CHƯƠNG 3
DANH SÁCH LIÊN KẾT

1. Giới thiệu về danh sách liên kết

- Danh sách liên kết là danh sách tuyến tính khi sử dụng cấu trúc lưu trữ phân tán. Các phần tử dữ liệu của danh sách được lưu trữ trong các phần tử nhớ mà ta gọi là nút (node). Trong mỗi nút nhớ, ngoài phần tử dữ liệu còn có địa chỉ của nút lân cận.
- Nếu giữa các nút nhớ có 1 liên kết thì ta có DSLK đơn, nếu giữa các nút có 2 liên kết thì ta có DSLK kép.

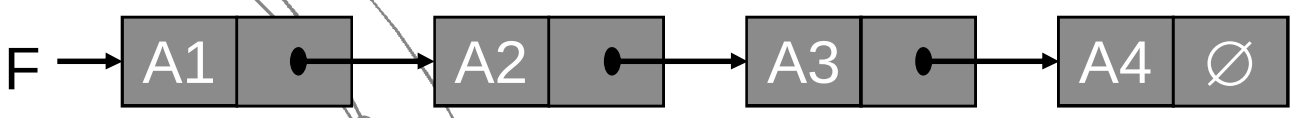
2. Danh sách liên kết đơn

2.1. Quy tắc tổ chức danh sách liên kết đơn

- Trong DSLK đơn, mỗi nút nhớ có cấu trực gồm hai trường, trường INFOR chứa phần tử dữ liệu và trường LINK chứa địa chỉ của nút đứng sau.

2.1. Quy tắc tổ chức danh sách liên kết đơn (*tiếp*)

- Nút cuối cùng trong danh sách không có nút đứng sau nên trường địa chỉ là rỗng, không chứa địa chỉ, ta ký hiệu là ∅.
- Dùng con trỏ F chứa địa chỉ nút đầu tiên để cho phép truy nhập vào tất cả nút trong danh sách.
- Khi danh sách rỗng thì F = ∅



2.1. Quy tắc tổ chức danh sách liên kết đơn (*tiếp*)

- Để tổ chức lưu trữ một danh sách liên kết thì phải có:
  - Phải có phương tiện chia bộ nhớ ra thành các nút và ở mỗi nút có thể truy nhập vào từng trường.
  - Phải có cơ chế để xác định một nút đang được sử dụng hoặc chưa được sử dụng (nút trống).
  - Phải có cơ chế cung cấp các nút trống khi có yêu cầu sử dụng và thu hồi lại các nút khi không cần dùng nữa.
- Ta ký hiệu:
  - P $\Leftarrow$ AVAIL là phép lấy ra một nút trống có địa chỉ là P (cấp phát một nút)
  - P $\Rightarrow$ AVAIL là phép thu hồi một nút có địa chỉ là P

2.2. Một số phép toán trên danh sách liên kết đơn

- Ký hiệu: Một nút có địa chỉ là p (được trỏ bởi p) thì Infor(p) và Link(p) tương ứng chỉ trường Infor và Link của nút đó.

a) Bổ sung một nút mới vào danh sách

Cho danh sách liên kết đơn F, M là con trỏ trỏ tới một nút trong danh sách. Viết thủ tục bổ sung phần tử dữ liệu x **vào sau** nút M.

2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

a) Bổ sung một nút mới vào danh sách:

- Vào: F, M, x
- Ra: Không có

{Thủ tục này bổ sung phần tử dữ liệu x **vào sau** nút trỏ bởi M trong danh sách liên kết đơn F}

Procedure SLInsert(Var F; M, x)

1. {Tạo nút mới}

   N ⇐ AVAIL

   infor(N):=x; link(N):= ∅;

2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

2. {Thực hiện bổ sung: Nếu danh sách rỗng thì bổ sung nút mới vào thành nút đầu tiên. Nếu danh sách không rỗng thì bổ sung nút mới vào sau nút M}

