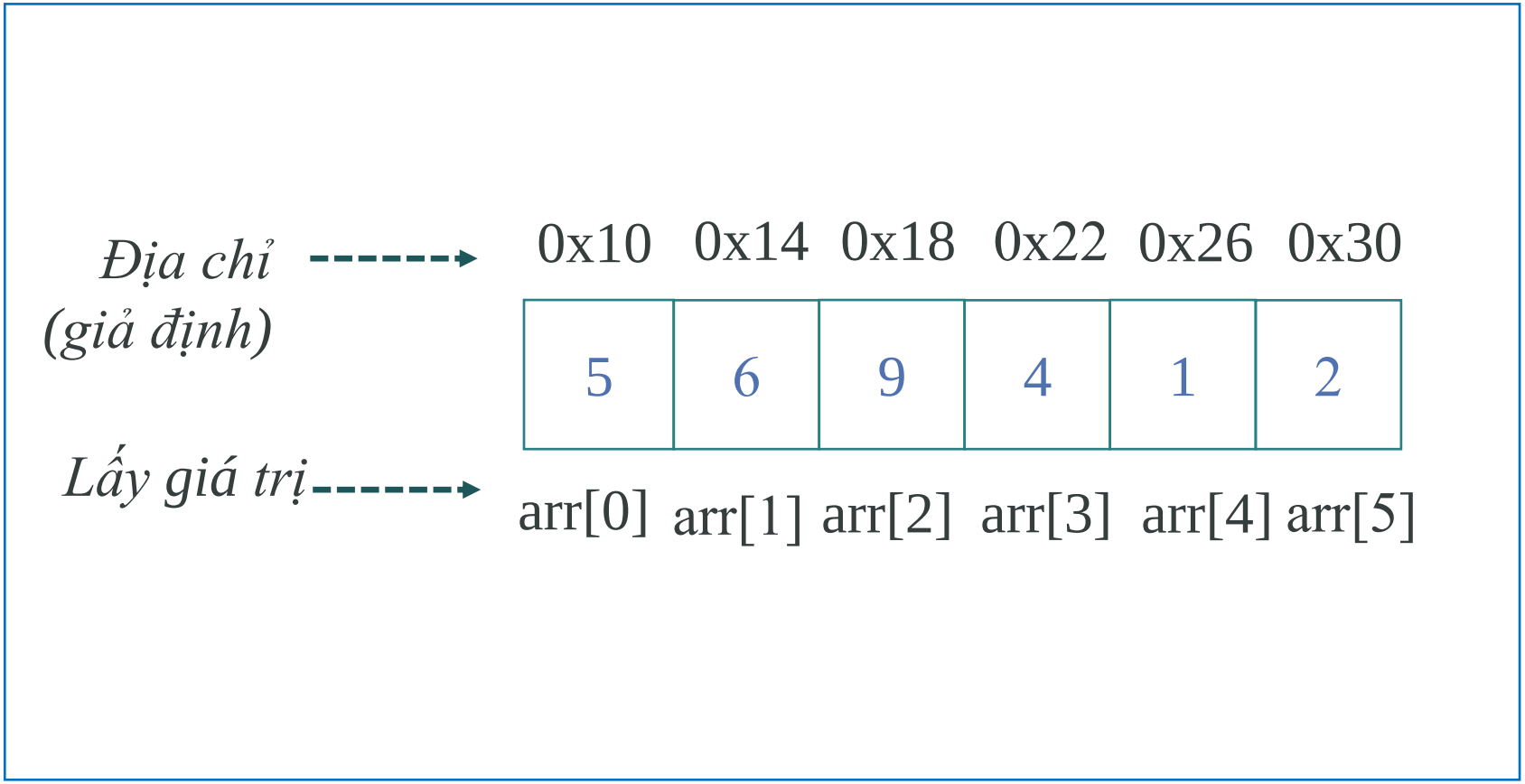
DANH SÁCH LIÊN KẾT ĐƠN (SINGLY LINKED LIST)

DATA STRUCTURES AND ALGORITHMS

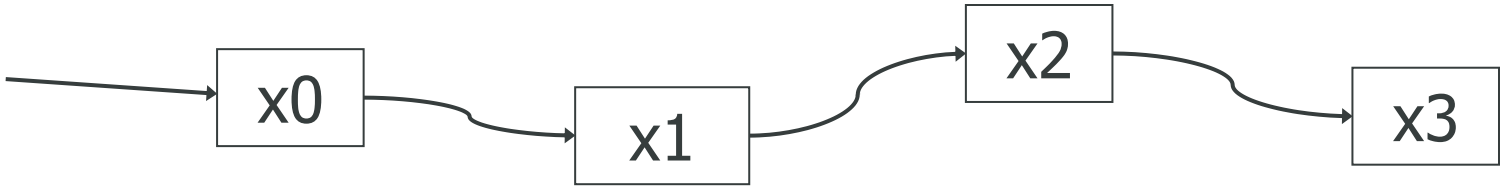
Nhắc lại: Lưu trữ Mảng 1 chiều

- Cho mảng 1 chiều: `int arr[6] = {5, 6, 9, 4, 1, 2};`



Memory Layout

Tổ chức của DSLK Đơn

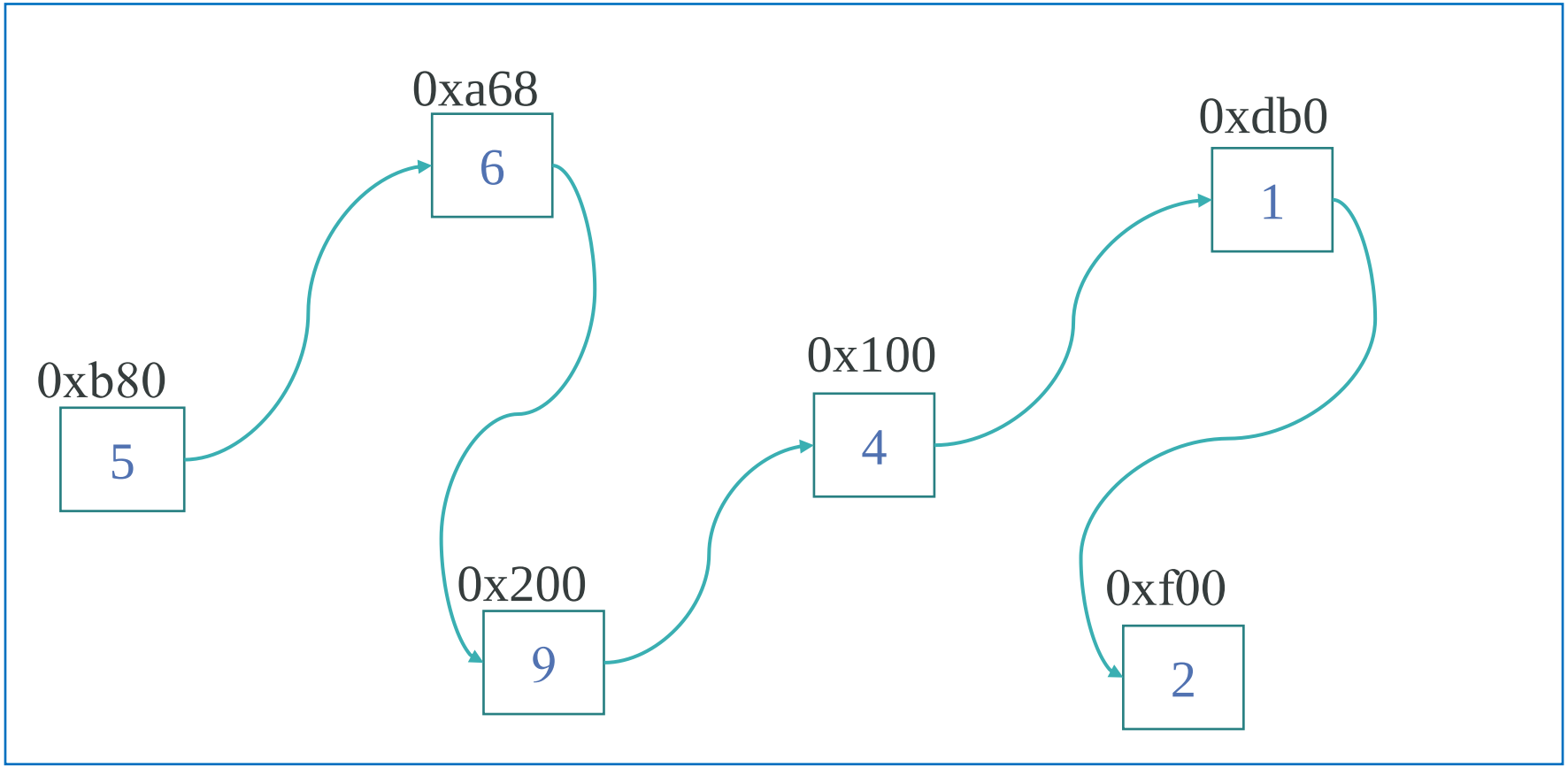


- Mỗi phần tử liên kết với phần tử đứng liền sau trong danh sách.

- Mỗi phần tử trong danh sách liên kết đơn là một cấu trúc có hai thành phần
  - **Thành phần dữ liệu**: Lưu trữ thông tin về bản thân phần tử
  - **Thành phần liên kết**: Lưu địa chỉ phần tử đứng sau trong danh sách hoặc bằng NULL nếu là phần tử cuối danh sách.

Tổ chức của DSLK Đơn

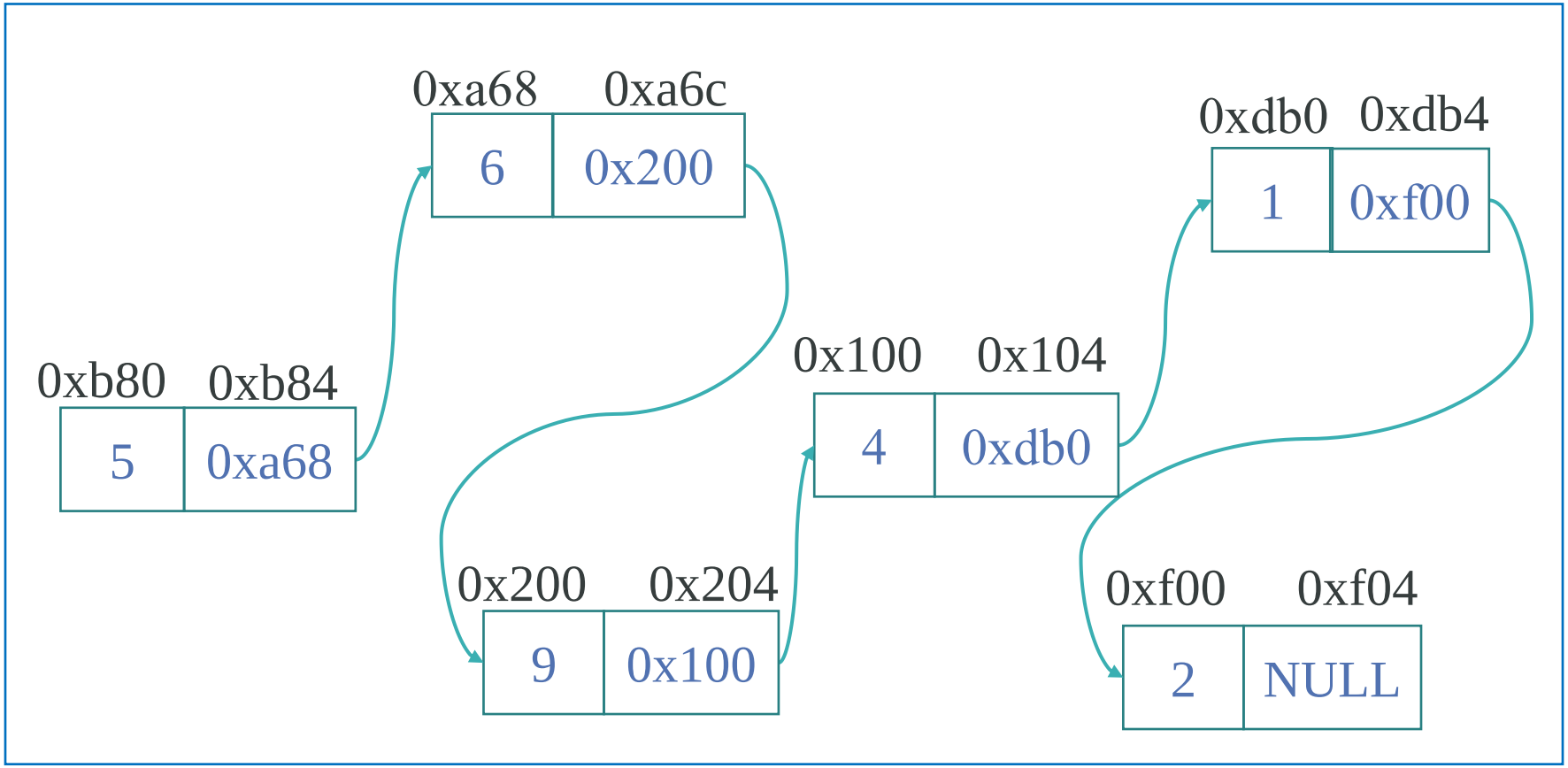
- Danh sách liên kết đơn lưu các giá trị {5, 6, 9, 4, 1, 2}



Memory Layout

Tổ chức của DSLK đơn

- Danh sách liên kết đơn lưu các giá trị {5, 6, 9, 4, 1, 2}



Memory Layout

CTDL của DSLK đơn

- Cấu trúc dữ liệu của 1 node trong List đơn

```
struct NODE {
    Data info;
    NODE* pNext;
};
```

- Cấu trúc dữ liệu của DSLK đơn

```
struct LIST {
    NODE* pHead;
    NODE* pTail;
};
```

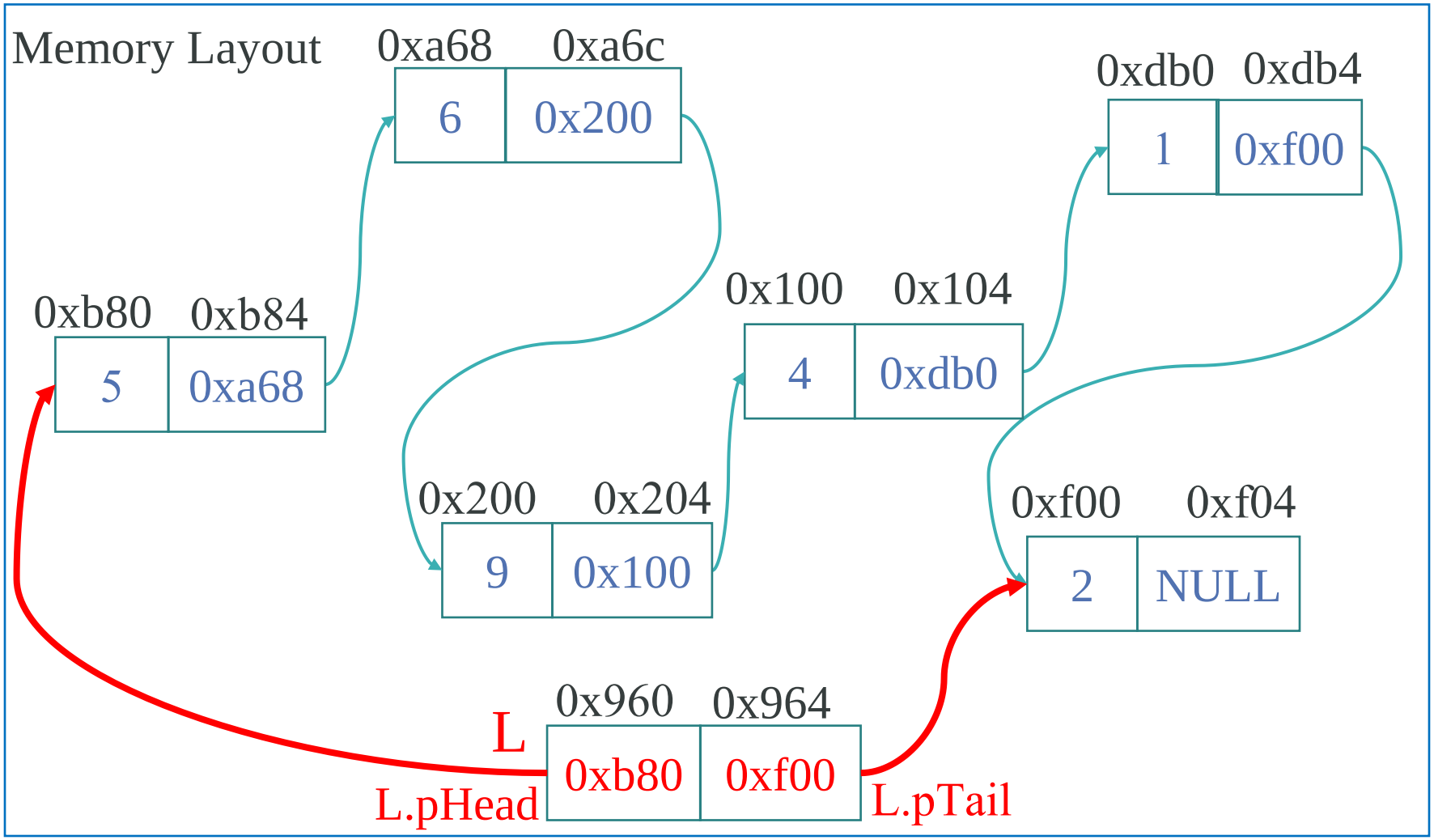
Ví dụ:

