Trường Đại Học Bách Khoa Hà Nội
Viện Điện Tử - Viễn Thông
Bộ môn Điện tử - Kỹ thuật máy tính

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

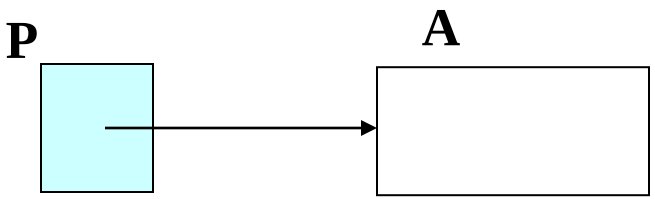
Cấu trúc Tuyến Tính
Phần 3: Danh sách móc nối

Nội dung

- Giới thiệu
  - Con trỏ và cấu trúc lưu trữ móc nối
  - Mô tả danh sách móc nối
  - Các loại danh sách móc nối
    - Danh sách nối đơn
      - Danh sách nối đơn thẳng
      - Danh sách nối đơn vòng
    - Danh sách nối kép
      - Danh sách nối kép thẳng
      - Danh sách nối kép vòng
- Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối
  - LIFO
  - FIFO

Giới thiệu chung

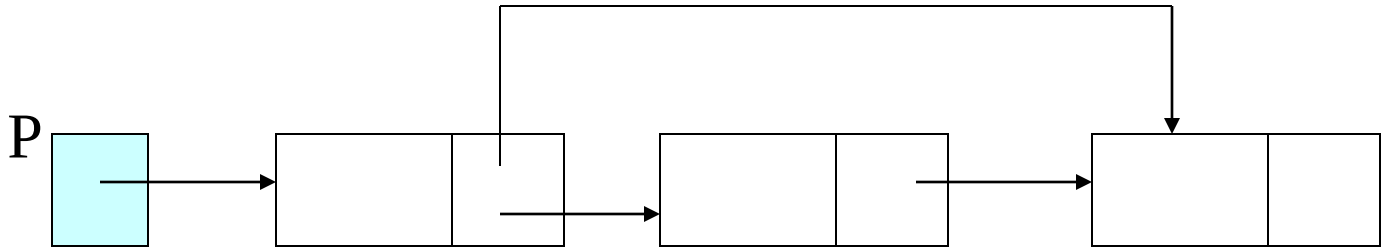
- Con trỏ & Cấu trúc lưu trữ móc nối
  - Con trỏ (pointer): là một kiểu dữ liệu (datatype) mà giá trị của nó chỉ dùng để chỉ đến một giá trị khác chứa trong bộ nhớ.



  - Các thao tác cơ bản
    - Khởi tạo (khai báo): ***int * P;***
    - Lấy địa chỉ 1 đối tượng ***int A; P = &A;***
    - Truy nhập vào đối tượng được trỏ: ****P = 20;***
    - Cấp phát bộ nhớ động cho đối tượng DL động: ***P = new int;***
    - Giải phóng đối tượng DL động: ***delete P;***

Giới thiệu chung

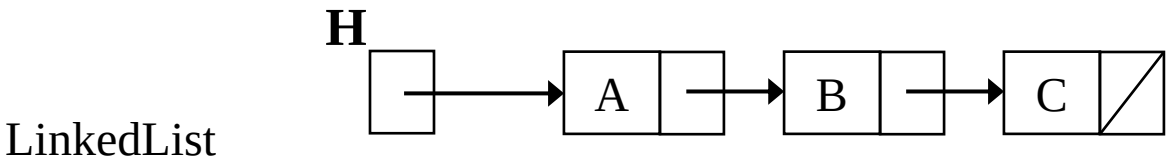
- Con trỏ & Cấu trúc lưu trữ móc nối
  - Cấu trúc lưu trữ móc nối – Cấu tạo



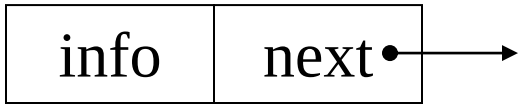
    - Con trỏ: trỏ đến các nút
    - Các nút: chứa thông tin về các phần tử và có thể cả con trỏ
  - Đặc điểm:
    - Cấu trúc lưu trữ động: cấp phát bộ nhớ trong khi chạy (run-time)
    - Linh hoạt trong tổ chức cấu trúc: các con trỏ trong cấu trúc có thể tùy ý thay đổi địa chỉ chỉ đến
    - Có ít nhất một điểm truy nhập: dùng con trỏ P

Giới thiệu chung

- Mô tả danh sách móc nối - Linked List
  - Tổ chức CTLT: gồm hai thành phần
    - Các nút (node): lưu trữ các phần tử của danh sách và các con trỏ để liên kết đến các phần tử khác.
    - Các con trỏ (pointer): biểu diễn các quan hệ trước sau giữa các phần tử. Có ít nhất một con trỏ đóng vai trò điểm truy nhập.



  - Cấu trúc 1 nút gồm hai phần:
    - Phần dữ liệu
    - Phần con trỏ chỉ đến nút tiếp theo



  - H: con trỏ trỏ vào nút đầu của danh sách
  - Thứ tự và độ dài danh sách có thể thay đổi tuỳ thích

Giới thiệu chung

- Cài đặt danh sách bằng CTLT móc nối
  - Khai báo cấu trúc: cấu trúc **tự móc nối**

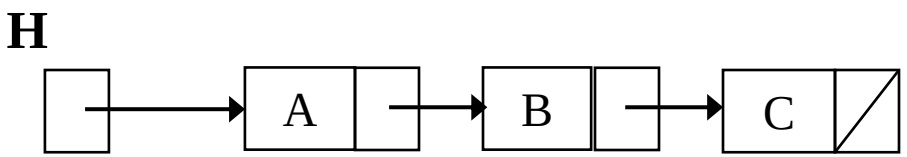
```
struct  Node {
   Type info;
   Node* next;
};
typedef Node*  LinkedList;
```

