**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 First và Last, First lưu chỉ số phần tử đầu Queue còn Last lưu chỉ số cuối Queue, khởi tạo Queue rỗng: First := 1 và Last := 0;

Để thêm một phần tử vào Queue, ta tăng Last lên 1 và đưa giá trị đó vào phần tử thứ Last.

Để loại một phần tử khỏi Queue, ta lấy giá trị ở vị trí First và tăng First lên 1.

Khi Last tăng lên hết khoảng chỉ số của mảng thì mảng đã đầy, không thể đẩy thêm phần tử vào nữa.

Khi First > Last thì tức là Queue đang rỗng

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

program QueueByArray;

const

max = 10000;

var

Queue: array[1..max] of Integer;

First, Last: Integer;

procedure QueueInit; {Khởi tạo một hàng đợi rỗng}

begin

First := 1; Last := 0;

end;

procedure Push(V: Integer); {Đẩy V vào hàng đợi}

begin

if Last = max then WriteLn('Overflow')

else

begin

Inc(Last);

Queue[Last] := V;

end;

end;

function Pop: Integer;

begin

if First > Last then WriteLn('Queue is Empty')

else

begin

Pop := Queue[First];

Inc(First);

end;

end;

begin

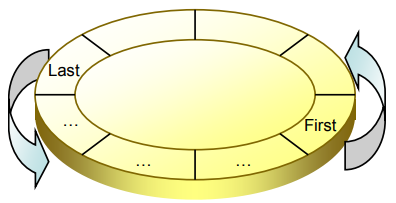
QueueInit;

<Test>;

end;

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 Last 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 Last 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 First tới Last là thuộc Queue, các phần tử từ vị trí 1 tới First - 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 First tới vị trí Last 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 Last 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 First rồi dịch chỉ số First theo vòng.



**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. (Ở đây dùng thêm biến n cho dễ hiểu còn trên thực tế chỉ cần hai biến First và Last là ta có thể kiểm tra được Queue tràn hay cạn rồi)

program QueueByCList;

const

max = 10000;

var

Queue: array[0..max - 1] of Integer;

i, n, First, Last: Integer;

procedure QueueInit; {Khởi tạo Queue rỗng}

begin

First := 0;

Last := 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

Last := (Last + 1) mod max;

Queue[Last] := V;

Inc(n);

end;

function Pop: Integer;

begin

if n = 0 then WriteLn('Queue is Empty')

else

begin

Pop := Queue[First];

First := (First + 1) mod max;

Dec(n);

end;

end;

begin

QueueInit;

<Test>;

end.

**5.2.2. Mô tả Queue bằng danh sách liên kết 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;

TNode = record

Value: Integer;

Link: PNode;

end;

var

First, Last: PNode;

procedure QueueInit;

begin

First := nil;

end;

procedure Push(V: Integer); {Đẩy giá trị V vào Queue}

var

P: PNode;

begin

New(P);

P^.Value := V;

P^.Link := nil;

if First = nil then First := P

else

Last^.Link := P;

Last := P;

end;

function Pop: Integer;

var

P: PNode;

begin

if First = nil then WriteLn('Queue is empty')

else

begin

Pop := First^.Value;

P := First^.Link;

Dispose(First);

First := P;

end;

end;

begin

QueueInit;

<Test>;

end.