HUST

TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

ET2100

ONE LOVE. ONE FUTURE.

Chương II :
Cấu trúc mảng và Danh sách tuyến tính

Giảng viên: TS. Đỗ Thị Ngọc Diệp
Khoa Kỹ thuật Truyền thông – Trường Điện Điện Tử

Các nội dung chính

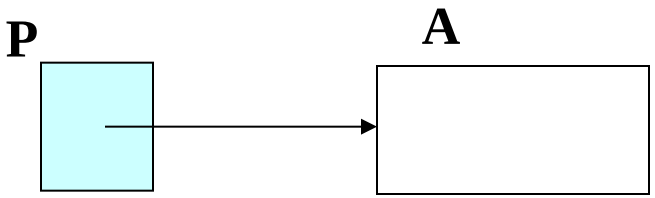
1. Cấu trúc mảng
   - 1.1. Mô tả
   - 1.2. Cấu trúc lưu trữ tuần tự
   - 1.3. Cài đặt mảng bằng cấu trúc lưu trữ tuần tự
2. Cấu trúc danh sách tuyến tính
   - 2.1. Mô tả
   - 2.2. Cài đặt danh sách bằng cấu trúc lưu trữ tuần tự
   - **2.3. Cài đặt danh sách bằng cấu trúc lưu trữ móc nối**
   - **2.4. Một số ứng dụng của ngăn xếp và hàng đợi**

2.3. Cài đặt danh sách bằng CTLT móc nối

- Nội dung
  - Con trỏ & Cấu trúc lưu trữ móc nối
  - Danh sách móc nối (liên kết)
    - Các loại danh sách móc nối
  - Cài đặt danh sách móc nối bằng CTLT móc nối
    - Khai báo cấu trúc
    - Cài đặt các thao tác cơ bản
  - Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

2.3.1 Con trỏ

- Con trỏ (pointer): là kiểu dữ liệu dùng để chỉ đến địa chỉ một đối tượng dữ liệu khác chứa trong bộ nhớ.

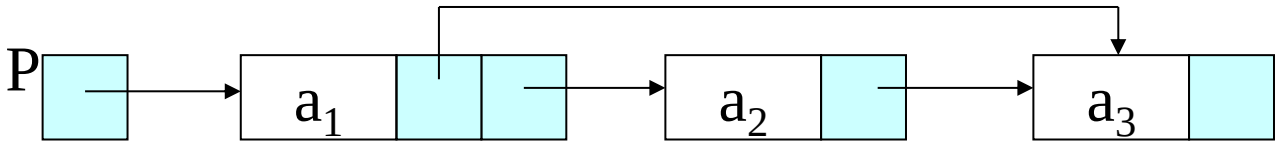


– Các thao tác cơ bản

- 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;***

2.3.1 Cấu trúc lưu trữ móc nối

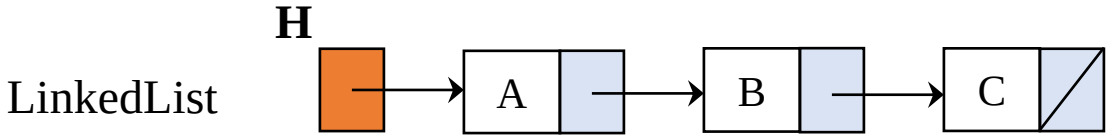
- Cấu tạo



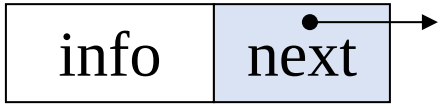
  - Các nút: chứa thông tin về các phần tử và có thể cả con trỏ
  - Con trỏ: trỏ đến các nút
- Đặc điểm:
  - Cấu trúc lưu trữ động: mỗi nút của CTLT là một đối tượng dữ liệu động (có thể được bổ sung hay loại bỏ trong lúc chạy chương trình).
  - Linh hoạt trong tổ chức cấu trúc: các con trỏ trong cấu trúc có thể tùy ý thay đổi địa chỉ chỉ đến
  - Truy nhập tuần tự: Có ít nhất một điểm truy nhập = con trỏ P, tốc độ truy nhập vào các phần tử của CTLT không đồng đều

2.3.2 Danh sách móc nối - Linked List

- Mô tả:
  - 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:
    - 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

2.3.2 Danh sách móc nối

- Ví dụ

H

1

	Data	Next
1	H	4
2		
3		
4	E	7
5		
6		
7	L	8
8	L	10
9		
10	O	–1

"HELLO"

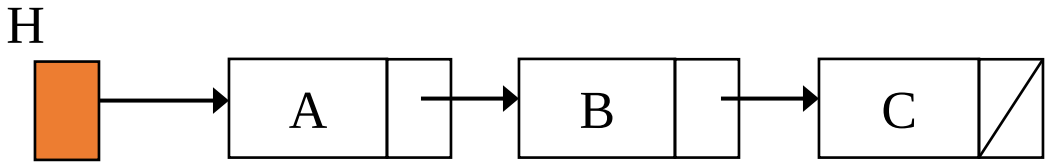
H

18

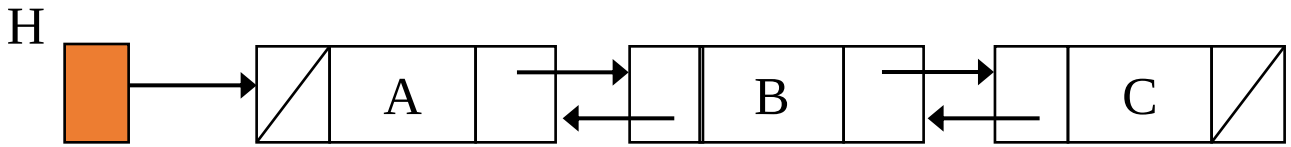
	Roll No	Name	Aggregate	Grade	Next
1	S01	Ram	78	Distinction	6
2	S02	Shyam	64	First division	14
3					
4	S03	Mohit	89	Outstanding	17
5					
6	S04	Rohit	77	Distinction	2
7	S05	Varun	86	Outstanding	10
8	S06	Karan	65	First division	12
9					
10	S07	Veena	54	Second division	–1
11	S08	Meera	67	First division	4
12	S09	Krish	45	Third division	13
13	S10	Kusum	91	Outstanding	11
14	S11	Silky	72	First division	7
15					
16					
17	S12	Monica	75	Distinction	1
18	S13	Ashish	63	First division	19
19	S14	Gaurav	61	First division	8

