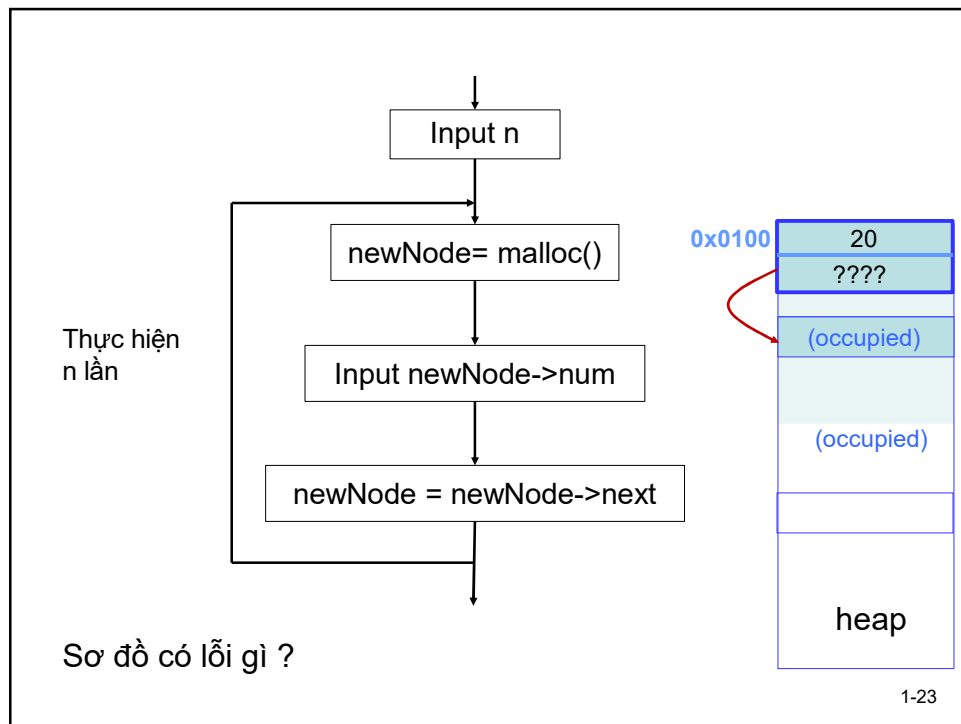
Tạo một danh sách liên kết có 3 nút

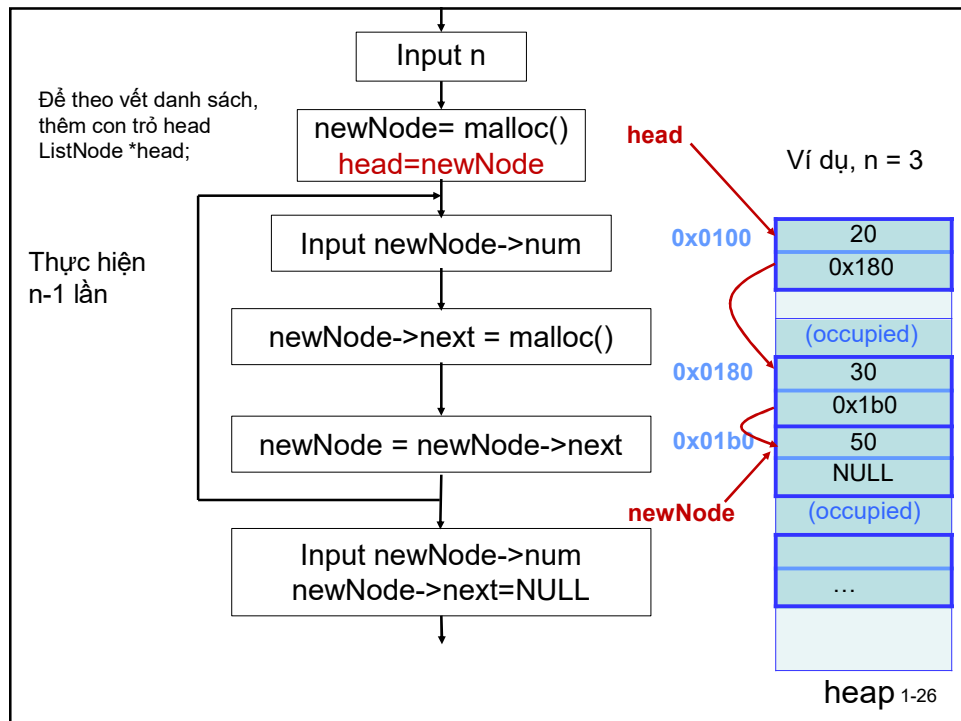
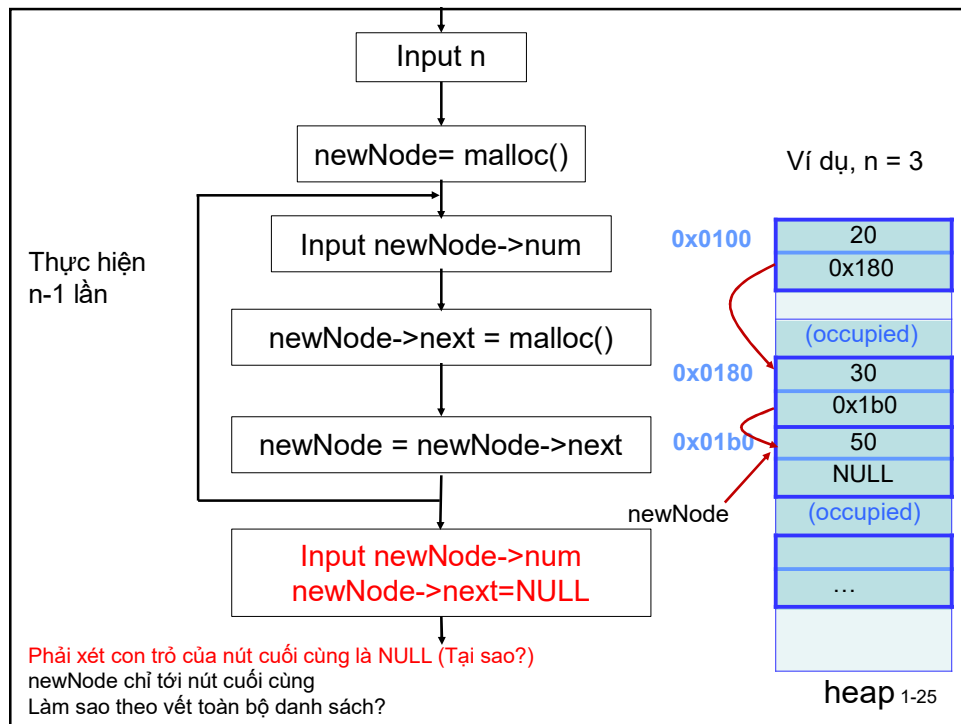