   If F=∅ then F := N

   Else begin

```
            LINK(N) := LINK(M);
            LINK(M) := N;
      end;
Return
```

2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

b) Loại bỏ một nút khỏi danh sách

- Vào: F, M
- Ra: Không

{Thủ tục này loại bỏ nút trỏ bởi M khỏi danh sách liên kết đơn F}

```
Procedure SLDelete(Var  F;  M)
 1. { Trường hợp danh sách rỗng}
  If F=∅  then begin
       Write('danh sách rỗng')
       Return
        end
```


2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

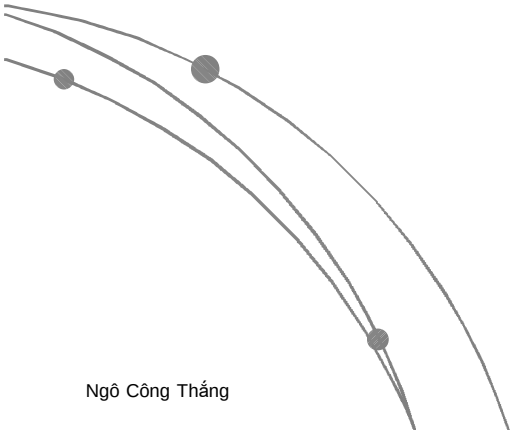
```
2. {Thay đổi liên kết, ngắt kết nối với nút M}
   {M là nút đầu tiên của danh sách}
   If M=F then F:=LINK(F)  Else
   begin
        {Tìm đến nút đứng trước nút M }
        P:=F;
        While LINK(P) # M do  P:=LINK(P);
        {Nối nút trước M với nút sau M}
        LINK(P):=LINK(M);
   end;
```

2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

3. {Hủy nút M}

M $\Rightarrow$ AVAIL;

Return



2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

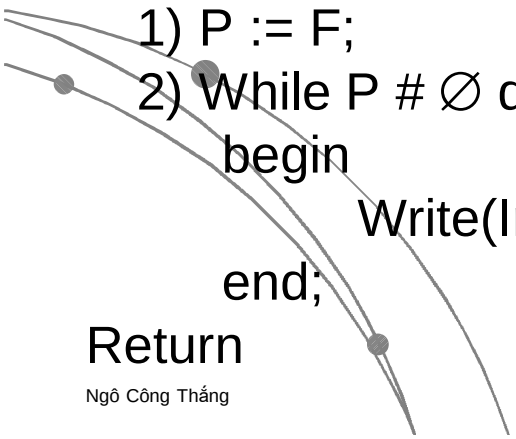
c) Duyệt danh sách

- Vào: F

- Ra: Không

{Thủ tục này duyệt danh sách liên kết đơn F và đưa ra các phần tử dữ liệu trong ds}

```
Procedure SLDisplay(F)
    1) P := F;
    2) While P # ∅ do
        begin
            Write(Infor(P)); P := Link(P);
        end;
Return
```



2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

d) Ghép hai danh sách liên kết đơn

Cho 2 danh sách liên kết đơn lần lượt trỏ bởi P và Q, ghép 2 danh sách trở thành một danh sách và cho P trỏ tới. Thuật toán có các bước sau:

```
Procedure SLConcat(P,Q)
  1. {Danh sách trỏ bởi q rỗng}
     If  Q = ∅ then Return
  2. {Trường hợp danh sách trỏ bởi p rỗng}
     If  P = ∅ then begin
         P:=Q
         return
     end
```