2.3.2 Danh sách móc nối

- Phân loại
  - 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

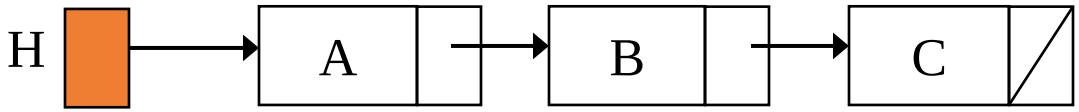


    - Danh sách nối kép (*double linked list*)
      - 2 con trỏ chỉ theo hai hướng trong danh sách

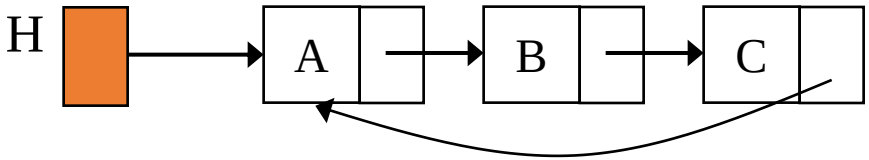


2.3.2 Danh sách móc nối

- Phân loại
  - Phân loại theo cách móc nối vòng hoặc thẳng
    - Danh sách nối thẳng (*grounded linked list*): truy cập vào danh sách thông qua điểm truy nhập H

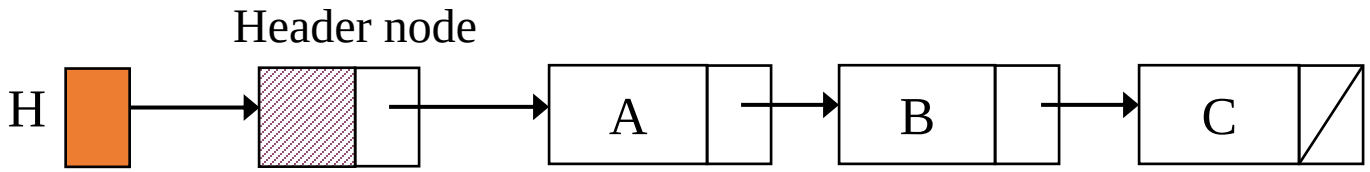


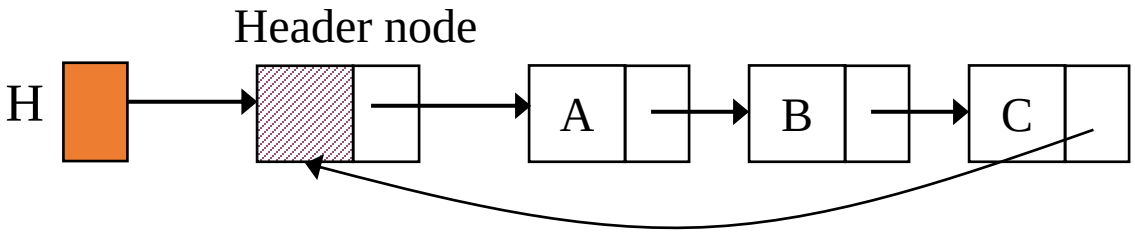
    - Danh sách nối vòng (*circular linked list*): bất cứ nút nào trong danh sách cũng có thể coi là nút đầu hay nút cơ sở (mọi nút có vai trò như nhau)



2.3.2 Danh sách móc nối

- Phân loại
  - *Header Linked Lists*





H → 5

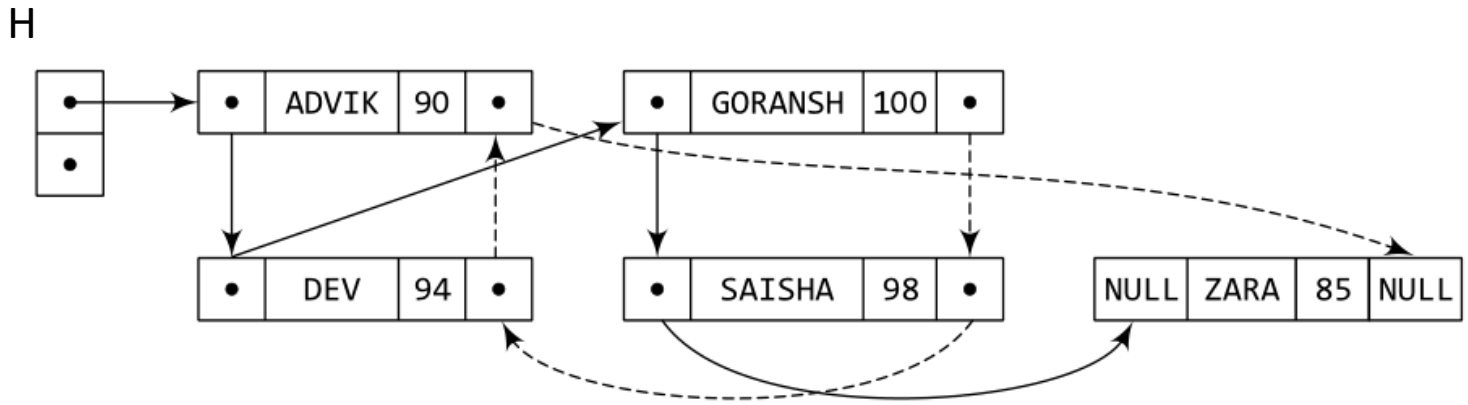
	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	0	−1

H → 5

	DATA	NEXT
1	H	3
2		
3	E	6
4		
5	1234	1
6	L	7
7	L	9
8		
9	0	5

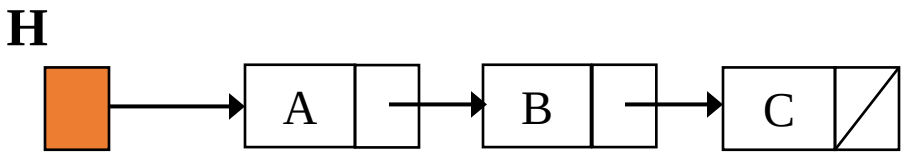
2.3.2 Danh sách móc nối

- Phân loại
  - *Multi-linked list*



Name: thứ tự tăng dần
Grade: thứ tự giảm dần

2.3.3 Cài đặt danh sách bằng CTLT móc nối

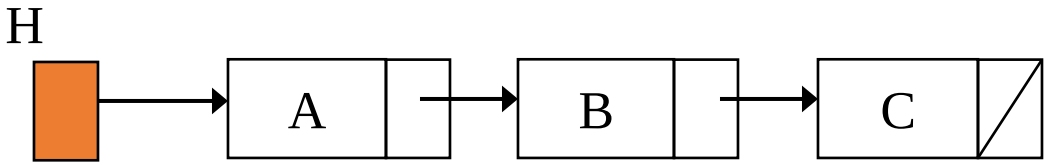


- Khai báo cấu trúc nút:

