CẤU TRÚC MẢNG VÀ DANH SÁCH TUYẾN TÍNH

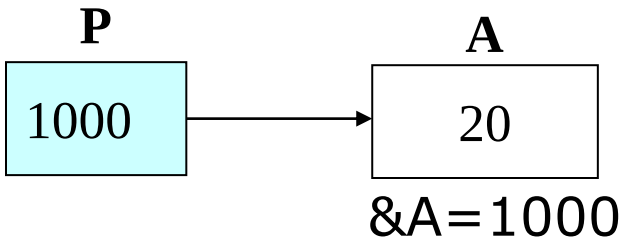
Phần 3: Danh sách móc nối

Các nội dung chính

- Con trỏ và cấp phát bộ nhớ cho đối tượng động
- Mô tả cấu trúc lưu trữ móc nối (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

Con trỏ và cấp phát bộ nhớ cho đối tượng động

- 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.

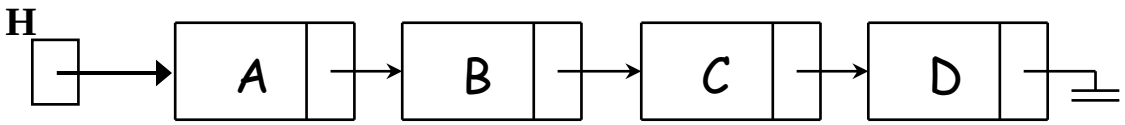


- Các thao tác cơ bản
  - Khởi tạo (khai báo): *int * P;*
  - Lấy địa chỉ 1 đối tượng: *int A=20; P = &A;*
  - Truy nhập vào đối tượng được trỏ: **P = 20;*
  - Cấp phát bộ nhớ động cho đối tượng DL động: *P = new int;*
  - Giải phóng đối tượng DL động: *delete P;*

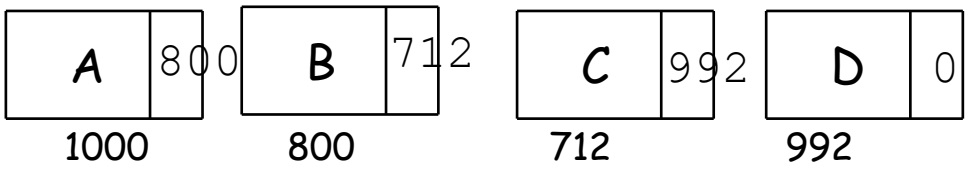
Mô tả cấu trúc lưu trữ móc nối (danh sách móc nối)

- Là tập hợp các phần tử dữ liệu không liên tục được kết nối với nhau thông qua một liên kết (thường là con trỏ)
- Cho phép ta quản lý bộ nhớ linh động
- Các phần tử được chèn vào danh sách và xóa khỏi danh sách một cách dễ dàng
- Tại mỗi nút có hai thành phần:
  - Dữ liệu trong nút
  - Con trỏ trỏ đến phần tử kế tiếp
- H: con trỏ trỏ vào nút đầu của danh sách

Mô tả

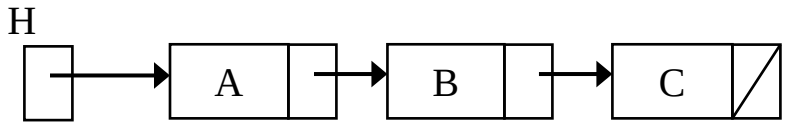


Trong bộ nhớ

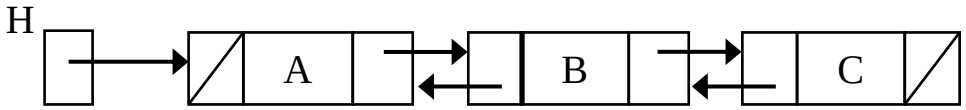


Phân loại danh sách móc nối

- 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

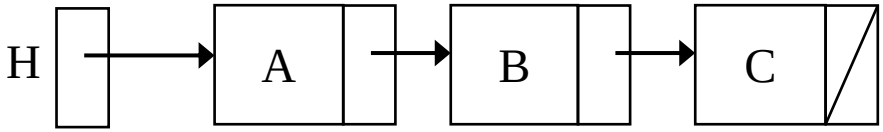


  - Danh sách nối kép (double linked list)
    - 2 con trỏ chỉ theo hai hướng trong danh sách

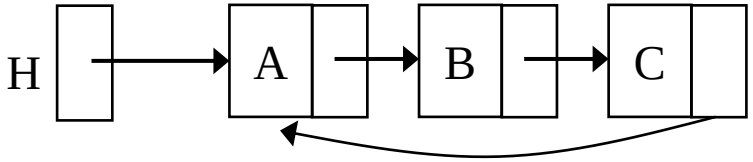


Phân loại danh sách móc nối

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

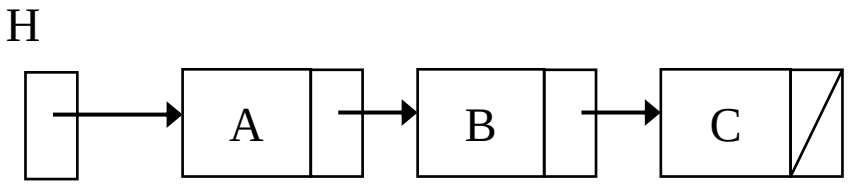


  - Danh sách nối vòng (circularly linked list): bất cứ nút nào trong danh sách cũng có thể coi là nút đầu hay nút cơ sở (mọi nút có vai trò như nhau)



