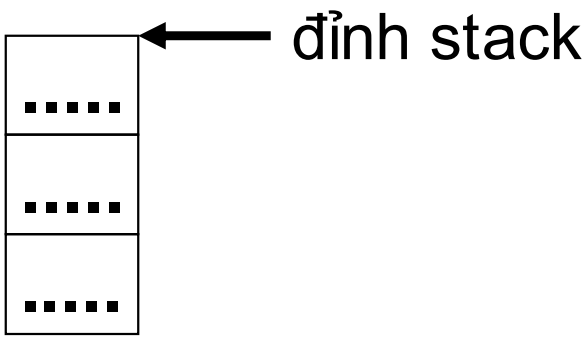
CẤU TRÚC DỮ LIỆU ĐỘNG

- ĐẶT VẤN ĐỀ
- KIỂU DỮ LIỆU CON TRỎ
- DANH SÁCH LIÊN KẾT
- DANH SÁCH ĐƠN
- **MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC**

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK



- Là cấu trúc dữ liệu cho phép lưu các phần tử chứa dữ liệu khác.
- Phần tử chứa dữ liệu được quản lý theo nguyên tắc LIFO (Last In First Out) trong đó phần tử được đưa vào stack trước sẽ được lấy ra khỏi stack sau
- Stack có đỉnh Stack luôn chỉ vào phần tử được thêm sau cùng.

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Stack có các thao tác đặc trưng sau:

- Push: thêm phần tử dữ liệu x vào stack.
- Pop: lấy đối tượng tại đỉnh ra khỏi stack.
- IsEmpty: kiểm tra stack có rỗng hay không.
- Top: lấy giá trị của phần tử tại đỉnh stack mà không hủy nó.

Stack có thể được cài đặt theo:

- Mảng
- Danh sách đơn

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Khai báo cấu trúc dữ liệu stack

```
#define MAX 100
struct Stack {
    TenDulieu data[MAX];
    int sp; //stack pointer
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Tạo Stack rỗng

```
void CreateStack(Stack &s) {
    s.sp = -1;
}
```

- Kiểm tra Stack có rỗng hay không

```
int IsEmpty(Stack &s) {
    return (s.sp == -1);
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Kiểm tra Stack đầy (do cài đặt trên mảng)

```
int IsFull(Stack &s) {
    return (s.sp >= MAX);
}
```

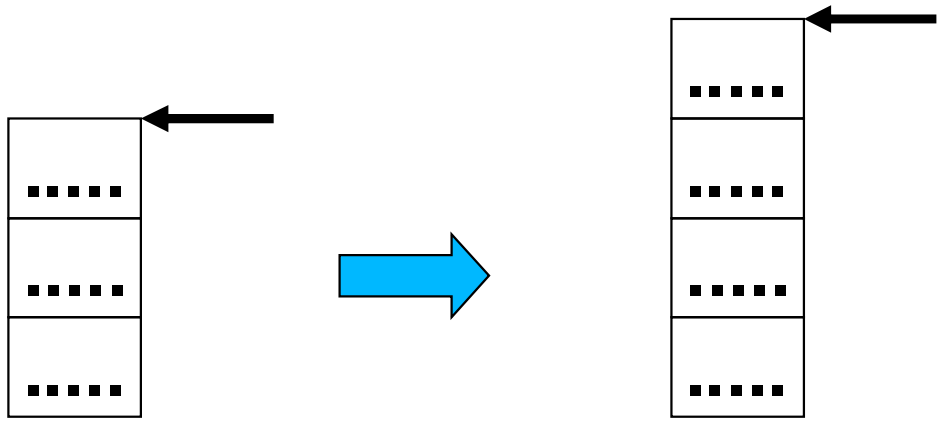
MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Đưa phần tử vào Stack

```
int Push(Stack &s, TenDulieu x) {
    if (IsFull(s))
        return 0;
    s.sp++;
    s.data[s.sp] = x;
    return 1;
}
```



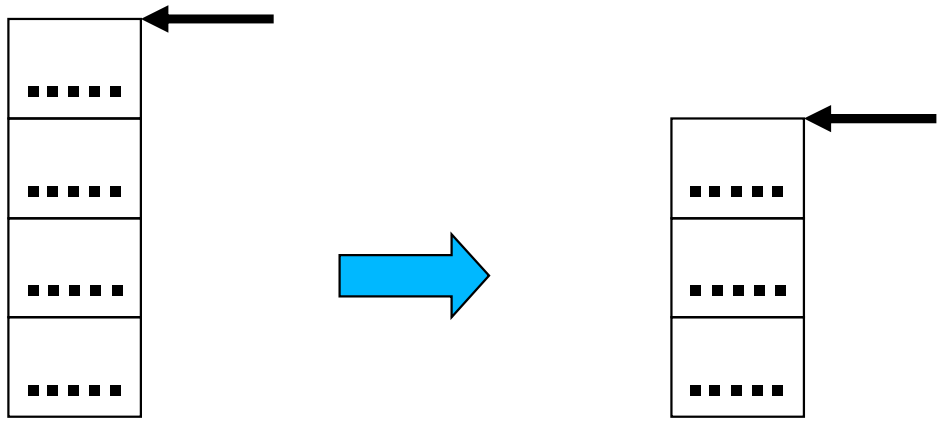
MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Lấy phần tử ra khỏi Stack

```
int Pop(Stack &s, TenDulieu &x) {
    if (IsEmpty(s))
        return 0;
    x = s.data[s.sp]
    s.sp--;
    return 1;
}
```



MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo mảng:

- Lấy giá trị phần tử ở đỉnh stack

```
int Top(Stack &s, TenDulieu &x) {
    if (IsEmpty(s)) return 0;
    x = s.data[sp];
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Ví dụ: Cho đoạn chương trình sau, cho biết kết quả trên màn hình.