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

CÀI ĐẶT danh sách NỐI ĐƠN

- Danh sách nối đơn thẳng (*single linked list*)




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

- Nút cuối : next = NULL
- Danh sách rỗng: H = NULL;
- Danh sách đầy: khi không còn đủ bộ nhớ để cấp phát

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 chèn một nút mới vào danh sách
      - Chèn vào đầu danh sách
      - Chèn vào cuối danh sách
      - Chèn vào trước/sau một vị trí trong danh sách
    - Phép xoá nút khỏi danh sách
      - Xóa ở đầu danh sách
      - Xóa ở cuối danh sách
      - Xóa ở trước/sau một vị trí trong danh sách
      - Xóa tất cả các nút
    - Phép tìm kiếm nút có dữ liệu = x
    - Phép duyệt danh sách: *Traverse*
    - Đếm số nút

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);
}
```

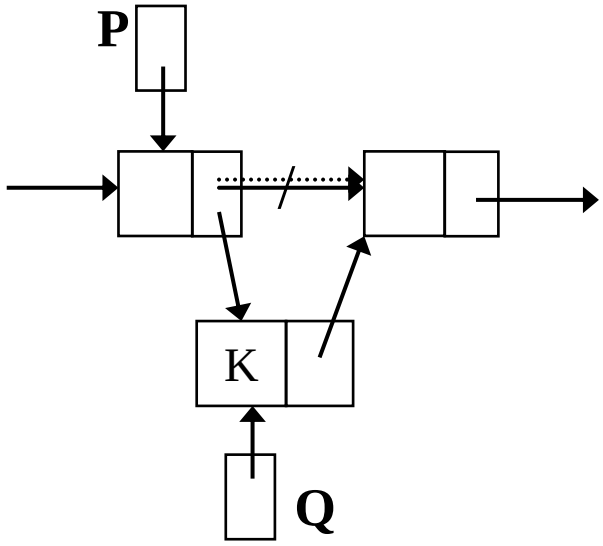
- Phép duyệt danh sách (có thể ứng dụng vào xử lý phần tử của danh sách):

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

CÀI ĐẶT danh sách NỐI ĐƠN

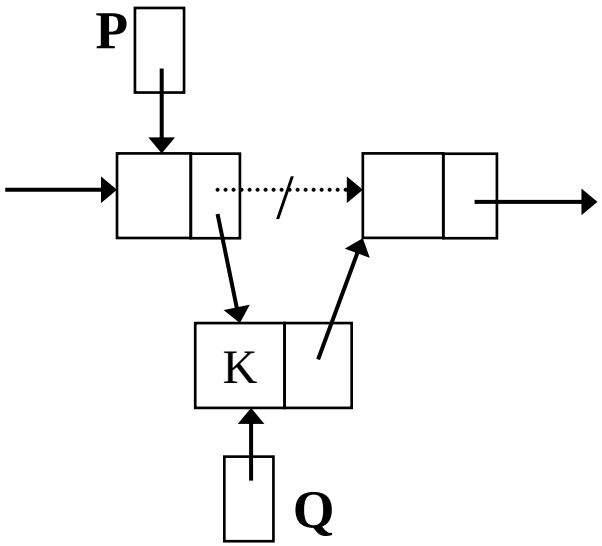
- Bổ sung một phần tử mới K vào sau phần tử hiện tại được trỏ bởi P trong d/s H. Trả về con trỏ trỏ vào nút vừa bổ sung

```
PNode InsertAfter(LinkedList H, PNode P, int K){
      Insert a new node Q contains value K
      if H is empty then
            Q->next = NULL;
            H=Q;
      else:
            Q->next=P->next;
            P->next = Q;
      return Q;
}
```



CÀI ĐẶT danh sách NỐI ĐƠN

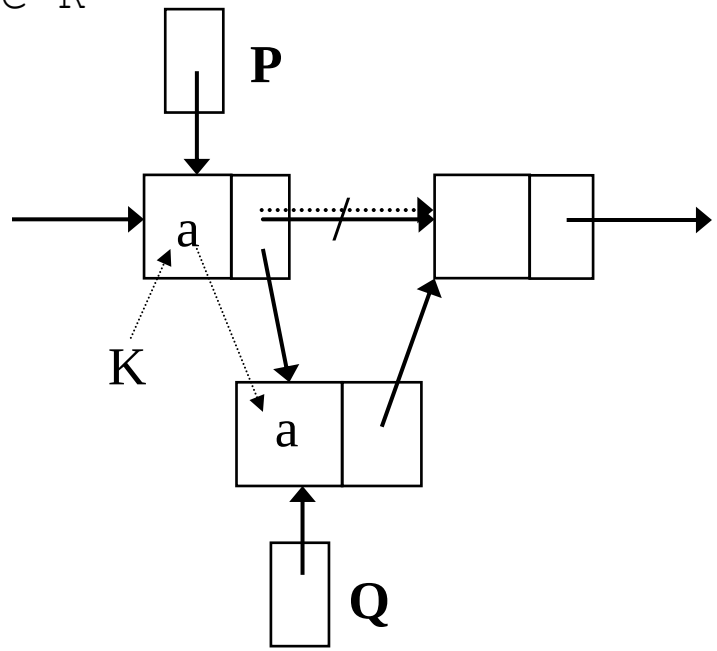
```
1.  PNode InsertAfter(LinkedList H, PNode P, int K){
2.    PNode Q = new Node;
3.    Q->info = K;
4.   if (P==NULL){
5.     return NULL;
6.   }else if (H==NUL){
7.          H = Q;
8.          Q->next = NULL;
9.    }else {
10.         Q->next = P->next;
11.         P->next = Q;
12.   }
13.   return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