  - Nút trống (null, nil): khi con trỏ không chỉ đến đâu:

```
next = NULL
```

  - Danh sách rỗng:

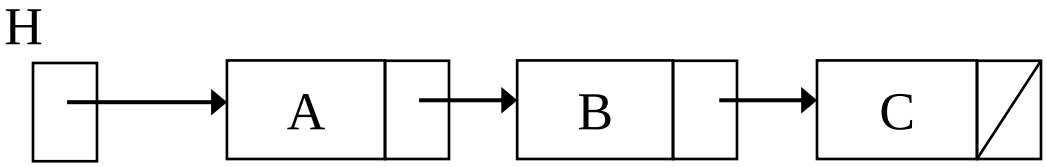
```
LinkedList H;
H = NULL;
```



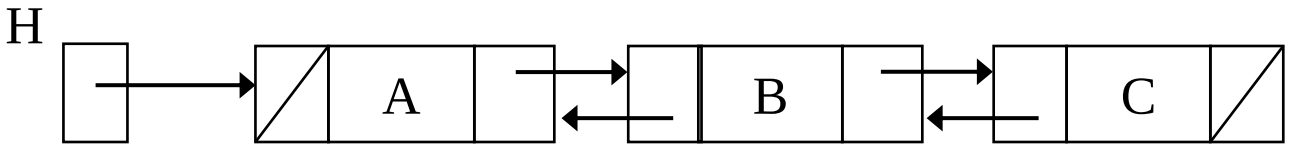
  - Danh sách đầy: khi không còng đủ bộ nhớ để cấp phát

Giới thiệu chung

- Phân loại danh sách móc nối
  Phân loại theo hướng con trỏ (hay số con trỏ trong 1 nút)
  - Danh sách nối đơn (*single linked list*):
    - con trỏ luôn chỉ theo một hướng trong danh sách

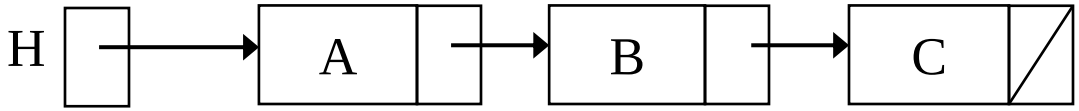


  - Danh sách nối kép (*double linked list*)
    - 2 con trỏ chỉ theo hai hướng trong danh sách

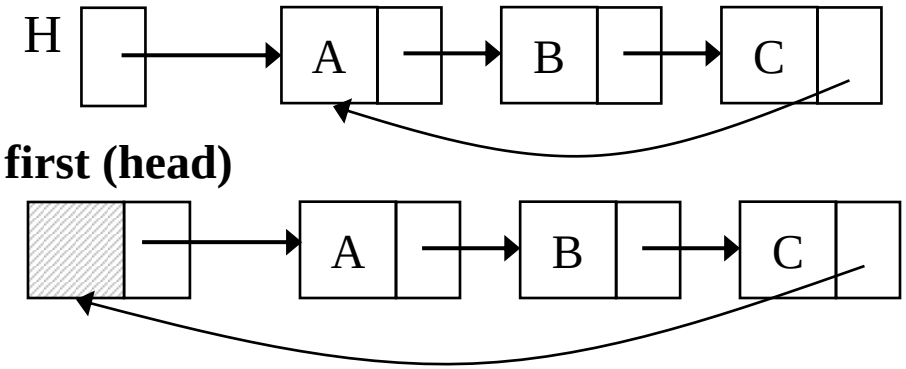


Giới thiệu chung

- Phân loại danh sách móc nối
  Phân loại theo cách móc nối vòng hoặc thẳng
  - Danh sách nối thẳng: truy cập vào danh sách thông qua điểm truy nhập H

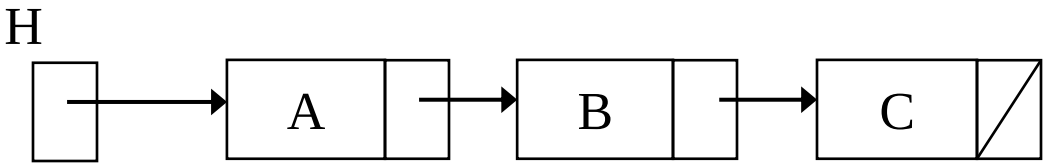


  - Danh sách nối vòng (*circularly linked list*): bất cứ nút nào trong danh sách cũng có thể coi là nút đầu hay nút cơ sở (mọi nút có vai trò như nhau)



Cài đặt danh sách nối đơn

- Mô tả
  - Danh sách nối đơn thẳng (*single linked list*):
    - Dùng 1 con trỏ luôn chỉ theo một hướng trong danh sách
    - Phần tử (nút) cuối của danh sách có con trỏ NULL
    - Các nút sắp xếp tuần tự trong danh sách




```
struct  Node {
    Type info;
    Node* next;
};
typedef Node* PNode;            //Kiểu con trỏ nút
typedef Node*  LinkedList; //Kiểu danh sách nối đơn
```

Cài đặt danh sách nối đơn

- Danh sách nối đơn thẳng
  - Các thao tác cơ bản

    - Khởi tạo danh sách: tạo ra một danh sách rỗng
    - Kiểm tra trạng thái hiện tại của DS:
      - Rỗng (Empty): khi con trỏ H = NULL
    - Phép xen một phần tử mới vào danh sách
      - Xen phần tử mới vào trước phần tử hiện tại Q: ***InsertAfter***
      - Xen phần tử mới vào sau phần tử hiện tại Q: ***InsertBefore***
    - Phép xoá phần tử khỏi danh sách: ***Delete***
    - Phép tìm kiếm phần tử có dữ liệu = x: ***Search***
    - Phép duyệt danh sách: ***Traverse***