2.2. Một số phép toán trên danh sách liên kết đơn (*tiếp*)

d) Ghép hai danh sách liên kết đơn

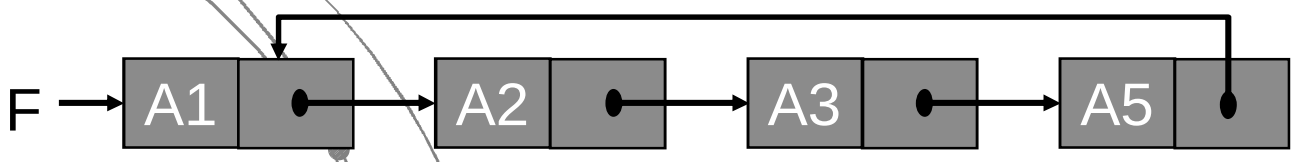
```
  3. {Tìm đến nút cuối danh sách p}
     P1:= P
     While  link(P1) # ∅ do P1:=link(P1);
  4. {Ghép}
     Link(P1):=Q;
  Return
```

Ưu nhược điểm của danh sách liên kết đơn

- Với danh sách tuyến tính động, trong quá trình xử lý luôn có bổ sung, loại bỏ thì tổ chức danh sách liên kết là hợp lý, tận dụng được các vùng nhớ nằm rải rác trong bộ nhớ.
- Chỉ có phần tử đầu tiên là truy nhập ngay được, các phần tử khác phải truy nhập qua phần tử đứng trước nó.
- Tốn bộ nhớ do phải lưu cả 2 trường infor và link ở mỗi nút.

3. Danh sách liên kết vòng

- Danh sách liên kết vòng (Circularly Linked List) là một dạng cải tiến của danh sách liên kết đơn.
- Trong danh sách liên kết vòng, trường địa chỉ của nút cuối cùng không phải là rỗng mà lại chứa địa chỉ của nút đầu tiên của danh sách.

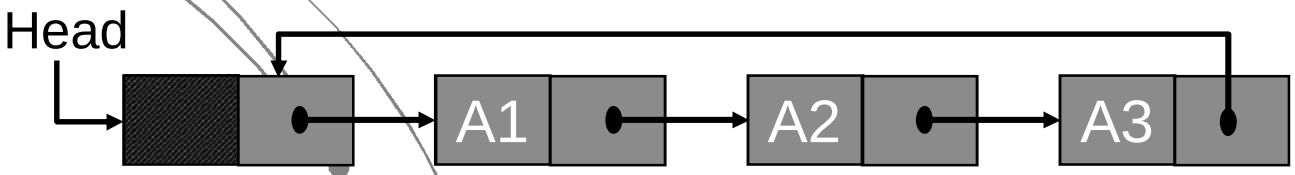


3. Danh sách liên kết vòng (*tiếp*)

- Ưu nhược điểm của danh sách nối vòng:
  - Danh sách nối vòng làm cho việc truy nhập vào các nút trong danh sách linh hoạt hơn. Ta có thể truy nhập vào danh sách bắt đầu từ một nút nào cũng được, không nhất thiết phải từ nút đầu tiên. Nút nào cũng có thể là nút đầu tiên và con trỏ F trỏ vào nút nào cũng được.
  - Nhược điểm của danh sách nối vòng là trong xử lý nếu không cẩn thận sẽ dẫn tới một chu trình không kết thúc.

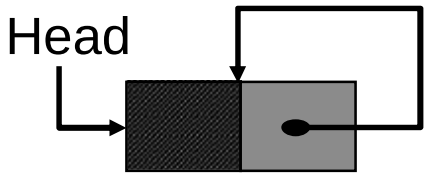
3. Danh sách liên kết vòng (*tiếp*)

- Để khắc phục nhược điểm của danh sách nối vòng ta đưa thêm vào một nút đặc biệt gọi là “nút đầu danh sách” (list head node). Trường Infor của nút này không chứa dữ liệu, con trỏ HEAD trỏ tới nút đầu danh sách này cho phép ta truy nhập vào danh sách.



3. Danh sách liên kết vòng (*tiếp*)

- Việc dùng thêm nút đầu danh sách đã làm cho danh sách luôn có ít nhất 1 nút nên không bao giờ rỗng. Danh sách có 1 nút HEAD có LINK(Head)= Head.



- Các phép toán bổ sung và loại bỏ nút trong danh sách liên kết vòng tương tự danh sách liên kết đơn .