Cài đặt danh sách nối đơn thẳng

- 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 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 sau phần tử hiện tại Q: InsertAfter
    - Xen phần tử mới vào trước 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 thẳng

- Khởi tạo danh sách: gán con trỏ H=Null

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

- Kiểm tra danh sách rỗng: kiểm tra con trỏ H có bằng Null không

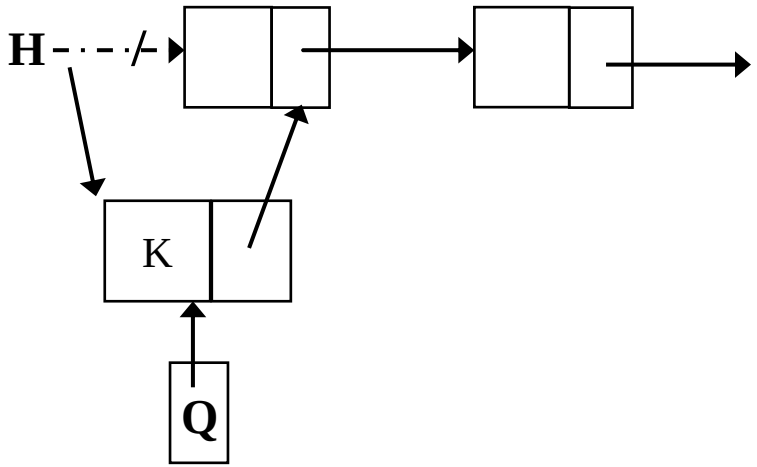
```
bool IsEmpty (LinkedList  H) {
    return (H == NULL);
}
```

Cài đặt danh sách nối đơn thẳng

- Thao tác bổ sung một phần tử mới K vào đầu danh sách H

Giải thuật

```
void InsertBegin(LinkedList & H, Type K){
    Tạo một nút mới Q để chứa K
     Nếu d/s rỗng (H=NULL) thì:
            Q->next = NULL;
            H=Q;
    Trái lại, thì:
           Q->next=H;
           H = Q;
}
```

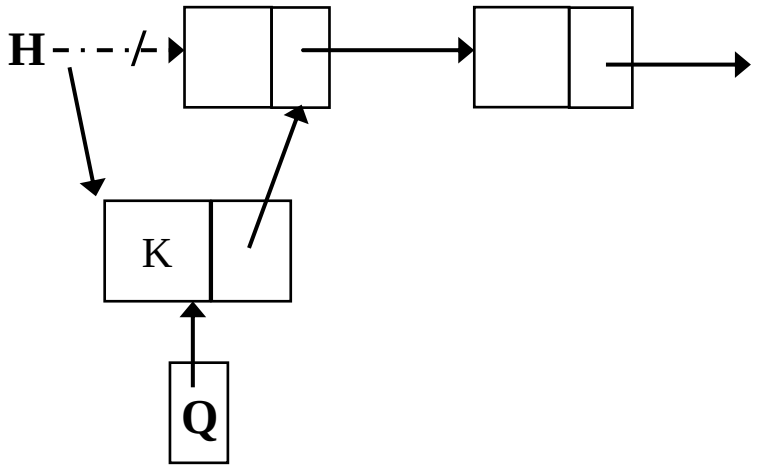


Cài đặt danh sách nối đơn thẳng

- Thao tác bổ sung một phần tử mới K vào đầu danh sách H

Cài đặt hàm

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

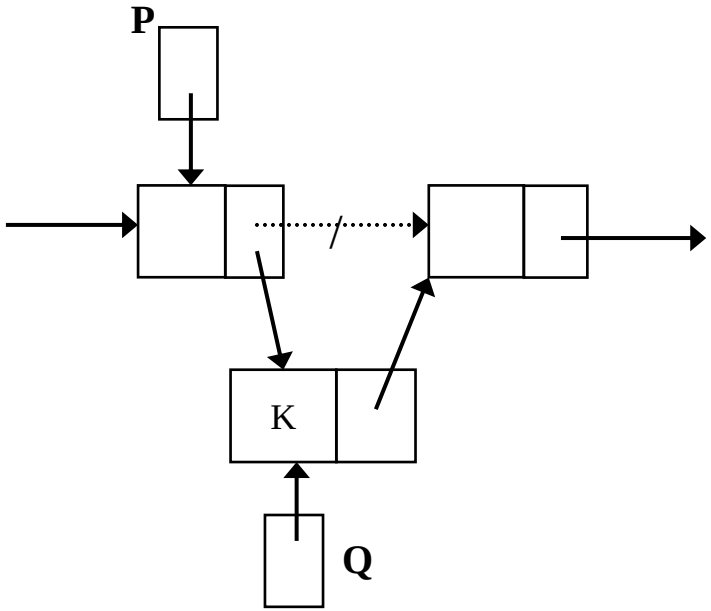


Cài đặt danh sách nối đơn thẳng

- 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. Nếu không cần trả về phần tử vừa bổ sung thì sửa thế nào?