Cài đặt danh sách nối đơn

- Khởi tạo danh sách:

```
void InitList (LinkedList & H) {
    H = NULL;
}
```

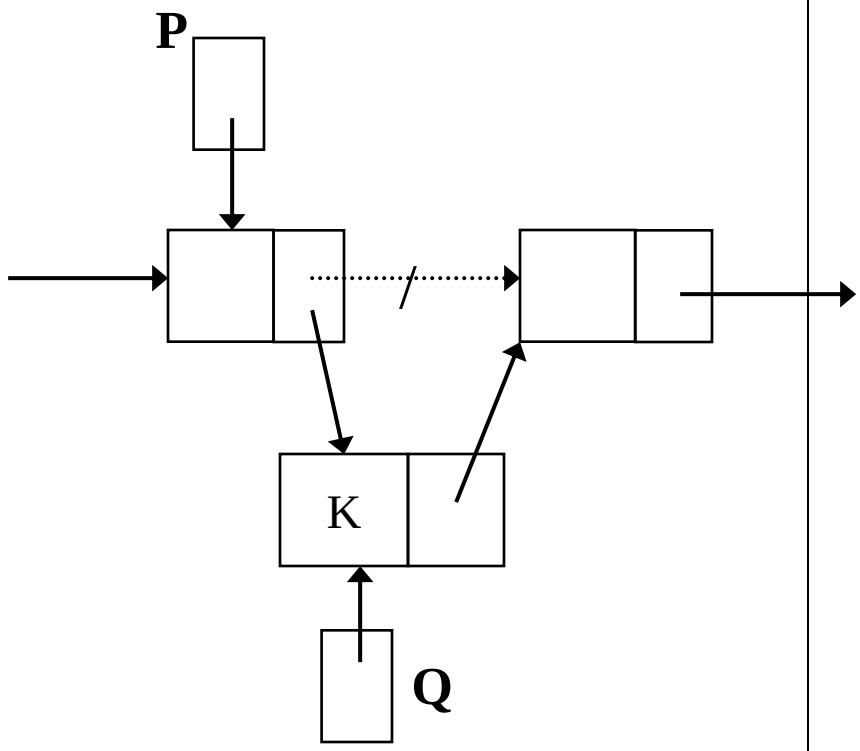
- Kiểm tra trạng thái:

```
bool IsEmpty (LinkedList  H) {
    return (H == NULL);
}
```

Danh sách nối đơn

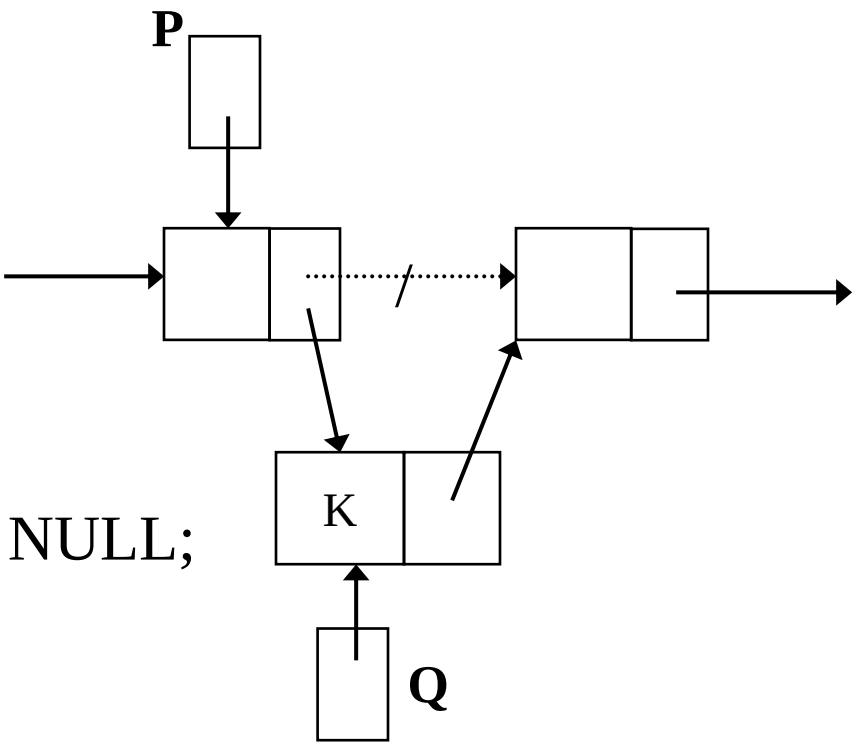
- Thao tác bổ sung một phần tử mới K vào sau phần tử hiện tại được trỏ bởi P trong d/s H. Thao tác này sau đó trả về con trỏ trỏ vào nút vừa bổ sung

```
PNode InsertAfter(LinkedList & H, PNode P, Type K){
    Bổ sung một nút Q để chứa K
    Nếu d/s H rỗng (H=P=NULL) thì:
        Q->next = NULL;
        H=Q;
    Trái lại, thì:
        Q->next=P->next;
        P->next = Q;
    return Q;
}
```