4. Danh sách liên kết kép

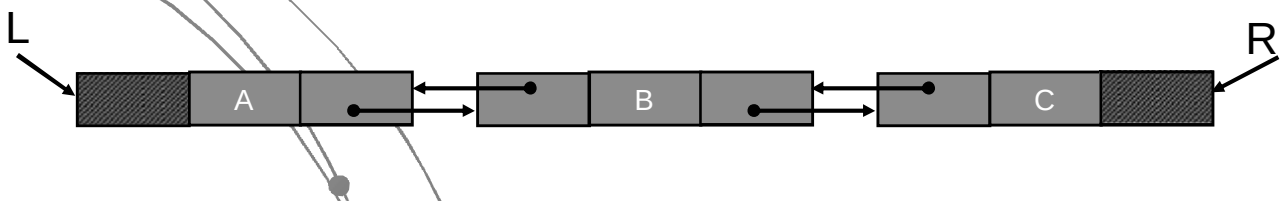
4.1. Giới thiệu

- Trong danh sách liên kết kép, mỗi nút nhớ có cấu trúc gồm 3 trường như sau:

Left: Con trỏ trỏ tới nút đứng trước
Right: Con trỏ trỏ tới nút đứng sau
Infor: Trường thông tin.

4.1. Giới thiệu (tiếp)

- Left của nút cực trái và Right của nút cực phải có giá trị là ∅.
- Để truy nhập vào danh sách cả 2 chiều ta phải dùng 2 con trỏ: Con trỏ L trỏ vào nút cực trái, con trỏ R trỏ vào nút cực phải.
- Khi danh sách rỗng thì L = R = ∅.



4.2. Các phép toán trên danh sách liên kết kép

a) Chèn thêm một nút vào danh sách

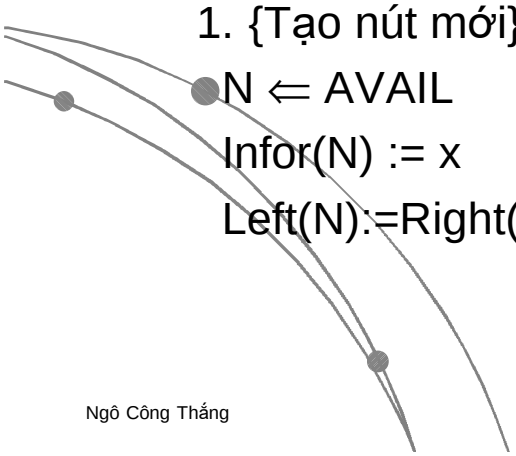
- Cho danh sách liên kết kép (L, R). M là con trỏ trỏ tới một nút trong danh sách. Bổ sung phần tử dữ liệu x **vào trước** nút M.

4.2. Các phép toán trên danh sách liên kết kép

- Vào: (L,R),M,x
- Ra: Không có

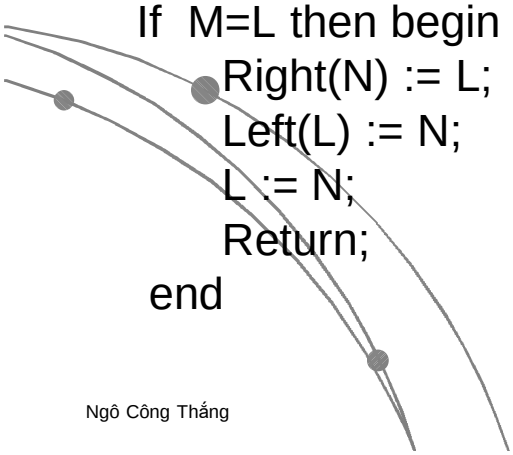
{Thủ tục này bổ sung phần tử x **vào trước nút M** trong DSLK kép (L, R)}

```
Procedure  DLInsert(Var  L,R;  M, x)
   1. {Tạo nút mới}
      N ⇐ AVAIL
      Infor(N) := x
      Left(N):=Right(N):= ∅
```