CÀI ĐẶT danh sách NỐI ĐƠN

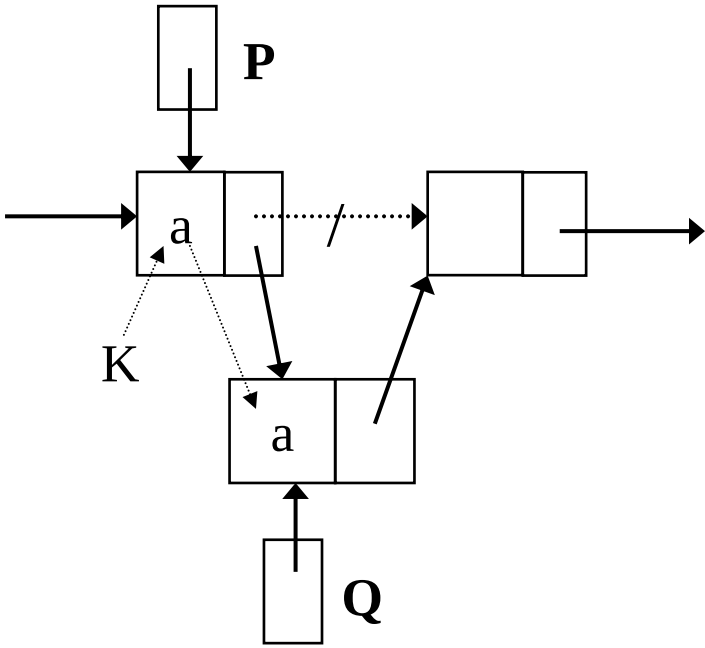
- Chèn một phần tử mới K vào vị trí phần tử hiện tại được trỏ bởi P trong d/s H, đẩy các phần tử đuôi lùi xuống. Trả về con trỏ trỏ vào nút vừa bổ sung

```
PNode InsertAtP(LinkedList H, PNode P, int K) {
    Insert a new node Q containing value K
    if H is empty then
       Q->next = NULL;
       H=Q;
       return Q;
    else:
       Move value a from P to Q
       Update value of P to K
       Q->next = P->next;
       P->next=Q;
       return P;
}
```

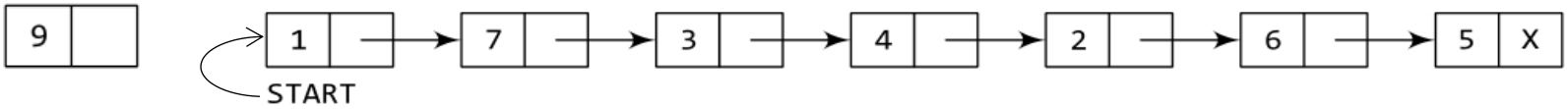


CÀI ĐẶT danh sách NỐI ĐƠN

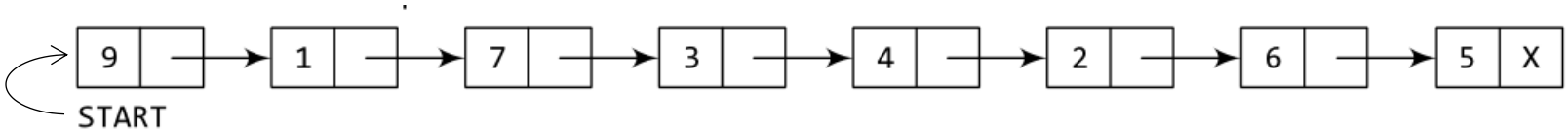
```
1.  PNode InsertAtP(LinkedList H, PNode P, int K){
2.    PNode Q = new Node;
3.    Q->info = K;
4.    if (H==NULL || P==NULL){
5.          H = Q;
6.          Q->next = NULL;
7.          return Q;
8.    }else {
9.          Q->info = P->info;
10.         P->info = K;
11.         Q->next = P->next;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



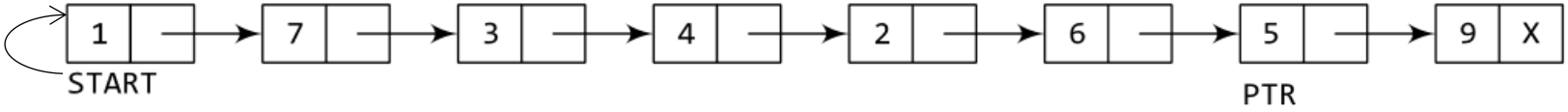
CÀI ĐẶT danh sách NỐI ĐƠN



- Bổ sung phần tử mới vào đầu danh sách



- Bổ sung phần tử mới vào sau cùng danh sách



CREATE NEW_NODE

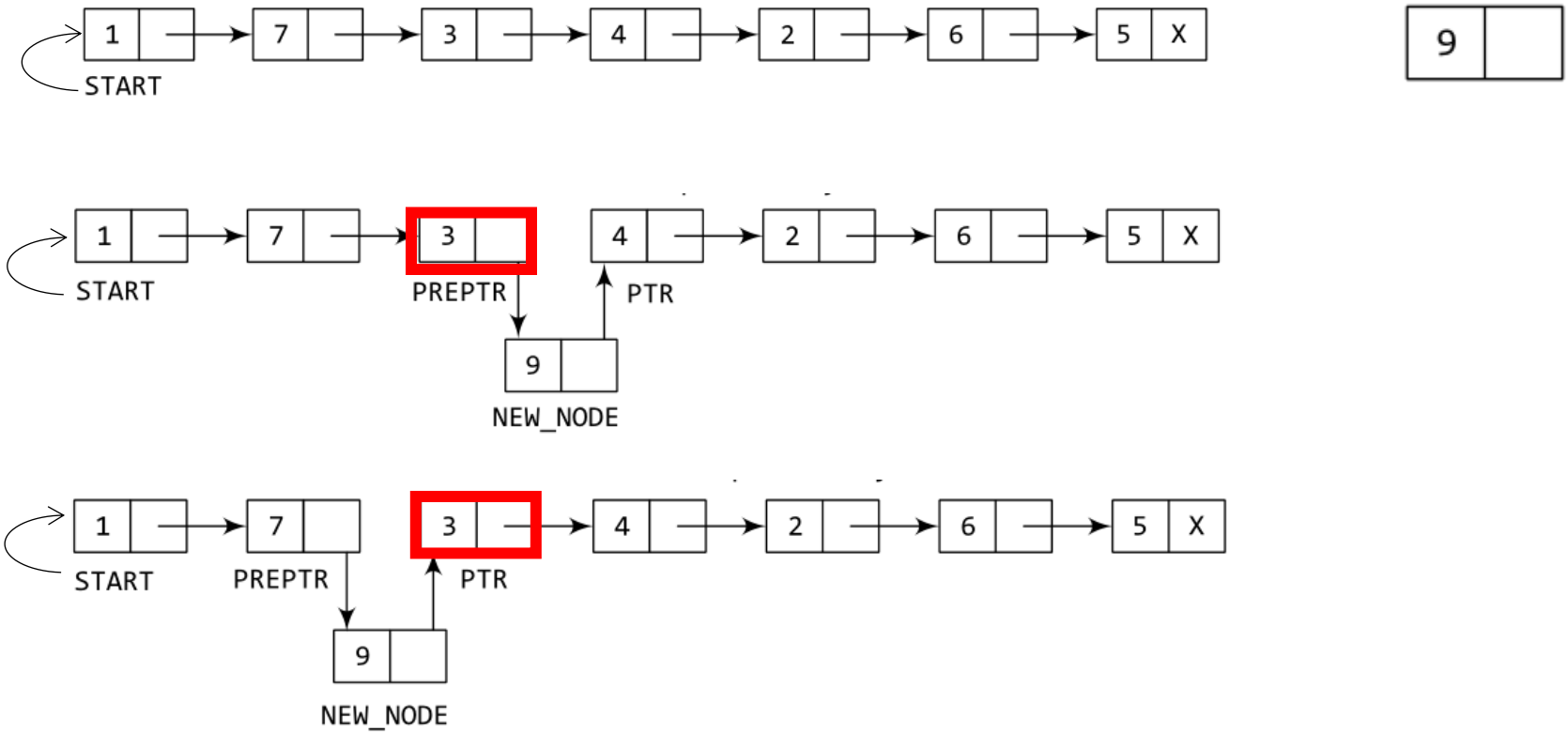
```
SET NEW_NODE -> DATA = VAL
SET NEW_NODE -> NEXT = START
SET START = NEW_NODE
```

CREATE NEW_NODE