```
#define MAX 30
struct Name {
    char dat[40];
};
struct NameStack {
    Name data[MAX];
    int sp;
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void CreateStack(NameStack &s) {
    s.sp = -1;
}
int IsEmpty(NameStack &s) {
    return s.sp == -1;
}
int IsFull(NameStack &s) {
    return s.sp >= MAX;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
int Push(NameStack &s, Name x) {
    if (IsFull(s))
        return 0;
    s.sp++;
    s.data[s.sp] = x;
    return 1;
}
int Pop(NameStack &s, Name &x) {
    if (IsEmpty(s))
        return 0;
    x = s.data[s.sp];
    s.sp--;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void main() {
    Name      n1 = {"mot"}, n2 = {"hai"}, n3 = {"ba"},
              n4 = {"bon"}, n;
    NameStack s;
    CreateStack(s);
    Push(s, n1);
    Push(s, n2);
    if (Pop(s, n))  cout << n.dat << endl;
    Push(s, n3);
    if (Pop(s, n))  cout << n.dat << endl;
    if (Pop(s, n))  cout << n.dat << endl;
    Push(s, n4);
    if (Pop(s, n))  cout << n.dat << endl;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn:

- Khai báo cấu trúc dữ liệu stack tương tự danh sách đơn

```
struct TenDulieu {
    // Các trường dữ liệu cần quản lý
};
struct Node {
    TenDulieu info;
    Node * pNext;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn: đỉnh stack là đầu danh sách đơn

- Khai báo cấu trúc dữ liệu stack tương tự danh sách đơn

```
struct Stack {
    Node * pHead, *pTail;
};
```

- Tạo Stack rỗng:

```
void CreateStack(Stack &s) {
    s.pHead = NULL; s.pTail = NULL;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn:

- Kiểm tra Stack rỗng

```
int IsEmpty(Stack &s) {
    return s.pHead == NULL;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn:

- Đưa phần tử dữ liệu vào stack

```
void Push(Stack &s, Node *p) {
    if (s.pHead == NULL) {
        s.pHead = p; s.pTail = p;
    } else {
        p->pNext = s.pHead; s.pHead = p;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn:

- Lấy phần tử dữ liệu ra khỏi stack

```
int Pop(Stack &s, TenDulieu &x) {
    Node *p;
    if (s.pHead == NULL)  return 0;
    p = s.pHead; s.pHead = p->pNext;
    if (s.pHead == NULL) s.pTail = NULL;
    x = p->info; delete p;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖STACK

Cài đặt stack theo danh sách đơn:

- Lấy dữ liệu của phần tử tại đỉnh stack

```
int Top(Stack &s, TenDulieu &x) {
    if (IsEmpty(s)) return 0;
    x = s.pHead->info;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

- STACK

Ví dụ: Viết hàm sắp xếp mảng theo phương pháp Quick Sort không dùng đệ quy

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
struct Range {
    int l, r;
};
struct Node {
    Range info;
    Node * pNext;
};
struct Stack {
    Node *pHead, *pTail;
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
Node * CreateNode(Range x) {
    Node *p = new Node;
    if (p != NULL) { p->info = x; p->pNext = NULL; }
    return p;
}
void CreateStack(Stack &s) {
    s.pHead = NULL; s.pTail = NULL;
}
int IsEmpty(Stack &s) {
    return s.pHead == NULL;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void Push(Stack &s, Node *p) {
    if (s.pHead == NULL) {  s.pHead = p; s.pTail = p; }
    else {  p->pNext = s.pHead; s.pHead = p; }
}
int Pop(Stack &s, Range &x) {
    Node *p;
    if (s.pHead == NULL)  return 0;
    p = s.pHead; s.pHead = p->pNext;
    if (s.pHead == NULL) s.pTail = NULL;
    x = p->info; delete p;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void QuickSort(int a[], int l, int r) {
    Stack s;
    Range range;
    Node *p;
    int k, i, j, t;
    CreateStack(s); range.l = l; range.r = r;
    p = CreateNode(range); Push(s, p);
    while (!IsEmpty(s)) {
        Pop(s, range); l = range.l; r = range.r;
        key = a[(l + r) / 2];
        i = l; j = r;
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
while (i <= j) {
    while (a[i] < key) i++;
    while (a[j] > key) j--;
    if (i <= j) {
        t = a[i]; a[i] = a[j]; a[j] = t;
        i++; j--;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
        if (l < j) {
            range.l = l; range.r = j; p = CreateNode(range);
            Push(s, p);
        }
        if (i < r) {
            range.l = i; range.r = r; p = CreateNode(range);
            Push(s, p);
        }
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

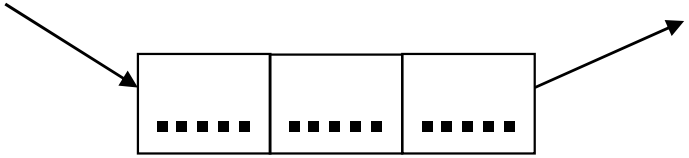
❖STACK

Ứng dụng của stack:

- Khử đệ quy đuôi
- Lưu vết các quá trình quay lui, vét cạn, tìm kiếm theo chiều sâu.

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI



- Là cấu trúc dữ liệu cho phép lưu các phần tử chứa dữ liệu khác.
- Phần tử chứa dữ liệu được quản lý theo nguyên tắc FIFO (First In First Out) trong đó phần tử được đưa vào hàng đợi trước sẽ được lấy ra trước

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi có các thao tác đặc trưng sau:

- EnQueue: thêm phần tử vào hàng đợi
- DeQueue: lấy phần tử ra khỏi hàng đợi
- IsEmpty: kiểm tra hàng đợi có rỗng không
- Front: lấy dữ liệu của phần tử đầu hàng đợi mà không hủy nó.

