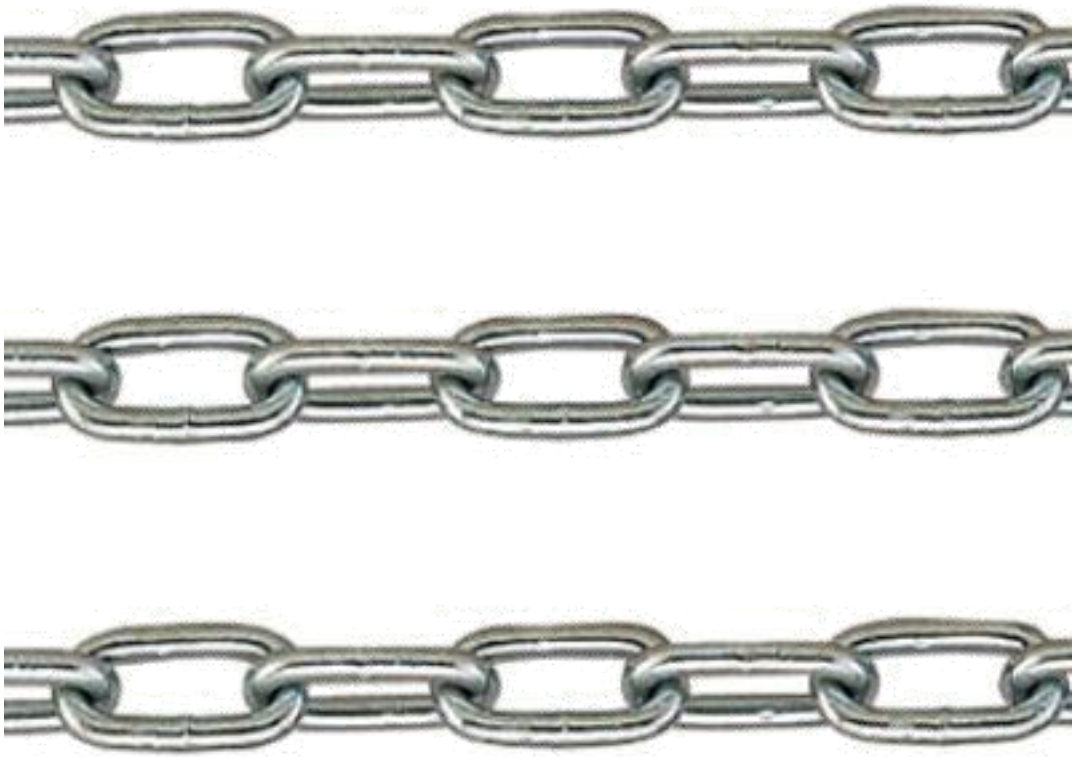
LECTURE 13

SINGLY LINKED LIST

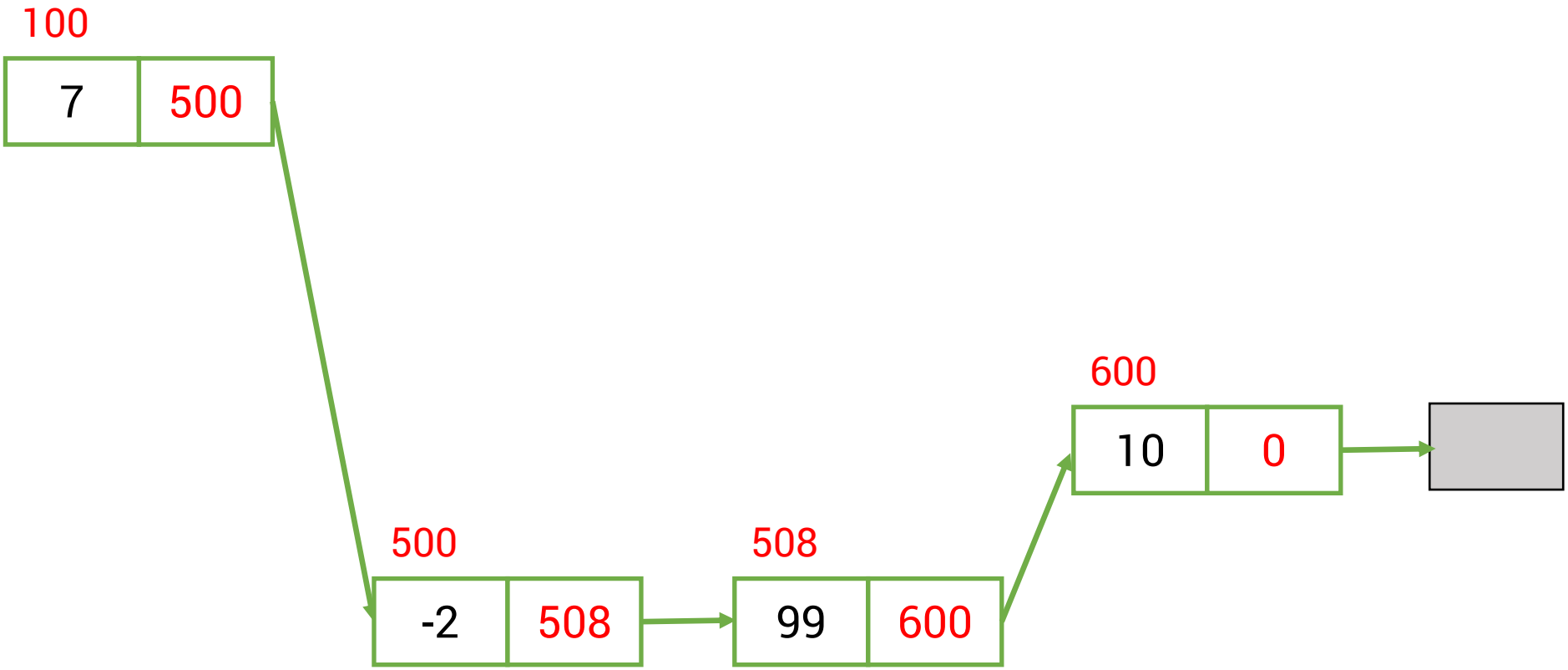
Big-O Coding

Website: www.bigocoding.com

Linked list trong thực tế

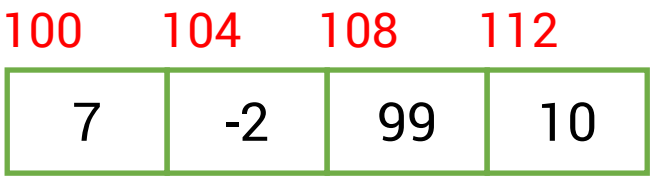


Linked list trong lập trình



Điểm mạnh & điểm yếu của array

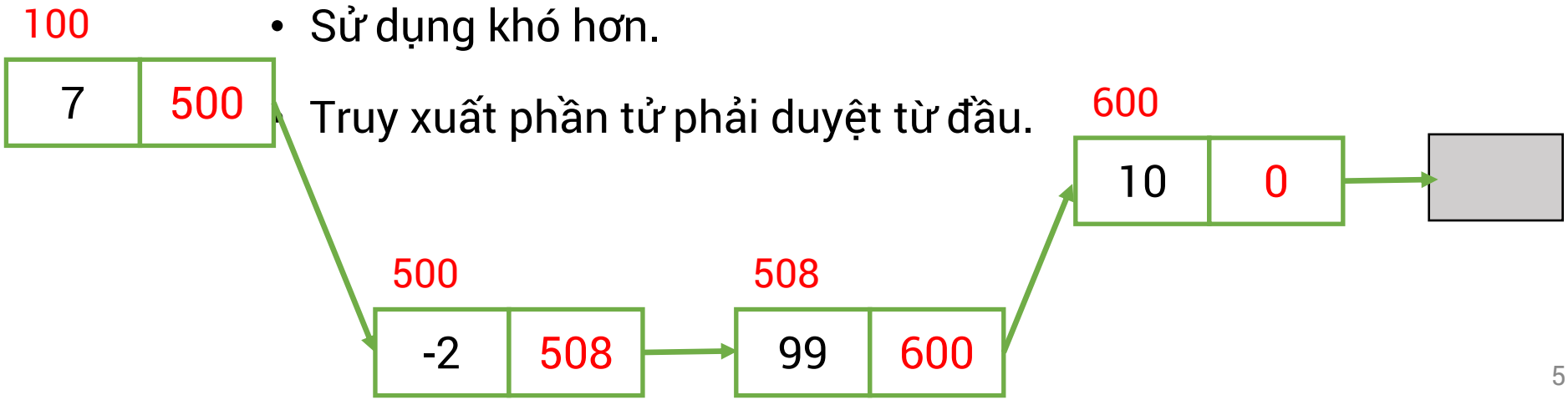
- Pros:
  - Sử dụng dễ dàng.
    - Học xong linked list rồi bạn sẽ thấy. ☺
  - Truy xuất phần tử thông qua chỉ số.
    - Lấy phần tử thứ năm trong mảng: a[5]
- Cons:
  - Số lượng phần tử cố định.
    - `int a[1000]; int *b = new int[n];`
  - Cấp phát vùng nhớ liên tục.