a) Chèn thêm một nút vào danh sách

```
2. {Trường hợp danh sách rỗng}
   If  L=R=∅  then begin
         L := R := N ;
         Return;
   end
3. {M trỏ tới nút cực trái}
   If  M=L then begin
         Right(N) := L;
         Left(L) := N;
         L := N;
         Return;
    end
```



a) Chèn thêm một nút vào danh sách

4. {Trường hợp còn lại}
   Right(Left(M)) := N
   Left(N) := Left(M)
   Right(N) := M
   Left(M) := N
5. {Kết thúc}
Return

b) Loại bỏ một nút ra khỏi danh sách liên kết kép

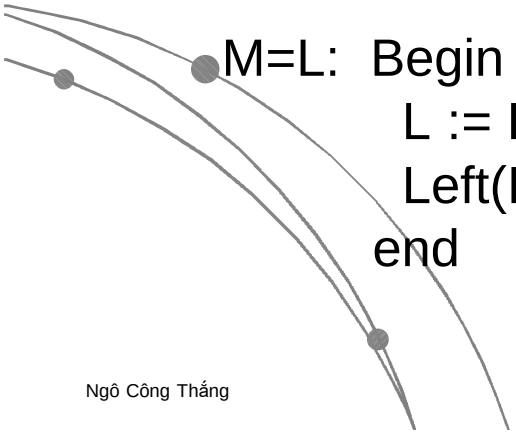
- Cho danh sách liên kết kép L, R. M là con trỏ trỏ tới một nút trong danh sách cần loại bỏ.
- Vào: (L,R), M
- Ra: Không có

{Thủ tục này loại bỏ nút trỏ bởi M trong DSLK kép L, R}

```
Procedure DLDelete(Var  L, R;  M)
1. { Trường hợp danh sách rỗng }
If L=R=∅  then  begin
    Write(' danh sach rong ')
    Return
  end
```

b) Loại bỏ một nút ra khỏi danh sách liên kết kép

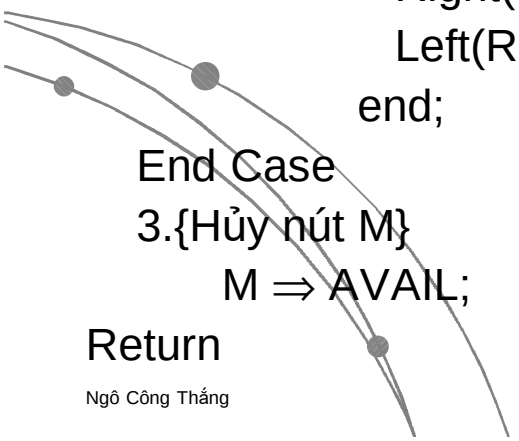
2. {Thay đổi liên kết}
   Case
      M=L= R: Begin {Danh sach chỉ còn 1 nút M}
                L:=R:= ∅;
              end



      M=L: Begin { Nút cực trái bị loại }
              L := Right(L)
              Left(L) := ∅
            end

b) Loại bỏ một nút ra khỏi danh sách liên kết kép

      M=R: Begin { Nút cực phải bị loại }
              R := Left(R)
              Right(R) := ∅
            end
      ELSE: begin
              Right(Left(M)):=Right(M)
              Left(Right(M)):=Left(M)
            end;
   End Case
   3.{Hủy nút M}
      M ⇒ AVAIL;
Return