Hàng đợi được cài đặt theo:

- Mảng
- Danh sách đơn

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo mảng:

- Khai báo kiểu dữ liệu

```
#define MAX 100
struct TenDulieu {
    // Dữ liệu cần quản lý
};
struct Queue {
    TenDulieu data[MAX];
    int Front, Rear;
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo mảng:

- Tạo hàng đợi rỗng

```
void CreateQueue(Queue &q) {
    q.Front = -1; q.Rear = -1;
}
```

- Kiểm tra hàng đợi có rỗng hay không

```
int IsEmpty(Queue &q) {
    return q.Front == -1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo mảng:

- Thêm phần tử vào hàng đợi

```
int EnQueue(Queue &q, TenDulieu x) {
    if ((q.Front == q.Rear+1) || (q.Rear-q.Front+1 == MAX))
        return 0; // hàng đợi đã đầy
    if (q.Front == -1) q.Front = 0;
    q.Rear++;
    if (q.Rear == MAX) q.Rear = 0;
    q.data[q.Rear] = x;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo mảng:

- Lấy phần tử ra khỏi hàng đợi

```
int DeQueue(Queue &q, TenDulieu &x) {
    if (IsEmpty(q)) return 0;
    x = q.data[q.Front];
    if (q.Front == q.Rear) { q.Front = -1; q.Rear = -1; }
    else
    {    q.Front++;
         if (q.Front >= MAX) q.Front = 0; }
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo danh sách đơn:

- Khai báo cấu trúc dữ liệu

```
struct TenDulieu {
    // Dữ liệu quản lý
};
struct Node {
    TenDulieu info;
    Node *pNext;
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo danh sách đơn:

- Khai báo cấu trúc dữ liệu

```
struct Queue {
    Node *pHead, *pTail;
};
```

- Tạo hàng đợi rỗng

```
void CreateQueue(Queue &q) {
    q.pHead = NULL; q.pTail = NULL;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo danh sách đơn:

- Kiểm tra hàng đợi có rỗng hay không

```
int IsEmpty(Queue &q) {
    return q.pHead == NULL;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo danh sách đơn:

- Thêm một phần tử vào hàng đợi

```
void EnQueue(Queue &q, Node *p) {
    if (q.pHead == NULL) {
        q.pHead = p; q.pTail = p;
    } else {
        q.pTail->pNext = p; q.pTail = p;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi cài đặt theo danh sách đơn:

- Lấy một phần tử ra khỏi hàng đợi

```
int DeQueue(Queue &q, TenDulieu &x) {
    Node *p;
    if (q.pHead == NULL) return 0;
    p = q.pHead; q.pHead = p->pNext;
    if (q.pHead == NULL) q.pTail = NULL;
    x = p->info; delete p;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖QUEUE - HÀNG ĐỢI

Hàng đợi được ứng dụng:

- Tổ chức lưu vết quá trình tìm kiếm theo chiều rộng, quay lui, vét cạn
- Tổ chức quản lý và phân phối công việc

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖ DANH SÁCH LIÊN KẾT CÓ THỨ TỰ

Danh sách liên kết có thứ tự (Ordered List) là danh sách mà các phần tử của nó phải đảm bảo một thứ tự nào đó. Vì vậy, việc thêm phần tử cần phải xét đến thứ tự của danh sách.

Đối với danh sách liên kết có thứ tự được cài đặt theo danh sách đơn, có hai thao tác cần được hiệu chỉnh là:

- Thêm phần tử vào danh sách
- Tìm kiếm phần tử trong danh sách

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖DANH SÁCH LIÊN KẾT CÓ THỨ TỰ

- Thêm phần tử vào danh sách

```
int Compare(TenDulieu x, TenDulieu y);
// trả về -1 nếu x < y, 0 nếu x = y, 1 nếu x > y
void Add(TenDS &l, Node *p) {
    Node *q = NULL, *h = l.pHead;
    while (h) {
        if (Compare(h->info, p->info) >= 0) break;
        q = h; h = h->pNext;
    }
    AddAfter(l, q, p);
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖DANH SÁCH LIÊN KẾT CÓ THỨ TỰ

- Tìm kiếm phần tử trong danh sách

```
Node * Search(TenDS &l, TenDulieu x) {
    Node *p = l.pHead;
    while (p) {
        if (Compare(p->info, x) == 0) break;
        else if (Compare(p->info, x) > 0) {p = NULL; break;}
        p = p->pNext;
    }
    return p;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

❖DANH SÁCH LIÊN KẾT CÓ THỨ TỰ

- Ví dụ: Viết chương trình nhập vào danh sách hình tròn với thông tin tọa độ tâm (x, y) và bán kính r đến khi nhập r <= 0. In ra danh sách thứ tự ưu tiên các đường tròn có bán kính nhỏ.

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
struct Circle {
    double x, y, r;
};
struct CircleNode {
    Circle info;
    CircleNode *pNext;
};
struct CircleList {
    CircleNode *pHead, *pTail;
};
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void CreateList(CircleList &l) {
    l.pHead = NULL; l.pTail = NULL;
}
CircleNode* CreateNode(Circle x) {
    CircleNode *p = new CircleNode;
    if (p != NULL) {
        p->info = x;
        p->pNext = NULL;
    }
    return p;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void AddFirst(CircleList &l, CircleNode *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    } else {
        p->pNext = l.pHead; l.pHead = p;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void AddAfter(CircleList &l, CircleNode *p, CircleNode *q) {
    if (q != NULL) {
        p->pNext = q->pNext; q->pNext = p;
        if (l.pTail == q) l.pTail = p;
    }
    else
        AddFirst(l, p);
}
int Compare(Circle x, Circle y) {
    if (x.r == y.r) return 0;
    if (x.r < y.r) return -1;
    return 1;
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void Add(CircleList &l, CircleNode *p) {
    CircleNode *q = NULL, *h = l.pHead;
    while (h) {
        if (Compare(h->info, p->info) >= 0) break;
        q = h; h = h->pNext;
    }
    AddAfter(l, p, q);
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
void Print(Circle x) {
    cout << '(' << x.x << ", " << x.y << "), " << x.r << ' ';
}
void PrintList(CircleList l) {
    CircleNode *p = l.pHead;
    while (p != NULL) {
        Print(p->info);
        p = p->pNext;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

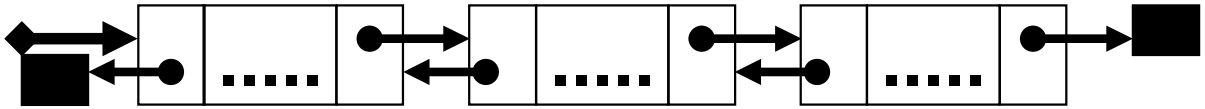
```
void InputList(CircleList &l) {
    CircleNode *p;
    Circle x;
    cin >> x.x >> x.y >> x.r;
    while (x.r > 0) {
        p = CreateNode(x);
        if (p == NULL)
            return;
        Add(l, p);
        cin >> x.x >> x.y >> x.r;
    }
}
```

MỘT SỐ DẠNG DANH SÁCH LIÊN KẾT KHÁC