Giải thuật

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

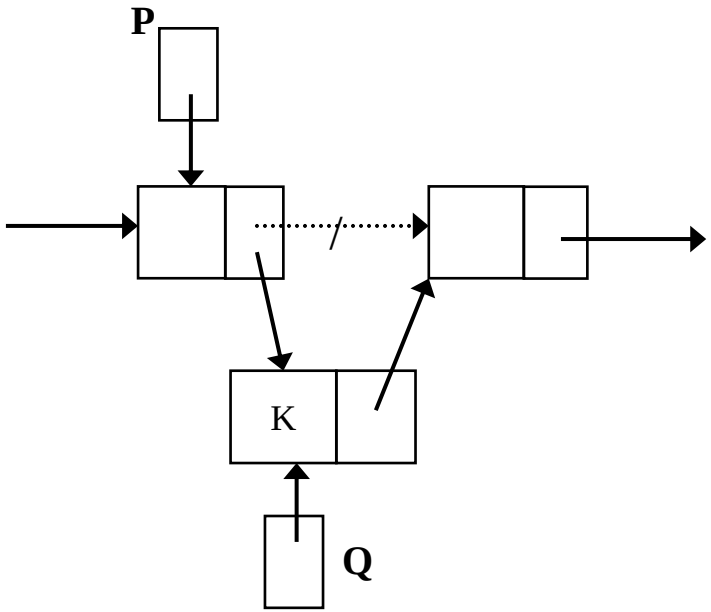


Cài đặt danh sách nối đơn thẳng

- 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. Nếu không cần trả về phần tử vừa bổ sung thì sửa thế nào?

Cài đặt hàm

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

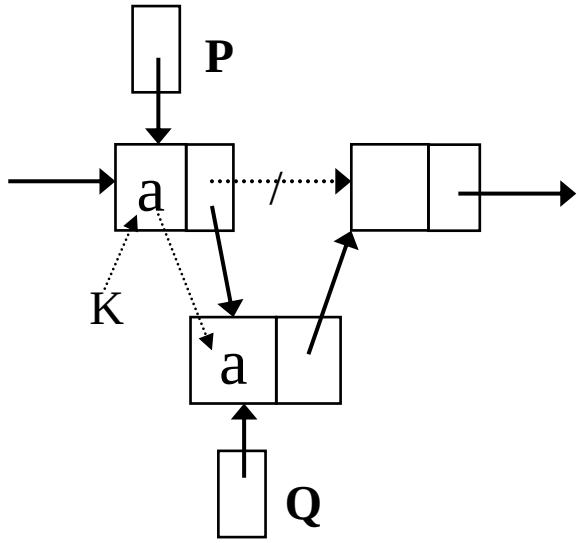


Cài đặt danh sách nối đơn thẳng

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

Giải thuật

```
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:
            H=Q;
            Q->next = NULL;
    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;
}
```

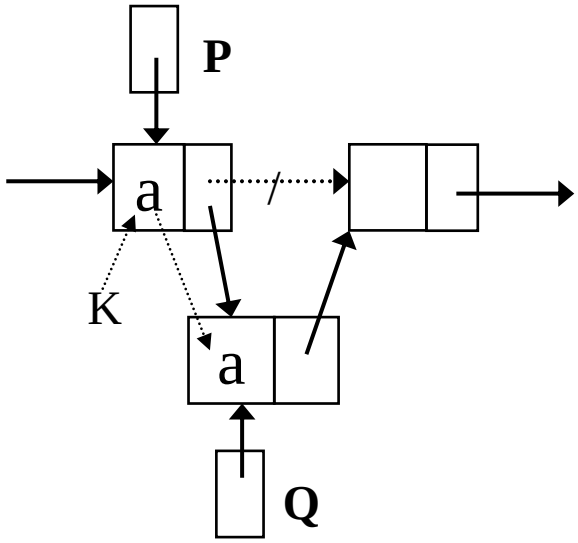


Cài đặt danh sách nối đơn thẳng

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

Cài đặt hàm

```
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. }
```

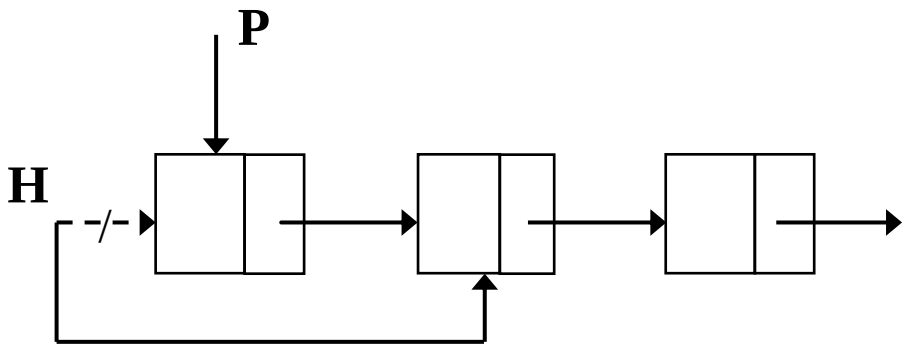


Cài đặt danh sách nối đơn thẳng

- Phép xóa phần ở đầu danh sách H

Giải thuật

```
void DeleteNode(LinkedList & H) {
    Nếu ds rỗng (H=NULL), đưa ra thông báo
    Trái lại
                P = H
                H = H->next;
                Giải phóng nút P: delete P
}
```

