§4. CẤU TRÚC DỮ LIỆU BIỂU DIỄN DANH SÁCH

4.1. KHÁI NIỆM DANH SÁCH

Danh sách là một tập sắp thứ tự các phần tử cùng một kiểu. Đối với danh sách, người ta có một số thao tác: Tìm một phần tử trong danh sách, chèn một phần tử vào danh sách, xoá một phần tử khỏi danh sách, sắp xếp lại các phần tử trong danh sách theo một trật tự nào đó v.v…

4.2. BIỂU DIỄN DANH SÁCH TRONG MÁY TÍNH

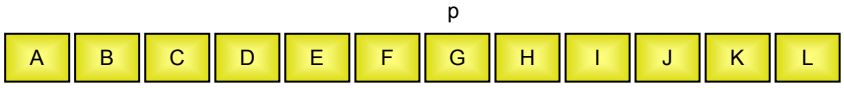
Việc cài đặt một danh sách trong máy tính tức là tìm một cấu trúc dữ liệu cụ thể mà máy tính hiểu được để lưu các phần tử của danh sách đồng thời viết các đoạn chương trình con mô tả các thao tác cần thiết đối với danh sách.

4.2.1. Cài đặt bằng mảng một chiều

Khi cài đặt danh sách bằng một mảng, thì có một biến nguyên n lưu số phần tử hiện có trong danh sách. Nếu mảng được đánh số bắt đầu từ 1 thì các phần tử trong danh sách được cất giữ trong mảng bằng các phần tử được đánh số từ 1 tới n.

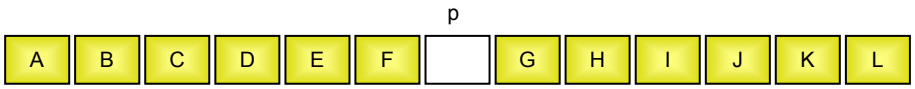
Chèn phần tử vào mảng:

Mảng ban đầu:

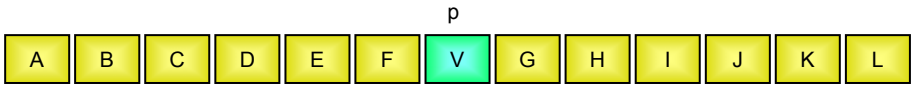


Nếu muốn chèn một phần tử V vào mảng tại vị trí p, ta phải:

❖ Dồn tất cả các phần tử từ vị trí p tới tới vị trí n về sau một vị trí:



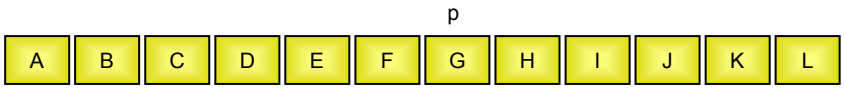
❖ Đặt giá trị V vào vị trí p:



❖ Tăng n lên 1

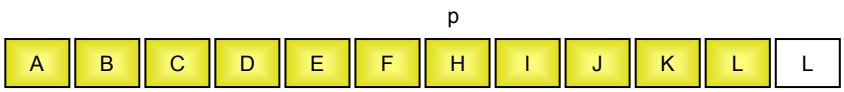
Xoá phần tử khỏi mảng

Mảng ban đầu:

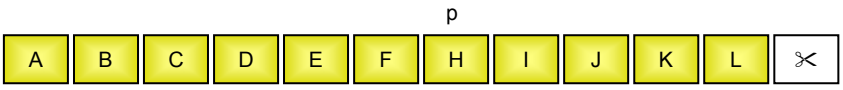


Muốn xoá phần tử thứ p của mảng mà vẫn giữ nguyên thứ tự các phần tử còn lại, ta phải:

❖ Dồn tất cả các phần tử từ vị trí p + 1 tới vị trí n lên trước một vị trí:



❖ Giảm n đi 1



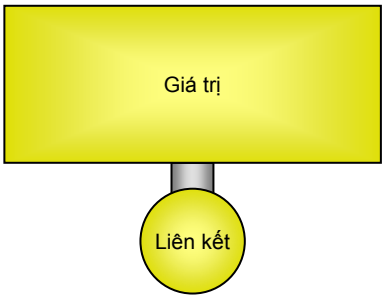
Trong trường hợp cần xóa một phần tử mà không cần duy trì thứ tự của các phần tử khác, ta chỉ cần đảo giá trị của phần tử cần xóa cho phần tử cuối cùng rồi giảm số phần tử của mảng (n) đi 1.