```
SET NEW_NODE -> DATA = VAL
SET NEW_NODE -> NEXT = NULL
SET PTR = START
Repeat          while PTR -> NEXT != NULL
    SET PTR = PTR -> NEXT
[END OF LOOP]
SET PTR -> NEXT = NEW_NODE
```

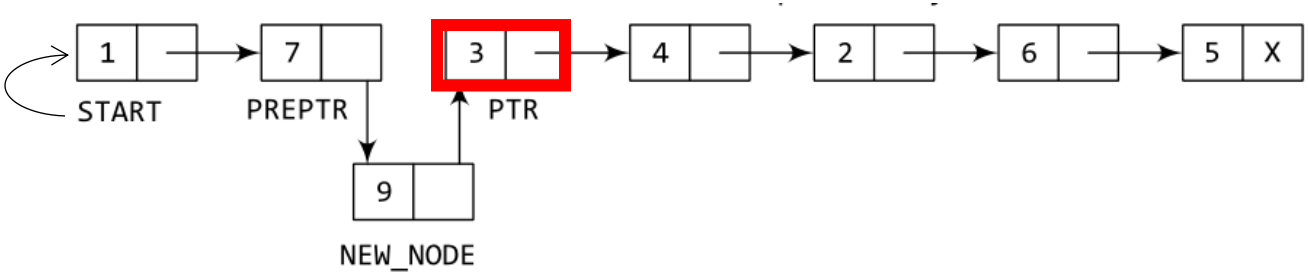
CÀI ĐẶT danh sách NỐI ĐƠN

- Bổ sung phần tử mới vào sau/trước 1 nút có giá trị X



CÀI ĐẶT danh sách NỐI ĐƠN

- Bổ sung phần tử mới vào sau/**trước** 1 nút có giá trị X

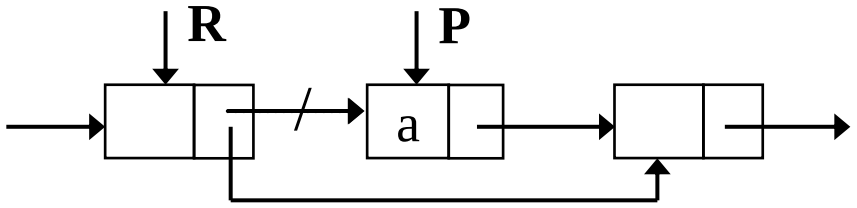



```
CREATE NEW_NODE
SET NEW_NODE->DATA = VAL
SET PTR = START
SET PREPTR = PTR
Repeat                  while PTR->DATA != NUM
    SET PREPTR = PTR
    SET PTR = PTR->NEXT
[END OF LOOP]
 PREPTR->NEXT = NEW_NODE
 SET NEW_NODE->NEXT = PTR
```

CÀI ĐẶT danh sách NỐI ĐƠN

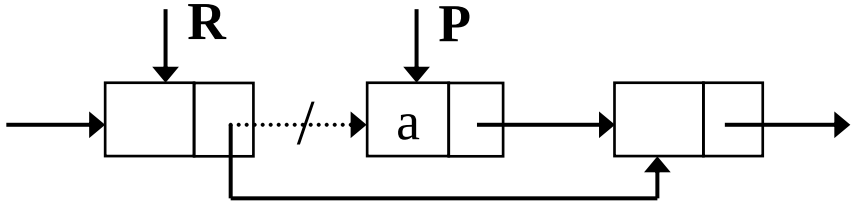
- Xóa phần tử hiện tại mà con trỏ P trỏ tới trong danh sách H.

```
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;
}
```

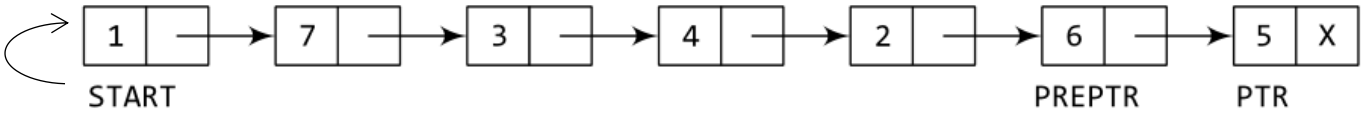


CÀI ĐẶT danh sách NỐI ĐƠN

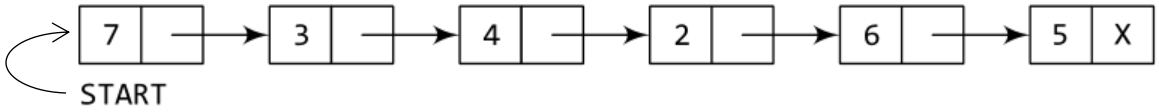
```
PNode DeleteNode(LinkedList H, PNode P){
    if (P==NULL) return NULL;
    if (H==P && P->next==NULL){
        H=NULL; delete P;
        return NULL;
    } else {
        if (H==P){  //P la nut dau tien
            H=P->next;
            P=NULL;
            return H;
        }else {
            PNode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