Cài đặt hàm bổ sung sau

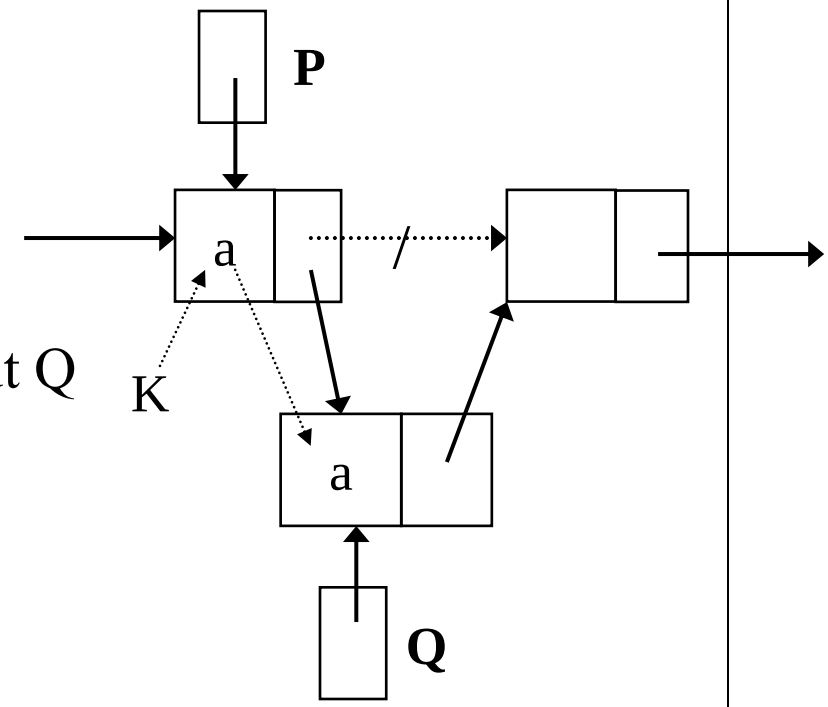
```
1.  PNode InsertAfter(LinkedList & H, PNode P, Type K){
2.      PNode Q = new Node;
3.      Q->info = K;
4.      if (H==NULL){
5.          H = Q;
6.          Q->next = NULL;
7.
8.      }else {
9.          if (P==NULL) return NULL;
10.         Q->next = P->next;
11.         P->next = Q;
12.     }
13.     return Q;
14. }
```



Danh sách nối đơn

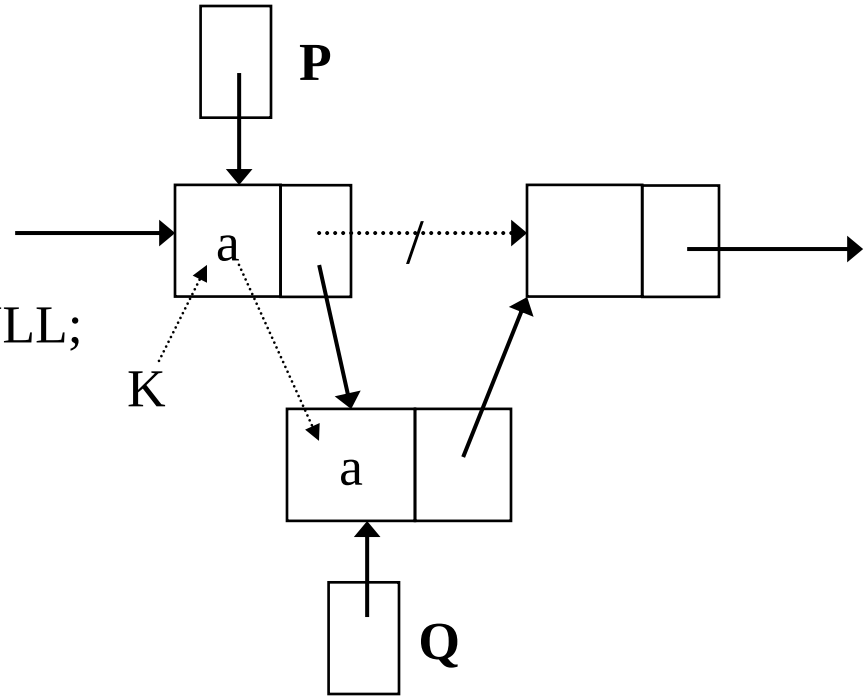
- Thao tác bổ sung một phần tử mới vào trước phần tử hiện tại P trong d/s H. Thao tác này sau đó trả về con trỏ trỏ vào nút vừa bổ sung

```
PNode InsertBefore(LinkedList & H, PNode P, Type K) {
    Bổ sung một nút Q để chứa K
    Nếu d/s H rỗng (H=P=NULL) thì:
        Q->next = NULL;
        H=Q;
   Trái lại, thì:
        Chuyển giá trị từ nút P sang nút Q
        Cập nhật giá trị của P bằng K
        Q->next = P->next;
        P->next=Q;
    return P;
}
```



Cài đặt hàm bổ sung trước

```
1.  PNode InsertBefore(LinkedList & H, PNode P, Type K){
2.      PNode Q = new Node;
3.      Q->info = K;
4.      if (H==NULL){
5.          H = Q;
6.          Q->next = NULL;
7.          return Q;
8.      }else {
9.          if (P==NULL) return NULL;
10.         Q->info = P->info;
11.         P->info = K;
12.         Q->next = P->next;
13.         P->next = Q;
14.     }
15.     return P;
16. }
```