Tạo danh sách liên kết có n nút

Viết một chương trình yêu cầu người dùng nhập vào số lượng số nguyên sẽ nhập n (giả sử $n > 0$) và sau đó hỏi giá trị từng số nguyên.

- Lặp n lần: dùng malloc() để tạo nút mới, sau đó thêm nút này vào danh sách liên kết





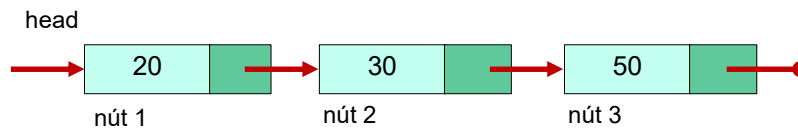
Tạo danh sách liên kết có $n > 0$ nút

```
int n;  
ListNode *newNode, *head;  
scanf("%d", &n);  
newNode=malloc(sizeof(ListNode));  
head=newNode;  
for (int i=1; i<n; i++){  
    scanf("%d", &newNode->num);  
    newNode->next=malloc(sizeof(ListNode));  
    newNode=newNode->next;  
}  
scanf("%d", &newNode->num); //the last num  
newNode->next=NULL;
```

1-27

Ví dụ

- $n = 3$, dữ liệu 20, 30, 50
- Kết quả chạy chương trình cho danh sách liên kết như sau



- Theo vết các phần tử trong danh sách:
 - head chỉ tới node1, head->num là 20
 - head->next chỉ tới node2, head->next->num là 30
 - head->next->next chỉ tới node3, head->next->next->num là 50
 - head->next->next->next là NULL

1-28

Tạo danh sách liên kết có $n \geq 0$ nút

```
int n;  
ListNode *newNode, *head=NULL;  
scanf("%d", &n);  
if (n>0) {  
    newNode=malloc(sizeof(ListNode));  
    head=newNode;  
    for (int i=1; i<n; i++){  
        scanf("%d", &newNode->num);  
        newNode->next=malloc(sizeof(ListNode));  
        newNode=newNode->next;  
    }  
    scanf("%d", &newNode->num); //the last num  
    newNode->next=NULL;  
}
```

1-29

Chương 2: Mảng và danh sách liên kết

- ❑ Cấu trúc lưu trữ mảng
- ❑ Danh sách liên kết
 - Giới thiệu danh sách liên kết
 - Cài đặt danh sách liên kết
 - **Các thao tác trên danh sách liên kết**
- ❑ Ngăn xếp
- ❑ Hàng đợi

1-30

Tạo danh sách liên kết có n nút

Viết một chương trình yêu cầu người dùng nhập vào số lượng số nguyên sẽ nhập n (giả sử $n > 0$) và sau đó hỏi giá trị từng số nguyên.