Điểm mạnh và điểm yếu của linked list

- Pros:
  - Không cần xác định trước số phần tử.
    - Cần thêm thì cứ thêm, cần xóa thì cứ xóa.
  - Vùng nhớ các phần tử không cần liên tục.
  - Cons:
    - Sử dụng khó hơn.

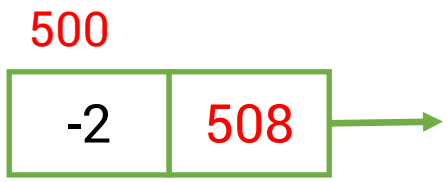
Truy xuất phần tử phải duyệt từ đầu.



struct Node

- Lưu thông tin 1 node bao gồm:
  - data: dữ liệu lưu trong Node đó: số nguyên, số thực, chuỗi, Fraction, Product…
  - next: con trỏ, lưu địa chỉ của node tiếp theo trong danh sách.

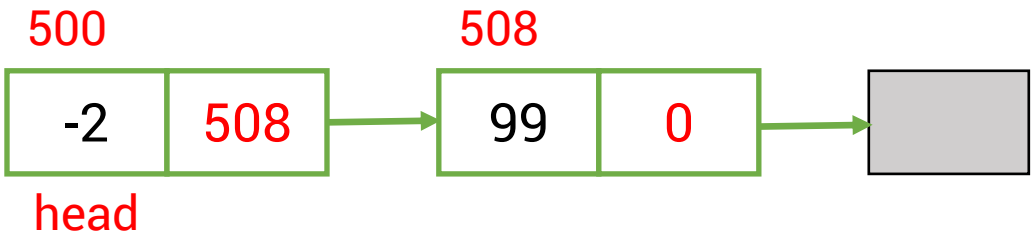
```
struct Node{
  int data;
  Node* next;
};
```



struct List

- Lưu thông tin của một linked list.
  - head: con trỏ, lưu địa chỉ của node đầu tiên.
  - Từ node đầu tiên, ta dùng con trỏ next để đi qua node tiếp theo, từ đó lại next để đi qua node tiếp theo. Cứ thế, ta sẽ duyệt đến phần tử cuối linked list (phần tử có next = NULL).

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



Các thao tác cơ bản trên linked list

- init(): khởi tạo list rỗng (không có phần tử nào).
- traverse(): duyệt qua các phần tử trong list.
- search(): tìm kiếm 1 hoặc nhiều phần tử theo tiêu chí nào đó.
- update(): tìm và cập nhật giá trị (data) của một phần tử.
- insertHead(): chèn phần tử mới vào đầu list.
- insertAfter(): chèn phần tử vào sau 1 phần tử.

Các thao tác cơ bản trên linked list

- removeHead(): xóa phần tử đầu ra khỏi list.
- removeAfter(): xóa 1 phần tử nằm sau phần tử x.
- clear(): xóa rỗng list.

BT1 - SỐ NHỎ NHẤT

- Cho danh sách liên kết đơn các số nguyên tìm số nhỏ nhất trong danh sách.

init()

- Khởi tạo list rỗng (ko có phần tử nào).

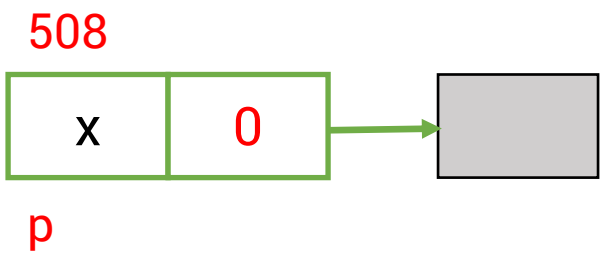
head

```
void init(List &lst){
    lst.head = NULL;
}
```

createNode()

- Tạo số node chứa data = x

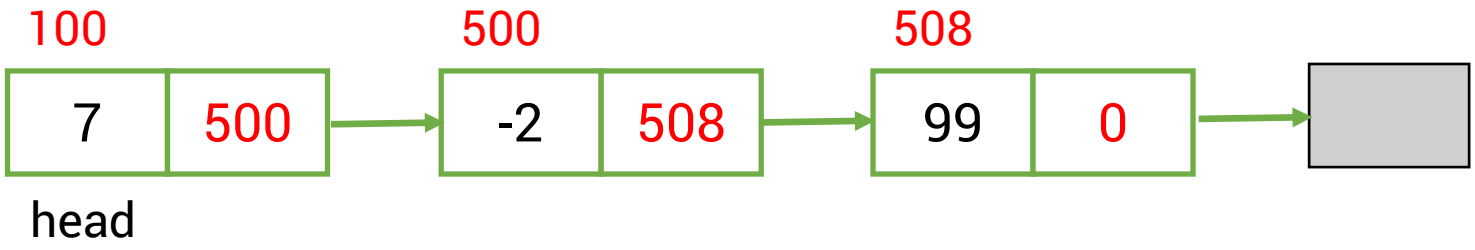
```
Node* createNode(x){
    Node *p = new Node;
    if(p == NULL)
        return NULL;
    p->data = x;
    p->next = NULL;
    return p;
}
```



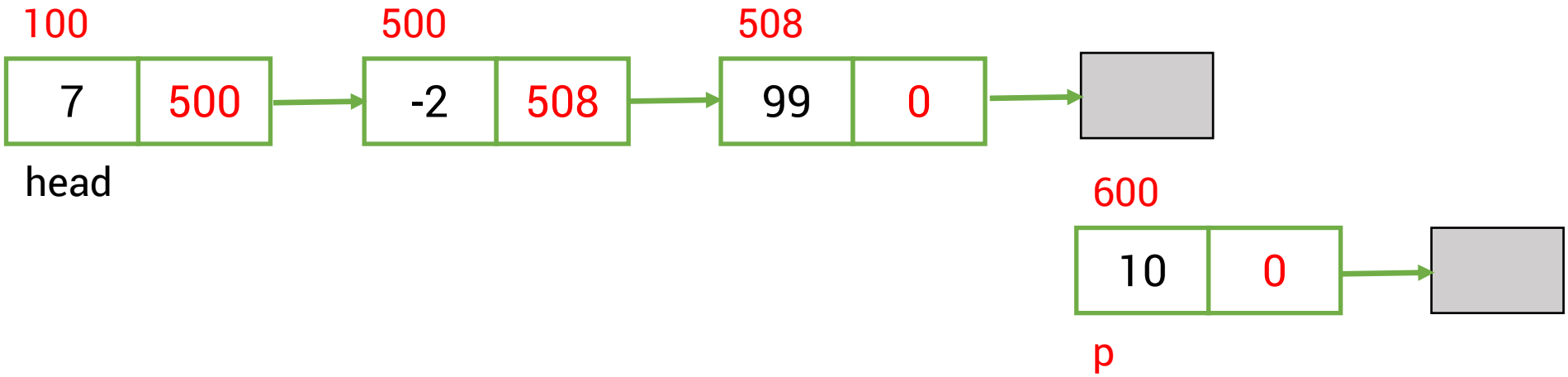
insertTail()

- Thêm phần tử x vào cuối list.
- Thực hiện:
    1. Gọi createNode(), để tạo Node *p cho x.
    2. Nếu p là NULL pointer, ko insert được.
    3. Nếu p ko phải NULL pointer.
        1. Nếu list rỗng, lst.head == NULL, cập nhật lst.head = p
        2. Ngược lại, sử dụng con trỏ Node *cur, bắt đầu từ đầu list, dừng lại ở phần tử cuối cùng trong list (cur->next == NULL).
        3. Gắn p vào phía sau cur.