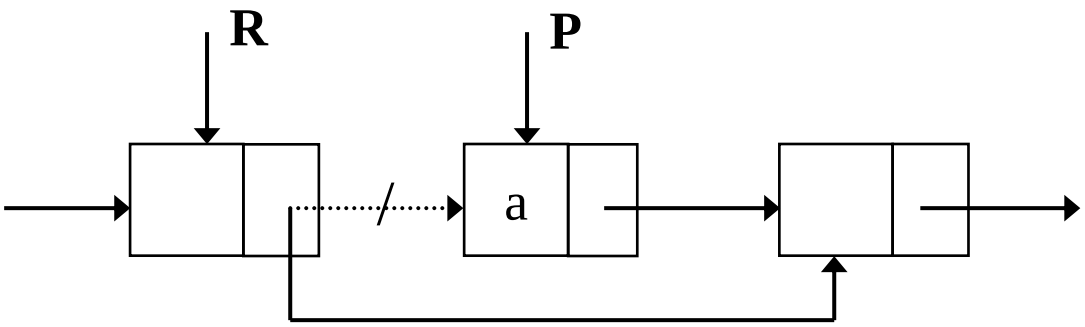


Danh sách nối đơn

- Phép xóa phần tử hiện tại mà con trỏ P trỏ tới trong danh sách H.

```
void DeleteNode(LinkedList & H, PNode P) {
    Nếu ds H chỉ có một phần tử (H=P và P->next = NULL)
      Cập nhật ds thành rỗng: H=NULL;
      Giải phóng nút P: delete P;
    Trái lại
      Nếu nút P là nút đầu ds (P = H)
                H = H->next;
                 Giải phóng nút P
      Trái lại
                Tìm đến nút R đứng ngay trước nút P;
                R->next= P->next;
                Giải phóng nút P;

}
```



Cài đặt hàm xóa một nút

- Hàm xóa phần tử hiện tại mà con trỏ P trỏ tới trong ds H. Nó trả về con trỏ trỏ đến nút kế tiếp của nút bị xóa

```
PNode DeleteNode(LinkedList & H, PNode P){
    if (P==NULL) return NULL;
    if (H==P &&P->next==NULL){//Neu ds H chi co 1 phan tu
            H=NULL;
            delete P;
            return NULL;
    }else {
            if (H==P){//Neu P la nut dau tien
                    H=P->next;
                    delete P;
                    return H;
            }else {
                    PNode R=H;
                    while (R->next != P) R=R->next;
                    R->next = P->next;
                    delete P;
                    return R->next;
            }
    }
}
```

Danh sách nối đơn

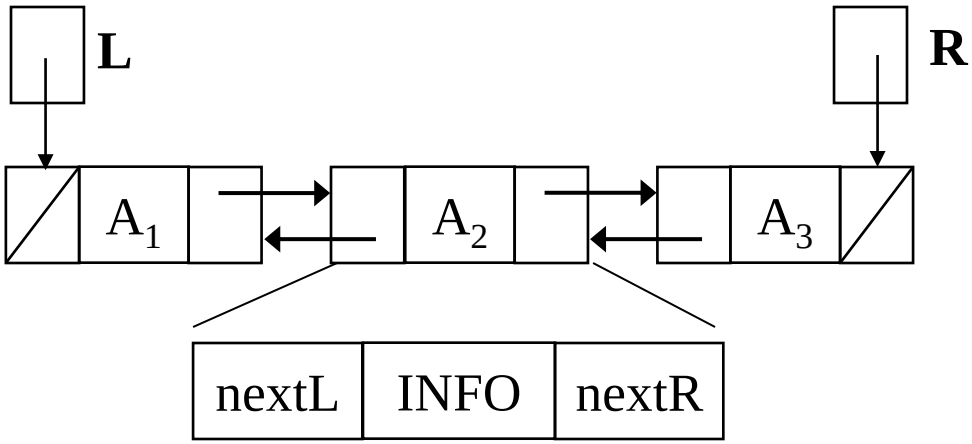
- Phép duyệt danh sách (có thể ứng dụng vào tính số phần tử của danh sách):

```
void Traverse (LinkedList H) {
    Pnode P;
    P = H;
    while (P != NULL) {
        Visit (P);
        P = P->next;
    }
}
```

  - Hàm Visit có thể là bất cứ chương trình nào đó làm việc với nút được lựa chọn (P). Nếu đơn thuần nó chỉ tăng lên 1 thì ta có hàm tính tổng số phần tử có trong danh sách.

Danh sách nối kép

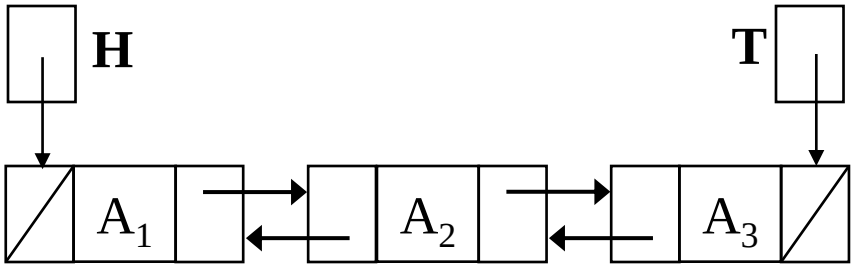
- Mô tả
  - Với danh sách đơn sử dụng một con trỏ, ta chỉ có thể duyệt danh sách theo một chiều
  - Danh sách nối kép (***double linked list***):
    - Con trỏ trái (nextL): trỏ tới thành phần bên trái (phía trước)
    - Con trỏ phải (nextR): trỏ tới thành phần bên phải (phía sau)
  - Đặc điểm
    - Sử dụng 2 con trỏ, giúp ta luôn xem xét được cả 2 chiều của danh sách
    - Tốn bộ nhớ nhiều hơn



Danh sách nối kép

- Định nghĩa danh sách nối kép thẳng

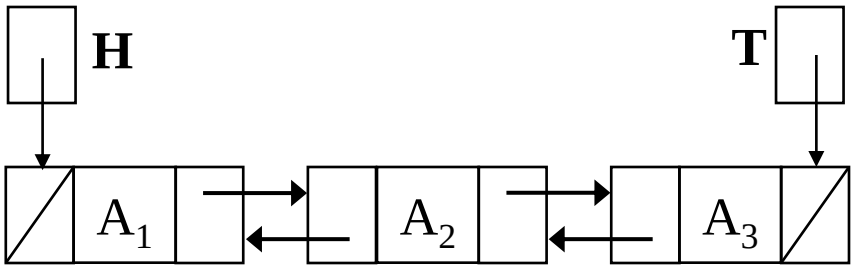
```
struct  DNode {
    Type info;
    DNode *nextL, *nextR;
};
typedef DNode*  PDNode;
typedef struct {
    PDNode  H; //con trỏ đầu
    PDNode  T; //con trỏ cuối
} DoubleLinkedList;
```



