Chương 3
DANH SÁCH

Ths. Phạm Thanh An
Bộ môn Khoa học máy tính - Khoa CNTT
Trường Đại học Ngân hàng TP.HCM

LOGO

Nội dung trình bày

- Danh sách và các phép toán trên danh sách
- Danh sách đặc
  - Định nghĩa, Cách biểu diễn và các phép toán
  - Ưu và nhược điểm của danh sách đặc
  - Tổ chức Stack và Queue theo kiểu danh sách đặc
- Danh sách liên kết
  - Khái niệm , Biểu diễn, Các phép toán
  - Ưu và nhược điểm
  - Tổ chức Stack và Queue theo kiểu danh sách liên kết
- Danh sách liên kết kép

Danh sách

- Định nghĩa danh sách
  - Danh sách là dãy hữu hạn có thứ tự bao gồm một số biến động các phần tử thuộc cùng một lớp đối tượng nào đó.
  - Mô tả danh sách : $L = (a_1, a_2, . . . ,a_n)$
  - Danh sách tuyến tính: là danh sách mà quan hệ lân cận giữa các phần tử được hiển thị

Lưu trữ danh sách

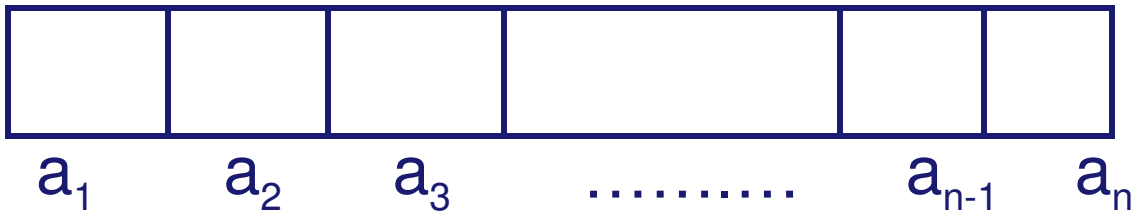
- Tổ chức lưu trữ danh sách trong bộ nhớ
  - Sử dụng mảng - Danh sách đặc
  - Đối tượng lớp - danh sách liên kết
    - Mỗi node là một đối tượng lớp

Các phép toán trên danh sách

- Thêm
- Loại bỏ
- Sắp xếp:
- Tìm kiếm
- Tách
- Ghép
- Duyệt:

Danh sách đặc (condensed list)

- Định nghĩa
  - Là danh sách có các phần tử được xếp kế tiếp nhau trong bộ nhớ



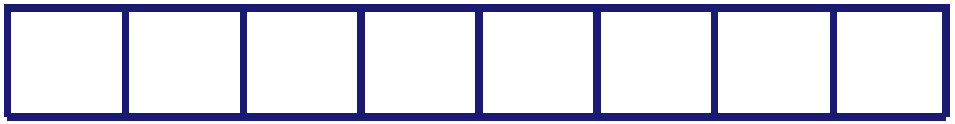
- Đặc điểm
  - d: chiều dài mỗi phần tử trong danh sách
  - l_0: địa chỉ của phần tử đầu tiên
    - ⇨ địa chỉ của phần tử thứ i là: $l_i=l_0+(i-1)d$

Danh sách đặc (condensed list)

- Ưu điểm
- Nhược điểm

Mảng danh sách đặc phổ biến

- Mảng một chiều a[ ]



Hình ảnh mảng

Hình ảnh một biến

- Khai báo:
  - Cách 1: <Kiểu dữ liệu> [] tên_mảng;
  - Tên_mảng = new <Kiểu dữ liệu>[size];
  - Ví dụ:
    - int[] myIntArray; myIntArray = new int[5];
    - int[] numbers; numbers = new int[] {0,1,2,3,4};

Mảng 2 chiều

- Mảng hai chiều a[,]
  - Khai báo mảng 2 chiều:
    int[,] grades = new int[2,3]; // 2 hàng, 3 cột