- Tạo danh sách liên kết đơn lưu các giá trị số tự nhiên.

```
// Cấu trúc của một node
struct NODE {
   int info;
   NODE* pNext;
};

// Cấu trúc của một DSLK
struct LIST {
   NODE* pHead;
   NODE* pTail;
};

LIST L;
```

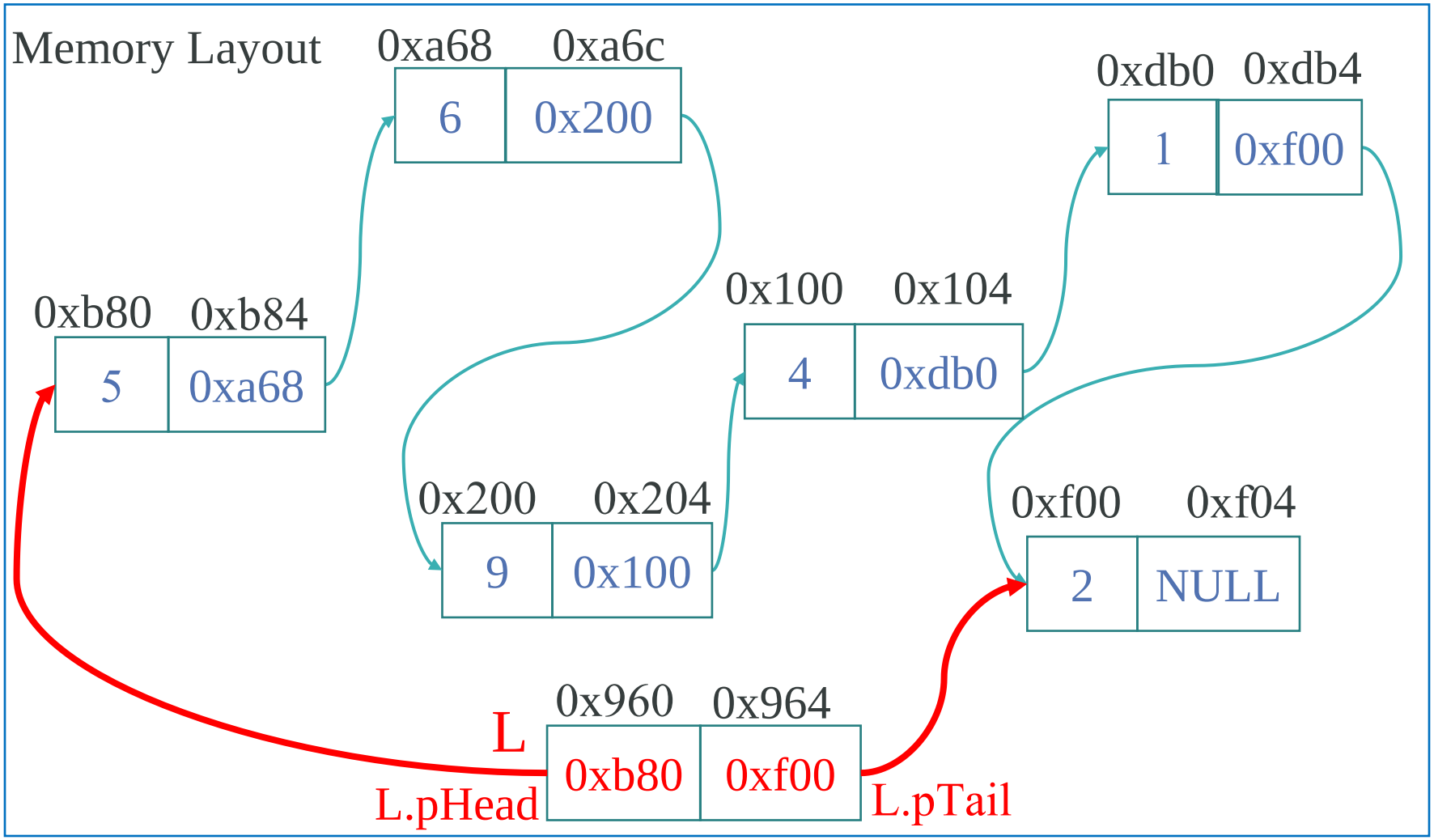
Ví dụ:

- Danh sách liên kết đơn lưu các giá trị {5, 6, 9, 4, 1, 2}



Ví dụ:

- Quá trình tạo danh sách liên kết đơn lưu các giá trị {5, 6, 9, 4, 1, 2} như sau:



Các thao tác cơ bản trên DSLK đơn

- Tạo 1 danh sách liên kết đơn rỗng
- Tạo 1 nút có trường `info` bằng x
- Thêm một phần tử có khóa x vào danh sách
- Duyệt danh sách
- Tìm một phần tử có `info` bằng x
- Hủy một phần tử trong danh sách
- Sắp xếp danh sách liên kết đơn
- ...

Khởi tạo danh sách liên kết RỖNG

- Địa chỉ của nút đầu tiên, địa chỉ của nút cuối cùng đều không có

```
void CreateEmptyList(LIST &l) {
    l.pHead = NULL;
    l.pTail = NULL;
}
```

Tạo 1 phần tử mới

- Hàm trả về địa chỉ phần tử mới tạo

```
NODE* CreateNode(int x) {
    NODE* p;
    p = new NODE;
    if (p == NULL)
          exit(1);
    p->info = x;
    p->pNext = NULL;
    return p;
}
```

Thêm 1 phần tử vào DSLK

- **Nguyên tắc thêm:** Khi thêm 1 phần tử vào List thì có làm cho pHead, pTail thay đổi.
- **Các vị trí cần thêm 1 phần tử vào List:**
  - Thêm vào đầu List đơn
  - Thêm vào cuối List
  - Thêm vào sau 1 phần tử Q trong List

Thuật toán: Thêm 1 phần tử vào đầu DSLK

- Thêm nút p vào đầu danh sách liên kết đơn

Bắt đầu:

Nếu List rỗng thì

+ pHead = p;

+ pTail = pHead;

Ngược lại

+ p->pNext = pHead;

+ pHead = p

Thuật toán: Code C/C++

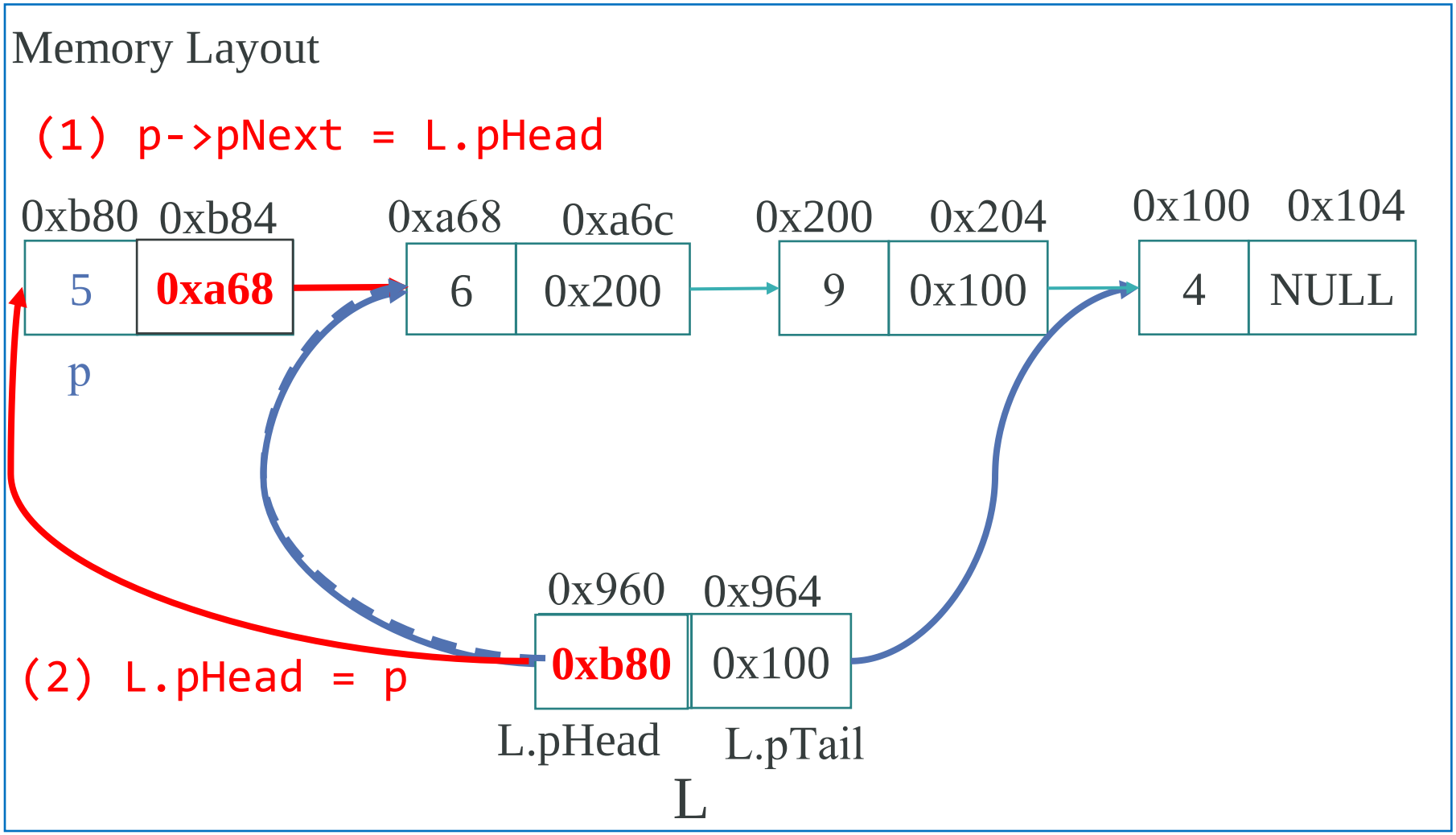
```
void AddHead(LIST &L, NODE* p) {
    if (L.pHead == NULL) {
        L.pHead = p;
        L.pTail = L.pHead;
    }
    else {
        p->pNext = L.pHead;
        L.pHead = p;
    }
}
```

Thuật toán: Code C/C++

```
void AddHead(LIST &L, int x) {
    NODE* p= CreateNode(x);
    if (L.pHead == NULL) {
        L.pHead = p;
        L.pTail = L.pHead;
    }
    else {
        p->pNext = L.pHead;
        L.pHead = p;
    }
}
```

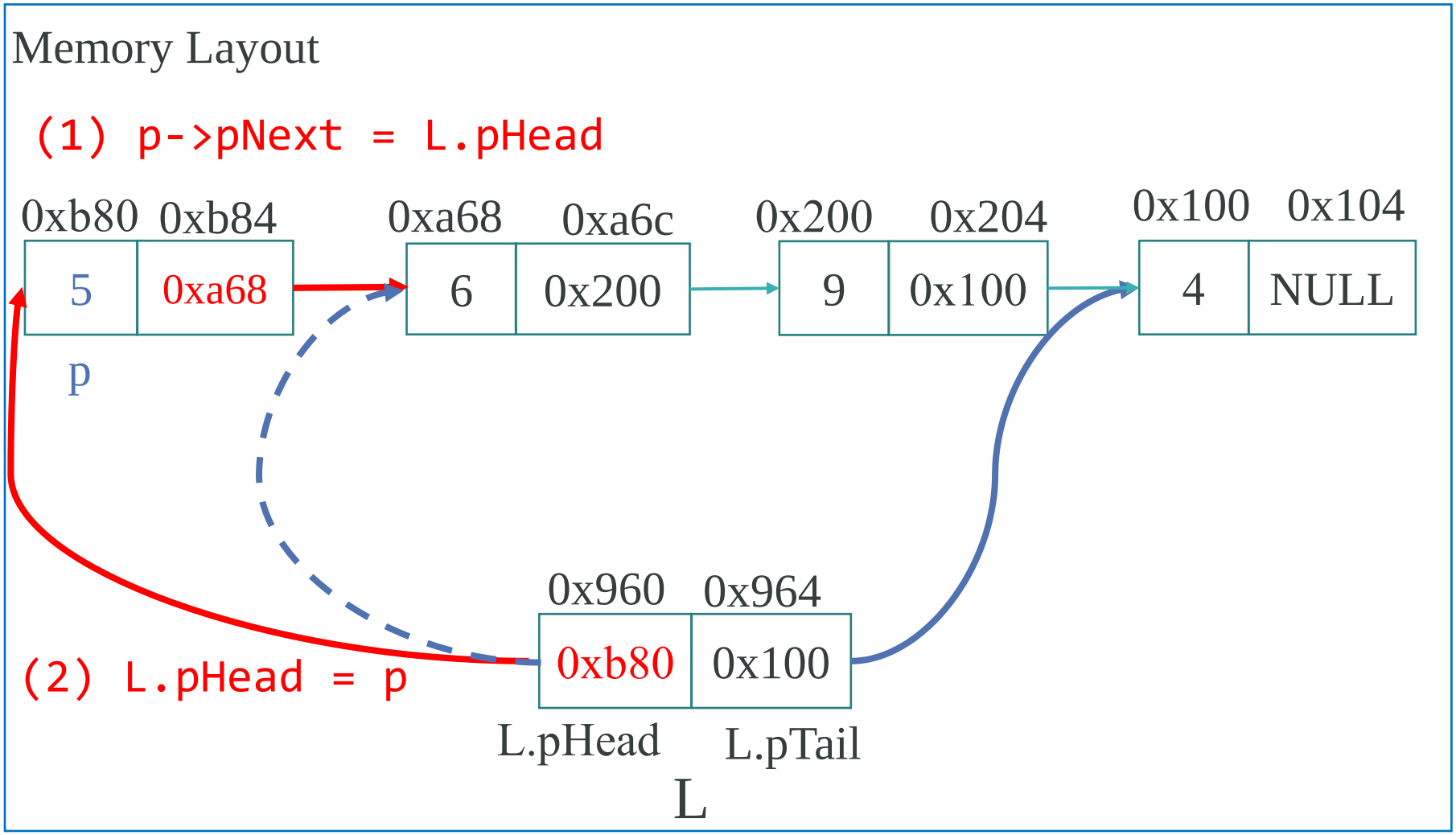
Thêm 1 phần tử vào đầu DSLK: Minh họa (tt)

- Thêm 5 vào đầu danh sách có thứ tự như sau: {6, 9, 4}



Thêm 1 phần tử vào đầu DSLK: Minh họa (tt)

- Thêm 5 vào đầu danh sách có thứ tự như sau: {6, 9, 4}



Thuật toán thêm vào cuối DSLK

- Ta cần thêm nút p vào cuối list đơn

Bắt đầu:

Nếu List rỗng thì

+ pHead = p;

+ pTail = pHead;

Ngược lại

+ pTail->pNext=p;