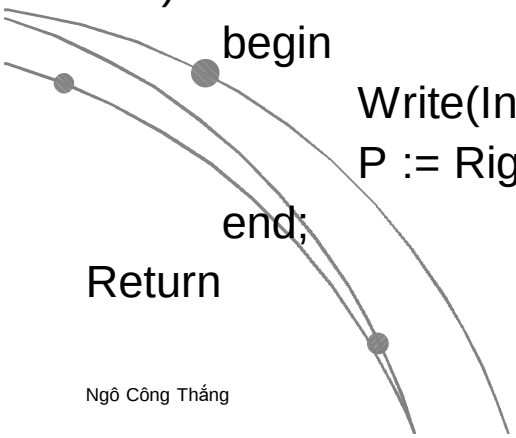


c) Duyệt danh sách liên kết kép và đưa ra các phần tử của danh sách

- Vào: L, R
- Ra: Không có

{Thủ tục này duyệt danh sách từ trái sang phải}

```
Procedute  DLDisplay(L, R);
   1) P:= L;
   2) While  P # ∅  do
         begin
                Write(Infor(P));
                P := Right(P);
         end;
Return
```



5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

5.1. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp

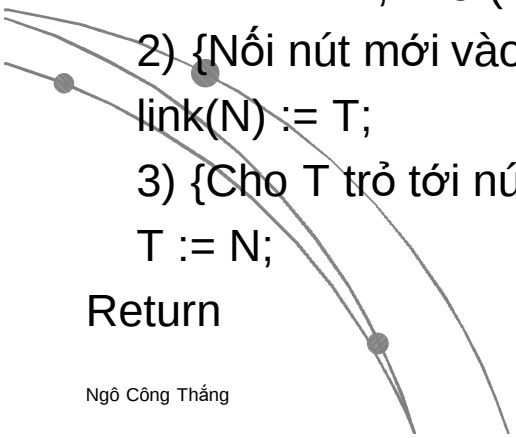
- Các phần tử dữ liệu của ngăn xếp lưu trữ trong các nút nhớ nằm rải rác khắp nơi trong bộ nhớ, mỗi nút nhớ có cấu trực gồm 2 trường
- Nút dưới cùng (nút đáy),

5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

- Vào: T, x
- Ra: Không có

{Thủ tục này bổ sung phần tử x vào ngăn xếp T lưu trữ phân tán}

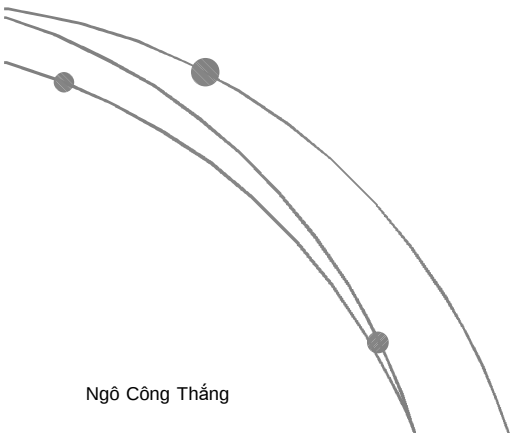
```
Procedure push(Var T;  x)
    1) {Tạo nút mới}
    N <= AVAIL; infor(N) := x; link(N) := ∅;
    2) {Nối nút mới vào trên nút T}
    link(N) := T;
    3) {Cho T trỏ tới nút mới}
    T := N;
Return
```



5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

- Vào: T
- Ra: Phần tử dữ liệu loại bỏ

{Hàm này loại bỏ phần tử đỉnh ngăn xếp T sử dụng cấu trúc lưu trữ phân tán và trả về phần tử này}



5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

```
Function pop(Var T)
   1) {Kiểm tra ngăn xếp rỗng}
        If T = ∅ then begin
                write('Ngan xep da rong'); return;  end;
   2) {Giữ lại phần tử đỉnh sẽ loại bỏ}
        tg := infor(T); P:=T;
   3) {Cho T trỏ xuống nút bên dưới}
        T := link(T);
   4) {Hủy nút đỉnh và trả về phần tử đỉnh}
        P => AVAIL; pop := tg;
Return
```


5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

- Vào: T
- Ra: TRUE nếu ngăn xếp rỗng, FALSE nếu không rỗng

```
{Hàm kiểm tra ngăn xếp T lưu trữ phân tán, trả về TRUE
   nếu n.xếp rỗng và FALSE nếu chưa rỗng}
Function isEmpty(T)
   If T = ∅ then isEmpty:=TRUE;
   Else isEmpty:=FALSE;
Return
```