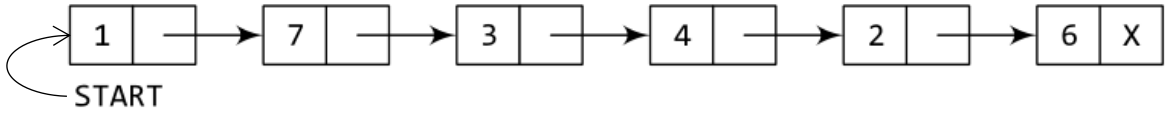


CÀI ĐẶT danh sách NỐI ĐƠN

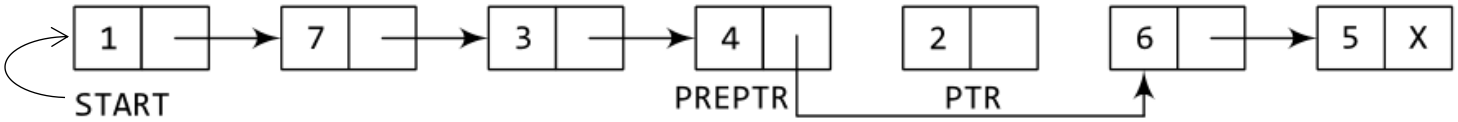


- Xóa phần tử đầu/cuối danh sách



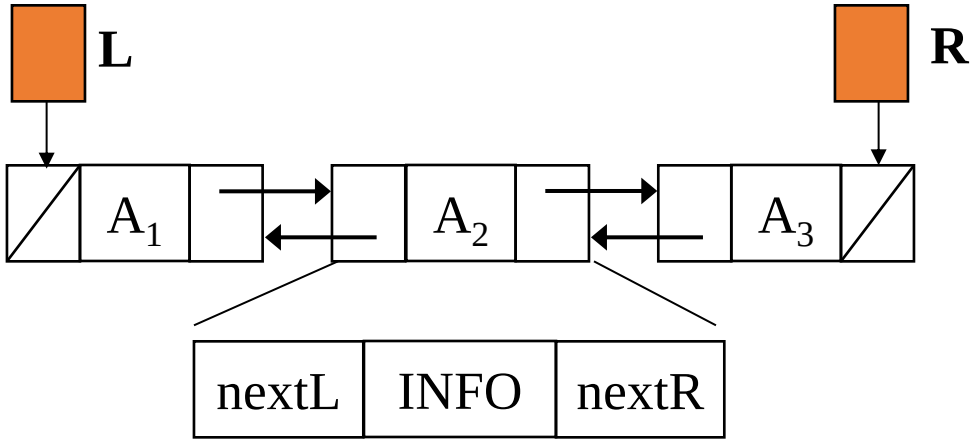


- Xóa phần tử ở sau/trước 1 nút cho giá trị X



Danh sách NỐI KÉP

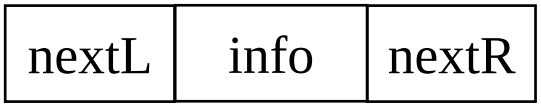
- Mô tả
  - Danh sách nối kép thẳng (***double linked list***):
    - Con trỏ trái (nextL): trỏ tới thành phần bên trái (phía trước)
    - Con trỏ phải (nextR): trỏ tới thành phần bên phải (phía sau)
  - Đặc điểm
    - Sử dụng 2 con trỏ, giúp ta luôn xem xét được cả 2 chiều của danh sách
    - Tốn bộ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

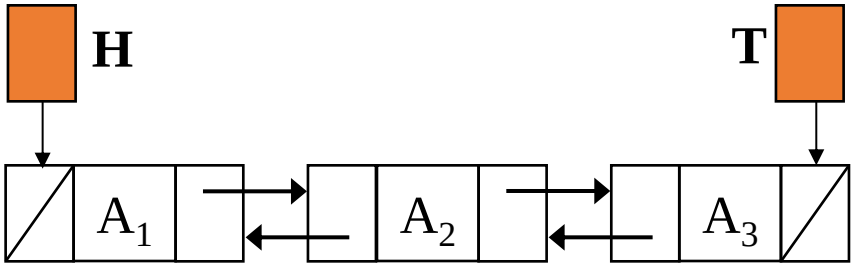


Danh sách NỐI KÉP

- Mô tả
  - Danh sách nối kép thẳng (*double linked list*):
    - Khai báo cấu trúc

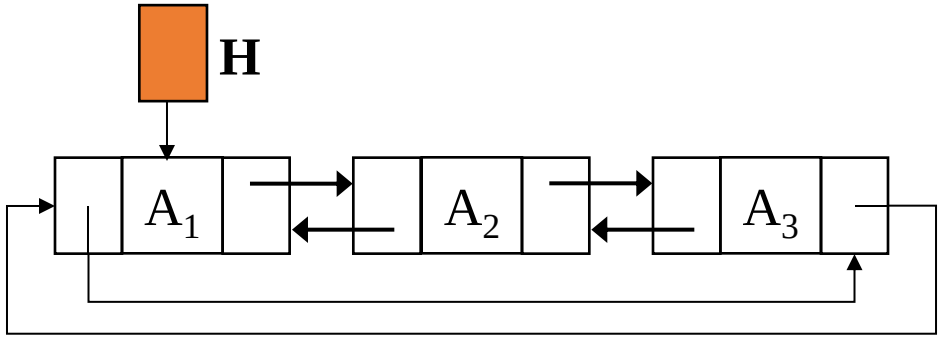
```
struct  DNode {
    Type info;
    DNode *nextL, *nextR;
};

typedef DNode*  PDNode;

typedef struct {
    PDNode  H; //con trỏ đầu
    PDNode  T; //con trỏ cuối
} DoubleLinkedList;
```