Sau đó người dùng có thể liên tục tạo các thay đổi: thêm mới, xóa phần tử trong danh sách

- Để tránh lặp lại mã chương trình, cần viết các hàm cho một số thao tác cơ bản

1-31

Các hàm cơ bản cho danh sách liên kết

□ Các thao tác cơ bản

○ Chèn nút mới

- Vào đầu
- Vào cuối
- Vào giữa

InsertNode()

○ Xóa một nút

- Ở đầu
- Ở cuối
- Ở giữa

RemoveNode()

○ Liệt kê toàn bộ các phần tử trong danh sách

PrintList()

○ Tìm kiếm nút có chỉ số i trong danh sách

FindNode()

1-32

PrintList()

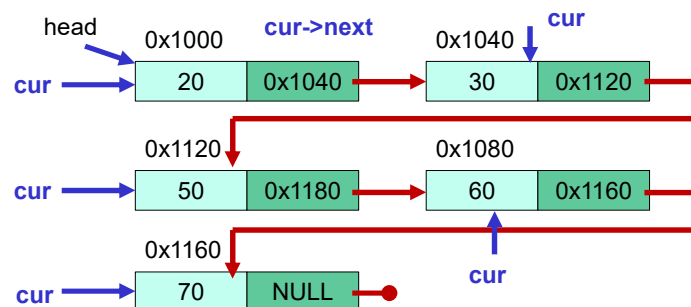
- ❑ Liệt kê tất cả các phần tử trong danh sách liên kết bắt đầu từ phần tử đầu tiên và duyệt danh sách tới phần tử cuối cùng
- ❑ Chuyển con trỏ head vào hàm
`void printList(ListNode *head)`
- ❑ Tại mỗi nút, sử dụng con trỏ next để di chuyển tới phần tử tiếp theo

1-33

PrintList()

```
void printList(ListNode *head){  
    ListNode *cur=head;  
    if (cur== NULL) return;  
    while (cur!= NULL){  
        printf("%d\n", cur ->num);  
        cur = cur ->next;  
    }  
}
```

20
30
50
60
70



1-34

findNode()

- ❑ Tìm con trỏ chỉ tới nút có chỉ số i
- ❑ Truyền con trỏ head vào hàm

`ListNode * findNode(ListNode *head, int i)`

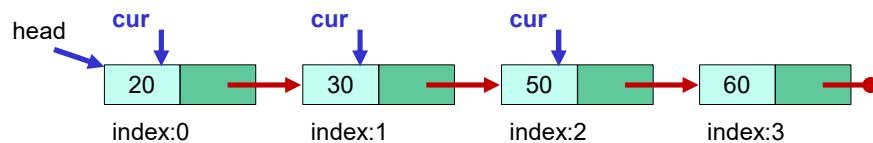
1-35

findNode()

```
ListNode *findNode(ListNode*head, int i) {  
    ListNode *cur=head;  
    if (head==NULL || i<0) return NULL;  
    while(i>0){  
        cur=cur->next;  
        if (cur==NULL) return NULL;  
        i--;  
    }  
    return cur;  
}
```

Khi danh sách trống hoặc chỉ số không hợp lệ

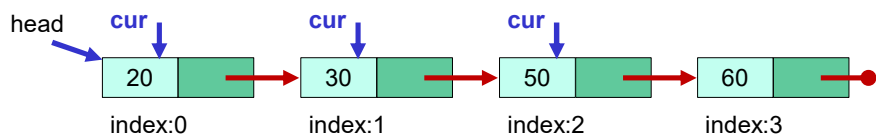
Khi danh sách ngắn hơn chỉ số



1-36

findNode()

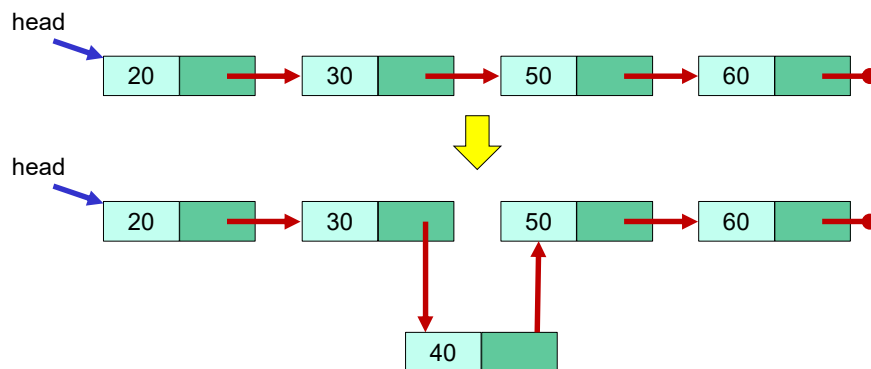
```
ListNode *findNode(ListNode*head, int i) {  
    ListNode *cur=head;  
    if (head==NULL || i<0) return NULL;  
    if (i == 0) return head;  
    while(i>0){  
        cur=cur->next;  
        if (cur==NULL) return NULL;  
        i--;  
    }  
    return cur;  
}
```