4.2.2. Cài đặt bằng danh sách nối đơn

Danh sách nối đơn gồm các nút được nối với nhau theo một chiều. Mỗi nút là một bản ghi (record) gồm hai trường:

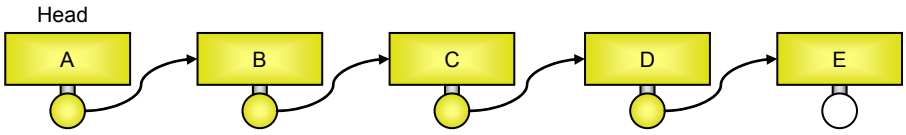
❖ Trường INFO chứa giá trị lưu trong nút đó

❖ Trường LINK chứa liên kết (con trỏ) tới nút kế tiếp, tức là chứa một thông tin đủ để biết nút kế tiếp nút đó trong danh sách là nút nào, trong trường hợp là nút cuối cùng (không có nút kế tiếp), trường liên kết này được gán một giá trị đặc biệt.



Hình 7: Cấu trúc nút của danh sách nối đơn

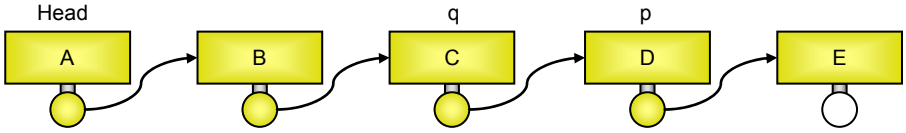
Nút đầu tiên trong danh sách được gọi là chốt của danh sách nối đơn (Head). Để duyệt danh sách nối đơn, ta bắt đầu từ chốt, dựa vào trường liên kết để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút cuối) thì dừng lại



Hình 8: Danh sách nối đơn

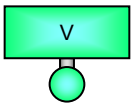
Chèn phần tử vào danh sách nối đơn:

Danh sách ban đầu:



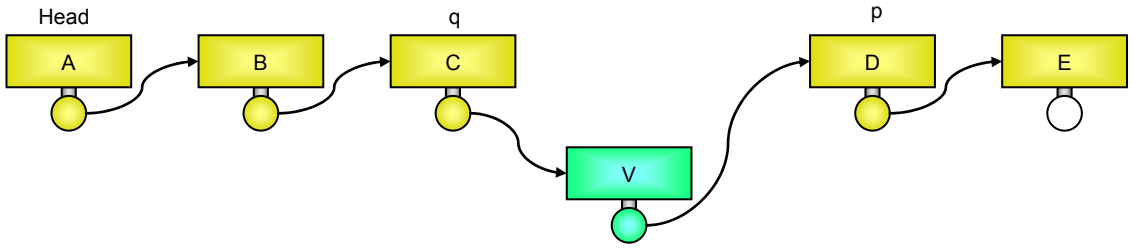
Muốn chèn thêm một nút chứa giá trị V vào vị trí của nút p, ta phải:

❖ Tạo ra một nút mới NewNode chứa giá trị V:



❖ Tìm nút q là nút đứng trước nút p trong danh sách (nút có liên kết tới p).