Danh sách nối kép

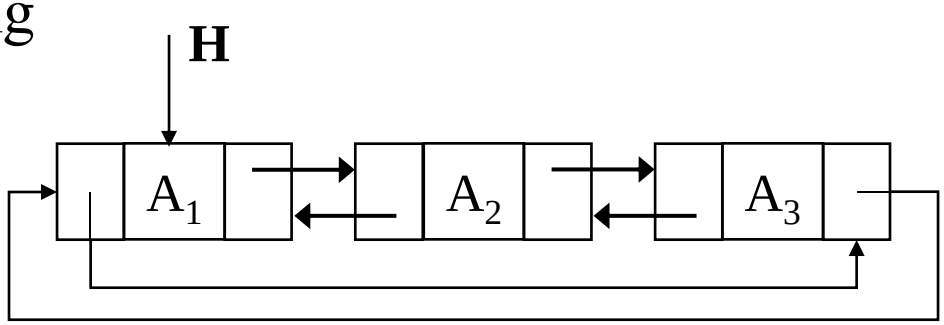
- Danh sách nối kép thẳng
  - Các phép toán:

- Khởi tạo danh sách: ***NewDList***
- Phép xen một phần tử mới vào danh sách
  - Xen phần tử mới vào trước phần tử hiện tại Q: ***InsertAfter***
  - Xen phần tử mới vào sau phần tử hiện tại Q: ***InsertBefore***
- Phép xoá phần tử khỏi danh sách: ***Delete***
- Phép tìm kiếm phần tử có dữ liệu = x: ***Search***
- Phép duyệt danh sách: ***Traverse***



Danh sách nối kép

- Danh sách nối kép vòng
  - Hai phần tử ở hai đầu không có con trỏ chỉ tới NULL
  - Danh sách rỗng nếu như:
    - H=NULL




```
struct  DNode {
    Type info;
    DNode *nextL, *nextR;
};
typedef DNode*  PDNode;
typedef PDNode CDoubleLinkedList;
```

Danh sách nối kép

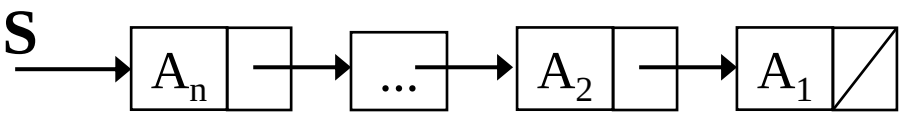
- Danh sách nối kép vòng
  - Các phép toán:

    - Khởi tạo danh sách: ***NewCDList***
    - Phép xen một phần tử mới vào danh sách
      - Xen phần tử mới vào trước phần tử hiện tại Q: ***InsertLeft***
      - Xen phần tử mới vào sau phần tử hiện tại Q: ***InsertRight***
    - Phép xoá phần tử khỏi danh sách: ***Delete***
    - Phép tìm kiếm phần tử có dữ liệu = x: ***Search***
    - Phép duyệt danh sách: ***Traverse***

LIFO, FIFO & CTLT móc nối

- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Cách tổ chức:
    - Chỉ cần một con trỏ S vừa là đỉnh, vừa là điểm truy nhập
  - Khai báo cấu trúc

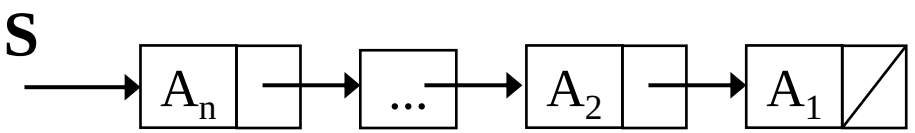
```
struct  Node {
    Type info;
    Node* next;
};
typedef Node* PNode;
typedef PNode Stack;
```



  - Nhắc lại: vì danh sách liên kết có kích thước động, không bị giới hạn trước, sự tăng kích thước chỉ phụ thuộc vào khả năng cấp phát bộ nhớ của máy tính cho nên ta có thể coi ngăn xếp có kích thước vô hạn.

LIFO, FIFO & CTLT móc nối

- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Các phép toán: là các trường hợp đặc biệt của DS nối đơn
    - ***Initialize***
    - ***isEmpty***
    - ***isFull***
    - ***Push***
    - ***Pop***
    - ***Traverse (Ứng dụng tính số phần tử: Size)***

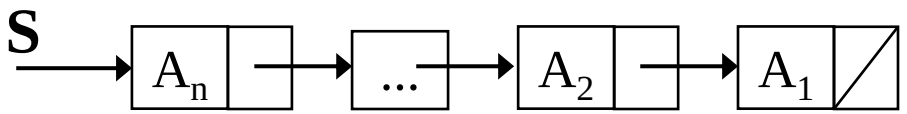



```
void Initialize (Stack & S) {
    S = NULL;
}
```

LIFO, FIFO & CTLT móc nối

- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Các phép toán: kiểm tra trạng thái của ngăn xếp

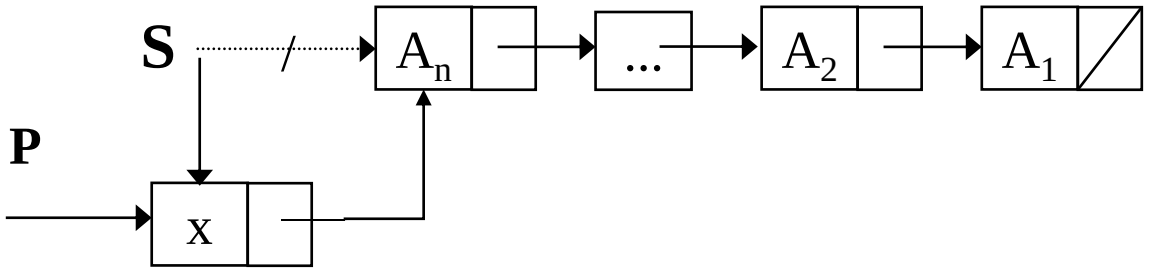
```
bool isEmpty (Stack S){
    return (S==NULL);
}
```