0	1	4
1	2	5

  - Truy cập phần tử của mảng <Tên mảng>[dòng, cột]

Khởi tạo mảng 2 chiều

```
int[,] grades = new int[,] {{1, 82, 74, 89, 100},
                            {2, 93, 96, 85, 86},
                            {3, 83, 72, 95, 89},
                            {4, 91, 98, 79, 88}}
```

Ví dụ cài đặt danh sách

```
class CArray {
    private int [] arr;
    private int upper;
    private int numElements;
    public CArray(int size) {
        arr = new int[size];
        upper = size-1;
        numElements = 0;
    }
```

Mảng danh sách đặc phổ biến

```
public void Insert(int item) {
    arr[numElements] = item;
    numElements++;
    }
public void DisplayElements() {
    for(inti=0;i<= upper; i++)
    Console.Write(arr[i]+"");
    }
```

Mảng danh sách đặc phổ biến

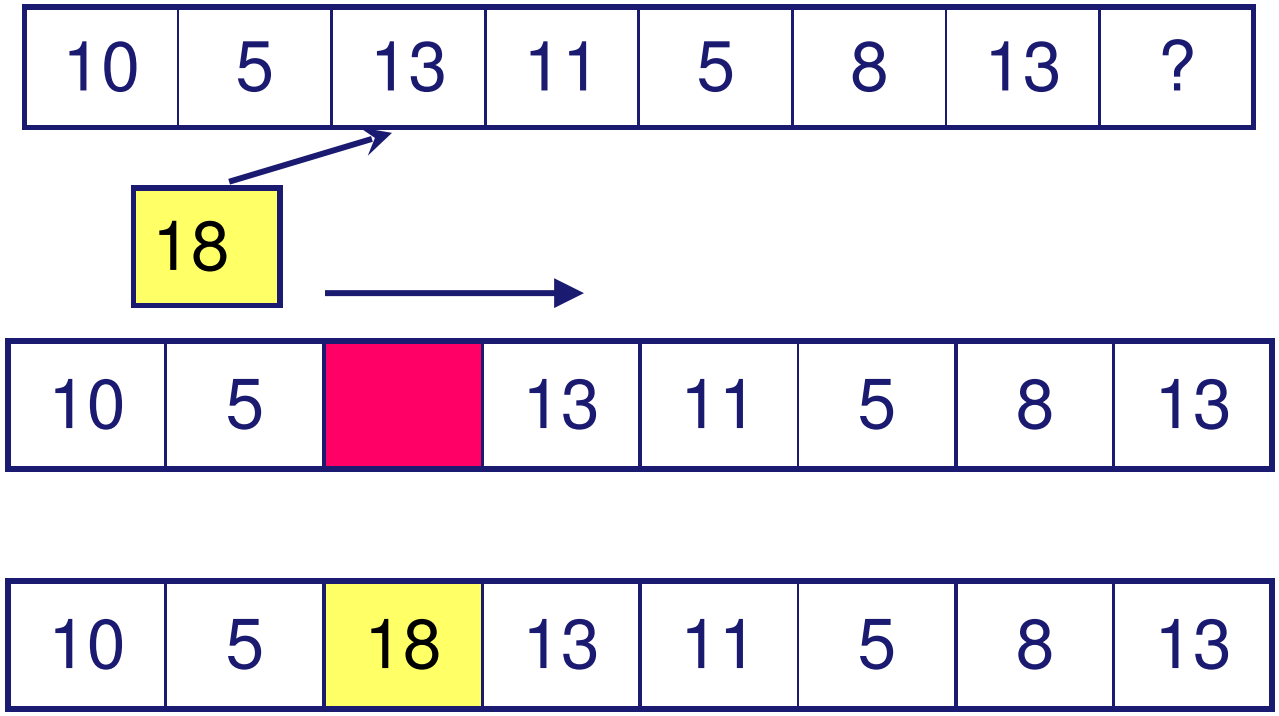
```
static void Main() {
    CArray nums = new CArray();
    for(inti=0;i<=49; i++)
    nums.Insert(i);
    nums.DisplayElements();
}
```