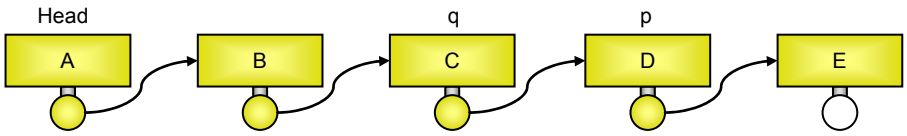
Nếu tìm thấy thì chỉnh lại liên kết: q liên kết tới NewNode, NewNode liên kết tới p



Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉnh lại liên kết: NewNode liên kết tới Head (cũ) và đặt lại Head = NewNode

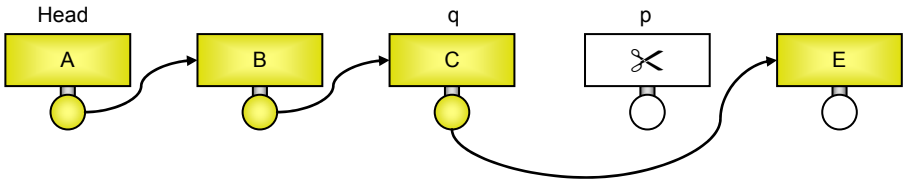
Xoá phần tử khỏi danh sách nối đơn:

Danh sách ban đầu:



Muốn huỷ nút p khỏi danh sách nối đơn, ta phải tìm nút q là nút đứng liền trước nút p trong danh sách (nút có liên kết tới p),

❖ Nếu tìm thấy thì chỉnh lại liên kết: q liên kết thẳng tới nút liền sau p, khi đó quá trình duyệt danh sách bắt đầu từ Head khi duyệt tới q sẽ nhảy qua không duyệt p nữa. Trên thực tế khi cài đặt bằng các biến động và con trỏ, ta nên có thao tác giải phóng bộ nhớ đã cấp cho nút p



❖ Nếu không có nút đứng trước nút p trong danh sách thì tức là p = Head, ta chỉ việc đặt lại Head bằng nút đứng kế tiếp Head (cũ) trong danh sách. Sau đó có thể giải phóng bộ nhớ cấp cho nút p (Head cũ)

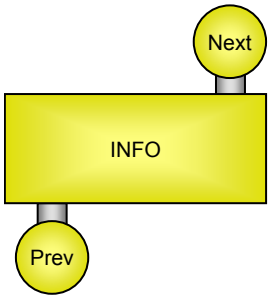
4.2.3. Cài đặt bằng danh sách nối kép

Danh sách nối kép gồm các nút được nối với nhau theo hai chiều. Mỗi nút là một bản ghi (record) gồm ba trường:

❖ Trường thứ nhất (Info) chứa giá trị lưu trong nút đó

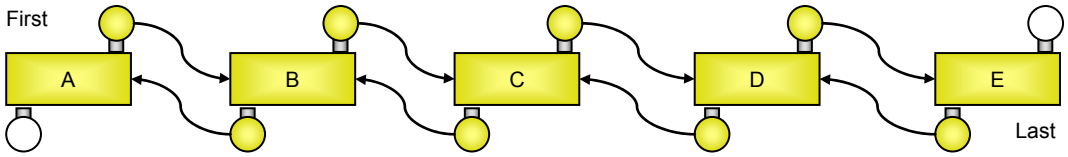
❖ Trường thứ hai (Next) chứa liên kết (con trỏ) tới nút kế tiếp, tức là chứa một thông tin đủ để biết nút kế tiếp nút đó là nút nào, trong trường hợp là nút cuối cùng (không có nút kế tiếp), trường liên kết này được gán một giá tị đặc biệt.

❖ Trường thứ ba (Prev) chứa liên kết (con trỏ) tới nút liền trước, tức là chứa một thông tin đủ để biết nút đứng trước nút đó trong danh sách là nút nào, trong trường hợp là nút đầu tiên (không có nút liền trước) trường này được gán một giá trị đặc biệt.