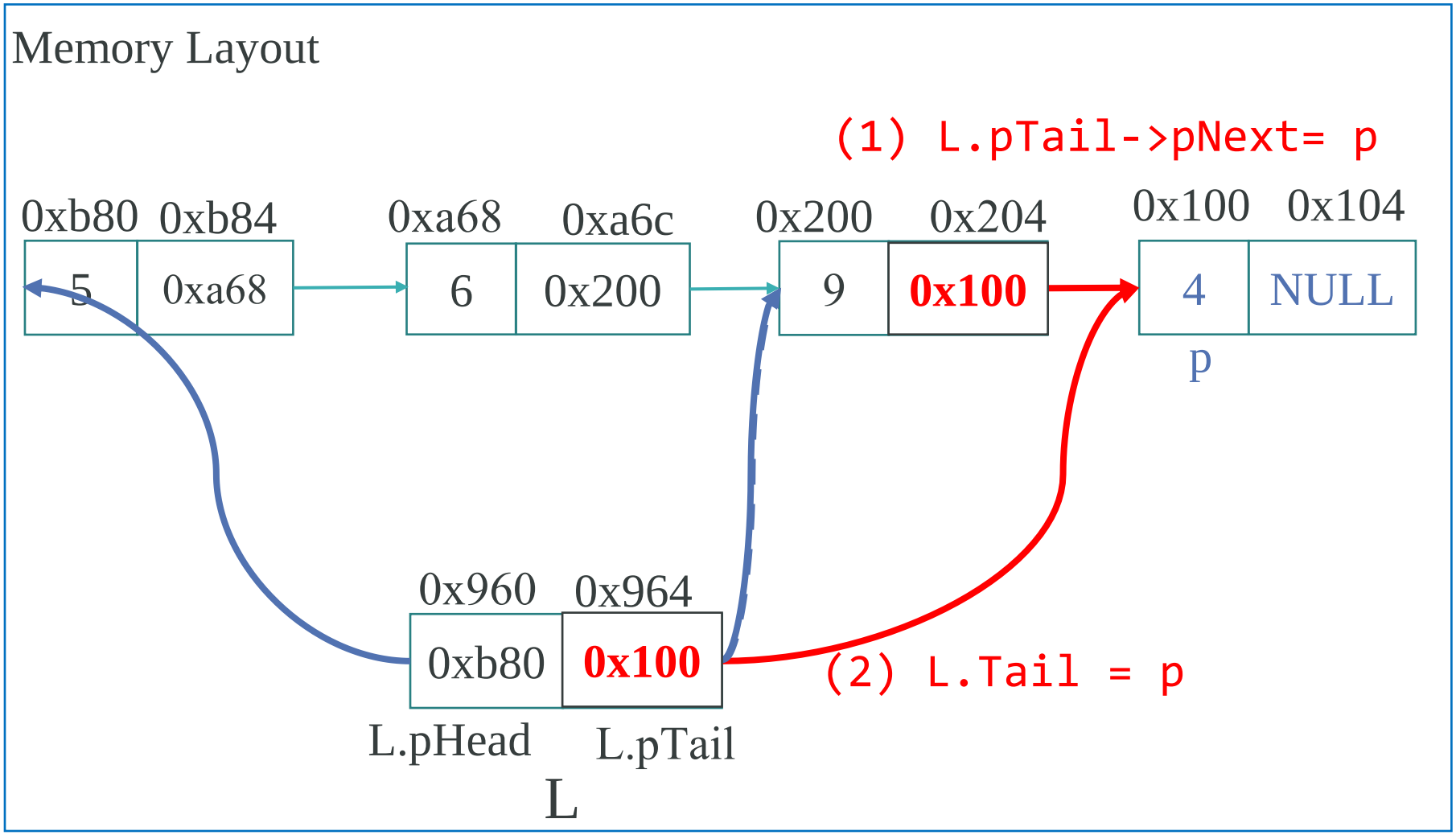
+ pTail=p

Cài đặt: Code C/C++

```
void AddTail(LIST &L, NODE* p) {
    if (L.pHead == NULL) {
        L.pHead = p;
        L.pTail = L.pHead;
    }
    else {
        L.pTail->pNext = p;
        L.pTail = p;
    }
}
```

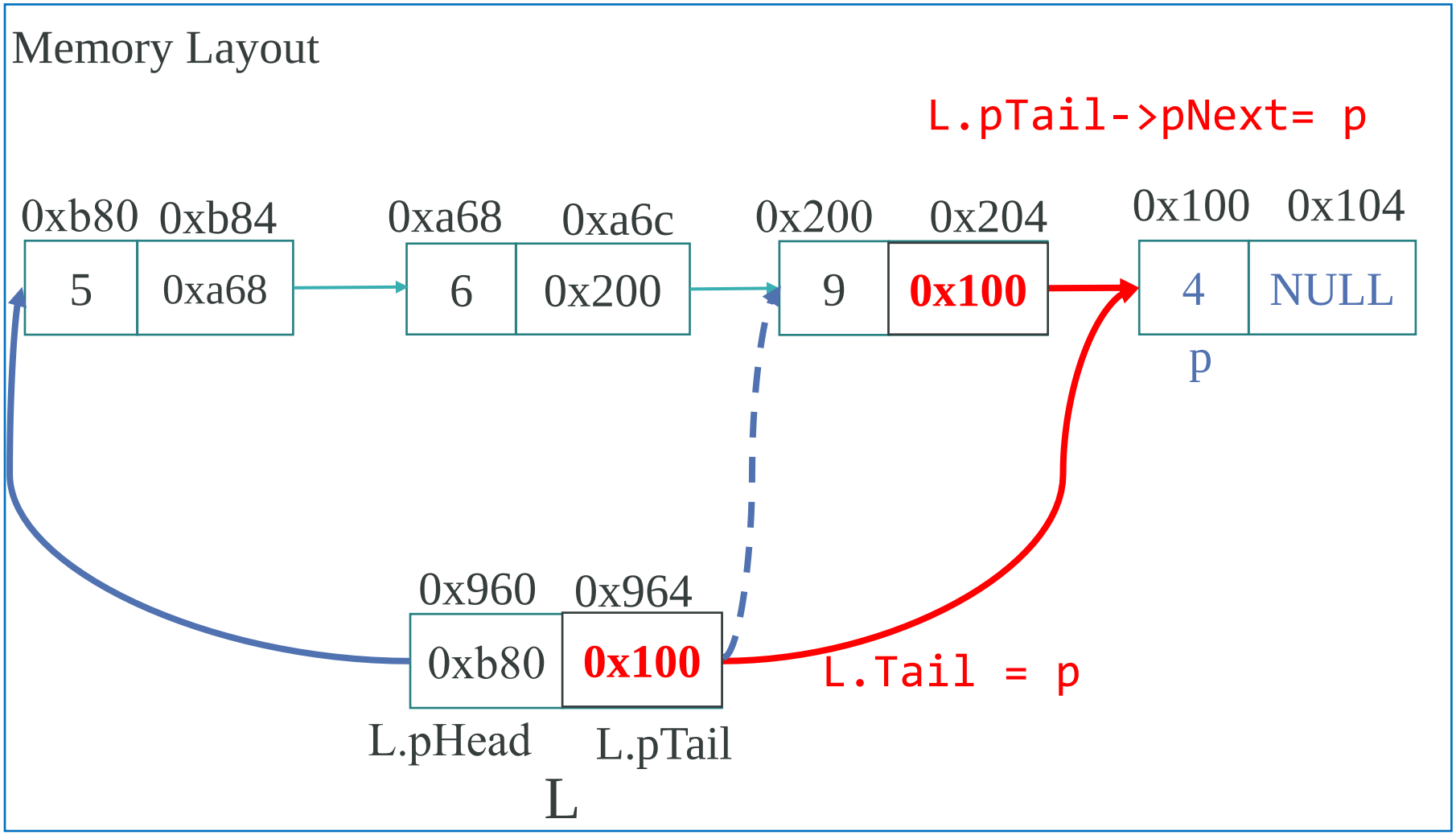
Thêm 1 phần tử vào cuối DSLK: Minh họa (tt)

- Thêm 4 vào cuối danh sách ban đầu có thứ tự như sau: {5, 6, 9}



Thêm 1 phần tử vào cuối DSLK: Minh họa (tt)

- Thêm 4 vào cuối danh sách ban đầu có thứ tự như sau: {5, 6, 9}

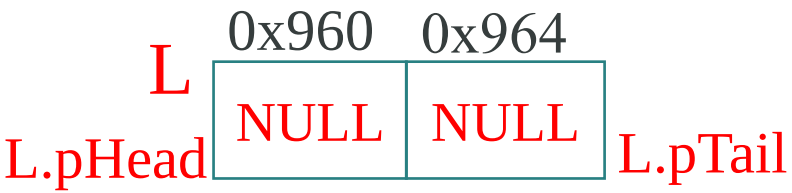


Ví dụ: Quá trình Thêm từng phần tử vào danh sách

- Tạo danh sách RỖNG.

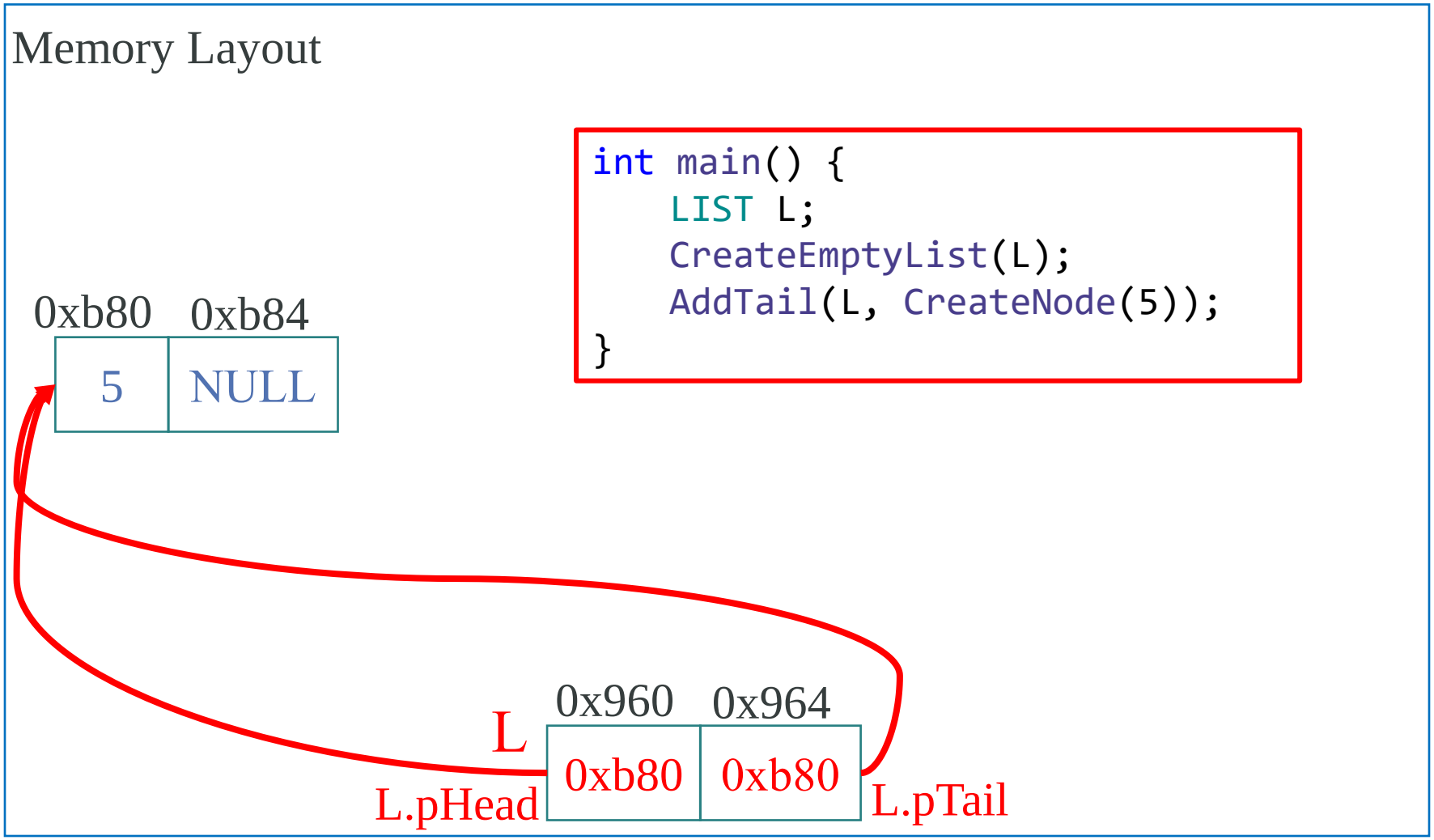
Memory Layout

```
int main() {
    LIST L;
    CreateEmptyList(L);
}
```



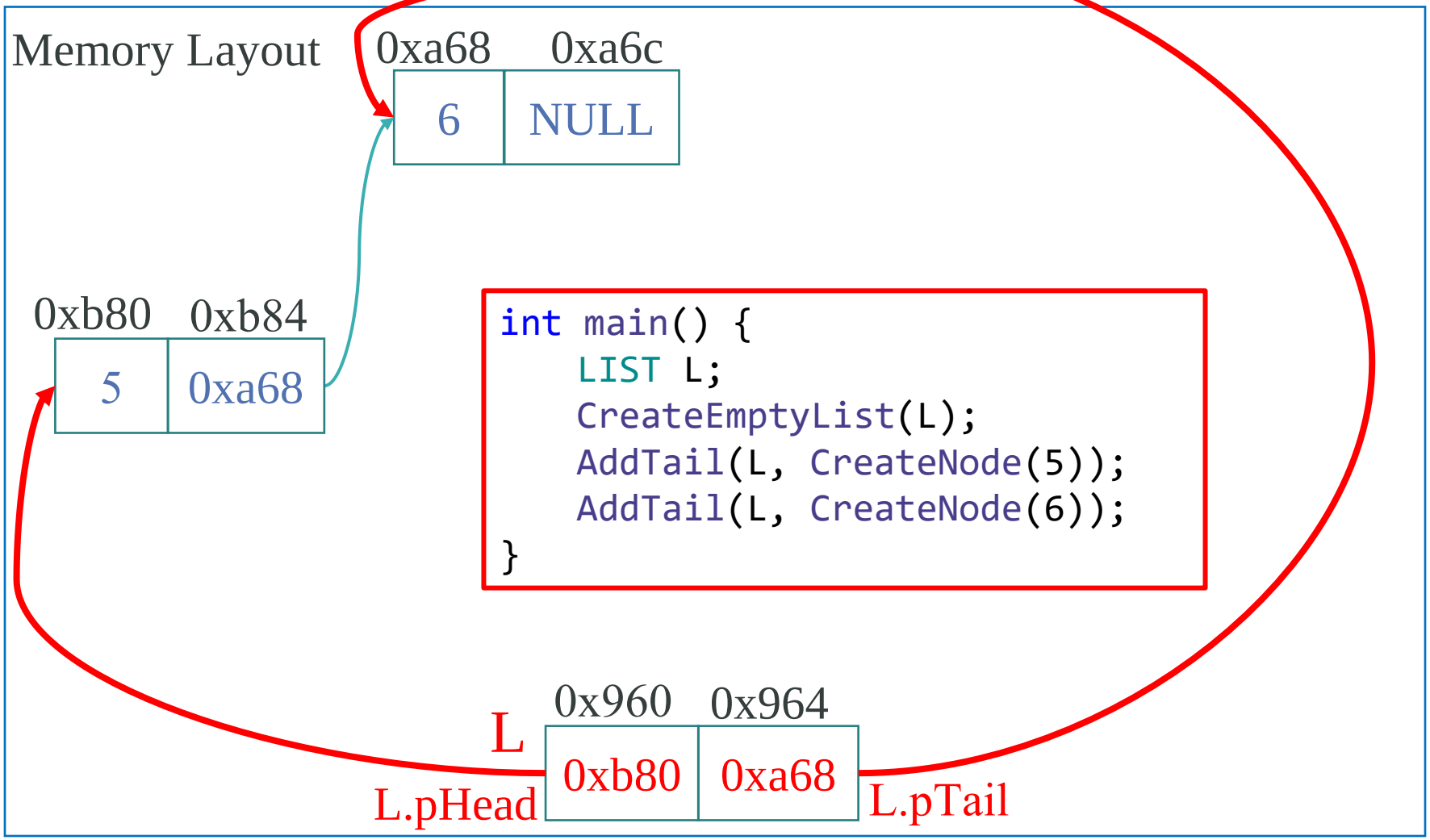
Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 5 vào danh sách.