Danh sách NỐI KÉP

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




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

Danh sách NỐI KÉP

- Ví dụ

H

1 →

	DATA	PREV	NEXT
1	H	−1	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	−1

H

1 →

	DATA	PREV	Next
1	H	9	3
2			
3	E	1	6
4			
5			
6	L	3	7
7	L	6	9
8			
9	0	7	1

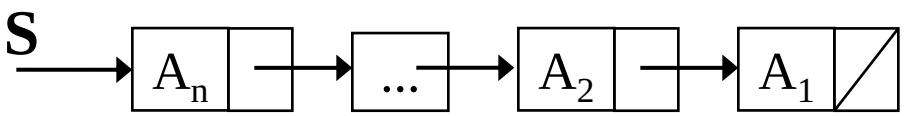
Danh sách NỐI KÉP

- Các phép toán chung:
  - Khởi tạo danh sách
  - Phép chèn một nút mới vào danh sách
  - Phép xoá nút khỏi danh sách
  - Phép tìm kiếm nút
  - Phép duyệt danh sách

2.3.4 Cài đặt LIFO, FIFO bằng CTLT móc nối

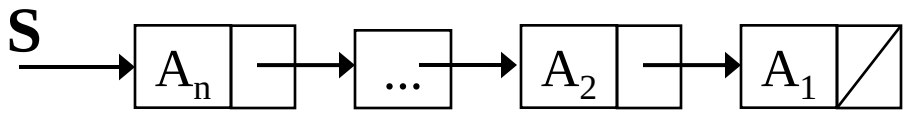
- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Cách tổ chức:
    - Chỉ cần một con trỏ S vừa là đỉnh, vừa là điểm truy nhập
  - Khai báo cấu trúc

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



Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ 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***

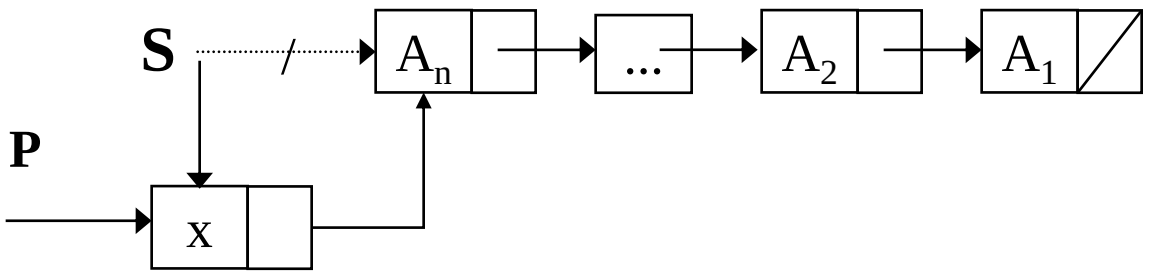



```
void Initialize (Stack S){
    S = NULL;
}
bool isEmpty (Stack S){
    return (S==NULL);
}
```

Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

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

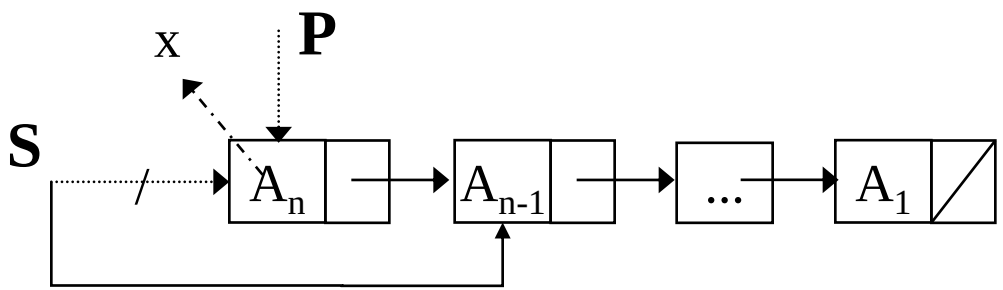
```
void Push (Type x, Stack S){
   Pnode P;
   P = new Node;
   P->info = x;
   P->next = S;
   S = P;
}
```



Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

- Biểu diễn LIFO hay ngăn xếp (Stack)
  - Các phép toán: loại bỏ 1 phần tử

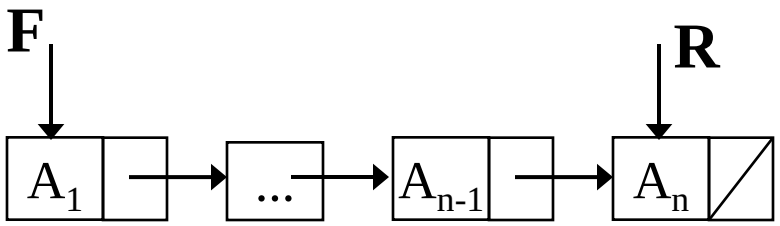
```
Type Pop (Stack S) {
   Item x ;
   if  (isEmpty (S)) return NULL;
   else {
     x = S->info;
     S = S->next;
     return x;
   }
}
```



Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

- Biểu diễn FIFO hay hàng đợi (Queue)
  - Khai báo cấu trúc

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



Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ 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***

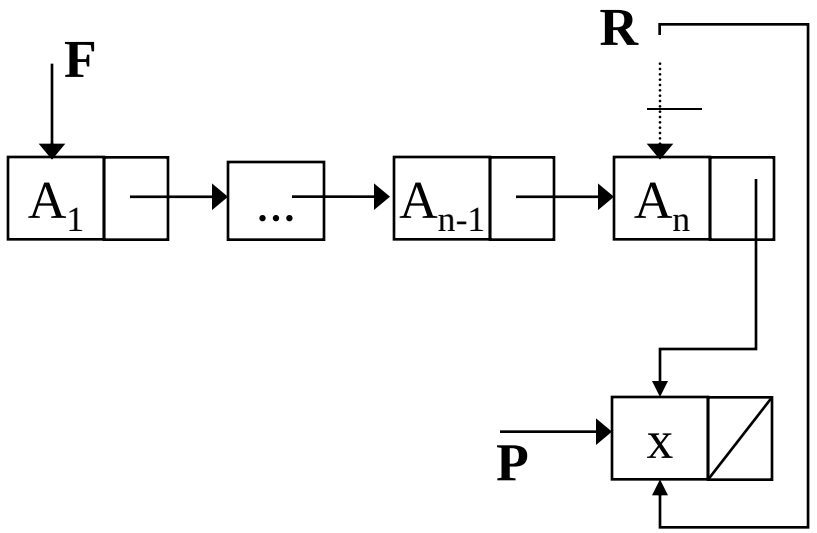
```
void Initialize (Queue Q){
    Q.F = Q.R = NULL;
}
```

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

Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

- Biểu diễn FIFO hay hàng đợi (Queue)