LIFO, FIFO & CTLT móc nối

- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Các phép toán: bổ sung 1 phần tử vào đỉnh

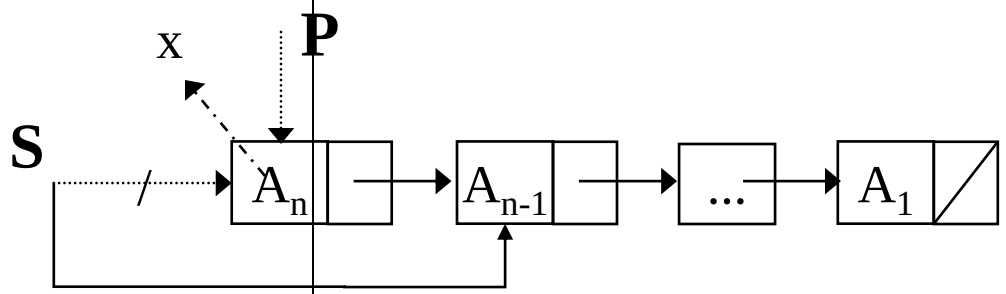
```
void Push (Item x, Stack & S){
    Pnode P;
    P = new PNode;
    P->info = x;
    P->next = S;
    S = P;
}
```



LIFO, FIFO & CTLT móc nối

- Biểu diễn LIFO ngăn xếp (Stack)
  - Các phép toán

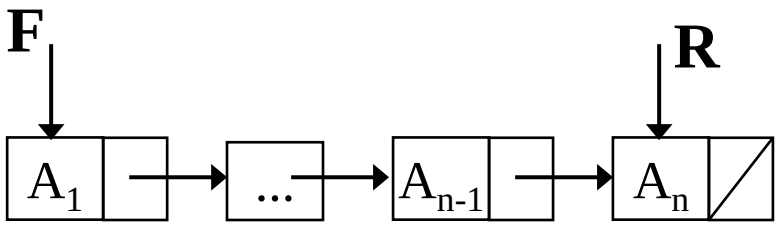
```
PNode Pop (Item & x, Stack & S) {
    Pnode P;
    if  (isEmpty (S)) return NULL;
    else {
      P = S;
      x = P->info;
      S = S->next;
      delete P;
      return S;
    }
}
```



LIFO, FIFO & CTLT móc nối

- Biểu diễn FIFO hay hàng đợi (Queue)
  - Cấu trúc

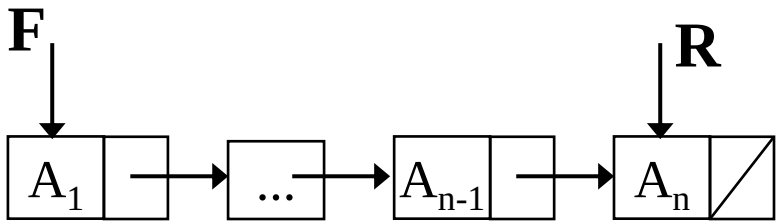
```
struct  Node {
    Type info;
    Node* next;
};
typedef Node* PNode;
typedef struct {
    PNode F, R;
} Queue;
```



  - Nhắc lại: vì danh sách liên kết có kích thước động, không bị giới hạn trước. Sự tăng kích thước chỉ phụ thuộc vào khả năng cấp phát bộ nhớ của máy tính cho nên ta có thể coi hàng đợi của chúng ta có kích thươc vô hạn.

LIFO, FIFO & CTLT móc nối

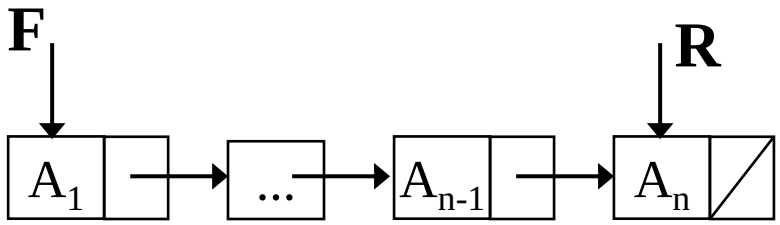
- Biểu diễn FIFO hay hàng đợi (Queue)
  - Các phép toán: là các trường hợp đặc biệt của DS nối đơn
    - ***Initialize***
    - ***isEmpty***
    - ***isFull***
    - ***InsertQ***
    - ***DeleteQ***
    - ***Traverse (Ứng dụng tính số phần tử: Size)***




```
void Initialize (Queue & Q){
    Q.F = Q.R = NULL;
}
```