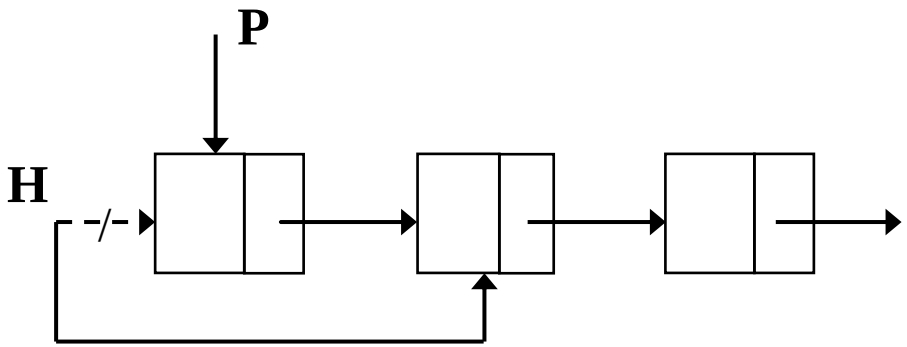


Cài đặt danh sách nối đơn thẳng

- Phép xóa phần ở đầu danh sách H

Cài đặt hàm

```
1. void DeleteNode(LinkedList & H){
2.   if (H==NULL) printf("danh sách rỗng");
3.   else {
4.         PNode P=H;
5.                H = H->next;
6.                delete P;
7.         }
8. }
```

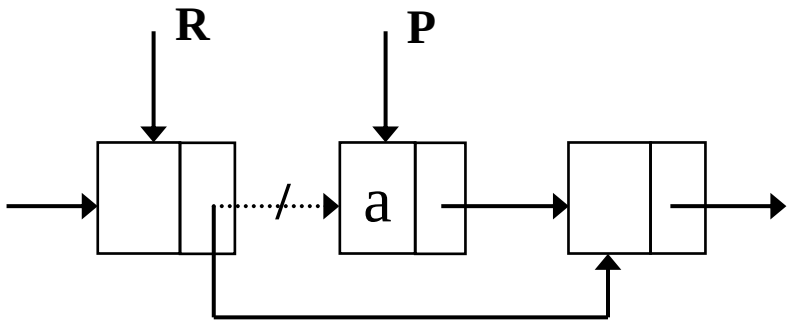


Cài đặt danh sách nối đơn thẳng

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

Giải thuật

```
void DeleteNode(LinkedList & H, PNode P) {
    Nếu ds rỗng (H=NULL), đưa ra thông báo
    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;
}
```

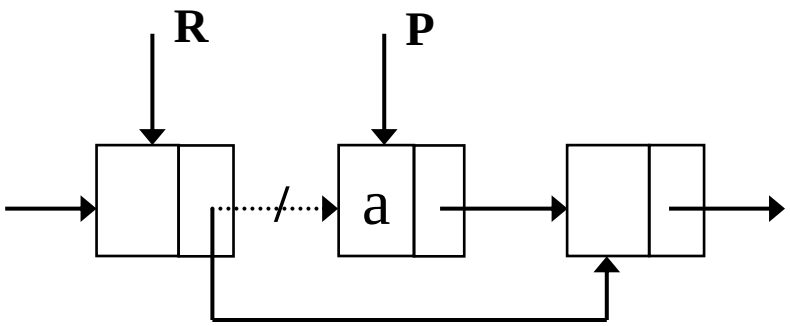


Cài đặt danh sách nối đơn thẳng

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

Cài đặt hàm

```
1.  void DeleteNode(LinkedList & H, PNode P){
2.    if (H==NULL) printf("danh sách rỗng");
3.    if (H==P &&P->next==NULL){//Neu ds H chi co 1 phan tu
4.          H=NULL;
5.          delete P;
6.    }else {
7.          if (H==P){//Neu P la nut dau tien
8.                H=P->next;
9.                delete P;
10.               }
11.         else {
12.               PNode R=H;
13.               while (R->next != P) R=R->next;
14.               R->next = P->next;
15.               delete P;
16.               }
17.  } }
```



Cài đặt danh sách nối đơn thẳng

- Phép tìm kiếm một phần tử trong danh sách H có dữ liệu bằng K cho trước. Hàm trả về địa chỉ của phần tử đó

Giải thuật

```
PNode SearchNode(LinkedList & H, type K) {
    Gán P là node đầu tiên
    Trong khi P còn chưa bằng Null thì
                    Nếu (P->info=K): return P
                    Trái lại: P=P->next
    Return 0;//Nếu không tìm thấy
}
```

Cài đặt danh sách nối đơn thẳng

- Phép tìm kiếm một phần tử trong danh sách H có dữ liệu bằng K cho trước. Hàm trả về địa chỉ của phần tử đó

Cài đặt hàm

```
1. PNode SearchNode(LinkedList & H, int K){
2.   PNode P=H;
3.   while (P!=0){
4.                 if (P->info==K)
5.               return P;
6.                 else p=p->next;
7.   }
8.          return 0;
9.  }
```