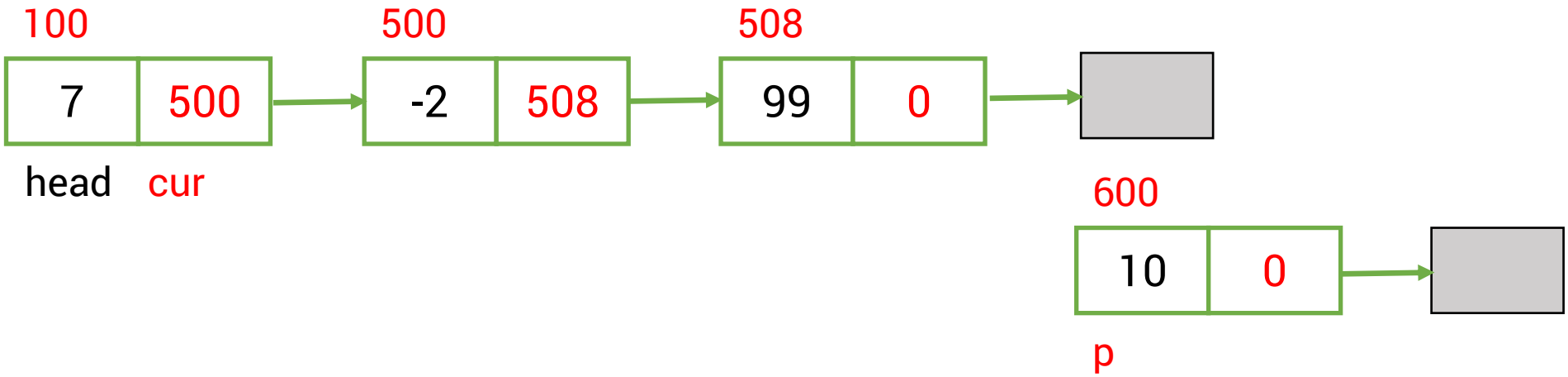
insertTail(10)



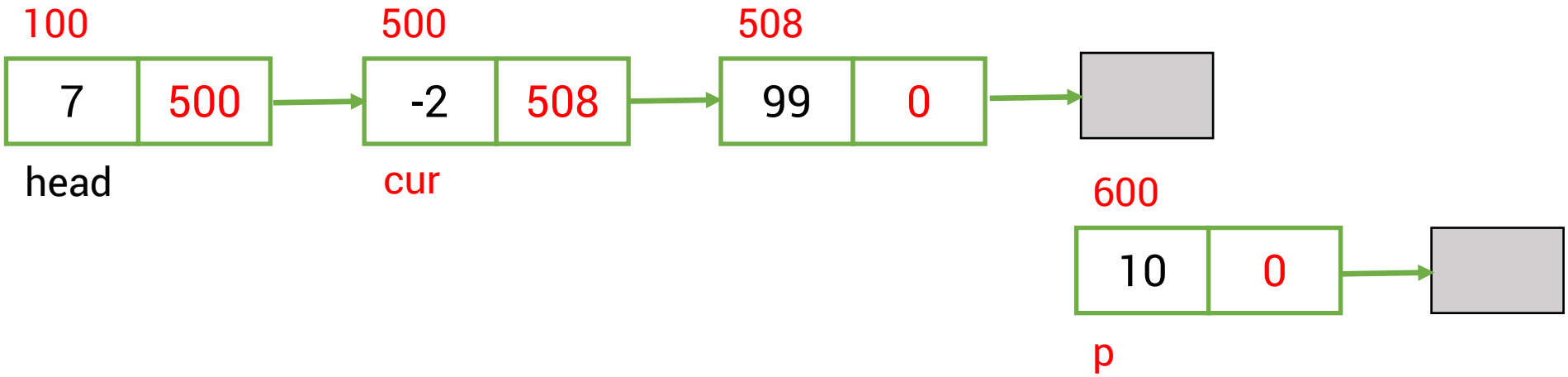
insertTail(10)



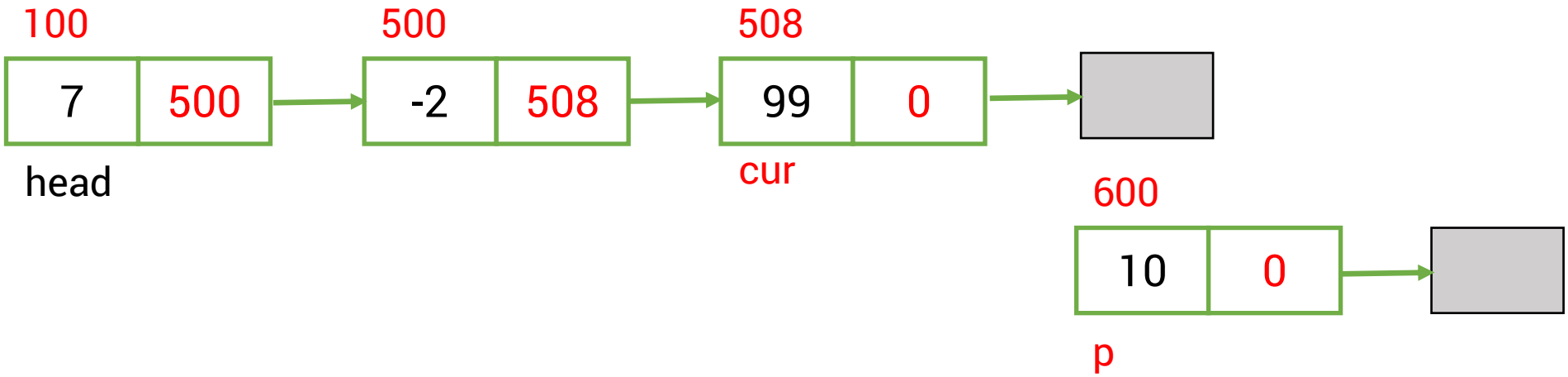
insertTail(10)



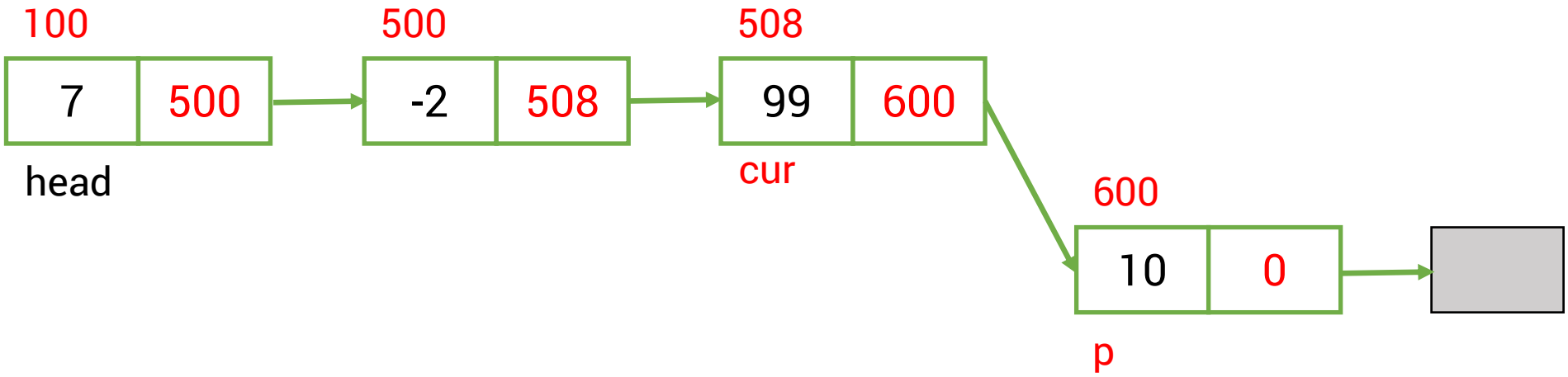
insertTail(10)