LIFO, FIFO & CTLT móc nối

- Biểu diễn FIFO hay hàng đợi (FIFO or Queue)
  - Các phép toán



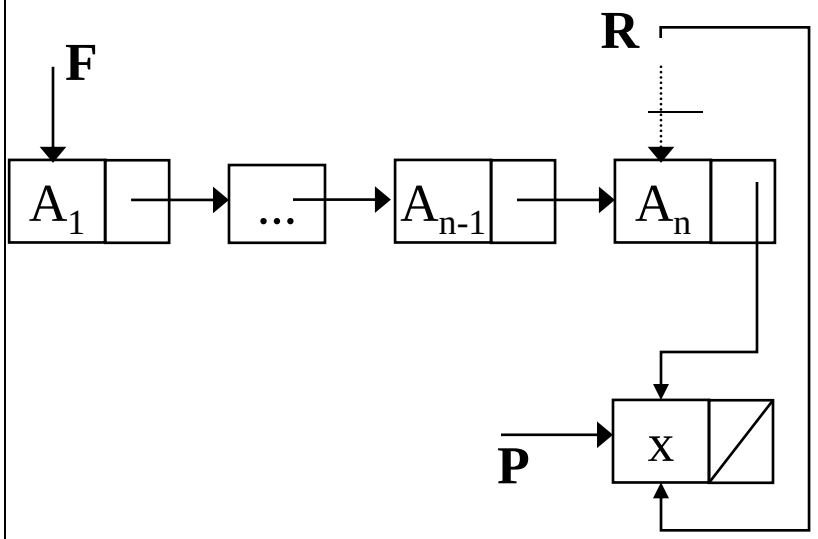

```
bool isFull (Queue Q) {
    return false;
}
```

```
bool isEmpty (Queue Q) {
    return  (Q.F == NULL);
}
```

LIFO, FIFO & CTLT móc nối

- Biểu diễn FIFO hay hàng đợi (FIFO or Queue)
  - Các phép toán

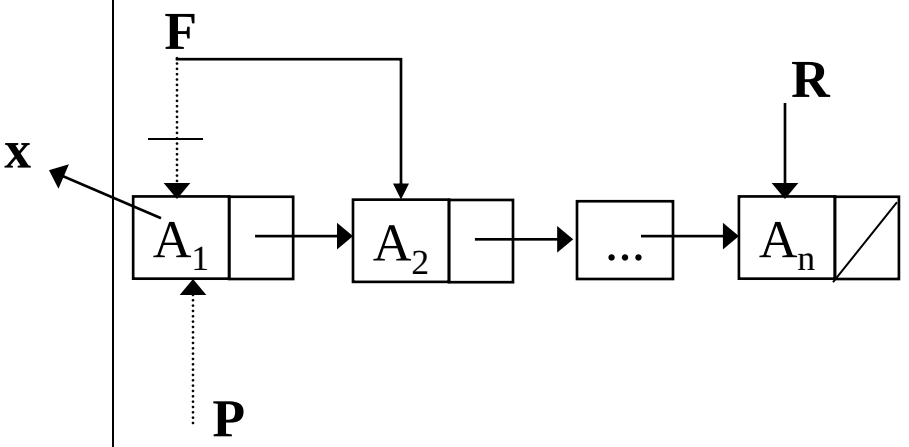
```
void InsertQ (Item x, Queue & Q){
    Pnode P;
    P = new PNode;
    P->info = x;
    P->next = NULL;
    if  (isEmpty (Q)) {
            Q.F = Q.R = P;
    }
    else  {
            Q.R->next = P;
            Q.R = P;
    }
}
```



LIFO, FIFO & CTLT móc nối

- Biểu diễn FIFO hay hàng đợi (FIFO or Queue)
  - Các phép toán

```
void DeleteQ (Item & x, Queue & Q){
    Pnode P;
    if  (isEmpty (Q)) return;
    else {
        P = Q.F;
        x = Q.F->info;
        Q.F = Q.F->next;
        delete P;
    }
}
```



So sánh CTLT tuần tự và CTLT móc nối trong cài đặt DS

- Về bộ nhớ, lưu trữ
  - CT móc nối đòi hỏi lưu trữ thêm con trỏ (pointer) trỏ tới các nút
  - CT tuần tự đòi hỏi xác định trước kích thước max cho các phần tử để chiếm chỗ trước cho chúng trong bộ nhớ. Gây ra lãng phí khi kích thước max đó không được dùng hết.
  - CT móc nối không đòi hỏi chiếm chỗ trước cho các phần tử. Điều này giúp chương trình tiết kiệm bộ nhớ khi chạy.

So sánh CTLT tuần tự và CTLT móc nối trong cài đặt DS

- Về thời gian
  - Hầu hết các phép toán của CT móc nối đòi hỏi nhiều lệnh hơn so với CT tuần tự, do vậy sẽ tốn nhiều thời gian hơn
  - CT tuần tự thường nhanh hơn trong trường hợp tìm kiếm và sửa đổi các phần tử nằm ở giữa danh sách
  - CT móc nối thường nhanh hơn trong trường hợp thêm hoặc bớt các phần tử vào giữa danh sách

Bài tập

- Bài 1: Cài đặt danh sách tổng quát bằng cấu trúc lưu trữ móc nối kép. Việc cài đặt bao gồm:
  - Nêu cách tổ chức danh sách
  - Định nghĩa cấu trúc
  - Cài đặt các hàm thực hiện các thao tác cơ bản: khởi tạo, bổ sung một phần tử vào trước 1 phần tử hiện tại, bổ sung một phần tử vào sau một phần tử hiện tại, loại bỏ một phần tử hiện tại.
- Bài 2: xây dựng lớp Stack và lớp Queue để cài đặt cho 2 cấu trúc dữ liệu trên bằng cấu trúc lưu trữ móc nối.
- Bài 3: Cài đặt Queue bằng cấu trúc móc nối kép:
  - Định nghĩa cấu trúc
  - Cài đặt các thao tác cơ bản: Khởi tạo, bổ sung, loại bỏ

- Bài 4: cài đặt một danh sách các môn học, mỗi môn học gồm các thông tin: mã môn, tên môn, số trình. Danh sách luôn được sắp xếp theo thứ tự tăng dần của số trình. Yêu cầu:
  - Sử dụng cấu trúc lưu trữ móc nối đơn để cài đặt danh sách
  - Cài đặt các thao tác: khởi tạo, bổ sung 1 môn, loại bỏ một môn có mã môn cho trước, in ra nội dung của DS.

Xin cảm ơn!