```
int main() {
    CircleList list;
    CreateList(list);
    InputList(list);
    PrintList(list);
    return 0;
}
```

DANH SÁCH KÉP

❖TỔ CHỨC

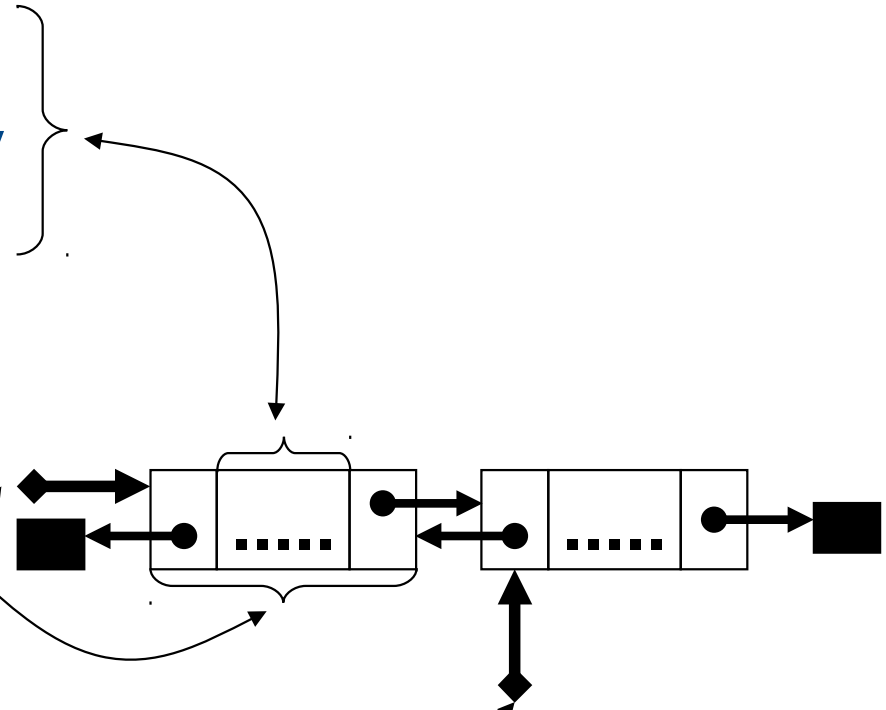


- Mỗi phần tử chứa liên kết đến phần tử đứng liền trước và sau nó
- Mỗi phần tử là một cấu trúc gồm 3 thành phần:
  - Thành phần dữ liệu: chứa thông tin cần quản lý
  - Hai thành phần liên kết: chứa địa chỉ của phần tử liền trước và sau nó, hoặc chứa giá trị NULL.

DANH SÁCH KÉP

❖TỔ CHỨC

```
struct TenDulieu {
    ... // Thông tin cần quản lý
};
struct Node {
    TenDulieu info;
    Node * pNext, * pPrev;
};
struct TenDS {
    Node *pHead, *pTail;
};
```



DANH SÁCH KÉP

❖TỔ CHỨC

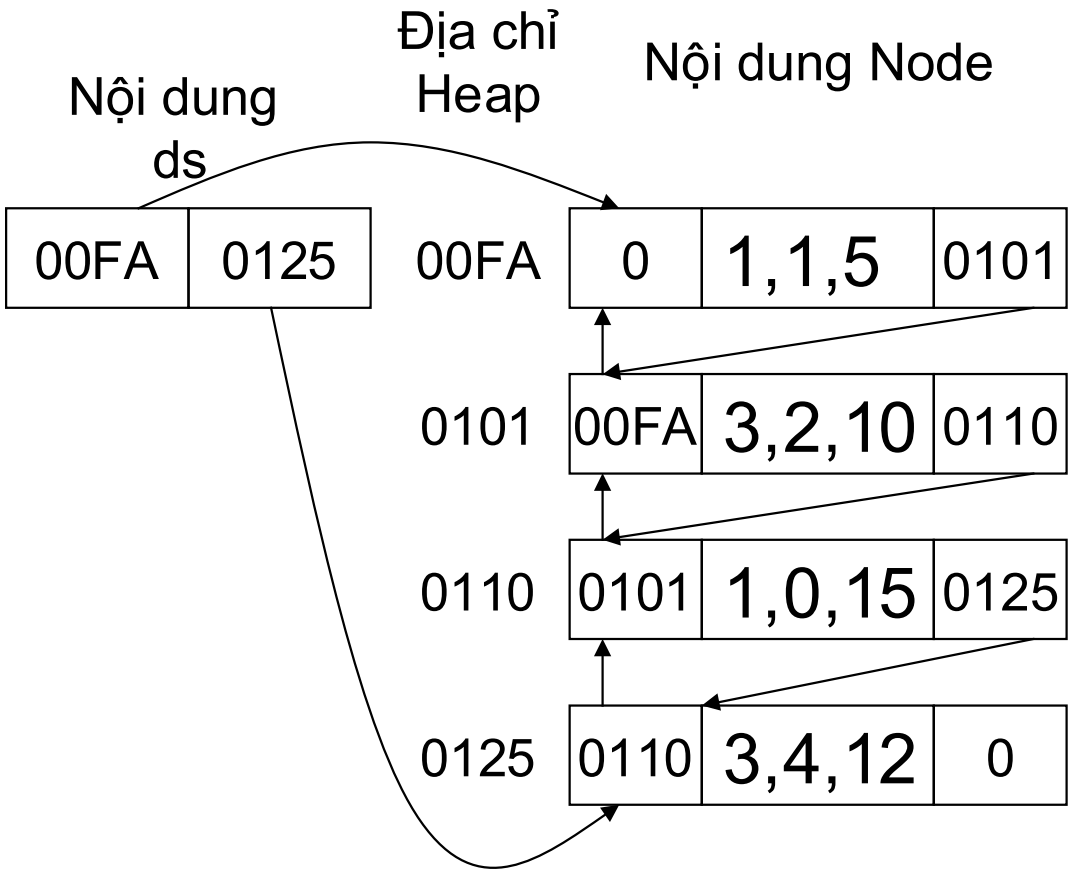
Ví dụ: Tổ chức dữ liệu cho một danh sách các hình tròn.