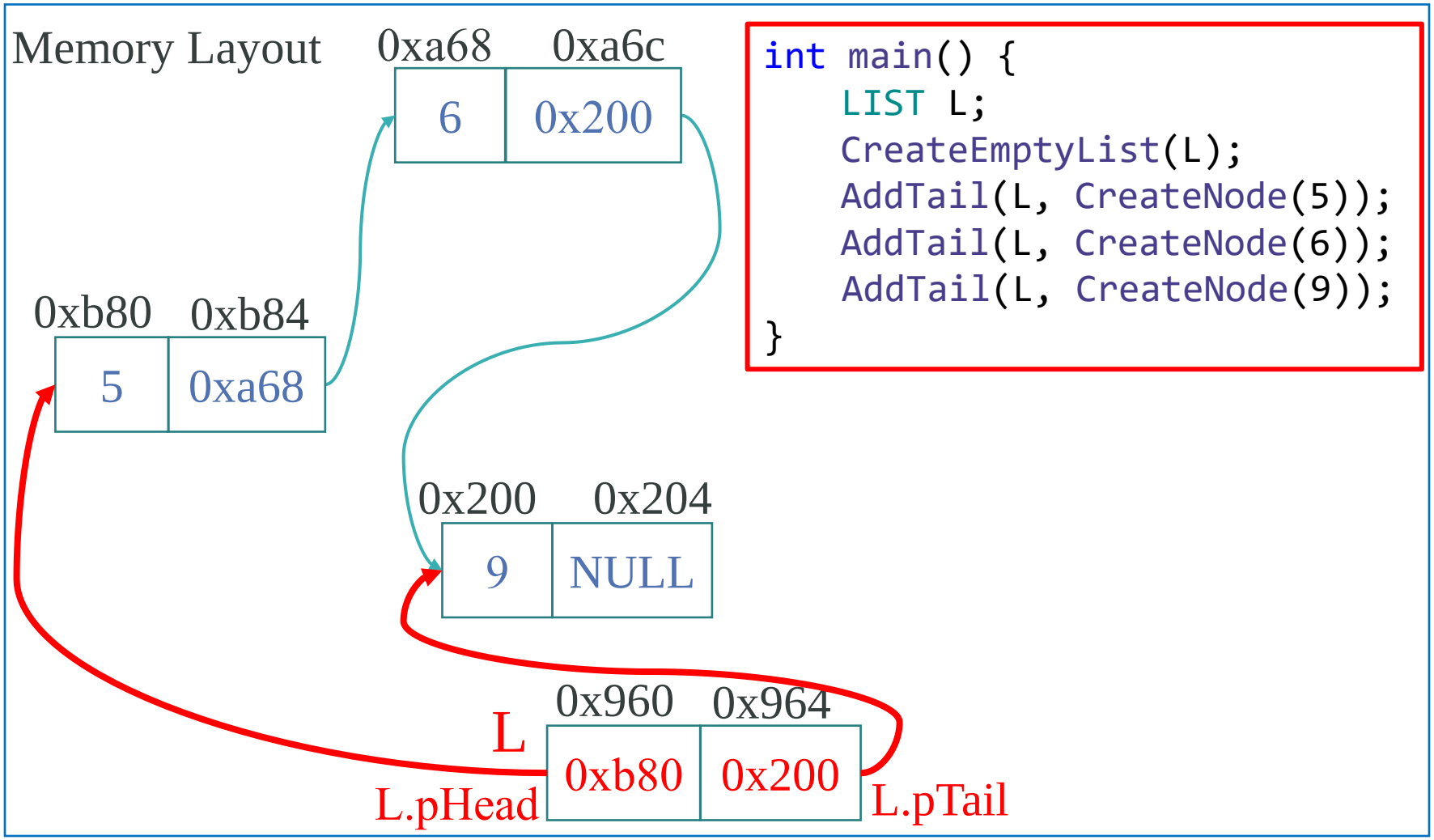
Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 6 vào danh sách.



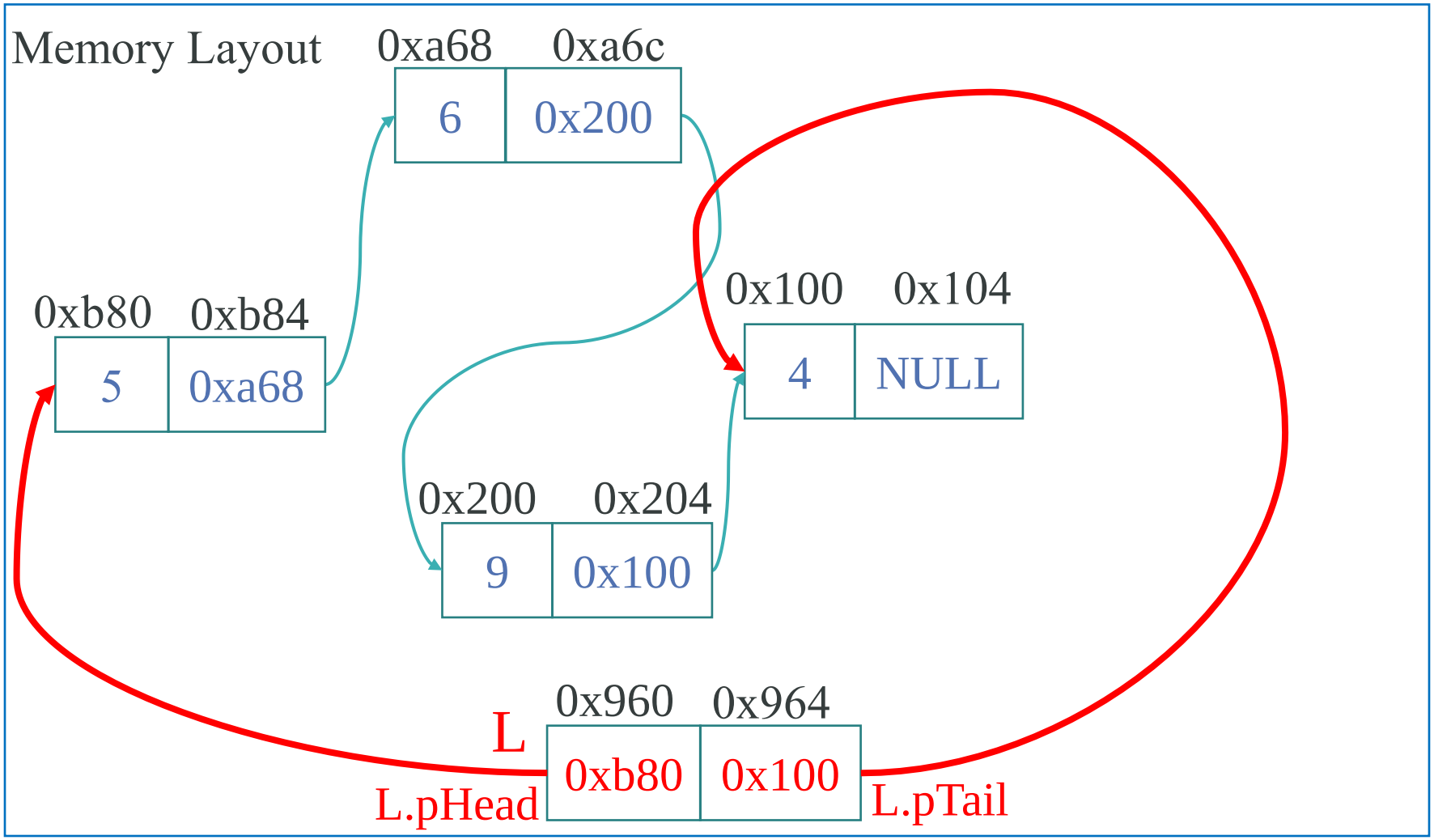
Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 9 vào danh sách.



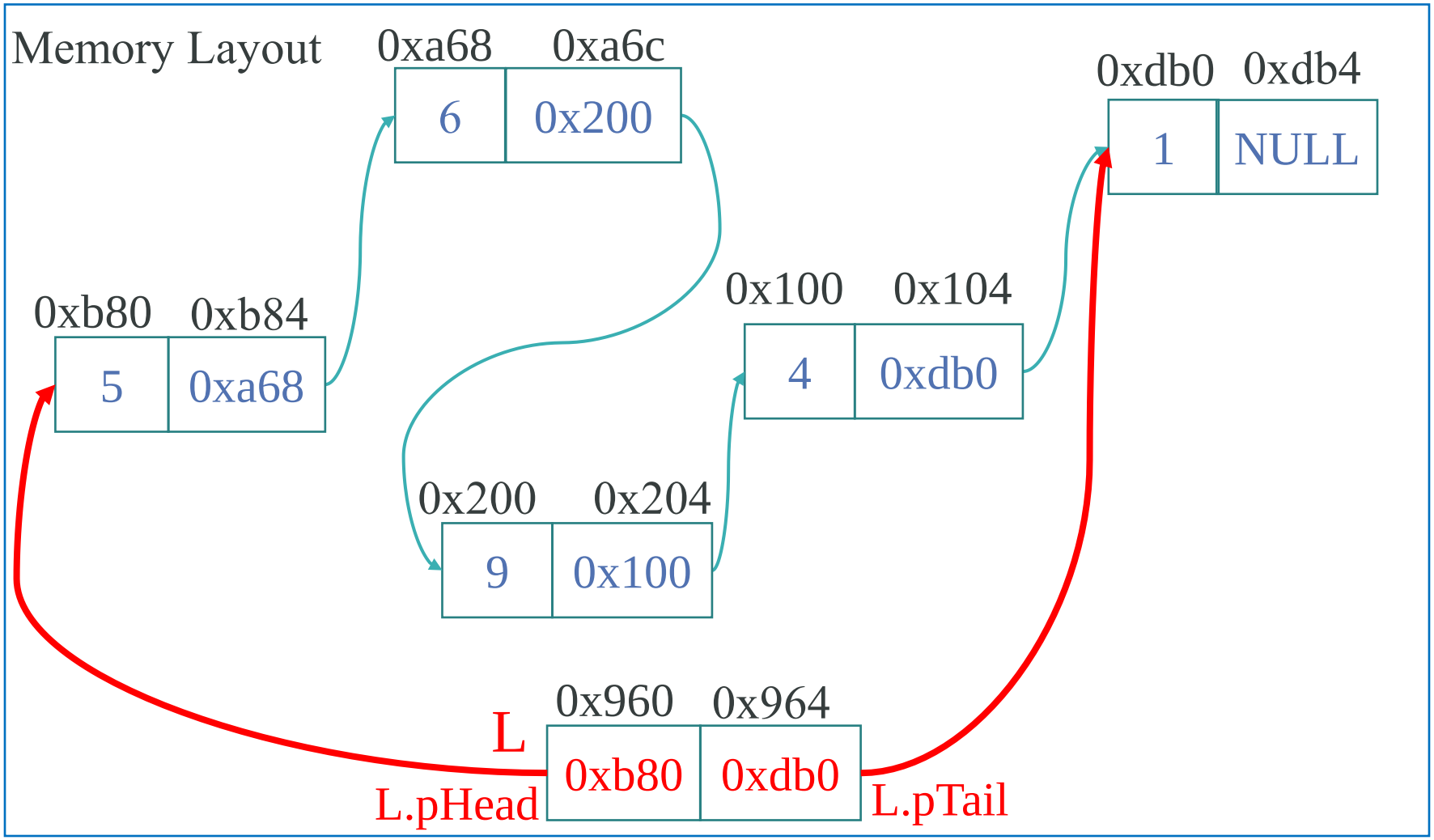
Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 4 vào danh sách.



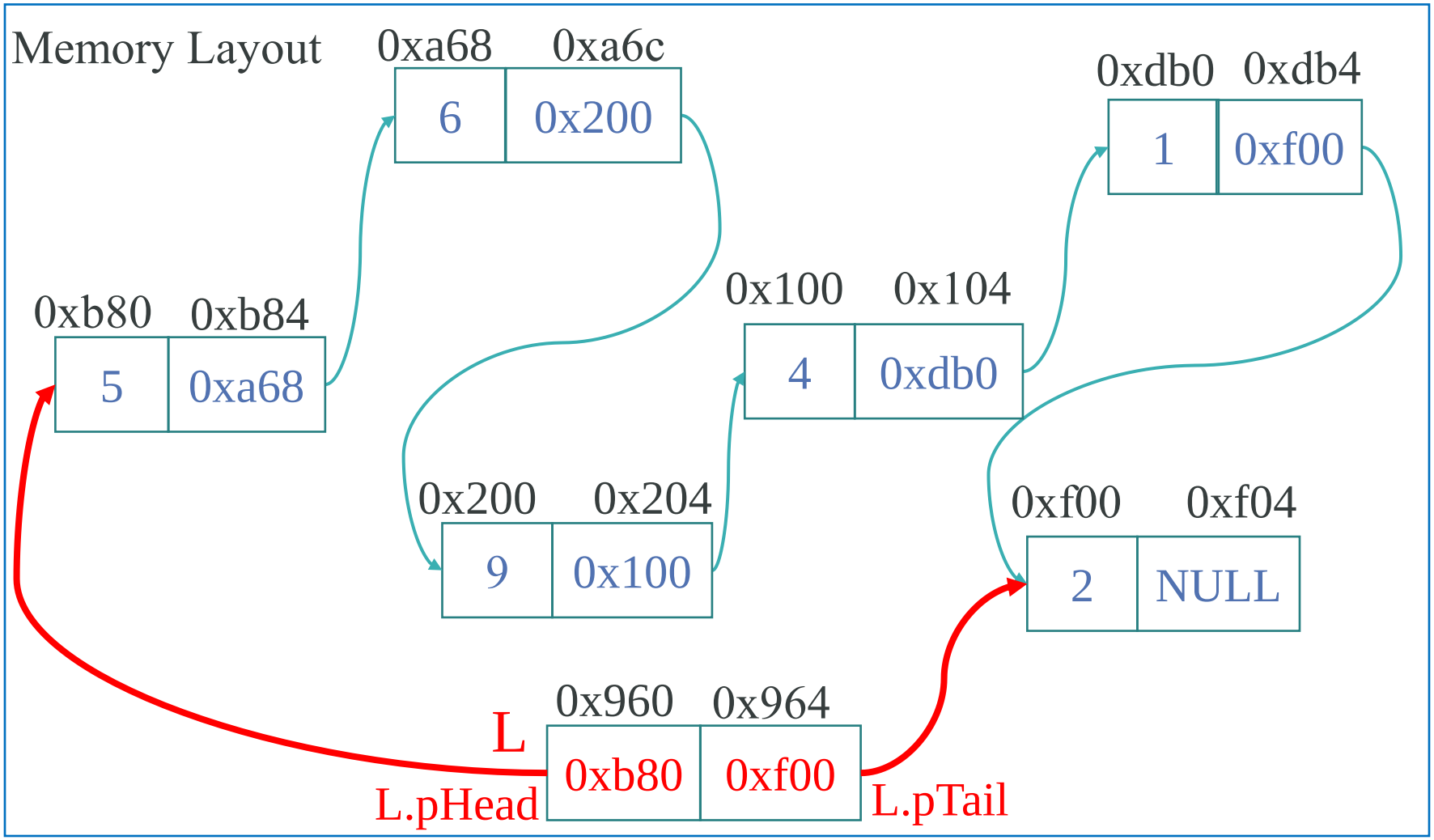
Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 1 vào danh sách.



Ví dụ: Quá trình Thêm từng phần tử vào danh sách (tt)

- Thêm 2 vào danh sách



Thuật toán: Thêm phần tử p vào sau phần tử Q

- Ta cần thêm nút p vào sau nút Q trong list đơn

Bắt đầu:

Nếu (Q != NULL) thì

B1: p->pNext = Q->pNext

B2:

+ Q->pNext = p

+ Nếu Q = pTail thì

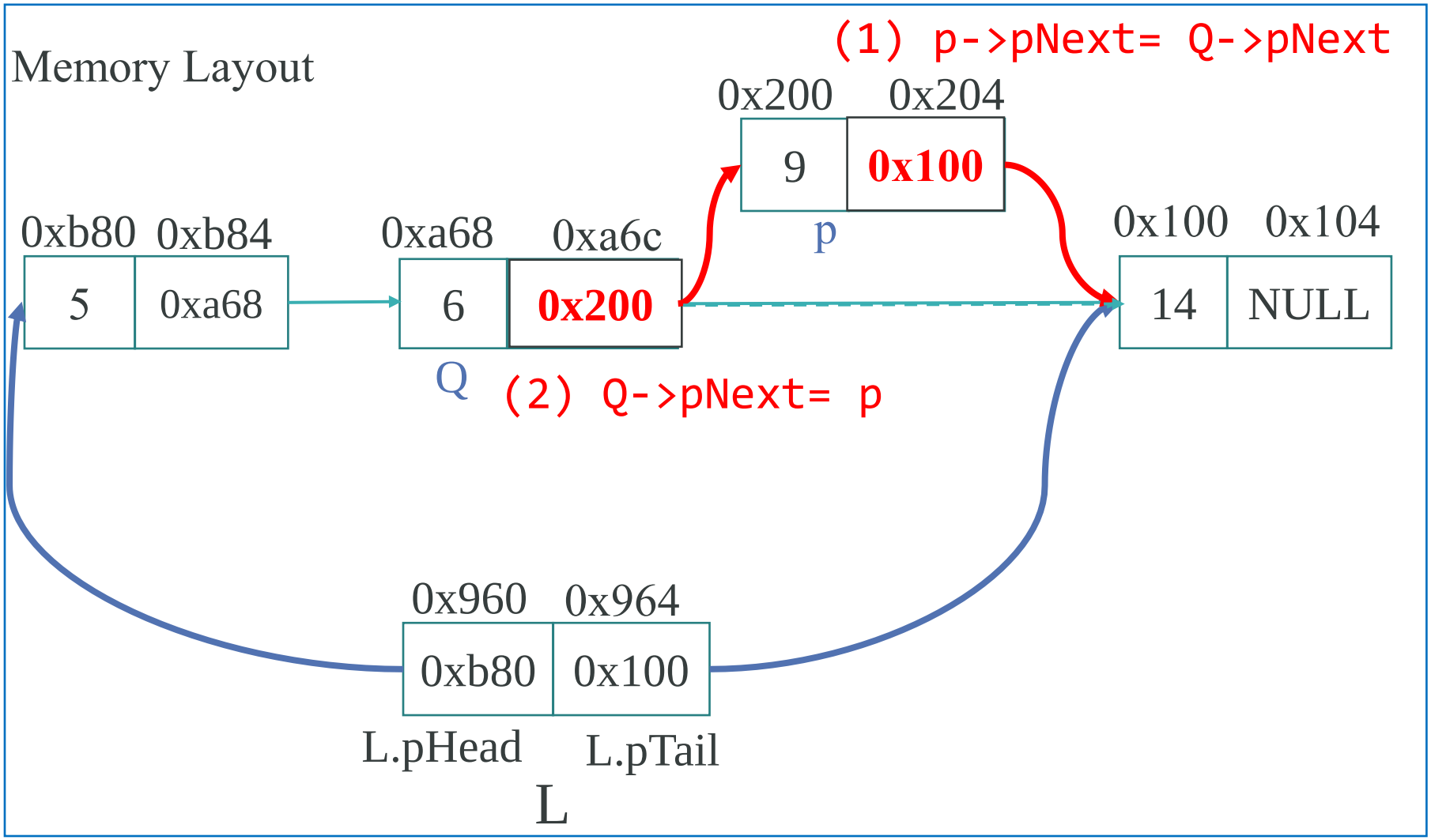
pTail=p

Thuật toán: Code C/C++

```
void AddAfterQ(LIST &L, NODE* p, NODE* Q) {
    if (Q != NULL) {
        p->pNext = Q->pNext;
        Q->pNext = p;
        if (L.pTail == Q)
                L.pTail = p;
    }
    else
         AddHead(L, p); // thêm q vào đầu list
}
```