Cài đặt danh sách nối đơn thẳng

- Thao tác duyệt 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 nhiệm vụ nào làm việc với nút P

Cài đặt danh sách nối đơn thẳng

- Thao tác duyệt danh sách, ứng dụng vào tính số phần tử của danh sách

```
int ListLength(LinkedList H) {
    Pnode P;
    P = H;
    count=0;
    while (P != NULL) {
                    count++;
                    P = P->next;
    }
    return count;
}
```

Bài tập

BT1. Thực hiện một danh sách liên đơn, mỗi node trong danh sách chỉ chứa một dữ liệu thuộc kiểu int. Hãy thực hiện các thao tác sau:

-tạo ds rỗng, kiểm tra ds rỗng, bổ sung phần tử K vào đầu ds, bổ sung phần tử K vào sau phần tử có giá trị bằng X, tìm kiếm phần tử có giá trị bằng X

- Bổ sung phần tử vào trước phần tử có giá trị bằng X

- Xóa phần tử ở đầu danh sách

-Xóa các phần tử có giá trị bằng K

-Tính giá trị trung bình của danh sách

-Hiển thị dữ liệu của danh sách ra màn hình

-Viết chương trình chính minh họa cách sử dụng danh sách liên kết đơn đó.

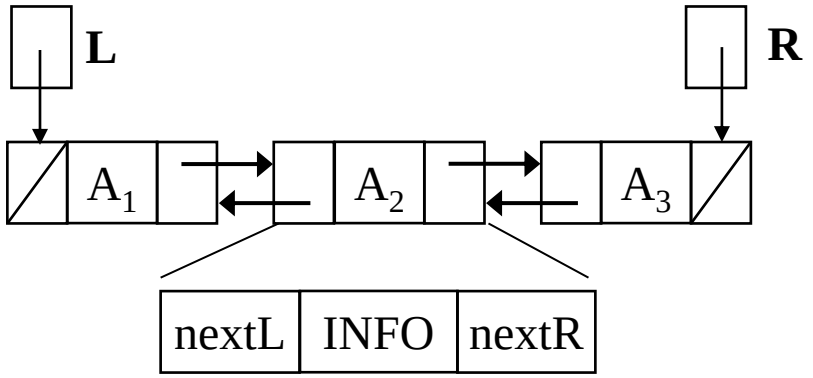
Bài tập

BT2. Thực hiện một cấu trúc list quản lý sinh viên. Thông tin về sinh viên được biểu diễn bởi một struct gồm mã sinh viên (int), họ tên sinh viên (string), lớp (string). Các thao tác cần thực hiện:

- Thêm SV vào đầu danh sách
- Thêm sinh viên vào cuối danh sách
- Tìm sinh viên theo tên
- Xóa sinh viên theo tên

Danh sách nối kép

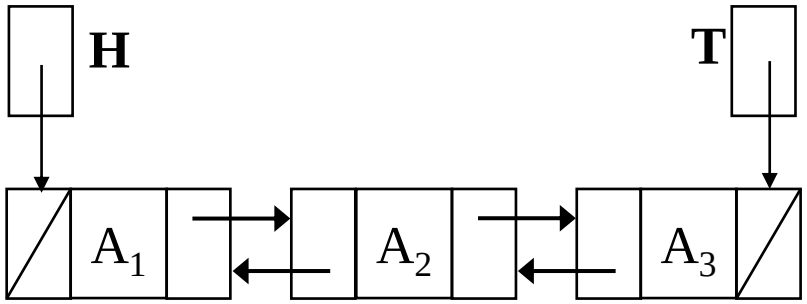
- Mô tả
  - Với danh sách đơn sử dụng một con trỏ, ta chỉ có thể duyệt danh sách theo một chiều
  - Danh sách nối kép (double linked list):
    - Con trỏ trái (nextL): trỏ tới thành phần bên trái (phía trước)
    - Con trỏ phải (nextR): trỏ tới thành phần bên phải (phía sau)
  - Đặc điểm
    - Sử dụng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