```
struct  HinhTron{
    double x, y, r;
};
struct NodeHinhTron {
    HinhTron info;
    NodeHinhTron *pNext, *pPrev;
};
```

DANH SÁCH KÉP

```
struct  DSHinhTron{
    NodeHinhTron *pHead,
    *pTail;
};
```

Giả sử có biến cấp phát tĩnh ds có kiểu DSHinhTron lưu trữ danh sách 4 hình tròn. Hình ảnh của ds như sau:



DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- Tạo danh sách rỗng
- Tạo một nút có trường info bằng x
- Thêm phần tử vào danh sách
- Duyệt danh sách
- Hủy phần tử trong danh sách
- Hủy danh sách
- Sắp xếp danh sách

Lưu ý: Các thao tác được thực hiện tương tự như danh sách đơn, cần duy trì liên kết với phần tử trước

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Tạo danh sách đơn rỗng**

  Danh sách rỗng có pHead và pTail trỏ đến NULL

```
void CreateList(TenDS &p) {
   p.pHead = NULL; p.pTail = NULL;
}
```

Ví dụ

```
void CreateDSHinhTron(DSHinhTron &p) {
   p.pHead = NULL; p.pTail = NULL;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

  Tạo nút bằng cách cấp phát động một biến có kiểu Node, sau đó gán giá trị x cho trường info. Lúc này, nút vừa tạo chưa thuộc danh sách nên mặc định pNext và pPrev mang giá trị NULL.

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

```
Node* CreateNode(TenDuLieu x) {
    Node *p = new Node; // cấp phát vùng nhớ
    if (p != NULL) { // kiểm tra kết quả cấp phát
        p->info = x;
        p->pPrev = NULL; p->pNext = NULL;
    }
    return p;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Tạo một nút có trường info bằng x**

Ví dụ

```
NodeHinhTron* CreateDSHinhTron(HinhTron x) {
    NodeHinhTron *p = new NodeHinhTron;
    if (p != NULL) {
        p->info = x;
        p->pPrev = NULL; p->pNext = NULL;
    }
    return p;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**

  Xét việc thêm phần tử vào danh sách theo các trường hợp sau:

  - Thêm phần tử vào đầu danh sách
  - Thêm phần tử vào cuối danh sách
  - Thêm phần tử vào ngay sau phần tử q trong danh sách.
  - Thêm phần tử vào ngay trước phần tử q trong danh sách.