Hình 9: Cấu trúc nút của danh sách nối kép

Khác với danh sách nối đơn, danh sách nối kép có hai chốt: Nút đầu tiên trong danh sách (First) và nút cuối cùng trong danh sách (Last). Có hai cách duyệt danh sách nối kép: Hoặc bắt đầu từ First, dựa vào liên kết Next để đi sang nút kế tiếp, đến khi gặp giá trị đặc biệt (duyệt qua nút Last) thì dừng lại. Hoặc bắt đầu từ Last, dựa vào liên kết Prev để đi sang nút liền trước, đến khi gặp giá trị đặc biệt (duyệt qua nút First) thì dừng lại

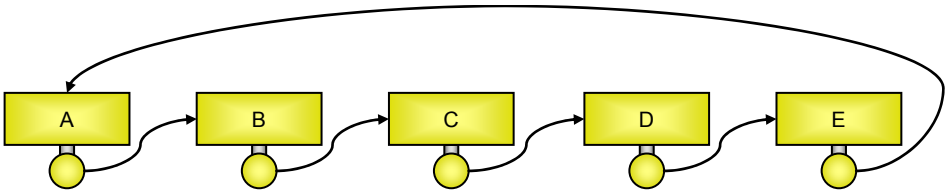


Hình 10: Danh sách nối kép

Việc chèn / xoá vào danh sách nối kép cũng đơn giản chỉ là kỹ thuật chỉnh lại các mối liên kết giữa các nút cho hợp lý, ta coi như bài tập.

4.2.4. Cài đặt bằng danh sách nối vòng một hướng

Trong danh sách nối đơn, phần tử cuối cùng trong danh sách có trường liên kết được gán một giá trị đặc biệt (thường sử dụng nhất là giá trị nil). Nếu ta cho trường liên kết của phần tử cuối cùng trỏ thẳng về phần tử đầu tiên của danh sách thì ta sẽ được một kiểu danh sách mới gọi là danh sách nối vòng một hướng.

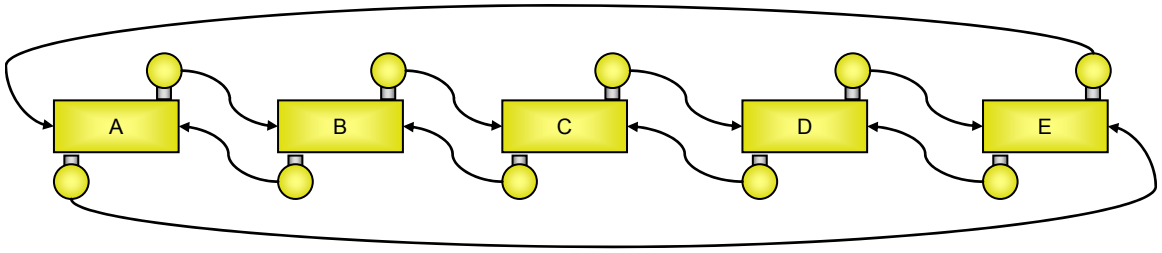


Hình 11: Danh sách nối vòng một hướng

Đối với danh sách nối vòng, ta chỉ cần biết một nút bất kỳ của danh sách là ta có thể duyệt được hết các nút trong danh sách bằng cách đi theo hướng của các liên kết. Chính vì lý do này, khi chèn xoá vào danh sách nối vòng, ta không phải xử lý các trường hợp riêng khi chèn xoá tại vị trí của chốt

4.2.5. Cài đặt bằng danh sách nối vòng hai hướng

Danh sách nối vòng một hướng chỉ cho ta duyệt các nút của danh sách theo một chiều, nếu cài đặt bằng danh sách nối vòng hai hướng thì ta có thể duyệt các nút của danh sách cả theo chiều ngược lại nữa. Danh sách nối vòng hai hướng có thể tạo thành từ danh sách nối kép nếu ta cho trường Prev của nút First trỏ thẳng tới nút Last còn trường Next của nút Last thì trỏ thẳng về nút First.



Hình 12: Danh sách nối vòng hai hướng

Bài tập

Bài 1

Lập chương trình quản lý danh sách học sinh, tuỳ chọn loại danh sách cho phù hợp, chương trình có những chức năng sau: (Hồ sơ một học sinh giả sử có: Tên, lớp, số điện thoại, điểm TB …)

Cho phép nhập danh sách học sinh từ bàn phím hay từ file.