1-37

insertNode()

- Thêm nút (40) vào giữa của danh sách liên kết không rỗng
- Thêm nút này vào vị trí có chỉ số 2, ngay sau nút có chỉ số 1



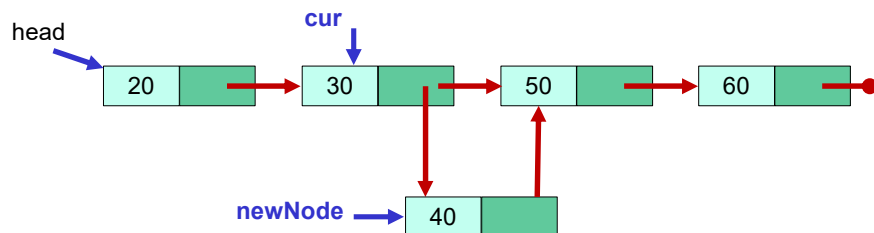
1-38

insertNode()

- ❑ Thêm nút (40) vào giữa của danh sách liên kết không rỗng
- ❑ Thêm nút này vào vị trí có chỉ số 2, ngay sau nút có chỉ số 1

```
newNode->next = cur->next;  
cur->next = newNode;
```

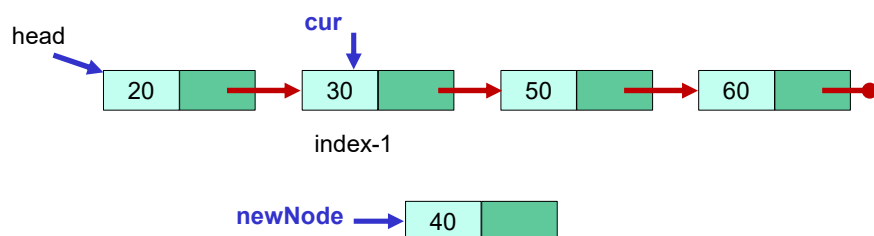
Chú ý thứ tự của lệnh
Điều gì xảy ra nếu thay đổi thứ
tự câu lệnh?



1-39

insertNode(): vị trí giữa, danh sách không trống

- ❑ Sử dụng findNode() để tìm địa chỉ của con trỏ cur
- ❑ Nếu chèn nút mới vào vị trí chỉ số 2, con trỏ cur cần chỉ tới nút có chỉ số 1
findNode(..., **index-1**)

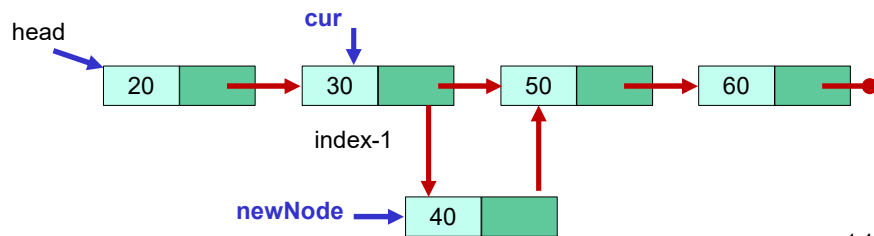


1-40

insertNode(): vị trí giữa, danh sách không trống

```
// Tìm nút trước vị trí cần chèn
// Tạo nút mới và liên kết lại
➡ if ((cur = findNode(*ptrHead, index-1)) != NULL) {
➡   newNode = malloc(sizeof(ListNode));
➡   newNode->num=40;
➡   newNode->next = cur->next;
➡   cur->next = newNode;
}
```

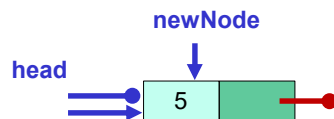
Đúng cả với trường hợp thêm vào cuối của danh sách



1-41

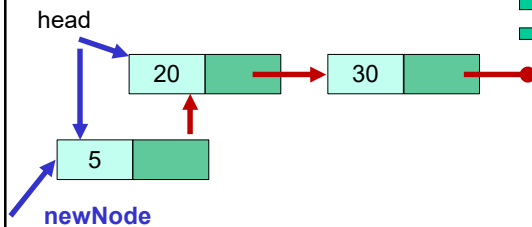
insertNode(): vị trí bất kì, danh sách trống/không trống

□ Trường hợp danh sách trống



```
➡ newNode=malloc(sizeof(ListNode));
➡ newNode->num = 5;
➡ newNode->next =head;
➡ head = newNode;
```

□ Trường hợp chèn vào vị trí chỉ số 0



```
➡ newNode=malloc(sizeof(ListNode));
➡ newNode->num = 5;
➡ newNode->next =head;
➡ head = newNode;
```

1-42

InsertNode()

- ❑ Danh sách tham số của hàm insertNode()?

```
int insertNode(ListNode *head, ... )
```

- ❑ Gợi ý: Có thể thay đổi địa chỉ lưu trong con trỏ head từ trong hàm insertNode() không?