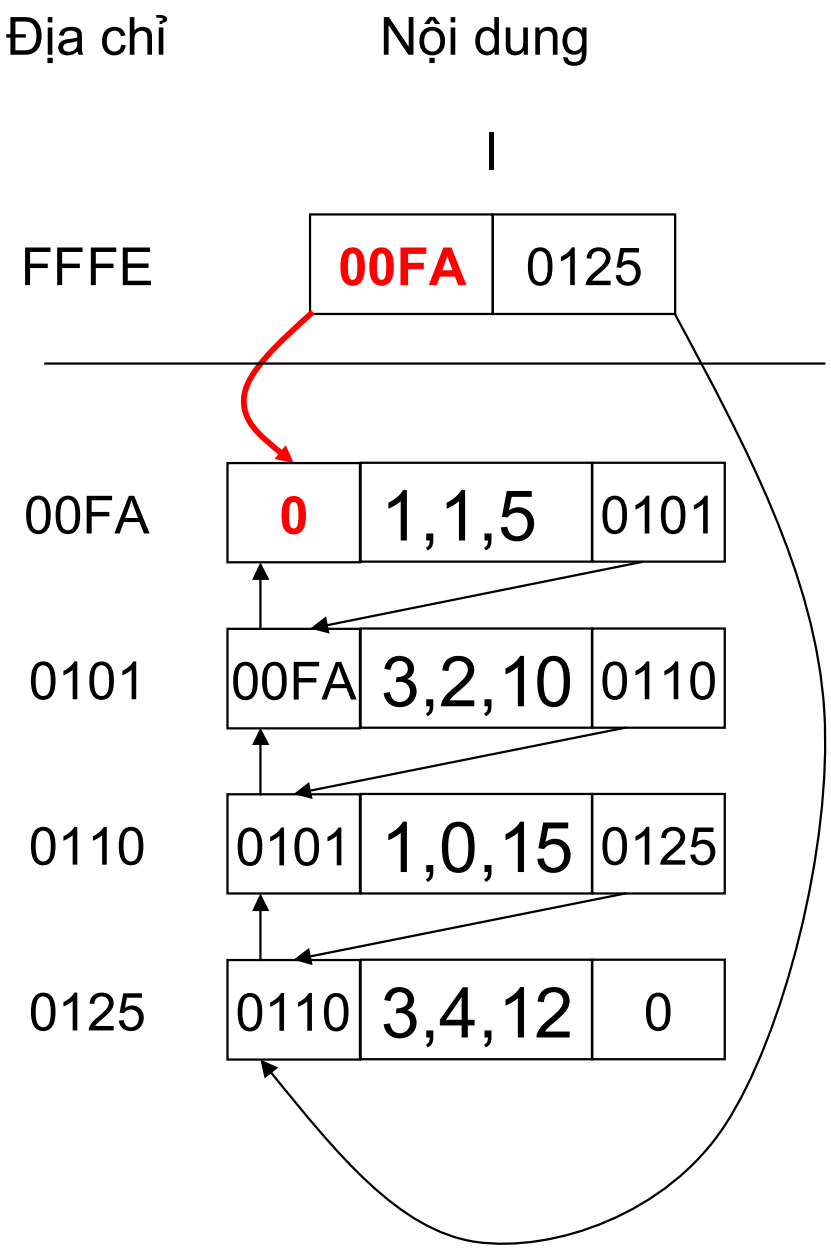
DANH SÁCH KÉP

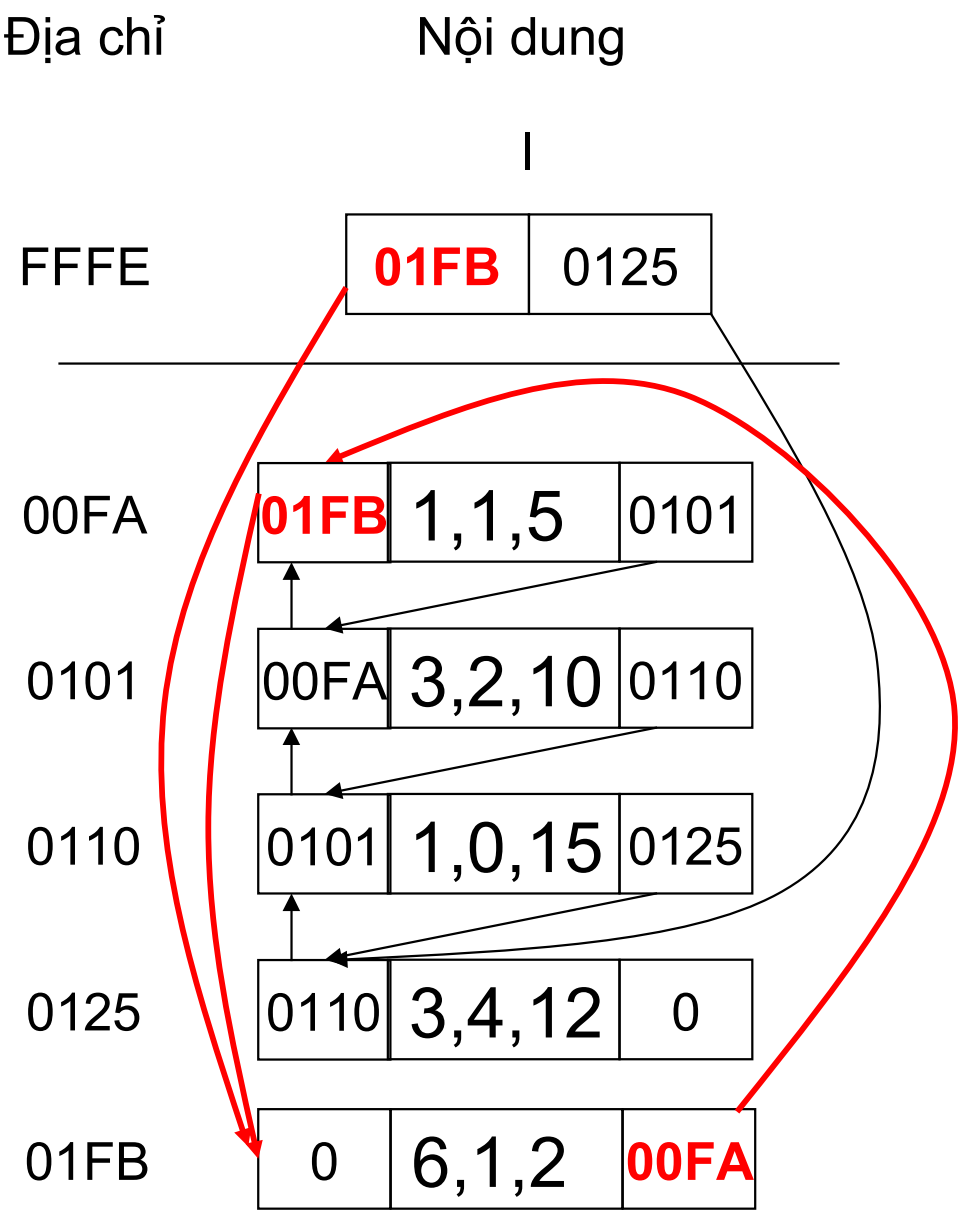
❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
- *Thêm vào đầu danh sách*

```
void AddFirst(TenDS &l, Node *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    }
    else {
        p->pNext = l.pHead; l.pHead->pPrev=p; l.pHead = p;
    }
}
```