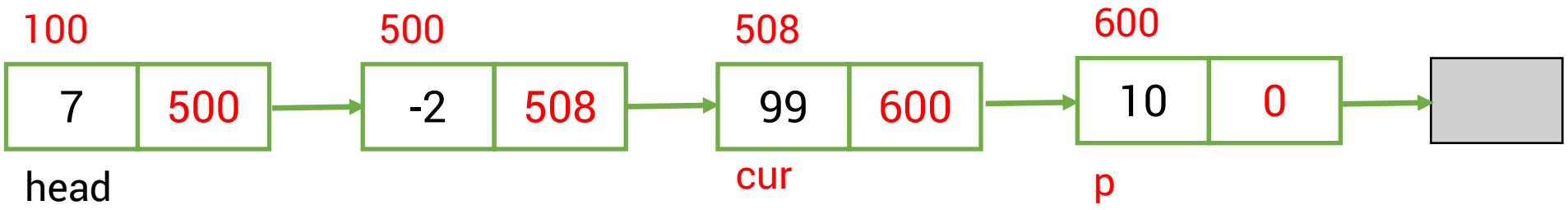
insertTail(10)



insertTail(10)



insertTail(10)



insertTail()

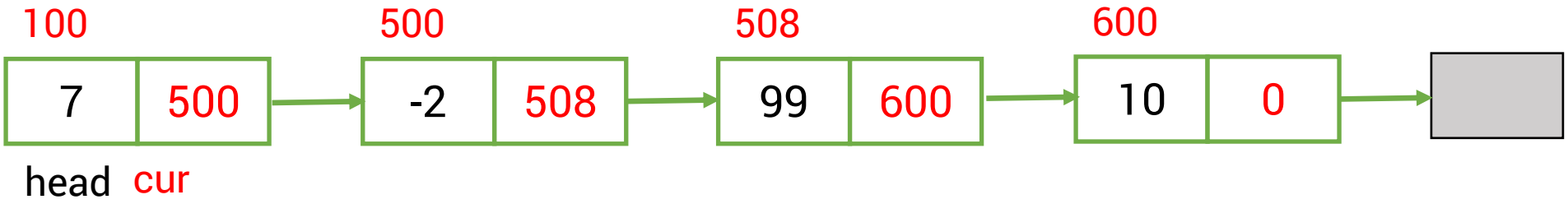
```
void insertTail(List &lst, int x){
    Node *p = createNode(x);
    if(p == NULL)
        return;
    if(lst.head == NULL) // empty list
        lst.head = p;
    else{
        Node *cur = lst.head;
        while(cur->next != NULL)
            cur = cur->next;
        cur->next = p;
    }
}
```

traverse()

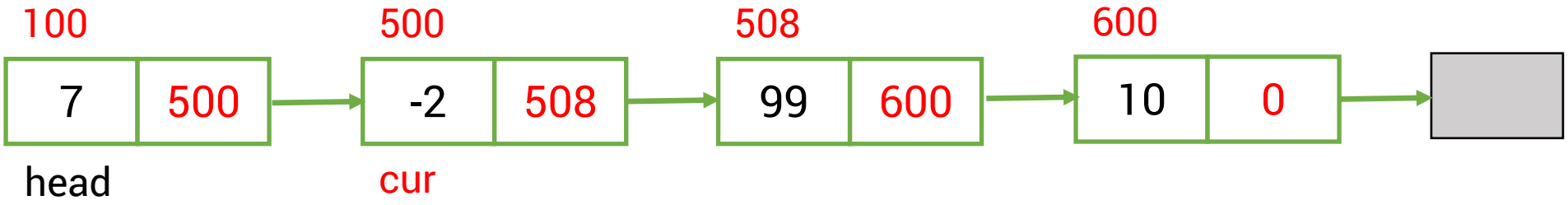
- Duyệt qua các phần tử trong list.

```
void traverse(List lst){
  if(lst.head == NULL)
    return;
  Node *cur = lst.head;
  while(cur != NULL){
    cout << cur->data << " ";
    cur = cur->next;
  }
}
```

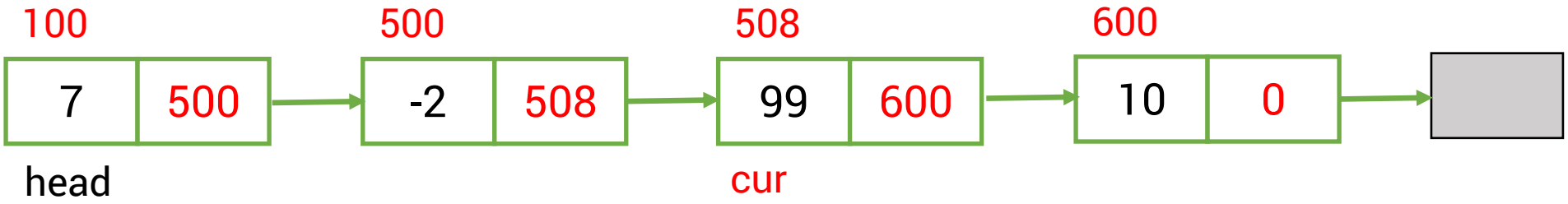
traverse()



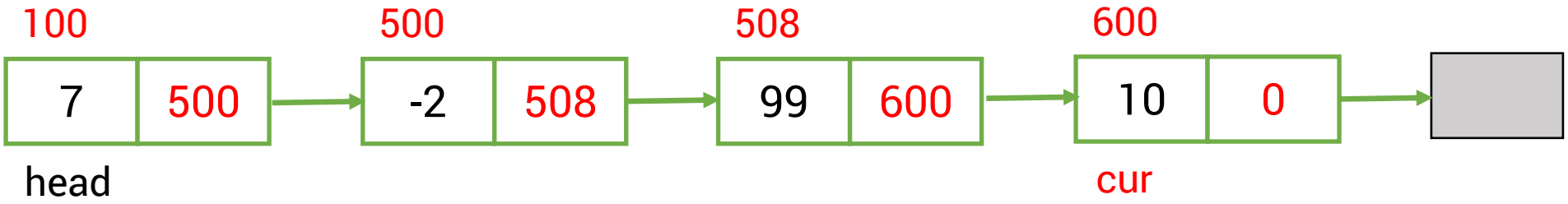
traverse()



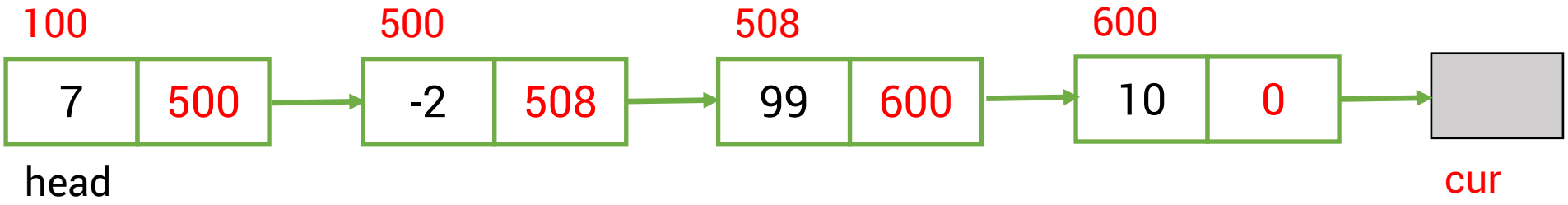
traverse()



traverse()