5. Sử dụng cấu trúc lưu trữ phân tán cho ngăn xếp và hàng đợi

- Vào: T
- Ra: Phần tử dữ liệu đỉnh ngăn xếp

{Hàm này trả về phần tử đỉnh ngăn xếp T lưu trữ phân tán}

```
Function top(T)
    1) {Kiểm tra ngăn xếp rỗng}
        If T = ∅ then begin
                write('Ngan xep da rong'); return;  end;
    2) {Trả về phần tử đỉnh}
        top := infor(T);
Return
```


5.2. Sử dụng cấu trúc lưu trữ phân tán cho hàng đợi

- Trong cấu trúc lưu trữ phân tán, các phần tử dữ liệu của hàng đợi được lưu trữ trong các nút nhớ nằm rải rác khắp nơi trong bộ nhớ nhưng có liên kết với nhau về địa chỉ. Mỗi nút nhớ có cấu trực gồm 2 trường, trường Infor chứa phần tử dữ liệu, trường Link chứa địa chỉ nút đứng sau.

5.2. Sử dụng cấu trúc lưu trữ phân tán cho hàng đợi

- Vào: (F, R), x
- Ra: Không có

{Thủ tục này bổ sung phần tử x vào lối sau của hàng đợi (F, R) sử dụng cấu trúc lưu trữ phần tán}

```
Procedure QInsert(Var  F,R;  x)
    1) {Tạo nút mới}
        N <= AVAIL;
        infor(N) := x; link(N) := ∅;
    2) {Trường hợp hàng đợi rỗng}
        If F=R=∅ Then begin F:=R:=N; return; end;
    3) {Nối nút mới vào sau R và cho R trỏ tới nút mới}
        link(R) := N;
        R := N;
Return
```


5.2. Sử dụng cấu trúc lưu trữ phân tán cho hàng đợi

- Vào: (F, R)
- Ra: Phần tử dữ liệu loại bỏ

{Hàm này loại bỏ phần tử ở lối trước của hàng đợi (F, R) lưu trữ phân tán và trả về phần tử loại bỏ}

```
Function QDelete(Var  F, R)
        1){Kiểm tra hàng đợi rỗng}
        If F=R=∅ then begin
                write('Hàng đợi đã rỗng.');
                return;
        end;
```

5.2. Sử dụng cấu trúc lưu trữ phân tán cho hàng đợi

2) {Giữ lại nút lối trước (nút đầu hàng)}

tg := tnfor(F); P := F;

3) {Cho F trỏ tới nút đứng sau}

If F=R then F:=R:=∅

Else F := link(F);

4) {Hủy nút và trả về phần tử dữ liệu}

P => AVAIL;

QDelete := tg;

Return

5.2. Sử dụng cấu trúc lưu trữ phân tán cho hàng đợi

- Vào: (F, R)
- Ra: True - rỗng, False - không rỗng

{Hàm này kiểm tra hàng đợi rỗng}

Function QIsEmpty(F, R)

If F=R=∅ then QIsEmpty := True

Else QIsEmpty := False;

Return

Bài tập

1. Thế nào là danh sách nối vòng. Nêu ưu nhược điểm của nó.
2. Để khắc phục hạn chế của danh sách nối vòng người ta làm thế nào.
3. Thế nào là danh sách nối kép? Qui ước biểu diễn một nút của danh sách nối kép.
4. Nêu ưu nhược điểm của danh sách nối kép.
5. Cài đặt Stack bằng danh sach nối đơn như thế nào. Cần chú ý gì khi thực hiện các phép bổ sung, loại bỏ phần tử.
6. Cài đặt Queue bằng danh sach nối đơn như thế nào. Cần chú ý gì khi thực hiện các phép bổ sung, loại bỏ phần tử.