Danh sách nối kép

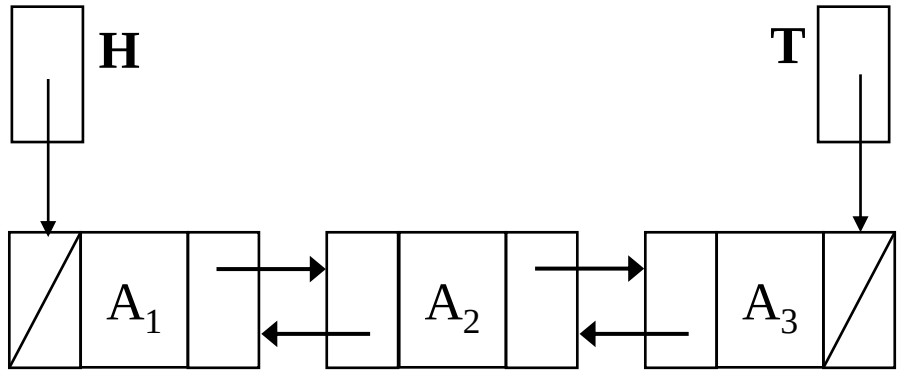
- Định nghĩa

```
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

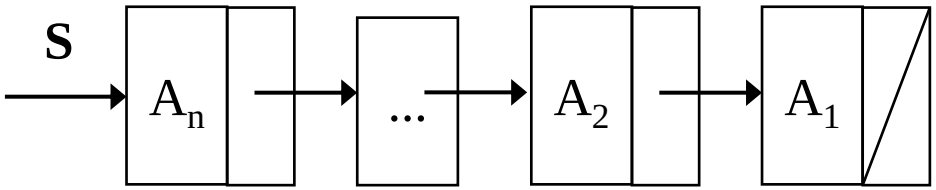
- Các phép toán
  - Khởi tạo danh sách: NewDList
  - Phép xen một phần tử mới vào danh sách
    - Xen phần tử mới vào trước phần tử hiện tại Q: InsertAfter
    - Xen phần tử mới vào sau phần tử hiện tại Q: InsertBefore
  - Phép xoá phần tử khỏi danh sách: Delete
  - Phép tìm kiếm phần tử có dữ liệu =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



Cài đặt LIFO (Stack) bằng CTLT móc nối

- Cách tổ chức:
  - Chỉ cần một con trỏ S vừa là đỉnh, vừa là điểm truy nhập
- Khai báo cấu trúc

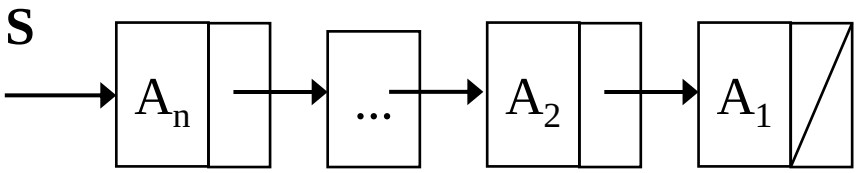
```
struct  Node {
    Type info;
    Node* next;
};
typedef Node* PNode;
typedef PNode Stack;
```



- Lưu ý: 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 có thể coi ngăn xếp có kích thước vô hạn.

Cài đặt LIFO (Stack) bằng CTLT móc nối

- Các phép toán:
  - Initialize
  - isEmpty
  - isFull
  - Push
  - Pop
  - Traverse (Ứng dụng tính số phần tử: Size)



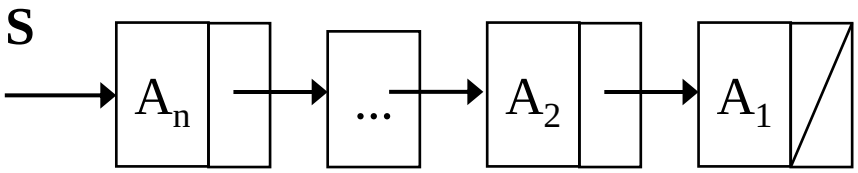

```
void Initialize (Stack & S) {
    S = NULL;
}
```

Cài đặt LIFO (Stack) bằng CTLT móc nối

- Kiểm tra trạng thái của ngăn xếp

Thuật toán

```
Procedure isEmpty (S){
    if S=NULL
        return true;
    else
        return false;
}
```



Cài đặt

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

Cài đặt LIFO (Stack) bằng CTLT móc nối

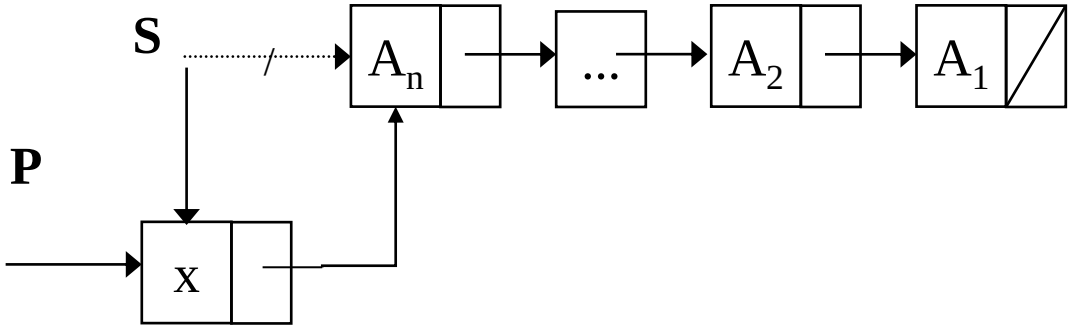
- Bổ sung 1 phần tử vào đỉnh ngăn xếp

Thuật toán

```
Procedure Push (x, S){
    Pnode P;
    P = new PNode;
    P->info = x;
    P->next = S;
    S = P;
}
```

Cài đặt

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

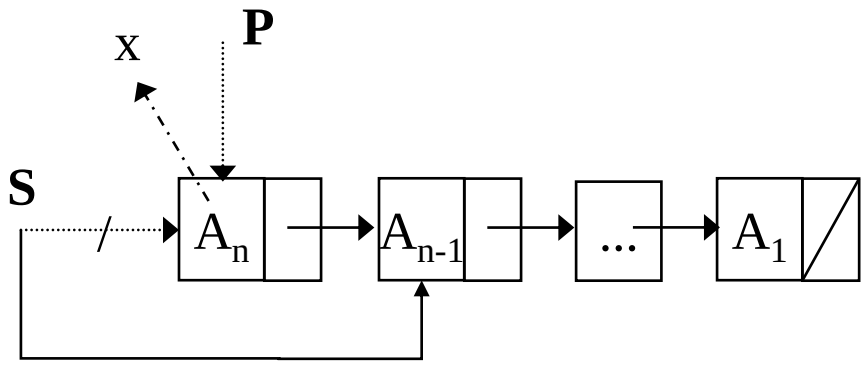


Cài đặt LIFO (Stack) bằng CTLT móc nối

- Phép toán lấy một phần tử khỏi ngăn xếp

Thuật toán