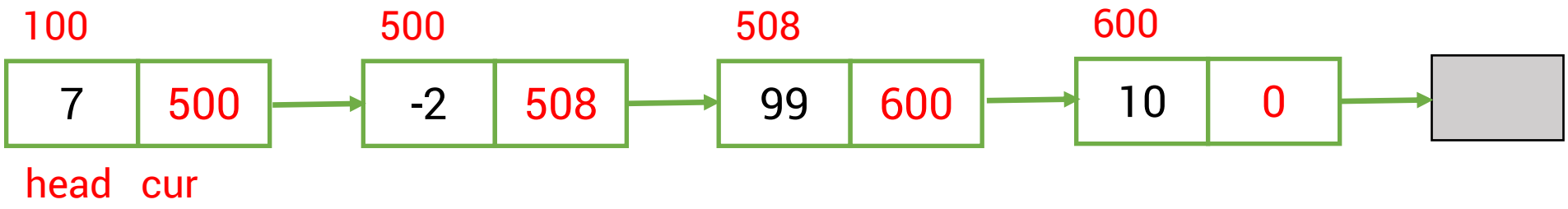
traverse()



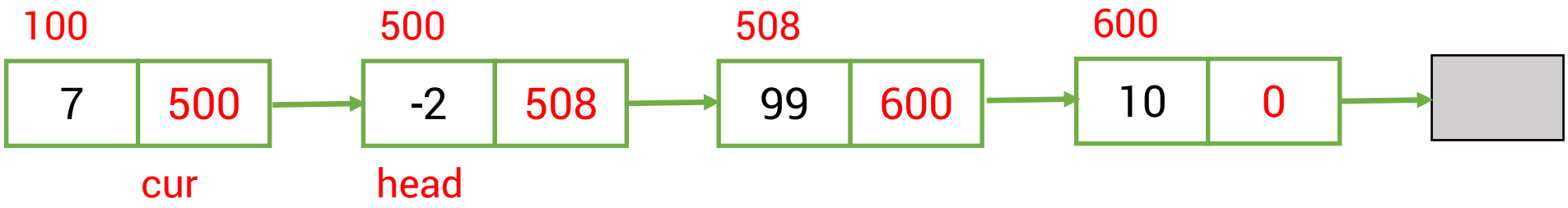
removeHead()

- Xóa phần tử đầu tiên ra khỏi list.
- Thực hiện:
    1. Nếu list rỗng, ko làm gì cả.
    2. Ngược lại,
        1. Cho Node *cur giữ phần tử đầu tiên.
        2. lst.head nhảy đến phần tử tiếp theo.
        3. delete cur.

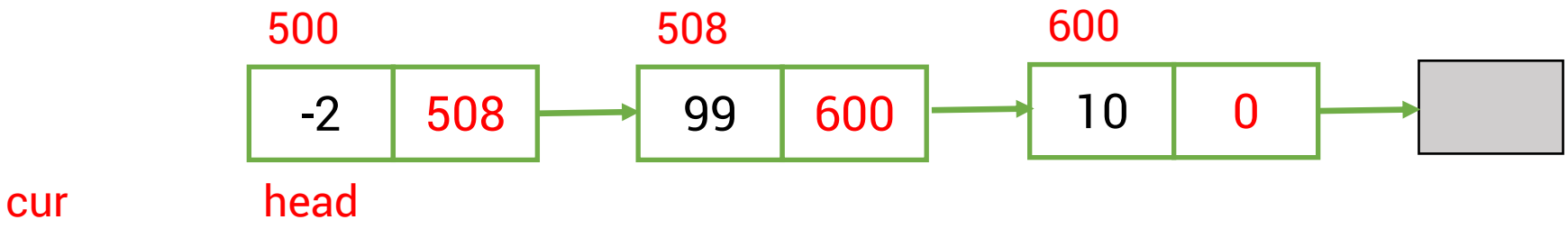
removeHead()



removeHead()



removeHead()



removeHead()

```
void removeHead(List &lst){
    if(lst.head == NULL)
        return;
    Node *cur = lst.head;
    lst.head = lst.head->next;
    delete cur;
}
```

clear()

- Xóa tất cả phần tử trong list.
- Thực hiện:
  - Liên tục removeHead() cho đến khi list rỗng.

```
void clear(List &lst){
    while(lst.head != NULL)
        removeHead(lst);
}
```

BT1 - Gợi ý

```
// ĐỌC TỪNG SỐ, INSERTTAIL
void inputList(List &lst){
  // Gọi hàm init, khởi tạo empty list
  // Đọc số đầu tiên vào x
  // Khai báo 2 con trỏ cur và p, khởi tạo bằng NULL pointer
  while(x != 0){
    // Gọi hàm createNode, tạo node p cho số nguyên x
    // Nếu p khác NULL
      // Nếu list rỗng
        // head chính là p
        // Đánh dấu cur là head cho các lần insert sau

      // Ngược lại, cur đang trỏ đến node cuối cùng của list,
        // next của cur là p
        // cur bằng p (cur giữ node cuối, để cho các lần insert sau)


    // Đọc số tiếp theo vào x, quay lại vòng lặp
  }
}
```

BT1 - Gợi ý

```
// ĐÁNH DẤU MIN LÀ PHẦN TỬ ĐẦU TIÊN,
// DUYỆT QUA CÁC PHẦN TỬ TRONG LIST, TÌM MIN
int min(List &lst){
  // Nếu list rỗng, min là 0, return;

  // Ngược lại, min là data của head
  // Dùng con trỏ cur, bắt đầu từ head
  while(cur != NULL){ // cur == NULL nghĩa là ta đã đi qua hết tất cả
các node trong list
    // Nếu data của cur lại nhỏ hơn min, thì cập nhật min
    cur = cur->next; // Đến node tiếp theo
  }
}

int main(){
  List lst;
  // Gọi hàm inputList
  // Gọi hàm min và in kết quả
  // Gọi hàm clear cho list
  return 0;
}
```

BT1 - Lời giải

```
void inputList(List &lst){
    init(lst);
    int x;
    cin>>x;
    Node *cur = NULL;
    Node *p = NULL;
    while(x != 0){
        p = createNode(x);
        if(p != NULL){
            if(lst.head == NULL){
                lst.head = p;
                cur = lst.head;
            }
            else{
                cur->next = p;
                cur = p;
            }
        }
        cin >> x;
    }
}
```

BT1 - Lời giải

```
int min(List &lst){
    if(lst.head == NULL)
        return 0;
    int ans = lst.head->data;
    Node* cur = lst.head;
    while(cur != NULL){
        if(cur->data < ans)
            ans = cur->data;
        cur = cur->next;
    }
    return ans;
}

int main(){
    List lst;
    inputList(lst);
    int ans = min(lst);
    cout << ans;
    return 0;
}
```

BT2 - ĐẾM SỐ NHỎ NHẤT

- Cho danh sách liên kết đơn các số thực, viết hàm đếm các số nhỏ nhất.

BT2 - Gợi ý

```
// DUYỆT QUA TỪNG NODE TRONG LIST, ĐẾM XEM X XUẤT HIỆN MẤY
LẦN
int count(List lst, int x){
  // Khởi tạo ans = 0
  // Node *cur trỏ đến head
  while(cur != NULL){
    // Nếu data của cur bằng x, tăng biến ans
    cur = cur->next;
  }
}

int main(){
  // Gọi hàm inputList
  // Gọi hàm min
  // Nếu x khác 0, gọi hàm count, in kết quả
  // Gọi hàm clear cho list
  return 0;
}
```

BT2 - Lời giải

```
int count(List lst, int x){
    int ans = 0;
    Node *cur = lst.head;
    while(cur != NULL){
        if(cur->data == x)
            ans++;
        cur = cur->next;
    }
    return ans;
}

int main(){
    List lst;
    inputList(lst);
    int x = min(lst);
    int ans = 0;
    if(x != 0)
        ans = count(lst, x);
    cout << ans;
    return 0;
}
```

BT3 - KIỆN HÀNG

- Cho danh sách liên kết đơn các kiện hàng {mã kiện hàng, cân nặng}. Hãy tìm kiện hàng nào có số cân nặng là nặng nhất.