Cho phép in ra danh sách học sinh gồm có tên và xếp loại

Cho phép in ra danh sách học sinh gồm các thông tin đầy đủ

Cho phép nhập vào từ bàn phím một tên học sinh và một tên lớp, tìm xem có học sinh có tên nhập vào trong lớp đó không ?. Nếu có thì in ra số điện thoại của học sinh đó

Cho phép vào một hồ sơ học sinh mới từ bàn phím, bổ sung học sinh đó vào danh sách học sinh, in ra danh sách mới.

Cho phép nhập vào từ bàn phím tên một lớp, loại bỏ tất cả các học sinh của lớp đó khỏi danh sách, in ra danh sách mới.

Có chức năng sắp xếp danh sách học sinh theo thứ tự giảm dần của điểm trung bình

Cho phép nhập vào hồ sơ một học sinh mới từ bàn phím, chèn học sinh đó vào danh sách mà không làm thay đổi thứ tự đã sắp xếp, in ra danh sách mới.

Cho phép lưu trữ lại trên đĩa danh sách học sinh khi đã thay đổi.

Bài 2

Có n người đánh số từ 1 tới n ngồi quanh một vòng tròn ($n \le 10000$), cùng chơi một trò chơi: Một người nào đó đếm 1, người kế tiếp, theo chiều kim đồng hồ đếm 2… cứ như vậy cho tới người đếm đến một số nguyên tố thì phải ra khỏi vòng tròn, người kế tiếp lại đếm bắt đầu từ 1:

Hãy lập chương trình

Nhập vào 2 số n và S từ bàn phím

- Cho biết nếu người thứ nhất là người đếm 1 thì người còn lại cuối cùng trong vòng tròn là người thứ mấy
- Cho biết nếu người còn lại cuối cùng trong vòng tròn là người thứ k thì người đếm 1 là người nào?.

Giải quyết hai yêu cầu trên trong trường hợp: đầu tiên trò chơi được đếm theo chiều kim đồng hồ, khi có một người bị ra khỏi cuộc chơi thì vẫn là người kế tiếp đếm 1 nhưng quá trình đếm ngược lại (tức là ngược chiều kim đồng hồ)

```
  ⟨Test⟩; {Đưa một vài lệnh để kiểm tra hoạt động của Stack}
end.
```

Khi cài đặt bằng mảng, tuy các thao tác đối với Stack viết hết sức đơn giản nhưng ở đây ta vẫn chia thành các chương trình con, mỗi chương trình con mô tả một thao tác, để từ đó về sau, ta chỉ cần biết rằng chương trình của ta có một cấu trúc Stack, còn ta mô phỏng cụ thể như thế nào thì không cần phải quan tâm nữa, và khi cài đặt Stack bằng các cấu trúc dữ liệu khác, chỉ cần sửa lại các thủ tục StackInit, Push và Pop mà thôi.

5.1.2. Mô tả Stack bằng danh sách nối đơn kiểu LIFO

Khi cài đặt Stack bằng danh sách nối đơn kiểu LIFO, thì Stack bị tràn khi vùng không gian nhớ dùng cho các biến động không còn đủ để thêm một phần tử mới. Tuy nhiên, việc kiểm tra điều này rất khó bởi nó phụ thuộc vào máy tính và ngôn ngữ lập trình. Ví dụ như đối với Turbo Pascal, khi Heap còn trống 80 Bytes thì cũng chỉ đủ chỗ cho 10 biến, mỗi biến 6 Bytes mà thôi. Mặt khác, không gian bộ nhớ dùng cho các biến động thường rất lớn nên cài đặt dưới đây ta bỏ qua việc kiểm tra Stack tràn.

```
program StackByLinkedList;
type
  PNode = ^TNode; {Con trỏ tới một nút của danh sách}
  TNode = record {Cấu trúc một nút của danh sách}
    Value: Integer;
    Link: PNode;
  end;
var
  Top: PNode; {Con trỏ đỉnh Stack}

procedure StackInit; {Khởi tạo Stack rỗng}
begin
  Top := nil;
end;

procedure Push(V: Integer); {Đẩy giá trị V vào Stack ⇔ thêm nút mới chứa V và nối nút đó vào danh sách}
var
  P: PNode;
begin
  New(P); P^.Value := V; {Tạo ra một nút mới}
  P^.Link := Top; Top := P; {Móc nút đó vào danh sách}
end;

function Pop: Integer; {Lấy một giá trị ra khỏi Stack, trả về trong kết quả hàm}
var
  P: PNode;
begin
  if Top = nil then WriteLn('Stack is empty')
  else
    begin
      Pop := Top^.Value; {Gán kết quả hàm}
      P := Top^.Link; {Giữ lại nút tiếp theo Top^ (nút được đẩy vào danh sách trước nút Top^)}
      Dispose(Top); Top := P; {Giải phóng bộ nhớ cấp cho Top^, cập nhật lại Top mới}
    end;
end;

begin
  StackInit;
  ⟨Test⟩; {Đưa một vài lệnh để kiểm tra hoạt động của Stack}
end.
```