DANH SÁCH KÉP





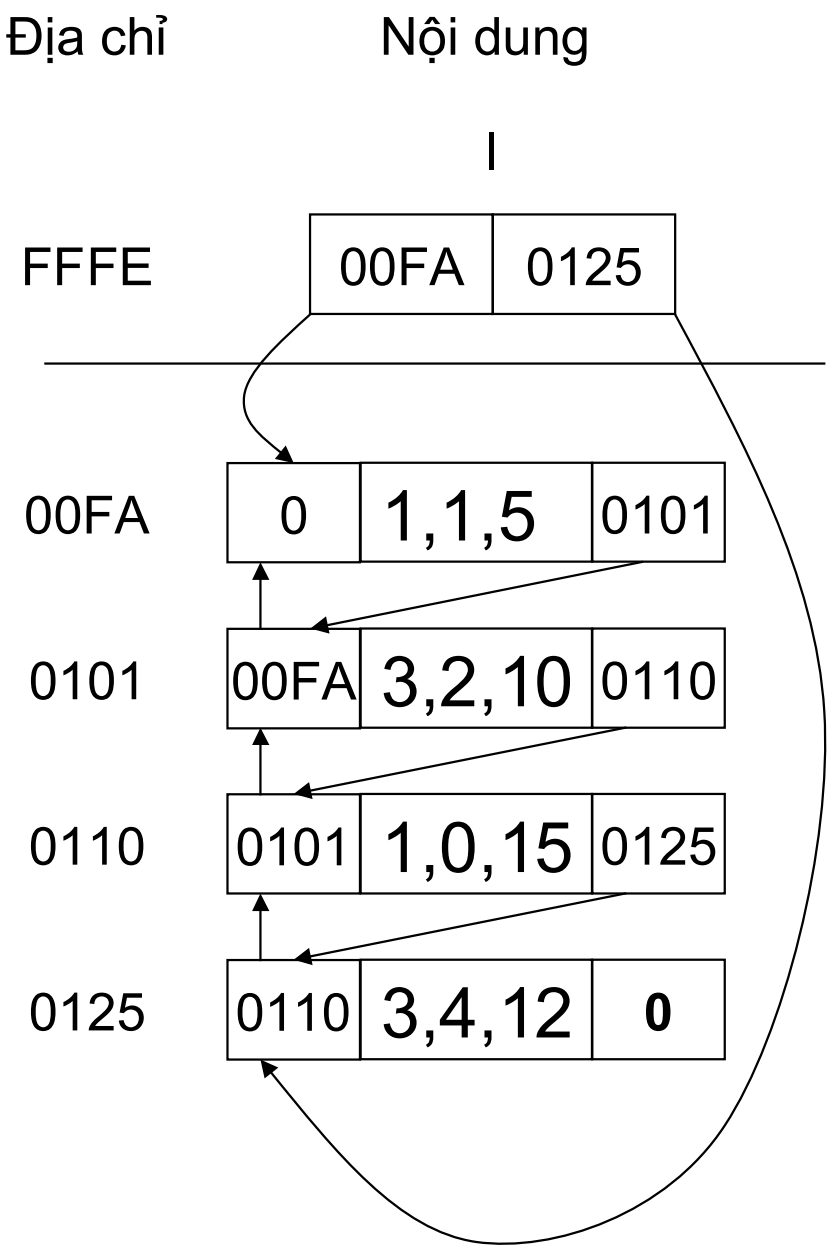
DANH SÁCH KÉP

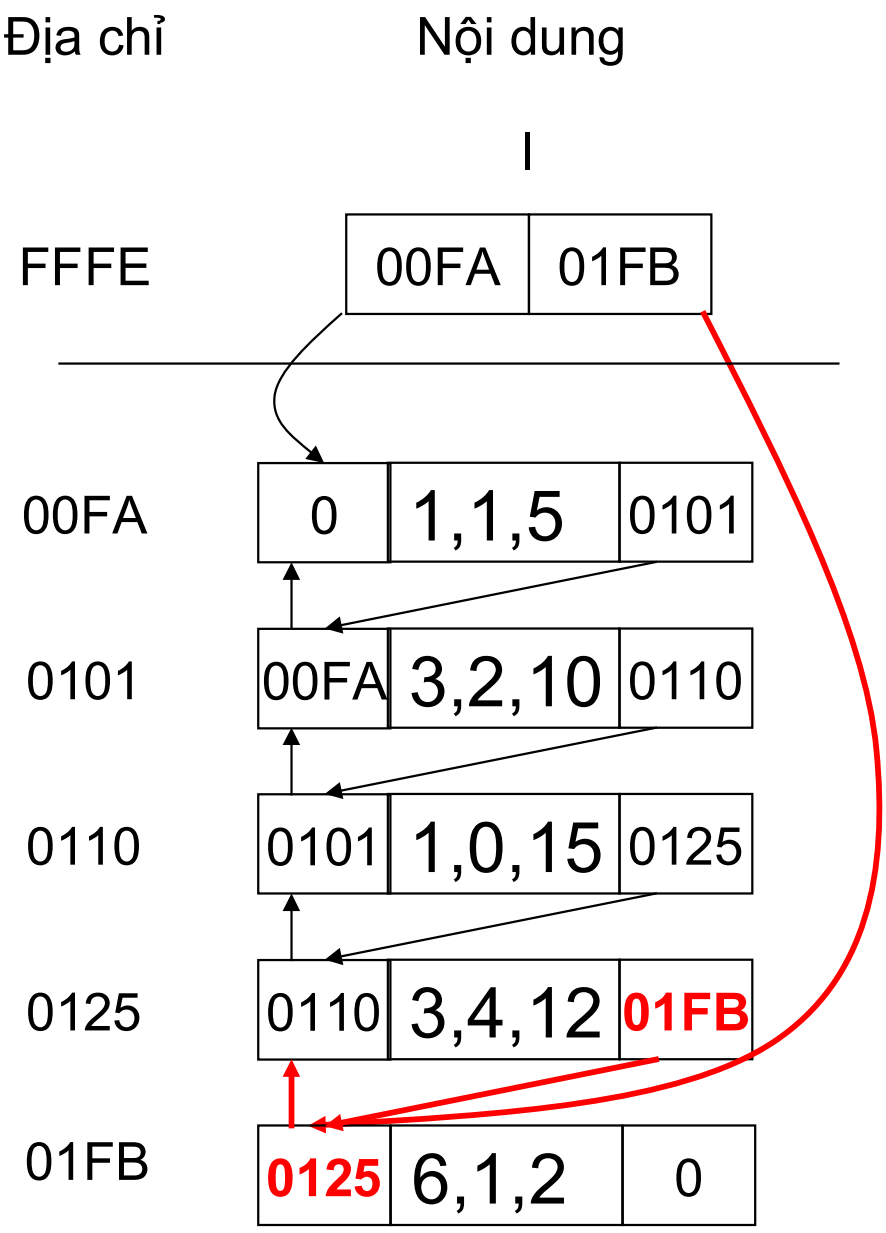
❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
- *Thêm vào cuối danh sách*

```
void AddLast(TenDS &l, Node *p) {
    if (l.pHead == NULL) {
        l.pHead = p; l.pTail = p;
    }
    else {
        l.pTail->pNext = p; p->pPrev = l.pTail; l.pTail = p;
    }
}
```

DANH SÁCH KÉP





DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào sau phần tử q trong danh sách*

```
void AddAfter(TenDS &l, Node *p, Node *q) {
    if (q != NULL) {
        p->pNext = q->pNext;
        if (q->pNext != NULL) q->pNext->pPrev = p;
        q->pNext = p; p->pPrev = q;
        if (l.pTail == q) l.pTail = p;
    } else
        AddFirst(l, p);
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Thêm phần tử vào danh sách**
  - *Thêm vào trước phần tử q trong danh sách*

```
void AddBefore(TenDS &l, Node *p, Node *q) {
    if (q != NULL) {
        p->pPrev = q->pPrev;
        if (q->pPrev != NULL) q->pPrev->pNext = p;
        q->pPrev = p; p->pNext = q;
        if (l.pHead == q) l.pHead = p;
    } else
        AddLast(l, p);
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách**

  Được thực hiện tuần tự từ phần tử đầu danh sách đến phần tử cuối danh sách. Duyệt danh sách nhằm mục đích đếm số phần tử, tìm phần tử thỏa điều kiện.