BT3 - Gợi ý

```
// TƯƠNG TỰ HÀM INPUT Ở P01, P02, P03 CŨNG INSERT TAIL
void inputList(List &lst){
  // Khởi tạo empty list
  Package pkg;
  // Đọc id và weight vào pkg
  Node *cur = NULL;
  Node *p = NULL;

  while(id != "0" && weight != 0){
    // Tạo node p từ pkg

    if(p != NULL){
      // Nếu list empty, xử lí như P01
      // Ngược lại, xử lí như P01
    }
    // Tiếp tục đọc id và weight vào pkg
  }
}
```

BT3 - Gợi ý

Big-O Green

```
// TƯƠNG TỰ HÀM MIN Ở P01,
// TA DÙNG NODE CUR DUYỆT QUA LIST, TÌM MAX
Node* maxWeight(List lst){
  // Nếu list rỗng, trả về NULL pointer

  // Node *ans trỏ đến head
  // Node* cur bắt đầu từ head
  while(cur != NULL){
    if(cur->data.weight > ans->data.weight
       || (cur->data.weight == ans->data.weight && cur->data.id > ans-
>data.id))
      // Cập nhật ans
  }
}

int main(){
  // Gọi hàm nhập list
  // Gọi hàm maxWeight
  // Nếu ans là NULL pointer, in "0"
  // Ngược lại in ans->data.id
  // clear list
  return 0;
}
```

BT3 - Lời giải

```
struct Package{
    string id;
    int weight;
};

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

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

BT3 - Lời giải

```
void inputList(List &lst){
    init(lst);
    string id;
    int weight;
    cin >> id;
    cin >> weight;
    Node *cur = NULL;
    Node *p = NULL;
    Package pkg;
    while(id != "0" && weight != 0){
        pkg.id = id;
        pkg.weight = weight;
        p = createNode(pkg);

    // to be continue
```

BT3 - Lời giải

```
        if(p != NULL){
            if(lst.head == NULL){
                lst.head = p;
                cur = lst.head;
            }
            else{
                cur->next = p;
                cur = p;
            }
        }
        cin >> id;
        cin >> weight;
    }
}
```

BT3 - Lời giải

```
Node* maxWeight(List lst){
    if(lst.head == NULL)
        return 0;
    Node* ans = lst.head;
    Node* cur = lst.head;
    while(cur != NULL){
        if(cur->data.weight < ans->data.weight
       || (cur->data.weight == ans->data.weight && cur-
>data.id > ans->data.id))
            ans = cur;
        cur = cur->next;
    }
    return ans;
}
```

BT3 - Lời giải

```
int main(){
    List lst;
    inputList(lst);
    Node* ans = maxWeight(lst);
    if(ans != NULL)
        cout << ans->data.id;
    else
        cout << "0";
    clear(lst);
    return 0;
}
```

BT4 - CHÈN SỐ CHÍNH PHƯƠNG

- Cho danh sách liên kết đơn có sẵn n phần tử và m số nguyên. Hãy kiểm tra xem số nào trong m số đó là số chính phương, chèn lần lượt các số chính phương vào phía sau các số nhỏ hơn hay bằng nó lần lượt trong danh sách liên kết đơn khi duyệt từ đầu đến cuối.

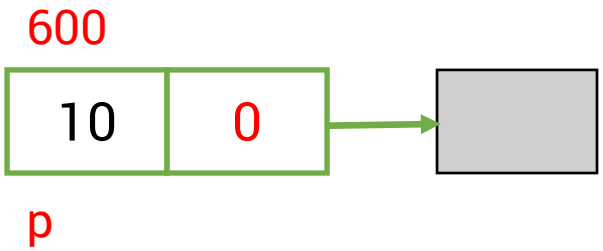
insertAfter()

- Thêm phần tử có giá trị y vào sau phần tử có giá trị x trong list.
- Thực hiện:
  1. Gọi createNode() tạo con trỏ Node *p cho x.
  2. Sử dụng con trỏ Node *cur, bắt đầu từ list.head, tìm đến node có giá trị y.
  3. Cho p->next trỏ đến cur->next.
  4. Cho cur->next trỏ đến p.

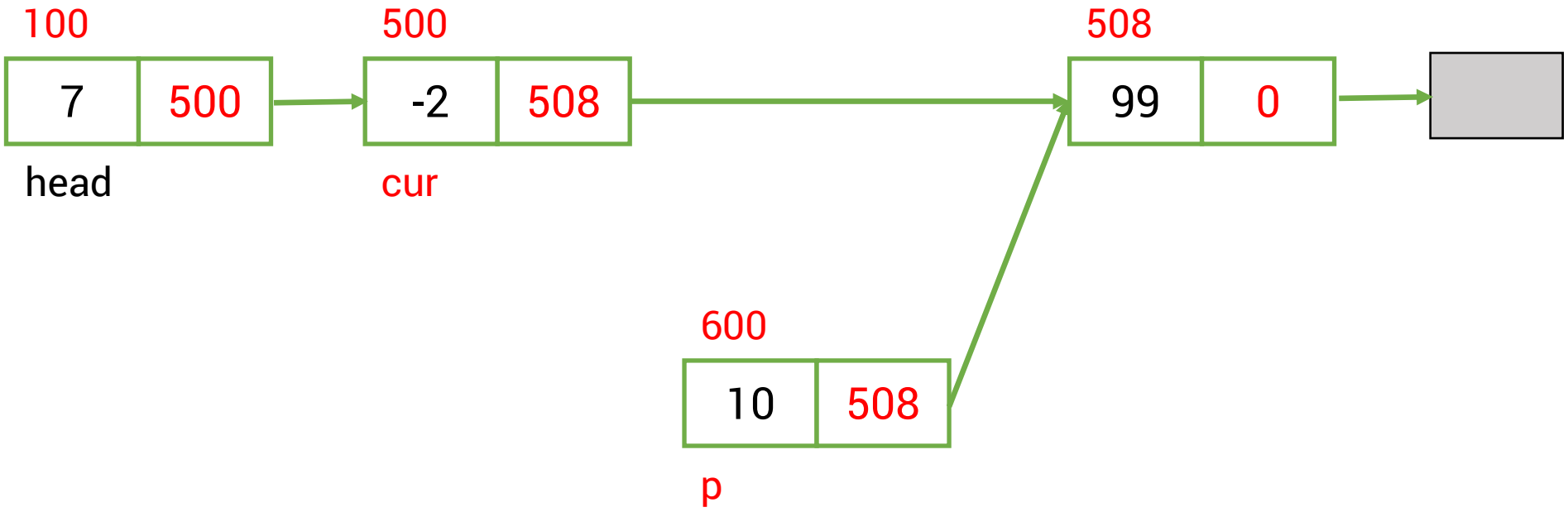
insertAfter() - x = -2, y = 10

insertAfter() - x = -2, y = 10

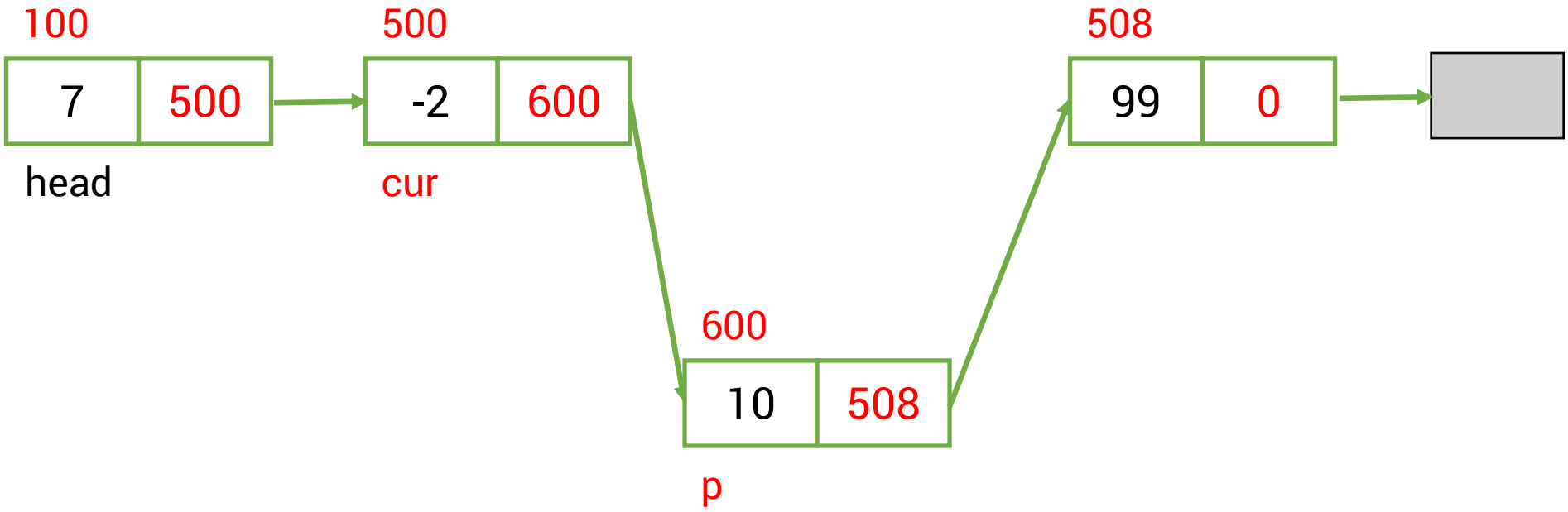
insertAfter() - x = -2, y = 10



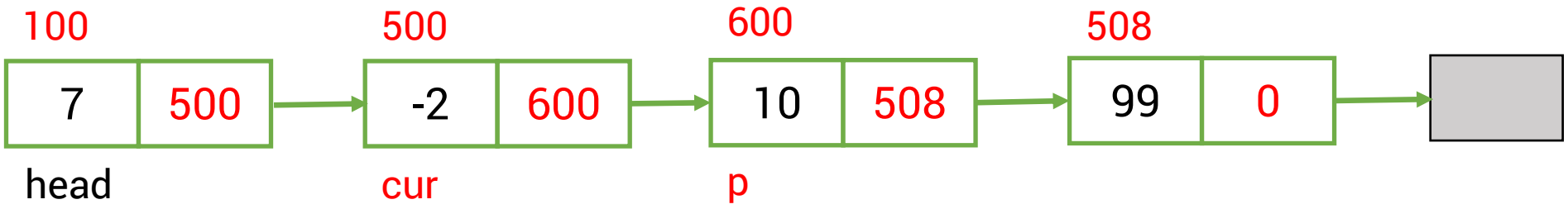
insertAfter() - x = -2, y = 10



insertAfter() - x = -2, y = 10



insertAfter() - x = -2, y = 10



insertAfter()