5.2. HÀNG ĐỢI (QUEUE)

Hàng đợi là một kiểu danh sách được trang bị hai phép toán **bổ sung một phần tử** vào cuối danh sách (Rear) và **loại bỏ một phần tử** ở đầu danh sách (Front).

Có thể hình dung hàng đợi như một đoàn người xếp hàng mua vé: Người nào xếp hàng trước sẽ được mua vé trước. Vì nguyên tắc "vào trước ra trước" đó, Queue còn có tên gọi là danh sách kiểu FIFO (First In First Out).

5.2.1. Mô tả Queue bằng mảng

Khi mô tả Queue bằng mảng, ta có hai chỉ số Front và Rear, Front lưu chỉ số phần tử đầu Queue còn Rear lưu chỉ số cuối Queue, khởi tạo Queue rỗng: Front := 1 và Rear := 0;

- ❖ Để thêm một phần tử vào Queue, ta tăng Rear lên 1 và đưa giá trị đó vào phần tử thứ Rear.
- ❖ Để loại một phần tử khỏi Queue, ta lấy giá trị ở vị trí Front và tăng Front lên 1.
- ❖ Khi Rear tăng lên hết khoảng chỉ số của mảng thì mảng đã đầy, không thể đẩy thêm phần tử vào nữa.
- ❖ Khi Front > Rear thì tức là Queue đang rỗng

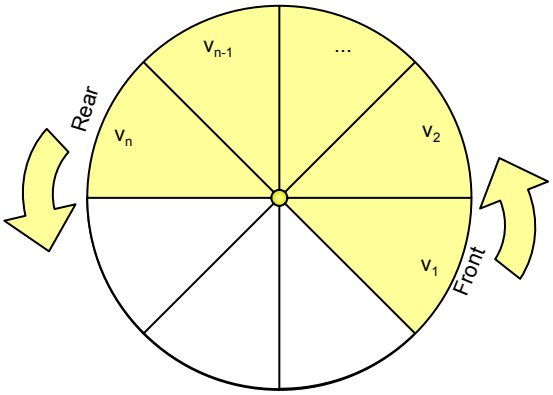
Như vậy chỉ một phần của mảng từ vị trí Front tới Rear được sử dụng làm Queue.

```
program QueueByArray;
const
  max = 10000;
var
  Queue: array[1..max] of Integer;
  Front, Rear: Integer;

procedure QueueInit; {Khởi tạo một hàng đợi rỗng}
begin
  Front := 1; Rear := 0;
end;

procedure Push(V: Integer); {Đẩy V vào hàng đợi}
begin
  if Rear = max then WriteLn('Overflow')
  else
    begin
      Inc(Rear);
      Queue[Rear] := V;
    end;
end;

function Pop: Integer; {Lấy một giá trị khỏi hàng đợi, trả về trong kết quả hàm}
begin
  if Front > Rear then WriteLn('Queue is Empty')
  else
    begin
      Pop := Queue[Front];
      Inc(Front);
    end;
end;

begin
  QueueInit;
  (Test); {Đưa một vài lệnh để kiểm tra hoạt động của Queue}
end.
```