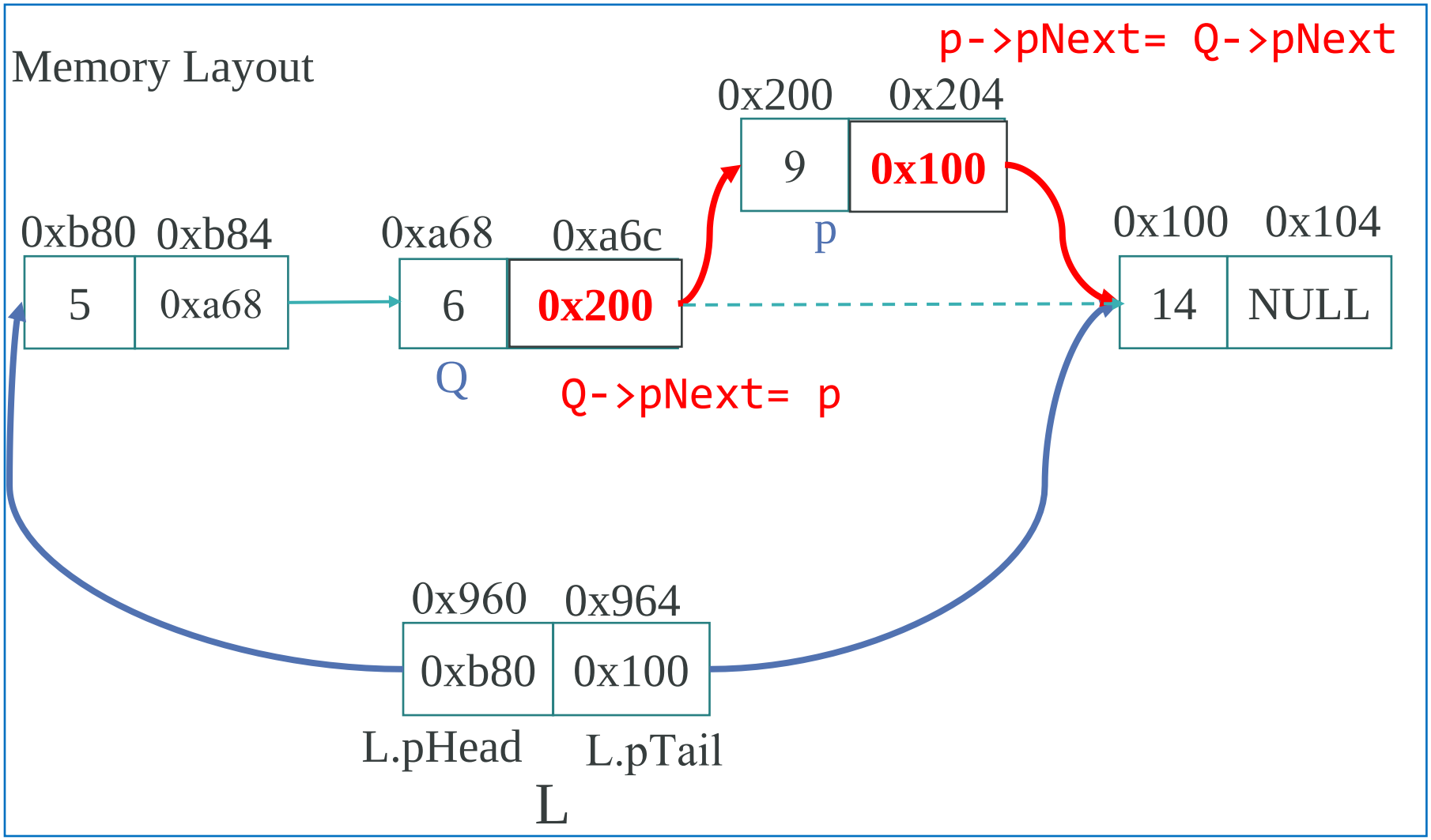
Minh họa

- Thêm 9 vào sau node Q có giá trị 6: {5, 6, 14}



Minh họa (tt)

- Thêm 9 vào sau node Q có giá trị 6: {5, 6, 14}



Hủy phần tử trong DSLK đơn

- **Nguyên tắc**: Phải cô lập phần tử cần hủy trước hủy.
- Các vị trị cần hủy
  - Hủy phần tử đứng đầu List
  - Hủy phần tử có khoá bằng x
  - Huỷ phần tử đứng sau q trong danh sách liên kết đơn
- Ở phần trên, các phần tử trong DSLK đơn được cấp phát vùng nhớ động bằng hàm new, thì sẽ được giải phóng vùng nhớ bằng hàm delete.

Thuật toán: Hủy phần tử đầu trong DSLK

- Bắt đầu:

Nếu (pHead!=NULL) thì

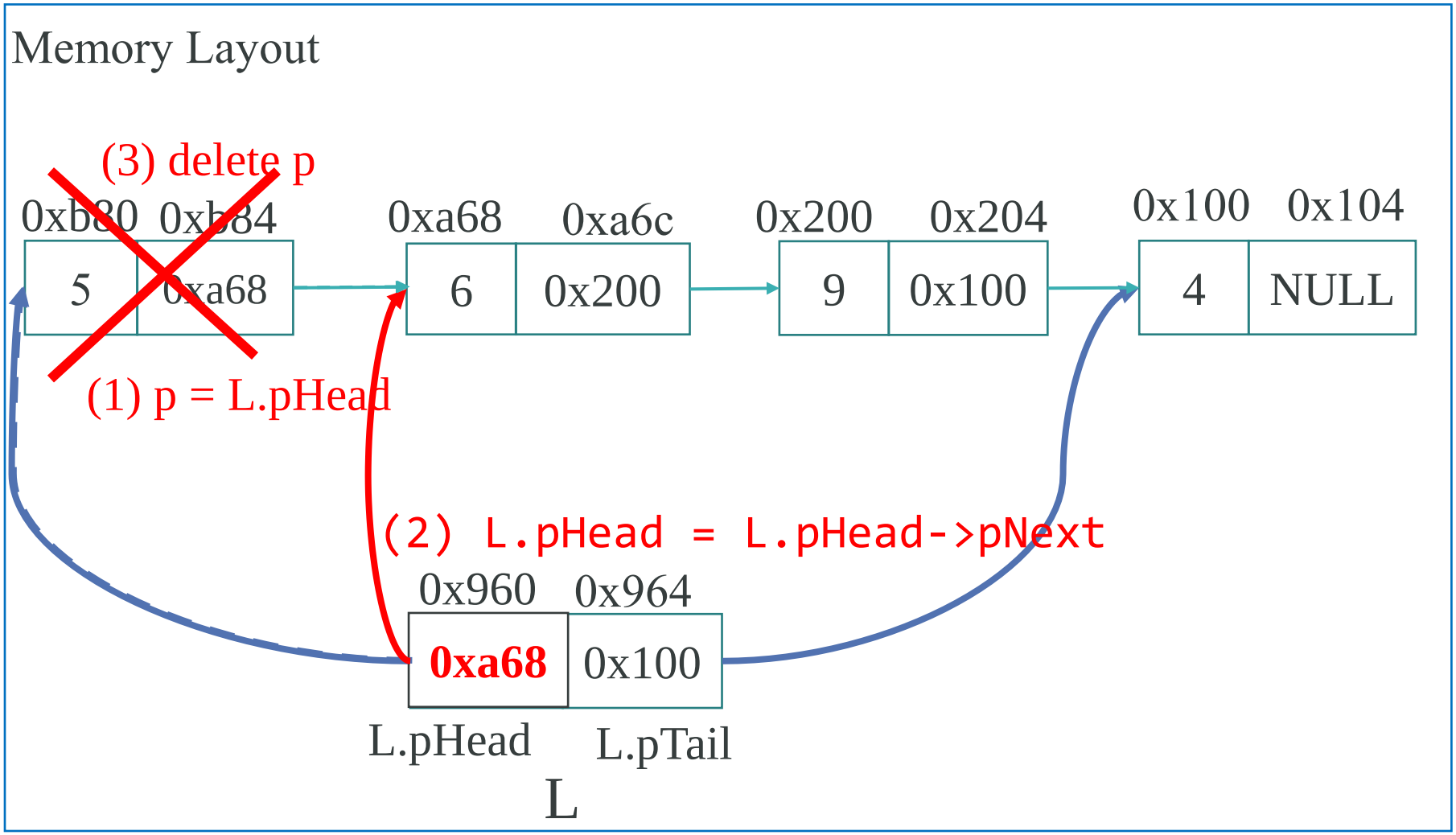
- B1: p=pHead
- B2:
  + pHead = pHead->pNext
  + delete (p)
- B3:
  Nếu pHead==NULL thì pTail=NULL

Thuật toán: Code C/C++

```
bool RemoveHead(LIST &L, int &x) {
    NODE *p;
    if (L.pHead !=NULL) {
        x = L.pHead->info;
        p = L.pHead;
        L.pHead = L.pHead->pNext;
        if (L.pHead == NULL)
                L.pTail = NULL;
        delete p;
        return 1; // Đã xóa phần tử
    }
    return 0; // List rỗng không có phần tử để xóa
}
```

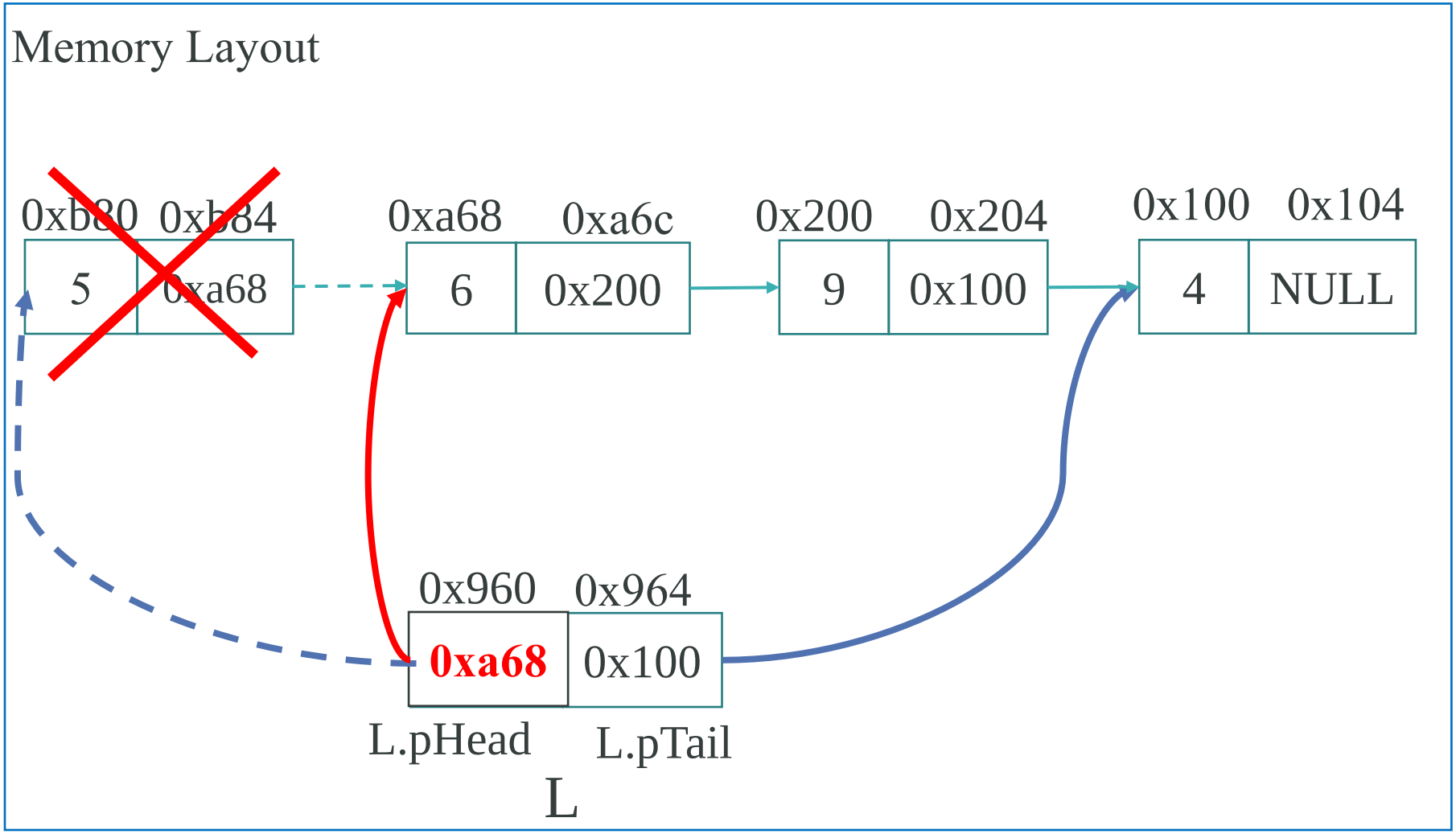
Hủy phần tử đầu trong DSLK: Minh họa

- Hủy phần tử đầu danh sách liên kết có thứ tự như sau: {5, 6, 9, 4}



Hủy phần tử đầu trong DSLK: Minh họa (tt)

- Hủy phần tử đầu danh sách liên kết có thứ tự như sau: {5, 6, 9, 4}



Thuật toán hủy phần tử đứng sau phần tử Q

- Bắt đầu

  Nếu (Q != NULL) thì: // Q tồn tại trong List

  - B1: p= Q ->pNext; // p là phần tử cần hủy
  - B2: Nếu (p != NULL) thì // Q không là phần tử cuối

    + Q ->pNext=p->pNext; // tách p ra khỏi xâu

    + Nếu (p == pTail) // nút cần hủy là nút cuối

      pTail= Q;