```
Procedure Pop (x, S) {
    Pnode P;
    if  (isEmpty (S)) write ("Empty");
    else {
      P = S;
      x = P->info;
      S = S->next;
      delete P;
      }
}
```

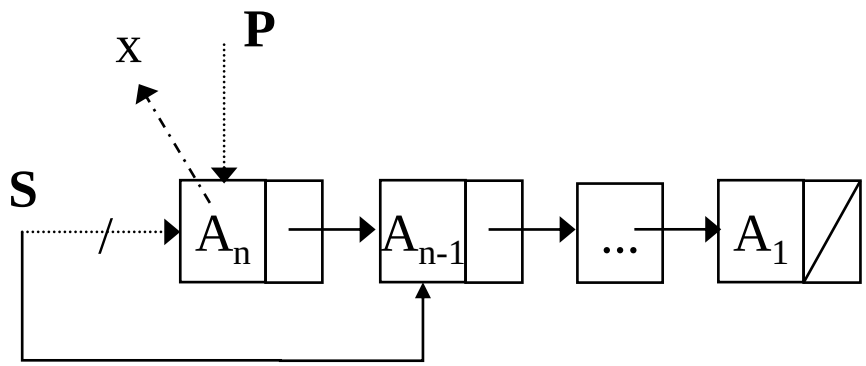


Cài đặt LIFO (Stack) bằng CTLT móc nối

- Phép toán lấy một phần tử khỏi ngăn xếp

Cài đặt

```
void Pop (Type & x, Stack & S) {
    Pnode P;
    if  (isEmpty (S)) printf("Empty");
    else {
      P = S;
      x = P->info;
      S = S->next;
      delete P;
      }
}
```



Cài đặt LIFO (Stack) bằng CTLT móc nối

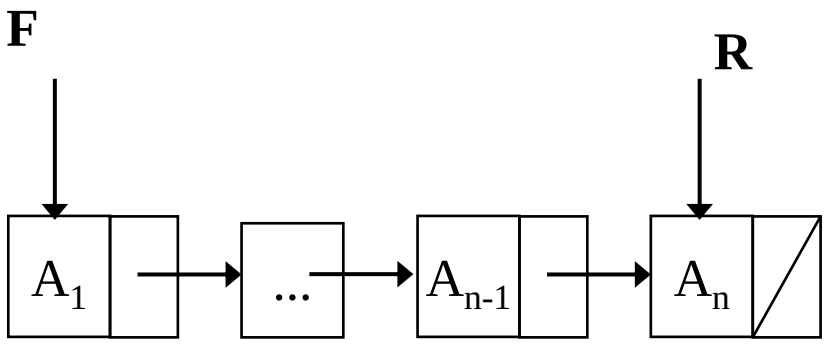
- Phép toán tính số phần tử của ngăn xếp