1-43

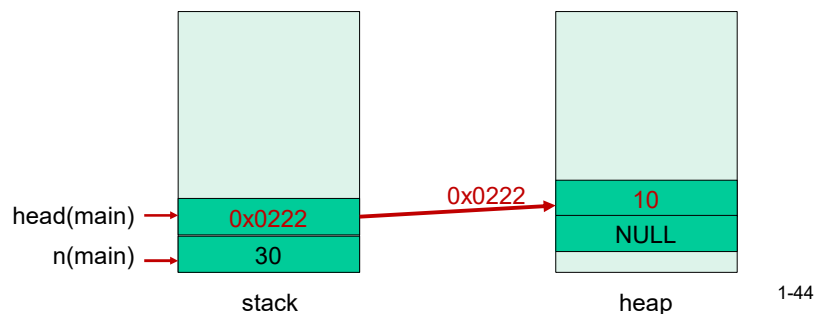
Con trỏ và truyền tham số

main()

```
int n;  
ListNode *head=NULL;  
head=malloc(sizeof(ListNode));  
head->num=10;  
...  
insertNode(head,n,0);
```

insertNode(ListNode *head, int n, int i)

```
if (head==NULL || i = 0) {  
    newNode=malloc(sizeof(ListNode));  
    newNode->num=n;  
    newNode->next =head;  
    head = newNode;  
}  
...
```



1-44

Con trỏ và truyền tham số

```

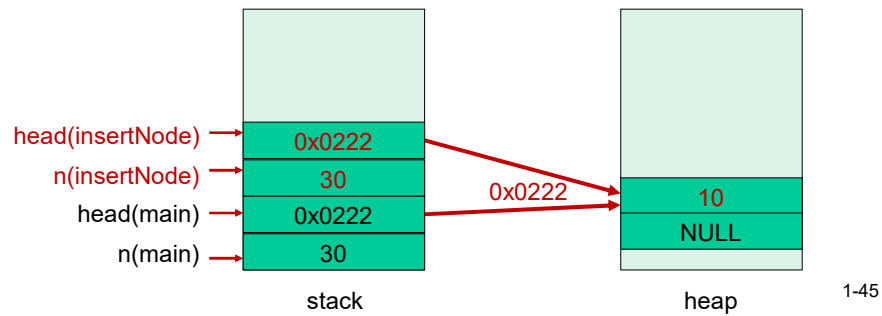
main()
{
    int n;
    ListNode *head=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    ...
    insertNode(head,n,0);
}

```

```

insertNode(ListNode *head, int n, int i)
{
    if (head==NULL || i==0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num=n;
        newNode->next=head;
        head = newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

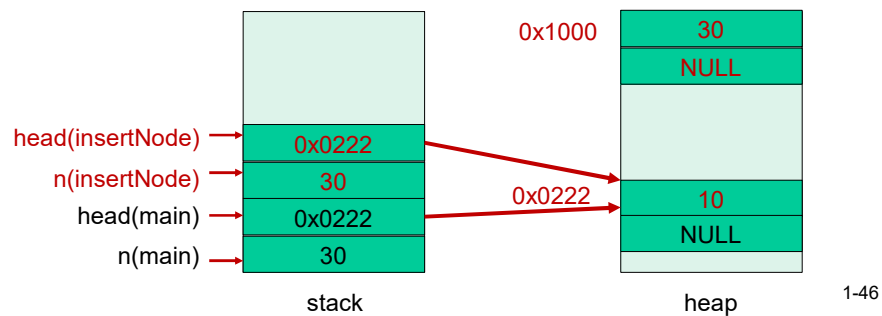
main()
{
    int n;
    ListNode *head=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    ...
    insertNode(head,n,0);
}

```

```

insertNode(ListNode *head, int n, int i)
{
    if (head==NULL || i==0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num=n;
        newNode->next=head;
        head = newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

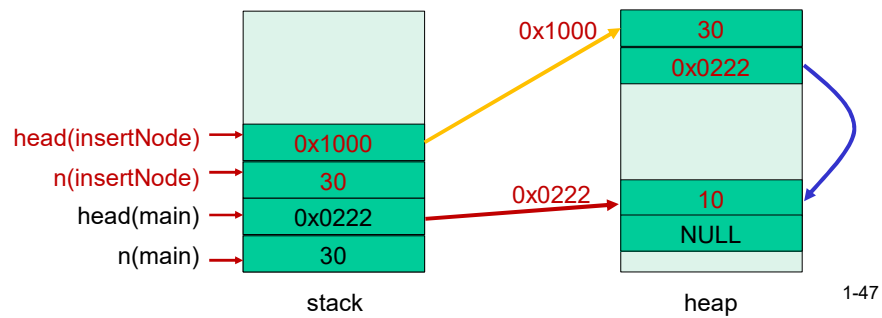
main()
{
    int n;
    ListNode *head=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    ...
    insertNode(head,n,0);
}