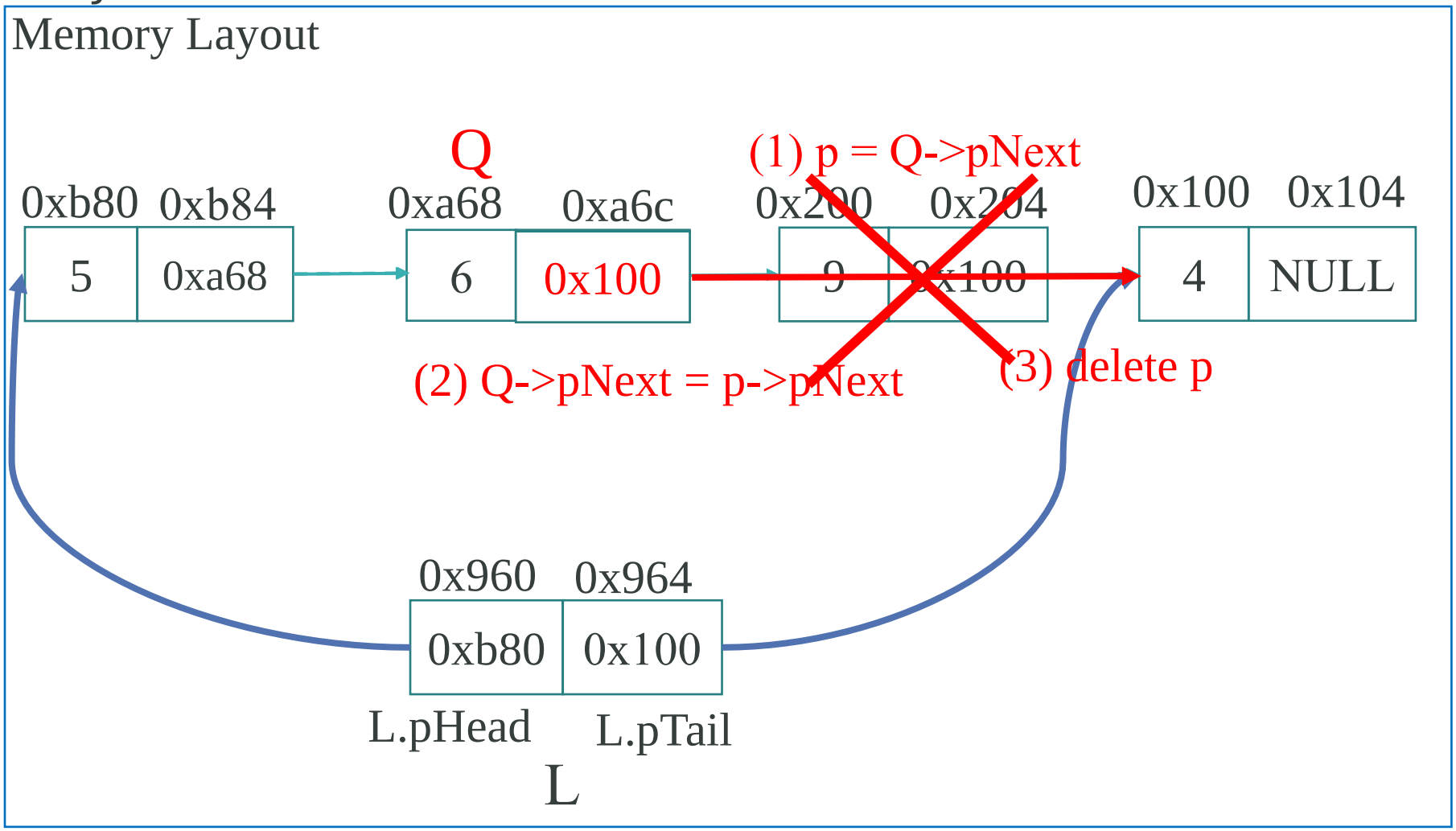
    + delete p; // hủy p

Thuật toán: Code C/C++

```
bool RemoveAfterQ(LIST &l, NODE *Q, int &x) {
    NODE *p;
    if (Q != NULL) {
        p = Q->pNext;           // p là nút cần xoá
        if (p != NULL) {        // q không phải là nút cuối
            if (p == l.pTail)   // nút cần xoá là nút cuối cùng
                l.pTail = Q;    // cập nhật lại pTail
            Q->pNext = p->pNext;
            x = p->info;
            delete p;
        }
        return 1;
    }
    return 0;
}
```

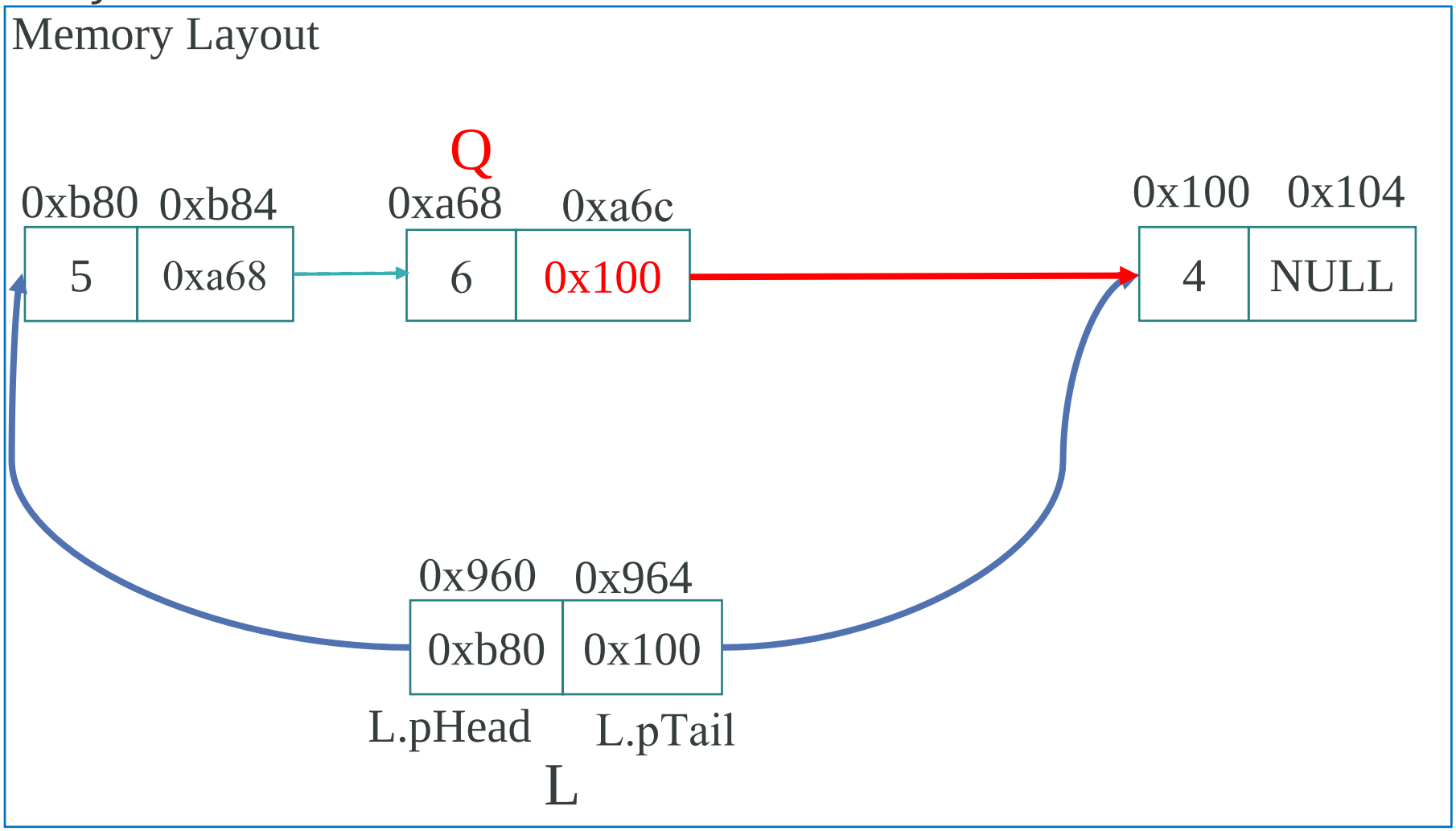
Thuật toán hủy phần tử đứng sau phần tử Q: Minh họa

- Hủy phần tử đứng sau node có giá trị 6 trong danh sách: {5, 6, 9, 4}



Thuật toán hủy phần tử đứng sau phần tử Q: Minh họa

- Hủy phần tử đứng sau node có giá trị 6 trong danh sách: {5, 6, 9, 4}



Thuật toán hủy phần tử có khoá x

Bước 1:

Tìm phần tử p có khoá bằng x, và Q đứng trước p

Bước 2:

Nếu (p!=NULL) thì //tìm thấy phần tử có khoá bằng x

Hủy p ra khỏi List bằng cách hủy phần tử đứng sau Q

Ngược lại: Báo không tìm thấy phần tử có khoá x

Thuật toán: Cài đặt C/C++

```
bool RemoveX(LIST &L, int x) {
    NODE *Q, *p;
    Q = NULL;
    p = L.pHead;
    while (p!=NULL && p->info!=x) {
        Q = p;
        p = p->pNext;
    }

    if (p == NULL) return 0;  // Tìm không thấy x
    if (Q != NULL) RemoveAfterQ(L, Q, x);
    else RemoveHead(L, x);
    return 1;
}
```

Tìm 1 phần tử trong DSLK đơn

- Tìm tuần tự (hàm trả về), các bước của thuật toán tìm nút có info bằng x trong list đơn.

Bước 1: p=pHead; // địa chỉ của phần tử đầu trong list đơn

Bước 2:

Trong khi p!=NULL và p->info!=x thì

p=p->pNext; // xét phần tử kế

Bước 3:

+ Nếu p!=NULL thì p lưu địa chỉ của nút có info = x

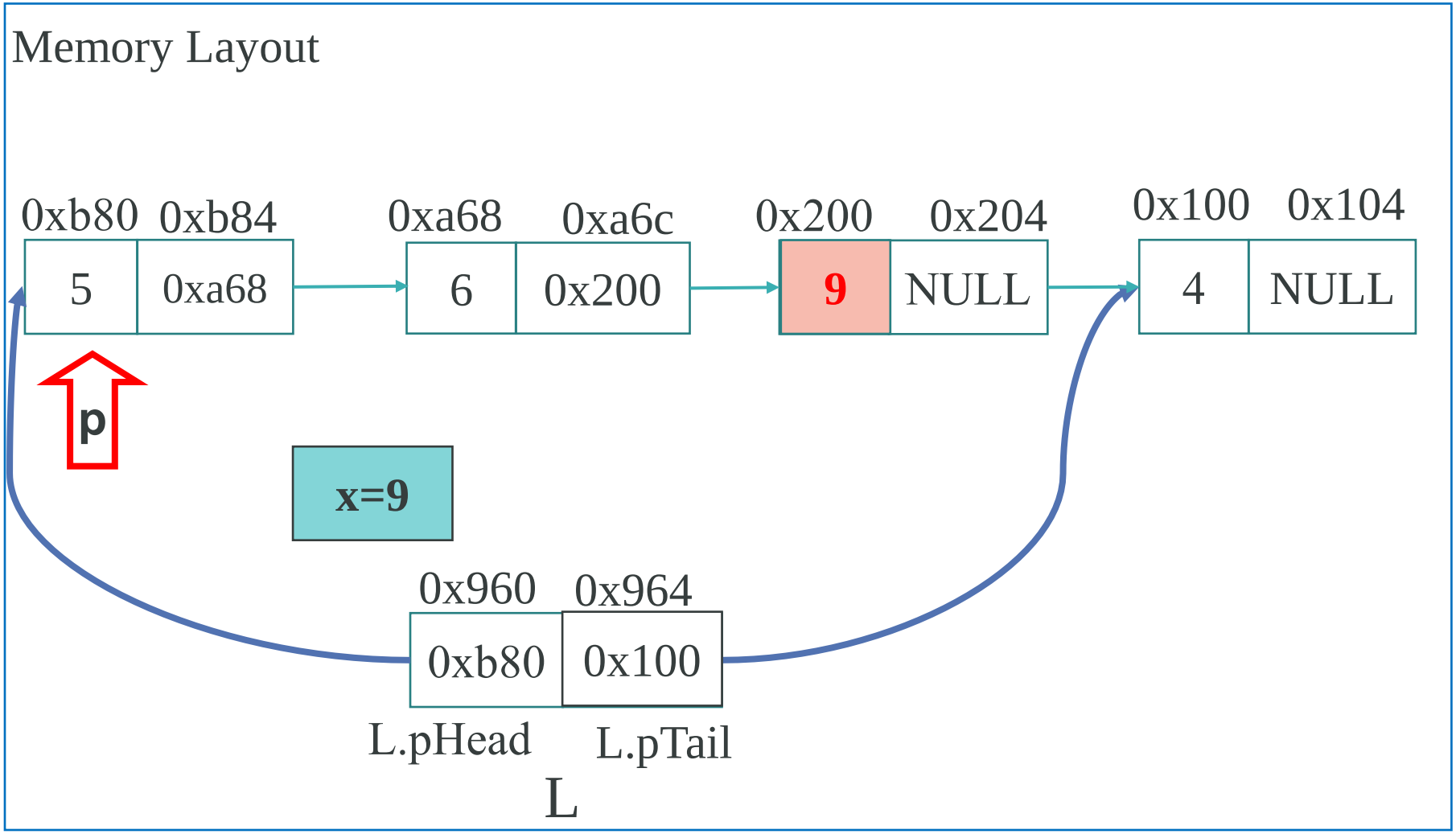
+ Ngược lại: Không có phần tử cần tìm

Thuật toán: Code C/C++

```
// Hàm trả về NULL: Không tìm thấy
// Hàm trả về khác NULL: Tìm thấy
NODE* SearchNode(LIST L, int x) {
    NODE* p = L.pHead;
    while (p != NULL && p->info != x)
        p = p->pNext;
    return p;
}
```

Tìm 1 phần tử trong DSLK đơn: Minh họa

- Tìm phần tử có giá trị bằng 9 trong DSLK gồm {5, 6, 9, 4}



Duyệt danh sách

- Duyệt danh sách là thao tác thường được thực hiện khi có nhu cầu cần xử lý các phần tử trong danh sách như:
  - Đếm các phần tử trong danh sách
  - Tìm tất cả các phần tử trong danh sách thảo điều kiện
  - Hủy toàn bộ danh sách

Thuật toán duyệt danh sách

- Bước 1:

  p = pHead;// p lưu địa chỉ của phần tử đầu trong List

- Bước 2:

  Trong khi (danh sách chưa hết) thực hiện

  + Xử lý phần tử p

  + p=p->pNext; // qua phần tử kế

Cài đặt in các phần tử trong List

```
void PrintList(LIST L) {
    NODE* p;
    if (L.pHead == NULL)
          cout << "\nDSLK rong.";
    else {
         p = L.pHead;
         while (p) {
            cout << p->info << "\t";
            p = p->pNext;
         }
    }
}
```

Hủy danh sách liên kết đơn

- Bước 1:

  Trong khi việc duyệt danh sách chưa kết thúc, thực hiện:

  - B1.1:

    p = pHead;

    pHead = pHead->pNext; // cập nhật pHead

  - B1.2:

    Hủy p

- Bước 2:

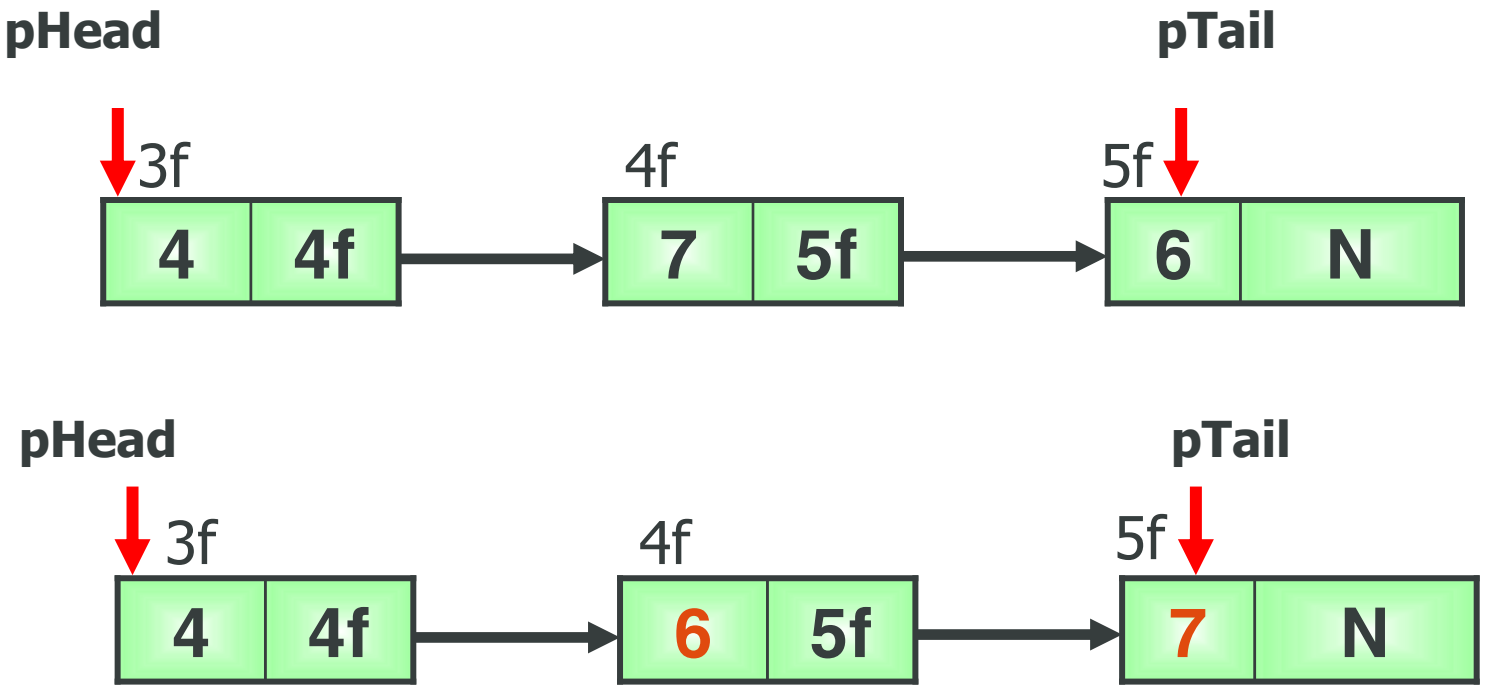
  pTail = NULL; // bảo toàn tính nhất quán khi xâu rỗng