```
void insertAfter(List &lst, int x, int y){
    if(lst.head == NULL)
        return;
    Node *p = createNode(y);
    if(p == NULL)
        return;
    Node *cur = lst.head;
    while(cur != NULL){
        if(cur->data == x)
            break;
        cur = cur->next;
    }
    if(cur == NULL) // x not found
        return;
    p->next = cur->next;
    cur->next = p;
}
```

BT4 - Gợi ý

Big-O Green

```
// KIỂM TRA X CÓ PHẢI SỐ CHÍNH PHƯƠNG KO
bool isSquare(int x) {
  return (int)sqrt(x)*(int)sqrt(x) == x;
}

// KHỞI TẠO LIST RỖNG
// ĐỌC N SỐ TRONG LIST
// ĐỌC TỪ SỐ TRONG M SỐ, TÌM VỊ TRÍ MÀ CUR->DATA <= X VÀ INSERT AFTER
// DUYỆT LIST, IN CÁC SỐ
// CLEAR LIST
int main(){
  // Khởi tạo empty list
  // cur và p khởi tạo là NULL pointer

  // Đọc n, m

  for (int i=0; i<n; i++) {
    // Đọc x
    // Tạo node p từ x
    if(p != NULL){
      // Đoạn này tương tự P01
    }
  }
```

BT4 - Gợi ý

```
// cur bắt đầu từ head
cur = lst.head;
for (int i=0; i<m; i++) {
  // Đọc x
  // Nếu x là số chính phương
    // Tạo node p từ x
    if(p != NULL){
      // Lặp while, cur->next != NULL
        // Nếu cur->data <= x, break vì cur là Node cần tìm
        cur = cur->next;
      // Thực hiện nối p vào sau cur:
      // next of p là next of cur
      // next of cur là p
      if(p->next != NULL)
        cur = p->next;
      else
        cur = p;
    }
  }
}
```

BT4 - Gợi ý

```
  // cur bắt đầu từ head
  while (cur!=NULL) {
    // In data của cur
  }

  // clear list

}
```

BT4 - Lời giải

```
bool isSquare(int x) {
    return (int)sqrt(x)*(int)sqrt(x) == x;
}

int main(){
    List lst;
    init(lst);
    Node *cur = NULL;
    Node *p = NULL;

    int n, m;
    cin >> n >> m;

 // to be continue
```

BT4 - Lời giải

```
    for (int i=0; i<n; i++) {
        int x;
        cin >> x;
        p = createNode(x);
        if(p != NULL){
            if (lst.head==NULL) {
                lst.head = p;
                cur = lst.head;
            }
            else {
                cur->next = p;
                cur = p;
            }
        }
  }
```

BT4 - Lời giải

```
    cur = lst.head;
    for (int i=0; i<m; i++) {
        int x;
        cin >> x;
        if (isSquare(x)) {
            p = createNode(x);
            if(p != NULL){
                while (cur->next != NULL){
                    if(cur->data <= x)
                        break;
                    cur = cur->next;
                }
                p->next = cur->next;
                cur->next = p;
                if(p->next != NULL)
                    cur = p->next;
                else
                    cur = p;
            }
        }
    }
```

BT4 - Lời giải

```
    cur = lst.head;
    while (cur!=NULL) {
        cout << cur->data << " ";
        cur = cur->next;
    }

    clear(lst);