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách**

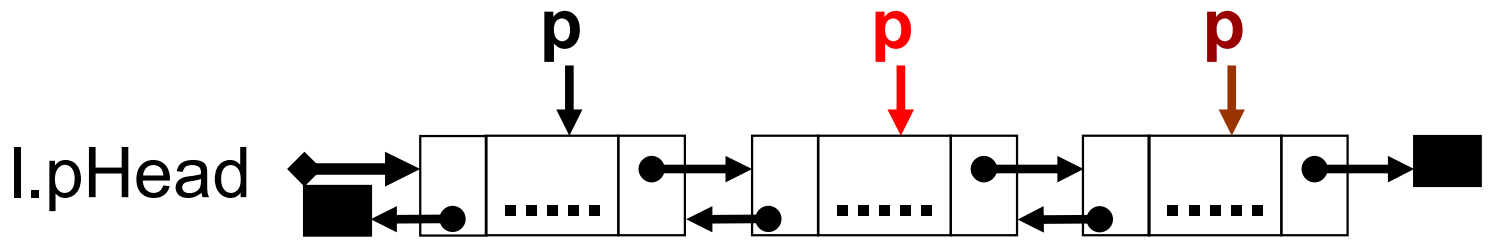
Nguyên tắc: Để duyệt danh sách l

B1) p ← l.pHead

B2) Nếu p = NULL qua B4

B3) Xử lý cho phần tử p, p ← p->pNext, qua B2.

B4) Kết thúc.



Lưu ý: Có thể duyệt từ phần tử cuối theo pPrev

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Duyệt danh sách: Tìm phần tử có trường info bằng x**

```
int Equal(TenDuLieu x, TenDuLieu y); // hàm so sánh
Node * Search(TenDS l, TenDuLieu x) {
    Node *p = l.pHead;
    while ((p != NULL) && (!Equal(p->info, x))
        p = p->pNext;
    return p;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:** Xét các trường hợp sau:
  - Hủy phần tử đầu danh sách
  - Hủy phần tử cuối danh sách
  - Hủy phần tử ngay sau phần tử q trong danh sách
  - Hủy phần tử ngay trước phần tử q trong danh sách
  - Hủy phần tử có khóa x

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử đầu danh sách*

```
int RemoveFirst(TenDS &l, TenDulieu &x) {
    Node *p = l.pHead;  int r = 0;
    if (l.pHead != NULL) {
        x = p->info; l.pHead = p->pNext; delete p; r = 1;
        if (l.pHead == NULL) l.pTail = NULL;
        else l.pHead->pPrev = NULL;
    }
    return r;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử cuối danh sách*

```
int RemoveLast(TenDS &l, TenDulieu &x) {
    Node *p = l.pTail;     int r = 0;
    if (l.pTail != NULL) {
        x = p->info; l.pTail = p->pPrev; delete p; r = 1;
        if (l.pTail == NULL) l.pHead = NULL;
        else l.pTail->pNext = NULL;
    }
    return r;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**

- *Hủy phần tử ngay sau phần tử q trong danh sách*

```
int RemoveAfter(TenDS &l, Node *q, TenDulieu &x) {
    Node *p;
    if (q != NULL) { p = q->pNext;
        if (p != NULL) {
            q->pNext = p->pNext;
            if (p==l.pTail) l.pTail=q; else p->pNext->pPrev=q;
            x = p->info; delete p;
    } return 1;} else return RemoveFirst(l, x);
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**

- *Hủy phần tử ngay trước phần tử q trong danh sách*

```
int RemoveBefore(TenDS &l, Node *q, TenDulieu &x) {
    Node *p;
    if (q != NULL) { p = q->pPrev;
        if (p != NULL) {
            q->pPrev = p->pPrev;
            if(p==l.pHead) l.pHead=q; else p->pPrev->pNext=q;
            x = p->info; delete p;
    } return 1;} return RemoveLast(l, x);
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy một phần tử trong danh sách:**
- *Hủy phần tử có khóa x*

```
int Remove(TenDS &l, TenDulieu &x) {
    Node *p = l.pHead, *q = NULL; int r = 0;
    while ((p != NULL) && (!Equal(p->info, x)))
    { q = p; p = p->pNext; }
    if (p != NULL)
        if (q == NULL) r = RemoveFirst(l,x);
        else r = RemoveAfter(l, q, x);
    return r; }
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Hủy danh sách:**

```
void RemoveList(TenDS &l) {
    Node *p;
    while (l.pHead != NULL)  {
        p = l.pHead; l.pHead = p->pNext; delete p;
    }
    l.pTail = NULL;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách:** Danh sách có thể được sắp xếp theo hai cách
- Hoán đổi thành phần info của các phần tử trong danh sách
- Thiết lập lại liên kết giữa các phần tử trong danh sách

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- *Quick Sort*

```
int Compare(TenDulieu x, TenDulieu y);
// so sánh khóa: -1 nếu x < y, 0 nếu x = y, 1 nếu x > y
void QuickSort(TenDS &l) {
    Node *p, *X;
    TenDS l1, l2;
    if (l.pHead == l.pTail) return;
    CreateList(l1); CreateList(l2);
    X = l.pHead; l.pHead = X->pNext;
```

DANH SÁCH KÉP

```
while (l.pHead != NULL) {
    p = l.pHead; l.pHead = p->pNext;
    p->pNext = NULL; p->pPrev = NULL;
    if (Compare(p->info, X->info) <= 0)
        AddLast(l1, p);
    else
        AddLast(l2, p);
}
l.pTail = NULL;
QuickSort(l1); QuickSort(l2);
```

DANH SÁCH KÉP

```
    if (l1.pHead != NULL) {
        l.pHead = l1.pHead; l1.pTail->pNext = X;
        X->pPrev = l1.pTail;
    } else
        l.pHead = X;
    X->pNext = l2.pHead;
    if (l2.pHead != NULL)
        { l.pTail = l2.pTail; l2.pHead->pPrev = X; }
    else
        l.pTail = X;
}
```

DANH SÁCH KÉP

❖CÁC THAO TÁC CƠ BẢN

- **Sắp xếp danh sách**
- *Merge Sort* (Sinh viên tự tìm hiểu)
- *Radix Sort* (Sinh viên tự tìm hiểu)