Thuật toán: Code C/C++

```
void RemoveList(LIST &L) {
    NODE * p;
    while (L.pHead != NULL) {
        p = L.pHead;
        L.pHead = p->pNext;
        delete p;
    }
    L.pTail = NULL;
}
```

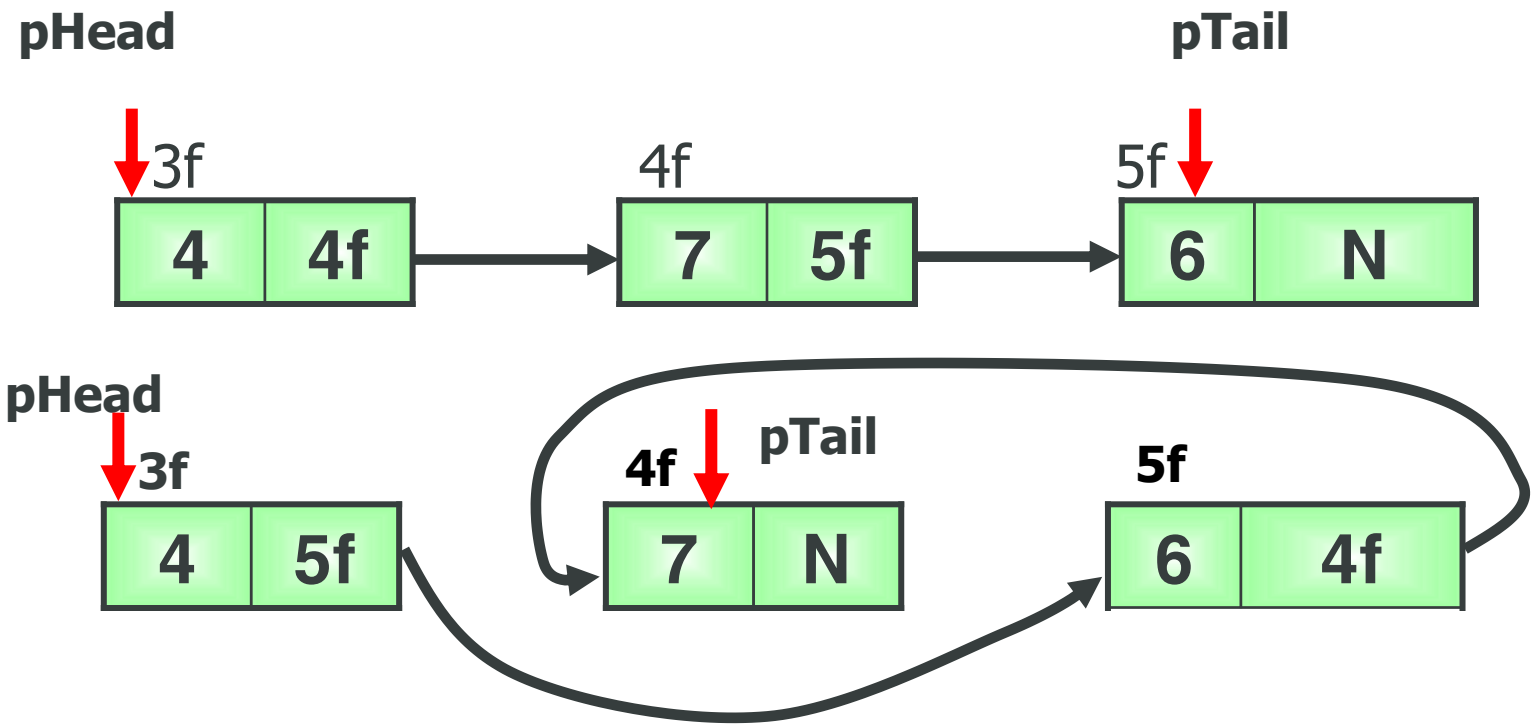
Sắp xếp danh sách

- Có hai cách tiếp cận
- **Cách 1**: Thay đổi thành phần Info



Sắp xếp danh sách

- **Cách 2**: Thay đổi thành phần pNext (thay đổi trình tự móc nối của các phần tử sao cho tạo lập nên được thứ tự mong muốn)



Ưu, nhược điểm của 2 cách tiếp cận

- Thay đổi thành phần info (dữ liệu)
  - Ưu: Cài đặt đơn giản, tương tự như sắp xếp mảng
  - Nhược:
    - Đòi hỏi thêm vùng nhớ khi hoán vị nội dung của 2 phần tử -> chỉ phù hợp với những xâu có kích thước info nhỏ
    - Khi kích thước info (dữ liệu) lớn chi phí cho việc hoán vị thành phần info lớn
    - ✓ Làm cho thao tác sắp xếp chậm
- Thay đổi thành phần pNext
  - Ưu:
    - Kích thước của trường này không thay đổi, do đó không phụ thuộc vào kích thước bản chất dữ liệu lưu tại mỗi nút.
    - ✓ Thao tác sắp xếp nhanh
  - Nhược: Cài đặt phức tạp

Selection Sort: Code C/C++

```
void SelectionSort(LIST &L) {
    NODE *p, *Q, *min;
    p = L.pHead;
    while (p != L.pTail) {
        min = p;
        Q = p->pNext;

        while (Q != NULL) {
            if (Q->info < min->info)
            min = Q;
            Q = Q->pNext;
        }

        if (min != p) Swap(min->info, p->info);

        p = p->pNext;
    }
}
```

Các thuật toán sắp xếp hiệu quả trên List

- Các thuật toán sắp xếp xâu (List) bằng các thay đổi thành phần pNext (thành phần liên kết) có hiệu quả cao như:
  - Thuật toán sắp xếp Quick Sort
  - Thuật toán sắp xếp Merge Sort
  - Thuật toán sắp xếp Radix Sort

Quick Sort

- Bước 1:

  Chọn X là phần tử đầu xâu L làm phần tử cầm canh

  Loại X ra khỏi L

- Bước 2:

  Tách xâu L ra làm 2 xâu L_1(gồm các phần tử nhỏ hơn hoặc bằng x) và L_2 (gồm các phần tử lớn hơn X)

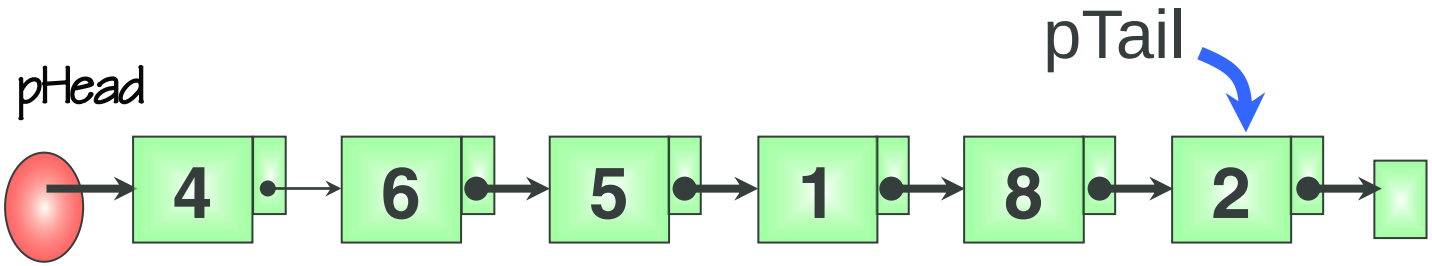
- Bước 3: Nếu (L_1 != NULL) thì QuickSort(L_1)

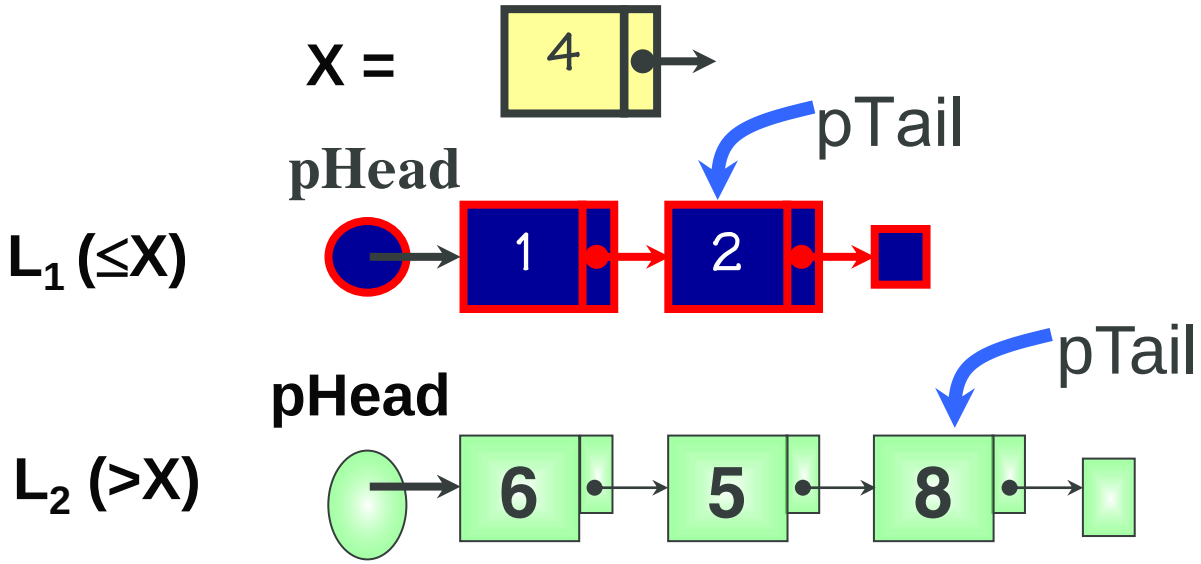
- Bước 4: Nếu (L_2 != NULL) thì QuickSort(L_2)

- Bước 5: Nối L_1, X, L_2 lại theo thứ tự ta có xâu L đã được sắp xếp

Quick Sort: Minh họa

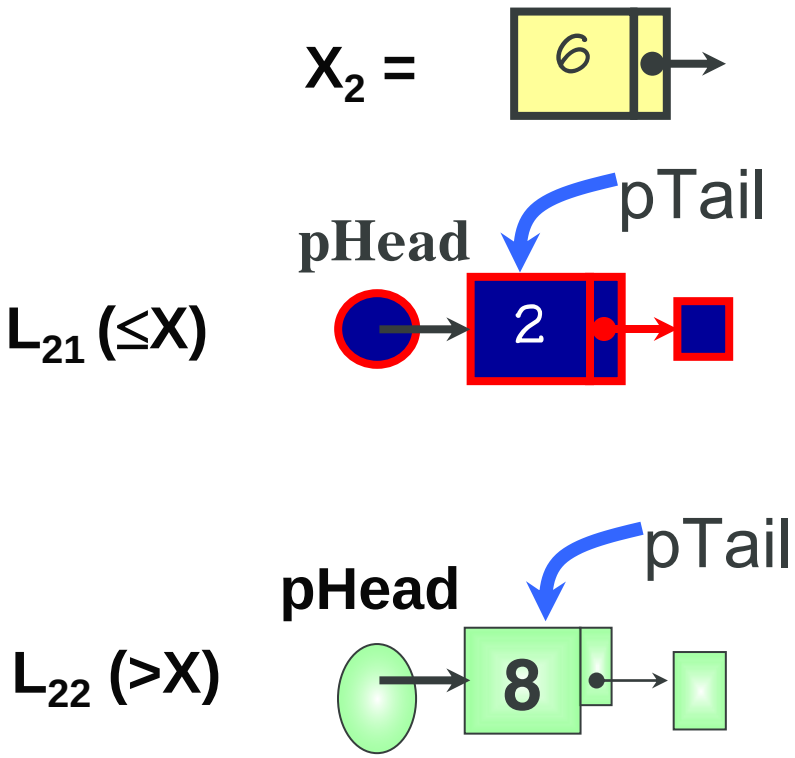
- Cho danh sách liên kết gồm các phần tử sau:





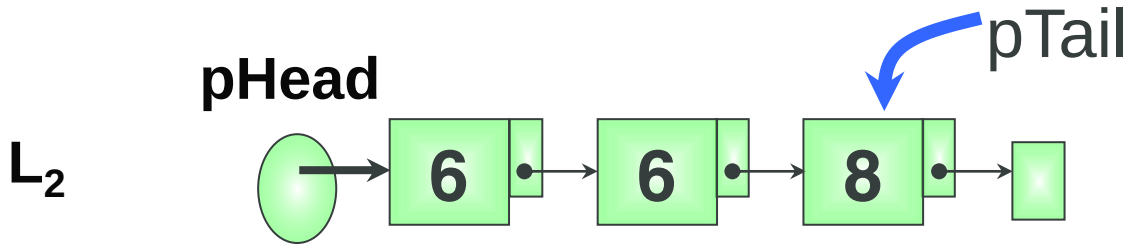
Quick Sort: Minh họa

- Sắp xếp L_1
- Sắp xếp L_2
  - Chọn x=6 cầm canh, và tách L_2 thành L_{21} và L_{22}

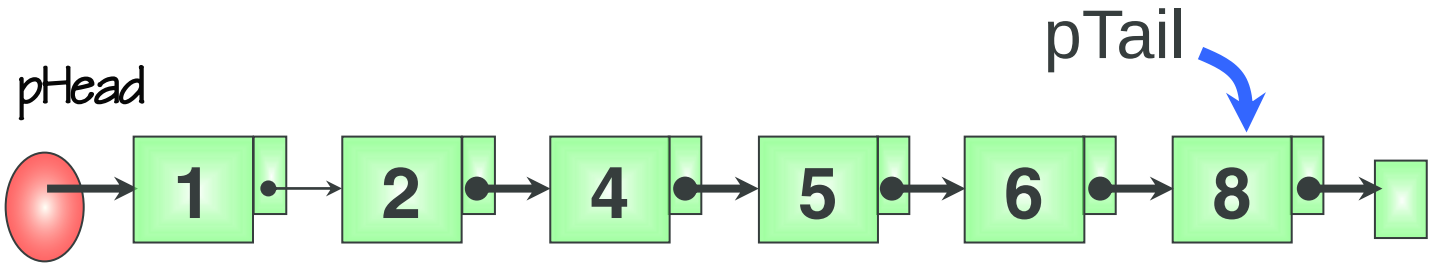


Quick Sort: Minh họa

- Nối L_{21}, X_2, L_{22} thành L_2



- Nối L1, X, L2 thành L



Thuật toán: Code C/C++

```
void QuickSort(LIST &L) {
    LIST L1, L2;
    CreateEmptyList(L1);
    CreateEmptyList(L2);

    NODE *pivot;
    Partition(L, L1, pivot, L2);

    if (L1.pHead != L1.pTail)
          QuickSort(L1);
    if (L2.pHead != L2.pTail)
          QuickSort(L2);
    Join(L, L1, pivot, L2);
}
```

Thuật toán: Code C/C++ (tt)

```
void Partition(LIST &L, LIST &L1, NODE *&pivot, LIST &L2) {
    NODE *p;

    if (L.pHead == NULL) return;

    pivot = SeparateHead(L);

    while (L.pHead != NULL) {
          p = SeparateHead(L);
          if (p->info <= pivot->info)
               AddTail(L1, p);
          else
               AddTail(L2, p);
    }
}
```

Thuật toán: Code C/C++ (tt)

```
void Join(LIST &L1, LIST &L2, NODE *pivot, LIST &L3) {
    NODE *p;

    while (L2.pHead != NULL) {
        p = SeparateHead(L2);
        AddTail(L1, p);
    }

    AddTail(L1, pivot);

    while (L3.pHead != NULL) {
        p = SeparateHead(L3);
        AddTail(L1, p);
    }
}
```

Thuật toán: Code C/C++ (tt)