}
```

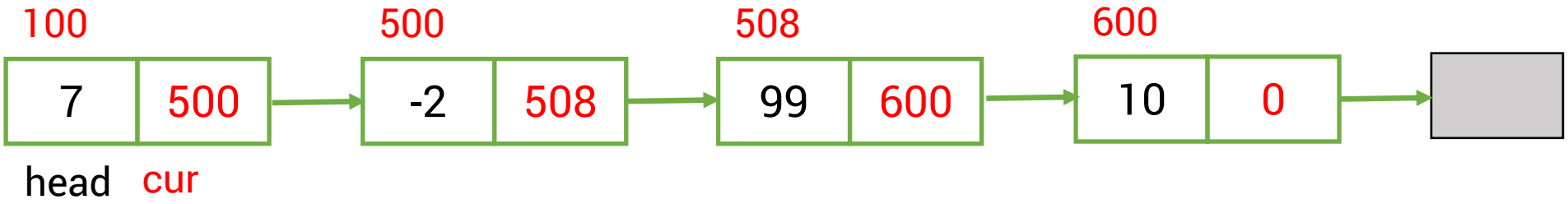
BT5 - XÓA TẬN CÙNG 5

- Cho danh sách liên kết đơn, hãy tìm và xóa những phần tử nào có tận cùng là 5.

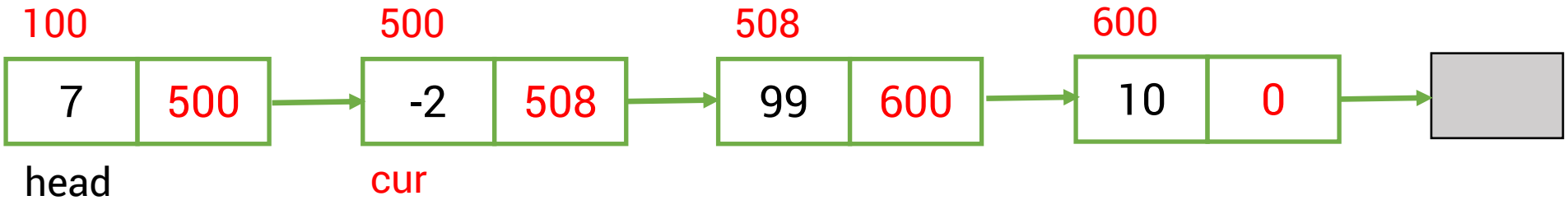
removeAfter()

- Xóa phần tử đứng sau phần tử x.
- Thực hiện:
  - Sử dụng con trỏ Node *cur, bắt đầu từ list.head, tìm đến node thỏa cur->data == x.
  - Cho con trỏ Node *p trỏ đến cur->next (là node ta cần xóa).
  - Cho cur->next = p->next.
  - delete p.

removeAfter(99)



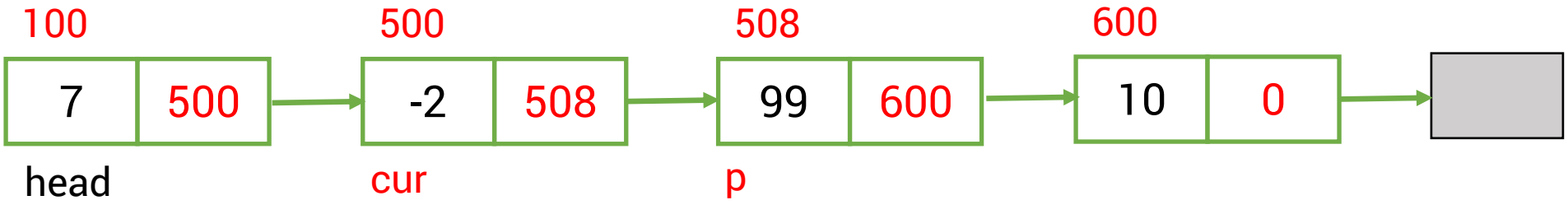
removeAfter(99)



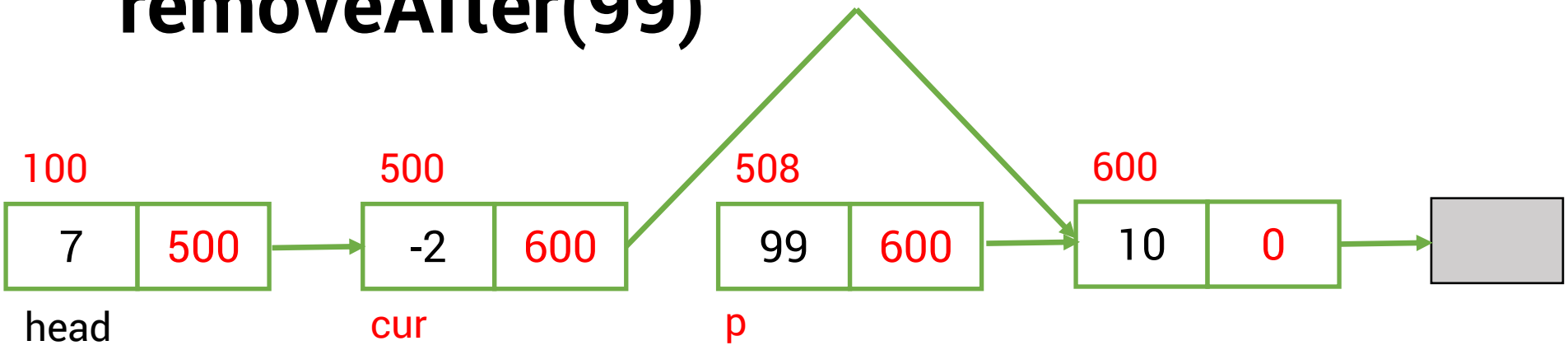
removeAfter()

```
void deleteAfter(List &lst, int x){
    if(lst.head == NULL)
        return;
    Node *cur = lst.head;
    while(cur != NULL){
        if(cur->data == x){
        Node *p = cur->next;
        cur->next = p->next;
        delete p;
        break;
        }
    }
}
```

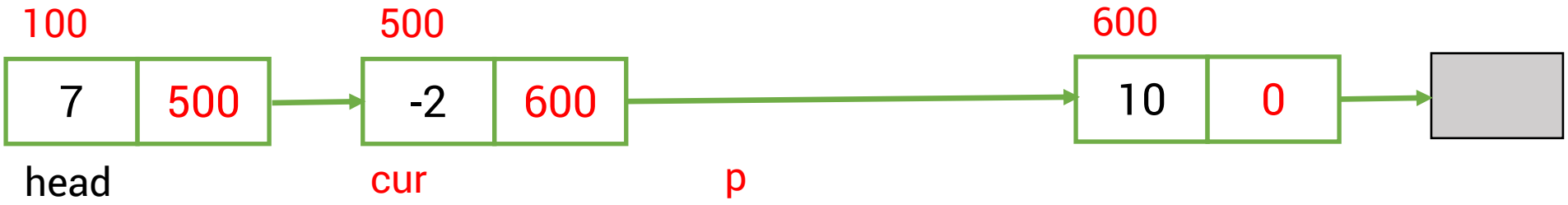
removeAfter(99)



removeAfter(99)



deleteAfter(99)



BT5 - Gợi ý

```
// KIỂM TRA X CÓ TẬN CÙNG LÀ 5 HAY KO
bool last5(int x) {
  if (x<0) x = -x;
  return x%10 == 5;
}

// KHỞI TẠO EMPTY LIST
// ĐỌC LIST, INSERT TAIL
// XỬ LÍ TRƯỜNG HỢP HEAD LÀ SỐ TẬN CÙNG 5 -> REMOVE HEAD
// TÌM CÁC NODE THỎA CUR->NEXT->DATA TẬN CÙNG 5 -> XÓA CUR->NEXT
// IN LIST
// CLEAR LIST
int main(){
  // Khởi tạo empty list
  // Đọc list, insert tail
```

BT5 - Gợi ý

```
// Nếu list rỗng, ko cần xóa gì cả, return

// Nếu head là số tận cùng 5, remove head liên tục
while(last5(lst.head->data)){
  // Xem đoạn code removeHead
  if(lst.head == NULL)
    break;
}
```

BT5 - Gợi ý

```
  // Nếu list rỗng, ko cần xóa gì cả, return

  while (cur->next != NULL) {
    // Nếu data của next của cur tận cùng 5
      // Dùng p giữ next
      // next của cur giờ là next của p
      // Xóa p
    }
    else
      cur = cur->next;
  }

  // In list

  // clear list

}
```

BT5 - Lời giải

```
bool last5(int x) {
    if (x<0)
        x = -x;
    return x%10 == 5;
}

int main(){
    List lst;
    init(lst);
    Node *cur = NULL;
    Node *p = NULL;

    int n;
    cin >> n;
```

BT5 - Lời giải

```
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        p = createNode(x);
        if (lst.head == NULL) {
            lst.head = p;
            cur = lst.head;
        }
        else {
            cur->next = p;
            cur = p;
        }
    }
    if(lst.head == NULL)
         return 0;
```

BT5 - Lời giải

```
while(last5(lst.head->data)){
    p = lst.head;
    lst.head = lst.head->next;
    delete p;
    if(lst.head == NULL)
        break;
}

if(lst.head == NULL)
    return 0;

cur = lst.head;
while (cur->next != NULL) {
    if (last5(cur->next->data)) {
        p = cur->next;
        cur->next = p->next;
        delete p;
    }
    else
        cur = cur->next;
}
```

BT5 - Lời giải

```
    cur = lst.head;
    while (cur!=NULL) {
        cout << cur->data << " ";
        cur = cur->next;
    }

    clear(lst);

}
```

Hỏi đáp