5.2.2. Mô tả Queue bằng danh sách vòng

Xem lại chương trình cài đặt Stack bằng một mảng kích thước tối đa 10000 phần tử, ta thấy rằng nếu như ta làm 6000 lần Push rồi 6000 lần Pop rồi lại 6000 lần Push thì vẫn không có vấn đề gì xảy ra. Lý do là vì chỉ số Top lưu đỉnh của Stack sẽ được tăng lên 6000 rồi lại giảm đến 0 rồi lại tăng trở lại lên 6000. Nhưng đối với cách cài đặt Queue như trên thì sẽ gặp thông báo lỗi tràn mảng, bởi mỗi lần Push, chỉ số cuối hàng đợi Rear cũng tăng lên và không bao giờ bị giảm đi cả. Đó chính là nhược điểm mà ta nói tới khi cài đặt: Chỉ có các phần tử từ vị trí Front tới Rear là thuộc Queue, các phần tử từ vị trí 1 tới Front - 1 là vô nghĩa.

Để khắc phục điều này, ta mô tả Queue bằng một danh sách vòng (biểu diễn bằng mảng hoặc cấu trúc liên kết), coi như các phần tử của mảng được xếp quanh vòng theo một hướng nào đó. Các phần tử nằm trên phần cung tròn từ vị trí Front tới vị trí Rear là các phần tử của Queue. Có thêm một biến n lưu số phần tử trong Queue. Việc thêm một phần tử vào Queue tương đương với việc ta dịch chỉ số Rear theo vòng một vị trí rồi đặt giá trị mới vào đó. Việc loại bỏ một phần tử trong Queue tương đương với việc lấy ra phần tử tại vị trí Front rồi dịch chỉ số Front theo vòng.



Hình 13: Dùng danh sách vòng mô tả Queue

Lưu ý là trong thao tác Push và Pop phải kiểm tra Queue tràn hay Queue cạn nên phải cập nhật lại biến n. (Thực ra có thể chỉ dùng hai biến Front và Rear là có thể kiểm tra được Queue tràn hay cạn).

```
program QueueByCList;
const
  max = 10000;
var
  Queue: array[0..max - 1] of Integer;
  i, n, Front, Rear: Integer;

procedure QueueInit; {Khởi tạo Queue rỗng}
begin
  Front := 0; Rear := max - 1; n := 0;
end;

procedure Push(V: Integer); {Đẩy giá trị V vào Queue}
begin
  if n = max then WriteLn('Queue is Full')
  else
```

```
    begin
      Rear := (Rear + 1) mod max; {Rear chạy theo vòng tròn}
      Queue[Rear] := V;
      Inc(n);
    end;
end;

function Pop: Integer; {Lấy một phần tử khỏi Queue, trả về trong kết quả hàm}
begin
  if n = 0 then WriteLn('Queue is Empty')
  else
    begin
      Pop := Queue[Front];
      Front := (Front + 1) mod max; {Front chạy theo vòng tròn}
      Dec(n);
    end;
end;

begin
  QueueInit;
  (Test); {Đưa một vài lệnh để kiểm tra hoạt động của Queue}
end.
```