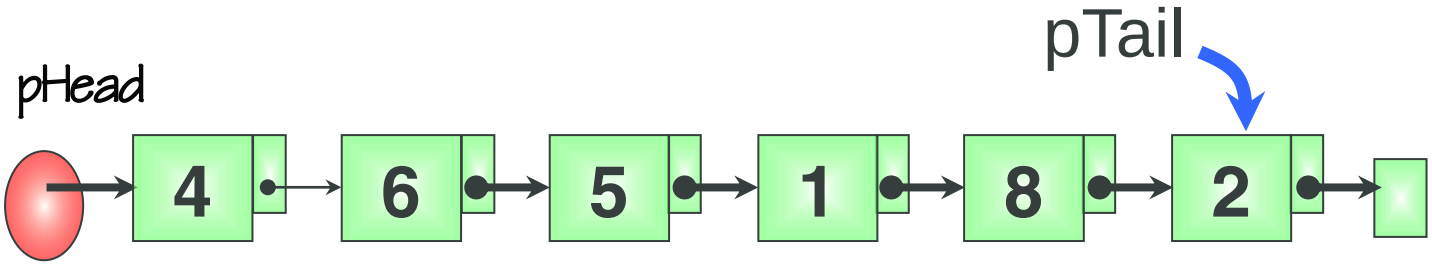
```
NODE* SeparateHead(LIST &L) {
    NODE*p = L.pHead;
    if (p == NULL) return NULL;

    L.pHead = L.pHead->pNext;
    if (L.pHead == NULL) L.pTail = NULL;

    p->pNext = NULL;
    return p;
}
```

Thuật toán sắp xếp Merge Sort

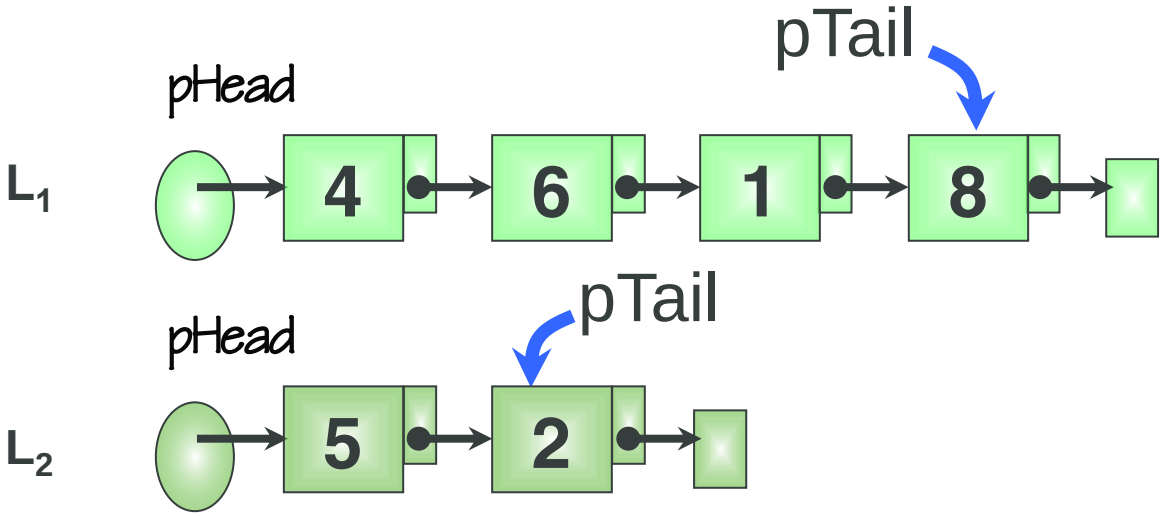
- Bước 1: Phân phối luân phiên từng đường chạy của xâu L vào 2 xâu con L_1 và L_2.
- Bước 2: Nếu L_1 != NULL thì Merge Sort (L_1).
- Bước 3: Nếu L_2 != NULL thì Merge Sort (L_2).
- Bước 4: Trộn L_1 và L_2 đã sắp xếp lại ta có xâu L đã được sắp xếp.
- Không tốn thêm không gian lưu trữ cho các dãy phụ

Minh họa thuật toán

- Cho danh sách liên kết gồm các phần tử sau:

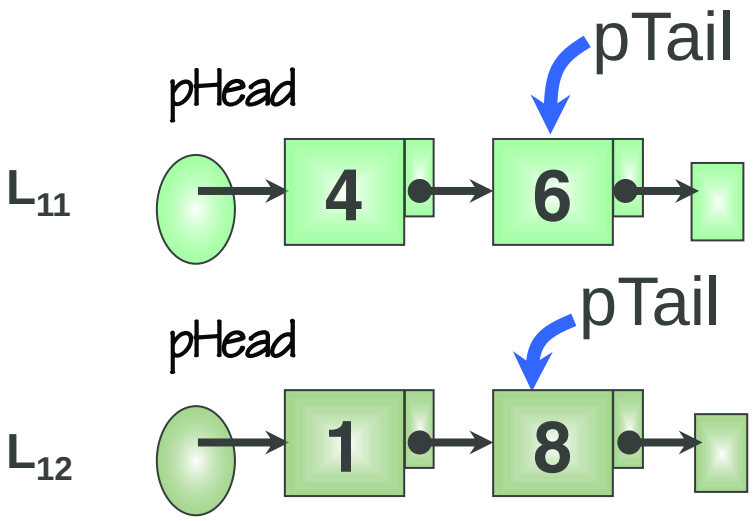


- Phân phối các đường chạy của L vào L_1, L_2

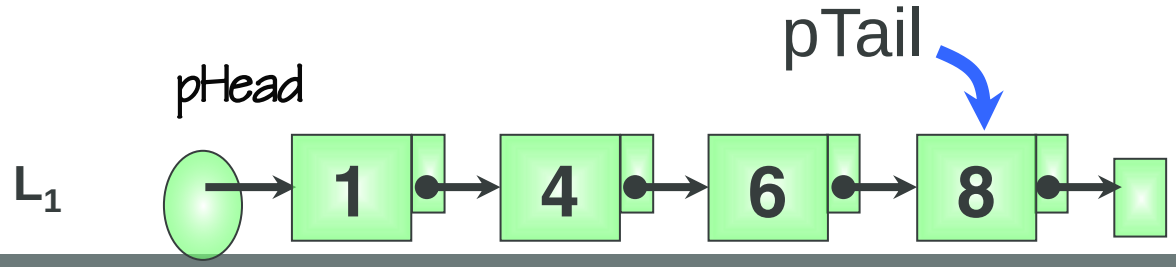


Minh họa thuật toán (tt)

- Sắp xếp L_1
  - Phân phối các đường chạy L_1 vào L_{11}, L_{12}

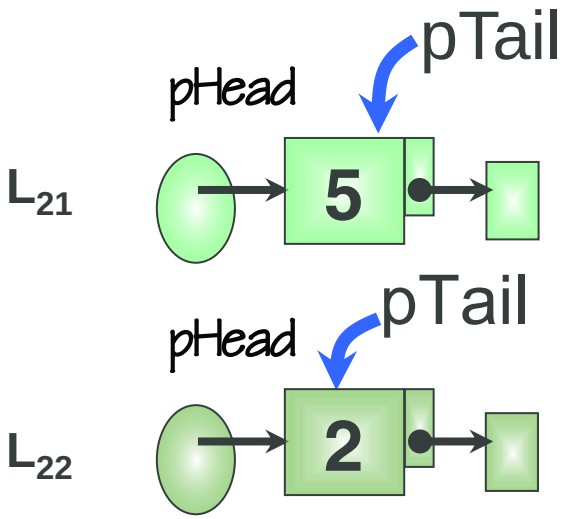


  - Trộn L_{11} và L_{12} vào L_1

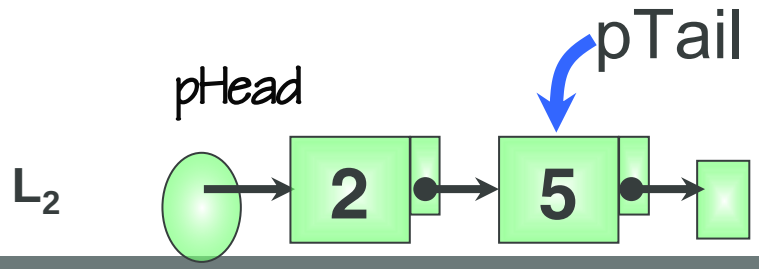


Minh họa thuật toán(tt)

- Sắp xếp L_2
  - Phân phối các đường chạy của L_2 vào L_{21}, L_{22}

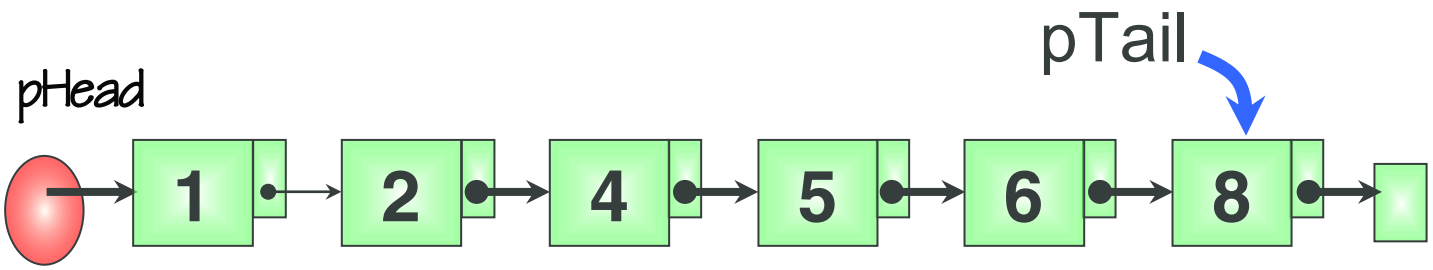


- Trộn L_{21}, L_{22} thành L_2



Minh họa thuật toán (tt)

- Trộn L_1, L_2 thành L



Merge Sort: Code C/C++

```
void MergeSort(LIST &L) {
    if (L.pHead == L.pTail) return;
    LIST L1, L2;
    MergeSplit(L, L1, L2);
    MergeSort(L1);
    MergeSort(L2);
    Merge(L, L1, L2);
}
```

Merge Sort: Code C/C++ (tt)

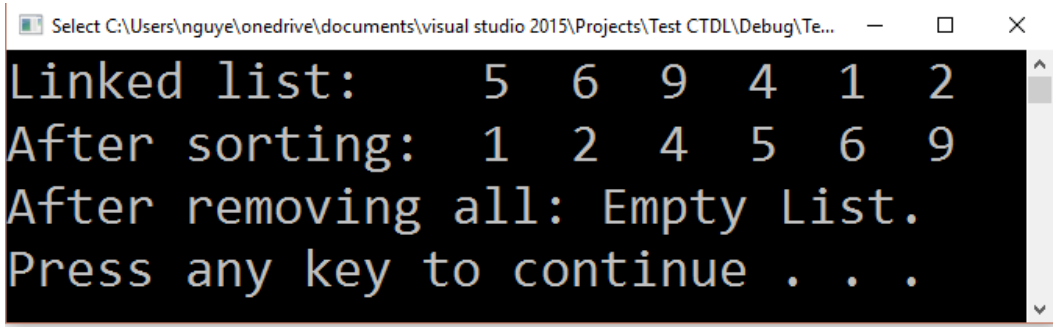
```
void MergeSplit(LIST &L, LIST &L1, LIST& L2) {
    NODE *p;
    CreateEmptyList(L1);
    CreateEmptyList(L2);
    while (L.pHead) {
        p = SeparateHead(L);
        AddTail(L1, p);
        if (L.pHead) {
                p = SeparateHead(L);
                AddTail(L2, p);
        }
    }
}
```

Merge Sort: Code C/C++ (tt)

```
void Merge(LIST &L, LIST &L1, LIST& L2) {
    NODE *p;
    CreateEmptyList(L);
    while (L1.pHead!=NULL || L2.pTail!=NULL) {
        if ( L2.pHead == NULL ||(L1.pHead != NULL && L1.pHead->info
        < L2.pHead->info) ) {
            p = SeparateHead(L1);
            AddTail(L, p);
        }
        else {
            p = SeparateHead(L2);
            AddTail(L, p);
        }
    }
}
```

Sử dụng danh sách trong hàm main?

```
int main() {
    LIST L;
    CreateEmptyList(L);

    AddTail(L, CreateNode(5));
    AddTail(L, CreateNode(6));
    AddTail(L, CreateNode(9));
    AddTail(L, CreateNode(4));
    AddTail(L, CreateNode(1));
    AddTail(L, CreateNode(2));

    cout << endl << "Linked list: "; PrintList(L);

    QuickSort(L);
    cout << endl << "After sorting: "; PrintList(L);

    RemoveList(L);
    cout << endl << "After removing all: “; PrintList(L);

    cout << endl;
    system("pause");
}
```



Cài đặt hàm main()

Yêu cầu: Viết chương trình thành lập 1 xâu đơn, trong đó thành phần dữ liệu của mỗi nút là 1 số nguyên dương.

1. Liệt kê tất thành phần dữ liệu của tất cả các nút trong xâu
2. Tìm 1 phần tử có khoá bằng x trong xâu.
3. Xoá 1 phần tử đầu xâu
4. Xoá 1 phần tử có khoá bằng x trong xâu
5. Sắp xếp xâu tăng dần theo thành phần dữ liệu (info)
6. Chèn 1 phần tử vào xâu, sao cho sau khi chèn xâu vẫn tăng dần theo trường dữ liệu

v.v..

Vài ứng dụng danh sách liên kết đơn

- Dùng xâu đơn để lưu trữ danh sách các học viên trong lớp học
- Dùng xâu đơn để quản lý danh sách nhân viên trong một công ty, trong cơ quan
- Dùng xâu đơn để quản lý danh sách các cuốn sách trong thư viện
- Dùng xâu đơn để quản lý các băng đĩa trong tiệm cho thuê đĩa.
- v..v.

Dùng xâu đơn để quản lý lớp học

- Yêu cầu: Thông tin của một sinh viên gồm, mã số sinh viên, tên sinh viên, điểm trung bình.

1. Hãy khai báo cấu trúc dữ liệu dạng danh sách liên kết để lưu danh sách sinh viên nói trên.
2. Nhập danh sách các sinh viên, và thêm từng sinh viên vào đầu danh sách (việc nhập kết thúc khi tên của một sinh viên bằng rỗng)
3. Tìm một sinh viên có trong lớp học hay không
4. Xoá một sinh viên có mã số bằng x (x nhập từ bàn phím)
5. Liệt kê thông tin của các sinh viên có điểm trung bình lớn hơn hay bằng 5.

Dùng xâu đơn để quản lý lớp học

6. Xếp loại và in ra thông tin của từng sinh viên, biết rằng cách xếp loại như sau:
   ĐTB <= 3.6: Loại yếu
   ĐTB >= 5.0 và ĐTB < 6.5: Loại trung bình
   ĐTB >= 6.5 và ĐTB < 7.0: Loại trung bình khá
   ĐTB >= 7.0 và ĐTB < 8.0: Loại khá
   ĐTB >= 8.0 và ĐTB < 9.0: Loại giỏi
   ĐTB >= 9.0: Loại xuất sắc
7. Sắp xếp và in ra danh sách sinh viên tăng theo điểm trung bình.
8. Chèn một sinh viên vào danh sách sinh viên tăng theo điểm trung bình nói trên, sao cho sau khi chèn danh sách sinh viên vẫn tăng theo điểm trung bình

..vv

Cấu trúc dữ liệu cho bài toán

- Cấu trúc dữ liệu của một sinh viên

```
typedef struct {
    char ten[40];
    char Maso[40];
    float ĐTB;
} SV;
```

- Cấu trúc dữ liệu của 1 nút trong xâu

```
typedef struct tagNode {
    SV info;
    struct tagNode *pNext;
} Node;
```

Câu hỏi và Bài tập

1. Nêu các bước để thêm một nút vào đầu, giữa và cuối danh sách liên kết đơn.

2. Nêu các bước để xóa một nút ở đầu, giữa và cuối danh sách liên kết đơn.

3. Viết thủ tục để in ra tất cả các phần tử của 1 danh sách liên kết đơn.

4. Viết chương trình thực hiện việc sắp xếp 1 danh sách liên kết đơn bao gồm các phần tử là số nguyên.

5. Viết chương trình cộng 2 đa thức được biểu diễn thông qua danh sách liên kết đơn.

Câu hỏi và Bài tập

1. Hãy nêu ưu điểm và hạn chế của CTDL tĩnh.
2. Hãy nêu ưu điểm và hạn chế của CTDL động.
3. Danh sách là gì? Cho ví dụ.
4. Hãy nêu ưu điểm và nhược điểm của danh sách liên kết ngầm và danh sách liên kết tường minh.

Chúc các em học tốt!