```

```

insertNode(ListNode *head, int n, int i)
{
    if (head==NULL || i==0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num=n;
        newNode->next=head;
        head = newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

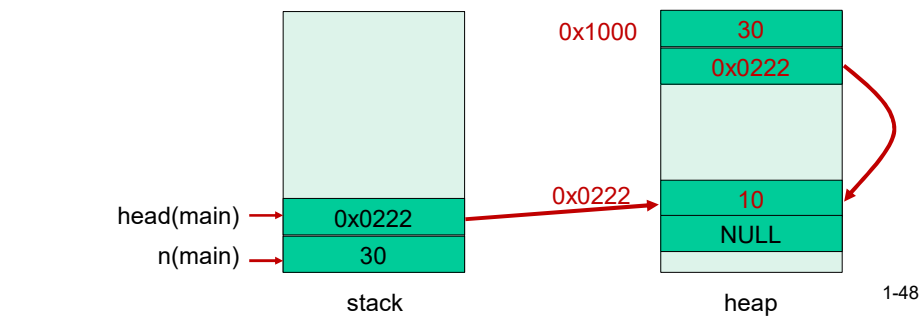
main()
{
    int n;
    ListNode *head=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    ...
    insertNode(head,n,0);
    ...
}

```

```

insertNode(ListNode *head, int n, int i)
{
    if (head==NULL || i==0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num=n;
        newNode->next=head;
        head = newNode;
    }
    ...
}

```



InsertNode()

head(insertNode)

n(insertNode)

head(main)

n(main)

0x1000	
30	
0x0222	
30	

stack

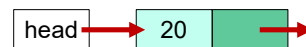
- ❑ Không thực hiện được
`int insertNode(ListNode *head, ...)`
- ❑ Nếu muốn chèn một nút vào một danh sách trống hoặc chèn một nút vào vị trí chỉ số 0 thì cần thay đổi địa chỉ chứa trong con trỏ pointer
- ❑ Tuy nhiên, chỉ giá trị của bản sao của head trong hàm insertNode() thay đổi, giá trị của con trỏ head ngoài insertNode() không thay đổi
- ❑ Cách thay đổi một biến trong hàm?

1-49

Con trỏ và truyền tham số

- ❑ Truyền vào một con trỏ, chỉ tới biến mà chúng ta muốn thay đổi
- ❑ Biến muốn thay đổi là con trỏ head

`ListNode *head`



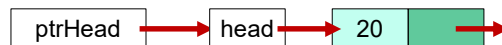
- ❑ Cần truyền vào con trỏ chỉ tới con trỏ head

`ListNode **head`



- ❑ Có thể viết lại như sau

`ListNode **ptrHead`



1-50

Con trỏ và truyền tham số

```

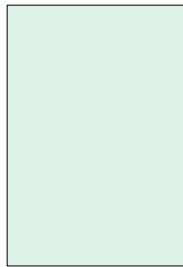
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead==NULL || i=0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num = n;
        newNode->next =*ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



stack



heap

1-51

Con trỏ và truyền tham số

```

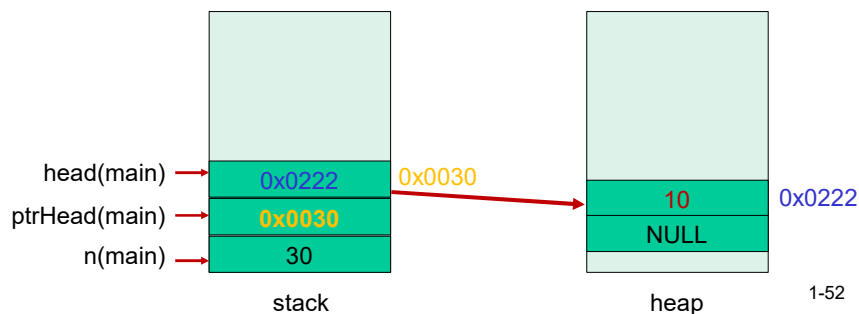
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead==NULL || i=0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num=n;
        newNode-> next =*ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