5.2.3. Mô tả Queue bằng danh sách nối đơn kiểu FIFO

Tương tự như cài đặt Stack bằng danh sách nối đơn kiểu LIFO, ta cũng không kiểm tra Queue tràn trong trường hợp mô tả Queue bằng danh sách nối đơn kiểu FIFO.

```
program QueueByLinkedList;
type
  PNode = ^TNode; {Kiểu con trỏ tới một nút của danh sách}
  TNode = record {Cấu trúc một nút của danh sách}
    Value: Integer;
    Link: PNode;
  end;
var
  Front, Rear: PNode; {Hai con trỏ tới nút đầu và nút cuối của danh sách}

procedure QueueInit; {Khởi tạo Queue rỗng}
begin
  Front := nil;
end;

procedure Push(V: Integer); {Đẩy giá trị V vào Queue}
var
  P: PNode;
begin
  New(P); P^.Value := V; {Tạo ra một nút mới}
  P^.Link := nil;
  if Front = nil then Front := P {Móc nút đó vào danh sách}
  else Rear^.Link := P;
  Rear := P; {Nút mới trở thành nút cuối, cập nhật lại con trỏ Rear}
end;

function Pop: Integer; {Lấy giá trị khỏi Queue, trả về trong kết quả hàm}
var
  P: PNode;
begin
  if Front = nil then WriteLn('Queue is empty')
  else
    begin
      Pop := Front^.Value; {Gán kết quả hàm}
      P := Front^.Link; {Giữ lại nút tiếp theo Front^ (Nút được đẩy vào danh sách ngay sau Front^)}
```

```
      Dispose(Front); Front := P; {Giải phóng bộ nhớ cấp cho Front^, cập nhật lại Front mới}
    end;
end;

begin
  QueueInit;
  ⟨Test⟩; {Đưa một vài lệnh để kiểm tra hoạt động của Queue}
end.
```

Bài tập

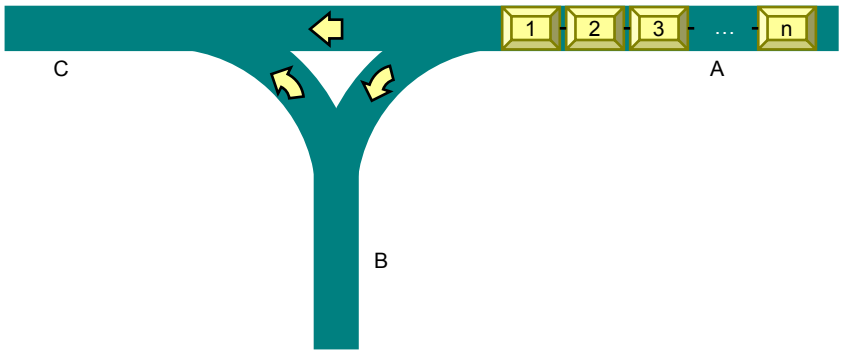
Bài 1

Tìm hiểu cơ chế xếp chồng của thủ tục đệ quy, phương pháp dùng ngăn xếp để khử đệ quy.

Viết chương trình mô tả cách đổi cơ số từ hệ thập phân sang hệ cơ số R dùng ngăn xếp

Bài 2

Hình 14 là cơ cấu đường tàu tại một ga xe lửa



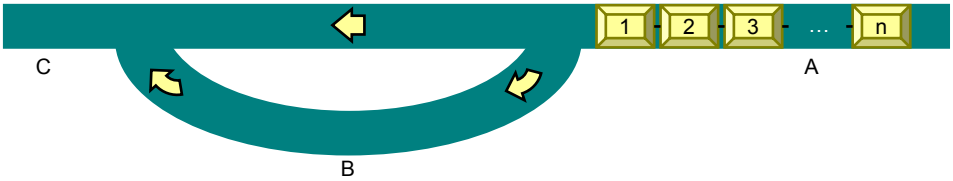
Hình 14: Di chuyển toa tàu

Ban đầu ở đường ray A chứa các toa tàu đánh số từ 1 tới n theo thứ tự từ trái qua phải, người ta muốn chuyển các toa đó sang đường ray C để được một thứ tự mới là một hoán vị của (1, 2, …, n) theo quy tắc: chỉ được đưa các toa tàu chạy theo đường ray theo hướng mũi tên, có thể dùng đoạn đường ray B để chứa tạm các toa tàu trong quá trình di chuyển.

a) Hãy nhập vào hoán vị cần có, cho biết có phương án chuyển hay không, và nếu có hãy đưa ra cách chuyển. Ví dụ: n = 4; Thứ tự cần có (1, 4, 3, 2), Cách di chuyển là: (A → C); (A → B); (A → B); (A → C); (B → C); (B → C).

b) Những hoán vị nào của thứ tự các toa là có thể tạo thành trên đoạn đường ray C với luật di chuyển như trên

c) Với hai yêu cầu trên, nhưng với sơ đồ đường ray như Hình 15:



Hình 15: Di chuyển toa tàu (2)