```
int StackLength(Stack S) {
    Pnode P;
    P = S;
    count=0;
    while (P != NULL) {
                    count++;
                    P = P->next;
    }
    return count;
}
```

Cài đặt FIFO (Queue) bằng CTLT móc nối

- Cấu trúc

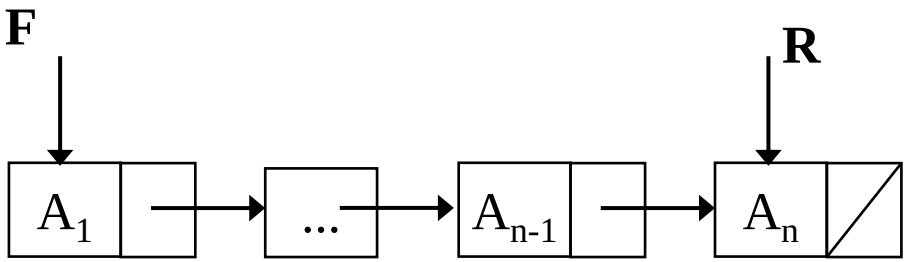
```
struct  Node {
    Type info;
    Node* next;
};
typedef Node* PNode;
typedef struct {
    PNode F, R;
} Queue;
```



- Lưu ý: 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 có thể coi hàng đợi của có kích thước vô hạn.

Cài đặt FIFO (Queue) bằng CTLT móc nối

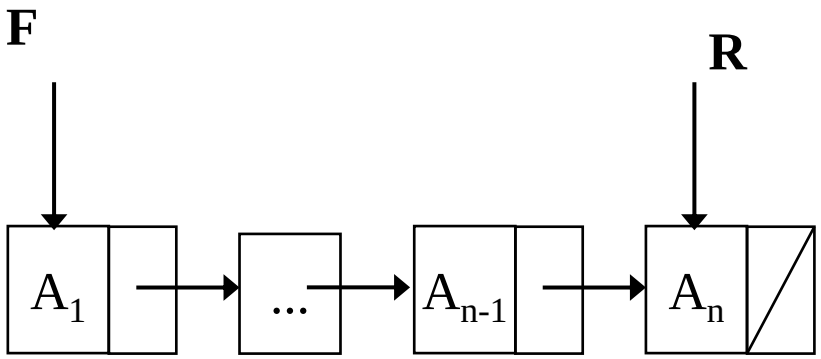
- Các phép toán:
  - Initialize
  - isEmpty
  - InsertQ
  - DeleteQ
  - Traverse (Ứng dụng tính số phần tử: Size)




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

Cài đặt FIFO (Queue) bằng CTLT móc nối

- Các phép toán

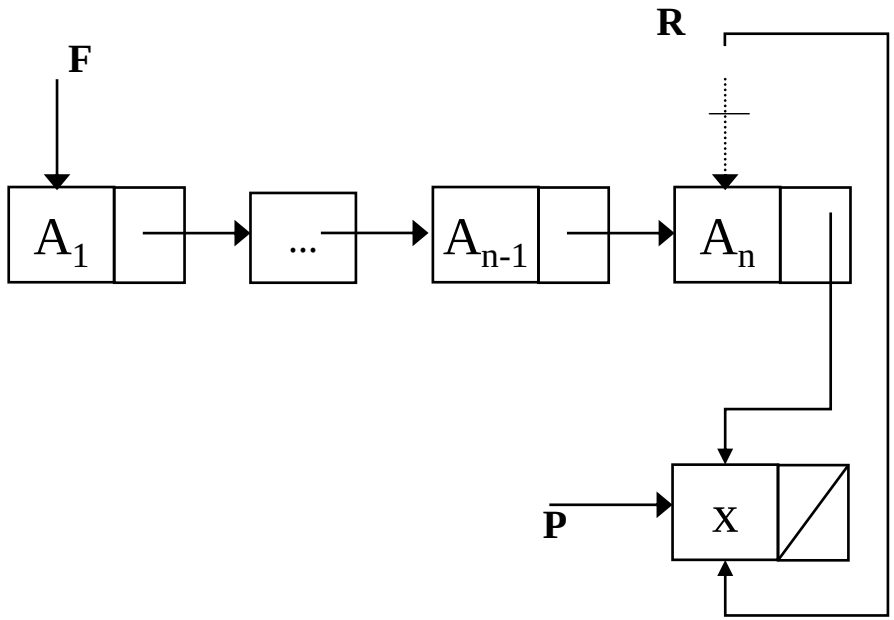



```
bool isEmpty (Queue Q) {
    return  (Q.F == NULL);
}
```

Cài đặt FIFO (Queue) bằng CTLT móc nối

- Phép toán thêm phần tử vào hàng đợi

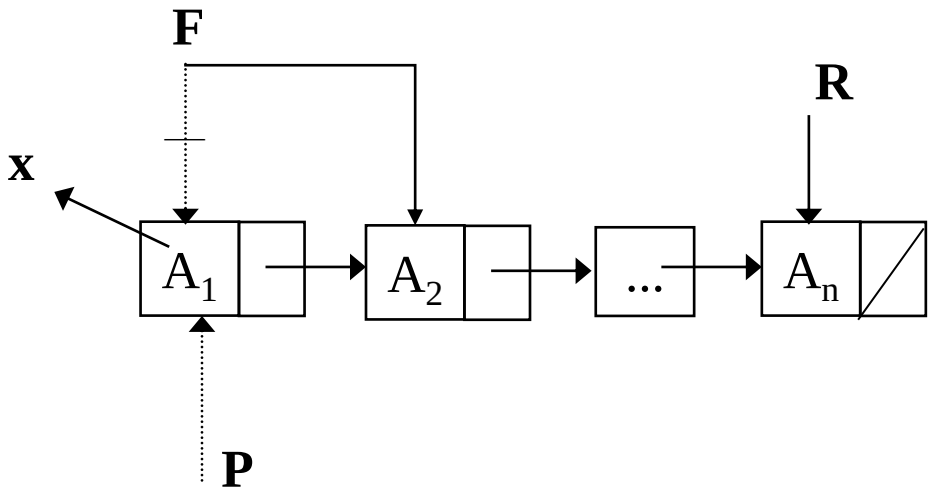
```
void InsertQ (Type x, Queue & Q){
    Pnode P;
    P = new PNode;
    P->info = x;
    P->next = NULL;
    if  (isEmpty (Q)) {
            Q.F = Q.R = P;
    }
    else {
            Q.R->next = P;
            Q.R = P;
    }
}
```



Cài đặt FIFO (Queue) bằng CTLT móc nối

- Phép toán xóa phần tử khỏi hàng đợi

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



@ Bài tập

- Bài 1: Cài đặt một danh sách số nguyên 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, bố 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: 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 tập (tiếp)

- Bài 3: cài đặt một danh sách các môn học, mỗi môn học gồm các thông tin: mã môn, tên môn, số tín chỉ. Danh sách luôn được sắp xếp theo thứ tự tăng dần của số tín chỉ. 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.