1-52

Con trỏ và truyền tham số

```

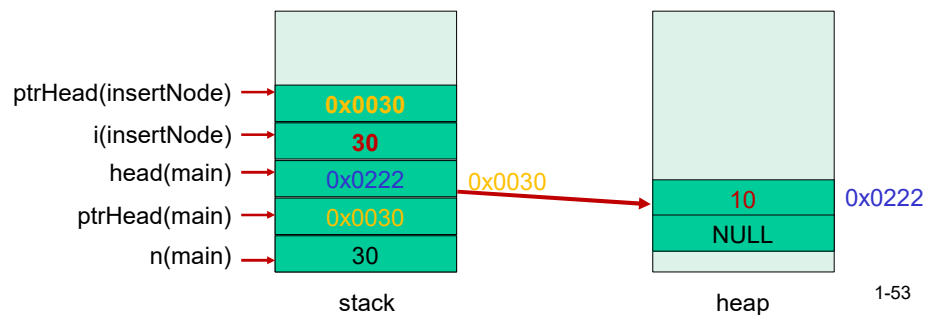
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead == NULL || i=0) {
        newNode = malloc(sizeof(ListNode));
        newNode->num = n;
        newNode->next = *ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

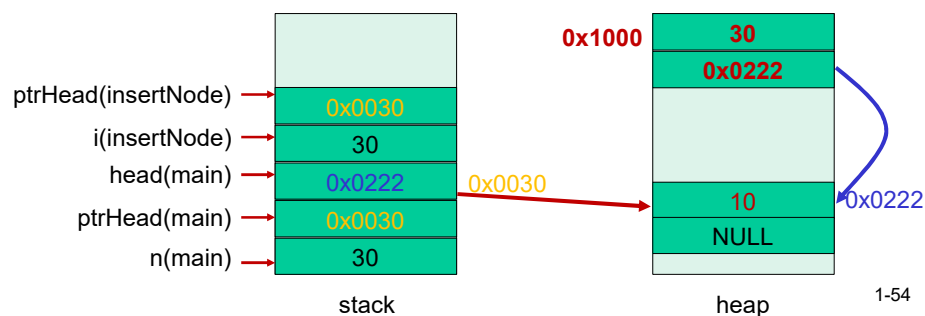
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead==NULL || i=0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num = n;
        newNode-> next = *ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

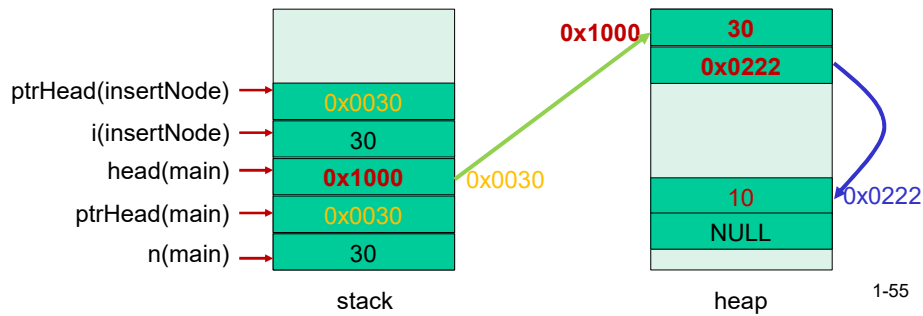
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead==NULL || i=0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num = n;
        newNode -> next = *ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



Con trỏ và truyền tham số

```

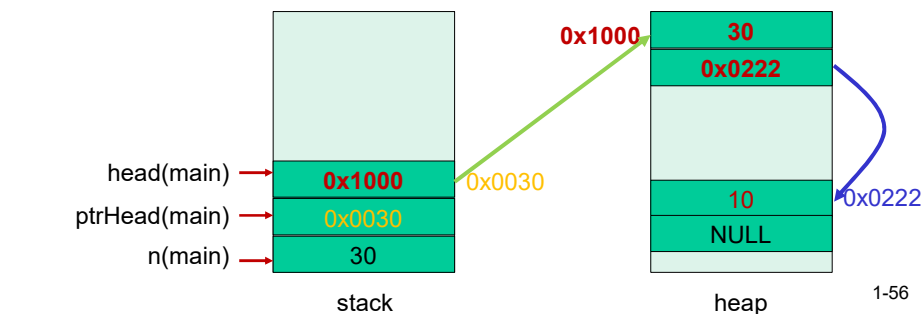
main()
{
    int n;
    ListNode **ptrHead=NULL;
    head=malloc(sizeof(ListNode));
    head->num=10;
    head->next= NULL;
    ...
    ptrHead=&head;
    insertNode(ptrHead,30,0);
}

```

```

insertNode(ListNode **ptrHead, int n, int
i)
{
    if (*ptrHead==NULL || i=0) {
        newNode=malloc(sizeof(ListNode));
        newNode->num = n;
        newNode -> next = *ptrHead;
        *ptrHead= newNode;
    }
    ...
}

```



InsertNode()

□ Chúng ta đã hoàn thành code cho hàm insertNode()

- Đã xem xét việc chèn một nút mới tại mọi vị trí
 - Trước
 - Sau
 - Giữa
- Đã xem xét tất cả các trạng thái của danh sách
 - Trống
 - Một nút
 - Nhiều nút

1-57

InsertNode()