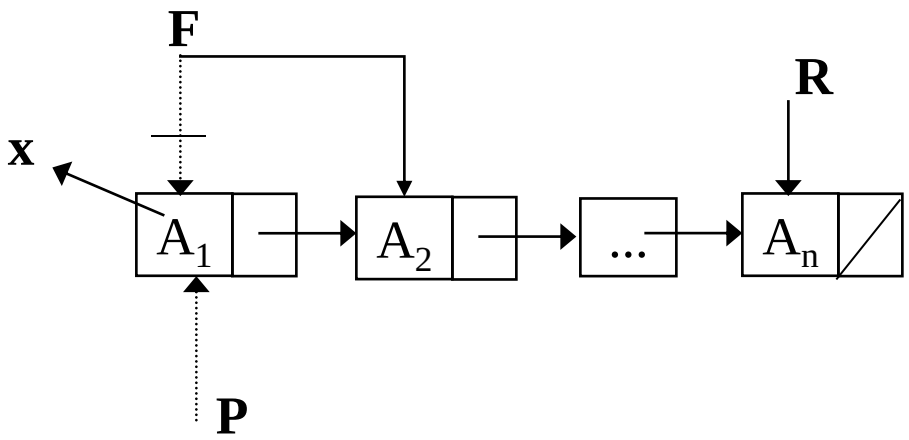
```
void InsertQ (Info x, Queue Q){
   Pnode P;
   P = new Node;
   P->info = x;
   P->next = NULL;
   if  (isEmpty (Q)) {
    Q.F = Q.R = P;
   }
   else{
    Q.R->next = P;
    Q.R = P;
   }
}
```



Cài đặt LIFO, FIFO bằng cấu trúc lưu trữ móc nối

- Biểu diễn FIFO hay hàng đợi (Queue)

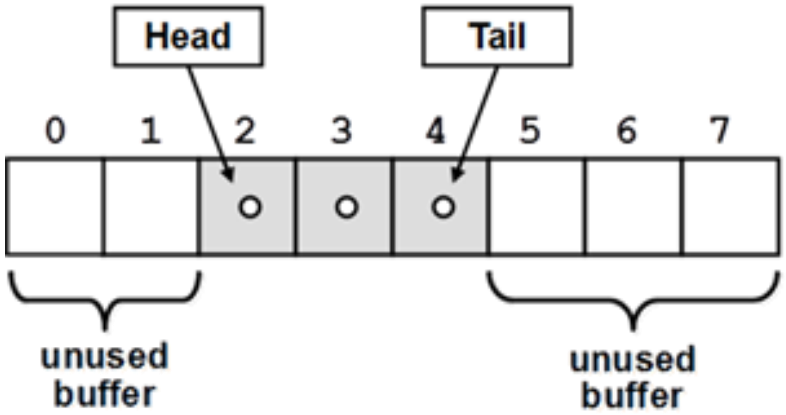
```
Info DeleteQ (Queue Q){
   Info x;
   if  (isEmpty (Q)) return;
   else {
    x = Q.F->info;
    Q.F = Q.F->next;
    return x;
   }
}
```

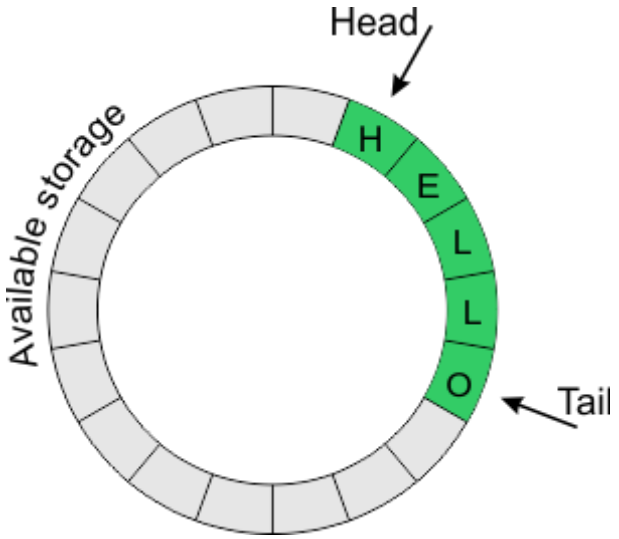


2.4. Một số ứng dụng của ngăn xếp và hàng đợi

- Ứng dụng của stack
  - Thứ tự các lệnh gọi hàm trong chương trình máy tính
  - Hoàn tác thao tác (undo) trong trình chỉnh sửa
  - Lịch sử trong trình duyệt web
  - Triển khai các hàm đệ quy
  - …

- Ứng dụng của queue
  - Truyền dữ liệu và bộ đệm giao tiếp
  - Bộ đệm đọc / ghi
  - Triển khai các dịch vụ máy khách / máy chủ





Tổng kết

- 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 => 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.
  - 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

Tổng kết

Operation	Array	Linked List
Insert to head	O(n)	O(1)
Insert to tail	O(n)	O(n)
Insert in the middle	**O(n)**	**O(1)**
Delete at head	O(n)	O(1)
Delete at tail	O(n)	O(n)
Delete in the middle	**O(n)**	**O(1)**
Delete all	O(1)	O(1)
Get element by index	**O(1)**	**O(n)**
Count elements	O(1)	O(n)
Iterate over elements	**O(n)**	**O(n)**

Bài tập – Tuần 4

- Bài 1: Viết chương trình triển khai queue sử dụng danh sách liên kết (và các hàm khởi tạo, thêm phần tử, xóa phần tử, lấy ra phần tử đầu tiên, hiển thị)
- Bài 2: Cài đặt danh sách nối kép thẳng. 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, nhập các nút vào danh sách, chèn 1 nút vào đầu/cuối danh sách, xóa 1 nút ở đầu/cuối danh sách, chèn 1 nút vào trước/sau 1 nút có giá trị X, xóa 1 nút ở trước/sau 1 nút có giá trị X, xóa toàn bộ danh sách.

Bài tập mở rộng

- **Điều chỉnh tổ chức của danh sách liên kết (*DSLK*)** Viết thuật toán lấy số nút của DSLK. Cho biết độ phức tạp của thuật toán.
  - Nếu bổ sung biến *_count* để lưu số nút thì cần cải thiện cài đặt DSLK thế nào?
- **Đổi cơ số đếm** Dùng danh sách liên kết để đổi cơ số đếm
  - *Đầu vào:* số nguyên không âm
  - *Đầu ra:* dãy các giá trị nhị phân
- **Phân tích xâu ký tự** Phân tích xâu ký tự thành các từ không rỗng
  - *Đầu vào:* xâu ký tự
  - *Đầu ra:* dãy các từ