```
void insertNode(ListNode **ptrHead, int index, int value){
    ListNode *cur, *newNode;
    // If empty list or inserting first node, need to update head pointer
    if (*ptrHead == NULL || index == 0){
        newNode = malloc(sizeof(ListNode));
        newNode->num = value;
        newNode->next = *ptrHead;
        *ptrHead = newNode;
    }
    // Find the nodes before and at the target position
    // Create a new node and reconnect the links
    else if ((cur = findNode(*ptrHead, index-1)) != NULL){
        newNode = malloc(sizeof(ListNode));
        newNode->num = value;
        newNode->next = cur->next;
        cur->next = newNode;
    }
    else printf(" can not insert the new item at index %d!\n", index);
}
```

1-58

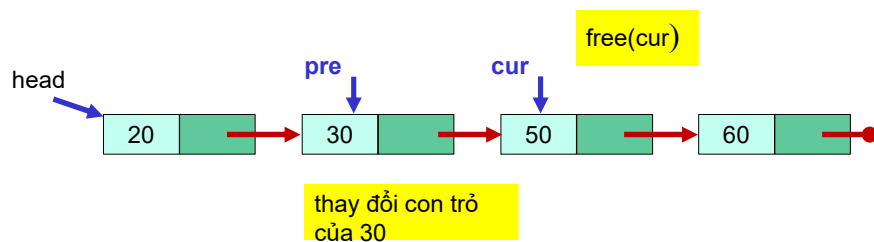
Phiên bản khác của InsertNode()

```
int insertNode(ListNode **ptrHead, int index, int value){
    ListNode *pre, *cur;
    // If empty list or inserting first node, need to update head pointer
    if (*ptrHead == NULL || index == 0){
        cur = *ptrHead;
        *ptrHead = malloc(sizeof(ListNode));
        (*ptrHead)->num = value;
        (*ptrHead)->next = cur;
        return 0;
    }
    // Find the nodes before and at the target position
    // Create a new node and reconnect the links
    if ((pre = findNode(*ptrHead, index-1)) != NULL){
        cur = pre->next;
        pre->next = malloc(sizeof(ListNode));
        pre->next->num = value;
        pre->next->next = cur;
        return 0;
    }
    return -1;
}
```

1-59

removeNode(..., 2)

- Tìm nút, loại bỏ
- Giải phóng bộ nhớ không sử dụng



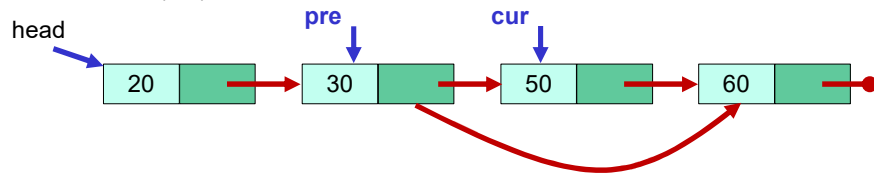
1-60

removeNode(..., 2)

- Tìm nút, loại bỏ
- Giải phóng bộ nhớ không sử dụng

Nếu $pre \rightarrow next$ là nút cần xóa

```
cur = pre->next;  
pre->next = cur->next;  
free(cur);
```

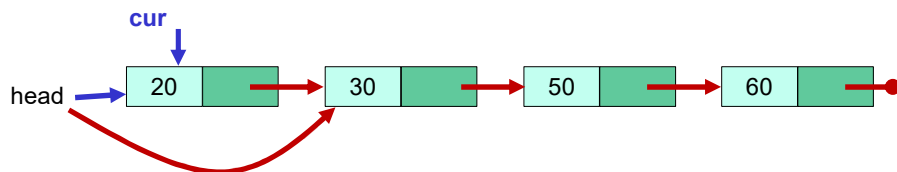


1-61

removeNode(..., 0)

- Nếu nút muốn xóa là nút đầu tiên

```
cur = head;  
head = cur->next;  
free(cur);
```



1-62

removeNode(..., i)

❑ void removeNode(ListNode **ptrHead, int index);

1-63

Chương 2: Mảng và danh sách liên kết

- ❑ Cấu trúc lưu trữ mảng
- ❑ Danh sách liên kết
 - Giới thiệu danh sách liên kết
 - Cài đặt danh sách liên kết
 - Các thao tác trên danh sách liên kết
- ❑ Ngăn xếp
- ❑ Hàng đợi

1-64

Cấu trúc dữ liệu và thuật toán

- Nội dung bài giảng được biên soạn bởi PGS. TS. Phạm Tuấn Minh.

1-65