Mảng danh sách đặc phổ biến

```
static void Main() {
    CArray nums = new CArray();
    Random rnd = new Random(100);
    for(inti=0;i<10; i++)
    nums.Insert((int)(rnd.NextDouble() * 100));
    nums.DisplayElements();
    }
```

Cài đặt danh sách bằng mảng

- Thêm một phần tử vào mảng



Cài đặt danh sách bằng mảng

- Xóa phần tử ra khỏi mảng

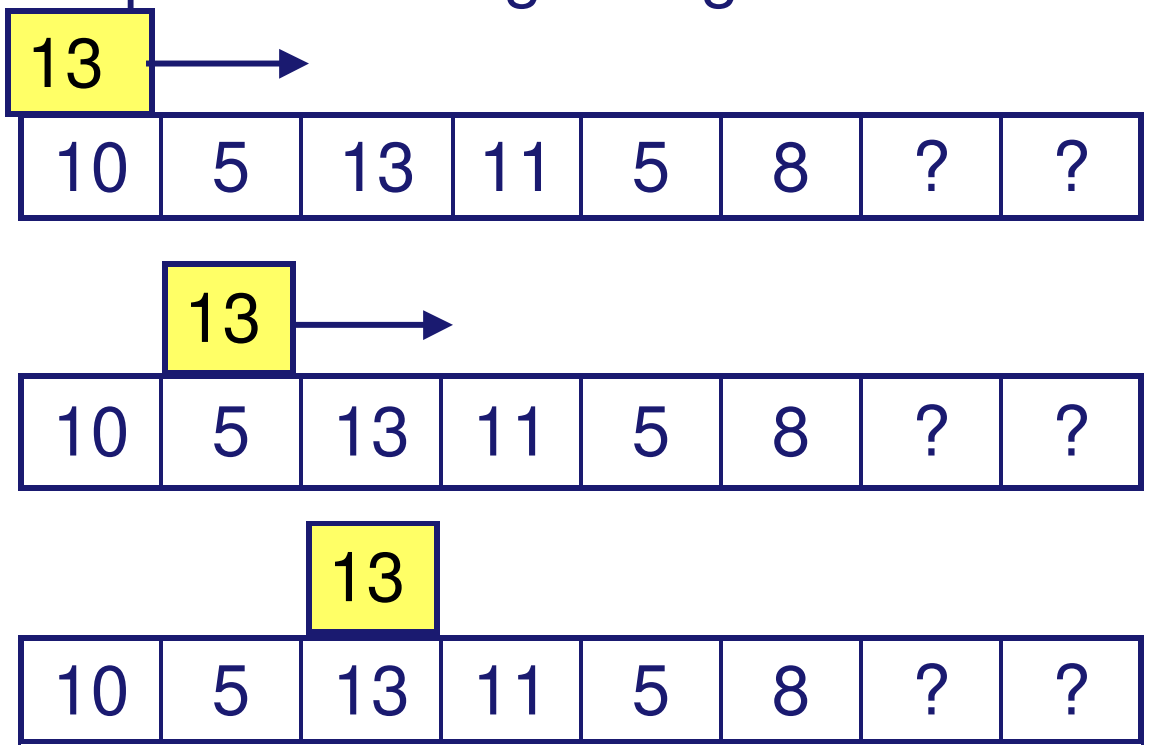
10	5	~~18~~	13	11	5	8	?

10	5		13	11	5	8	?

10	5	13	11	5	8	?	?

Cài đặt danh sách bằng mảng

- Tìm kiếm phần tử trong mảng



Bài tập

- Nhập một dãy số nguyên từ bàn phím, và sắp xếp chúng theo thứ tự tăng dần

Input: 5 2 4 18 9 1

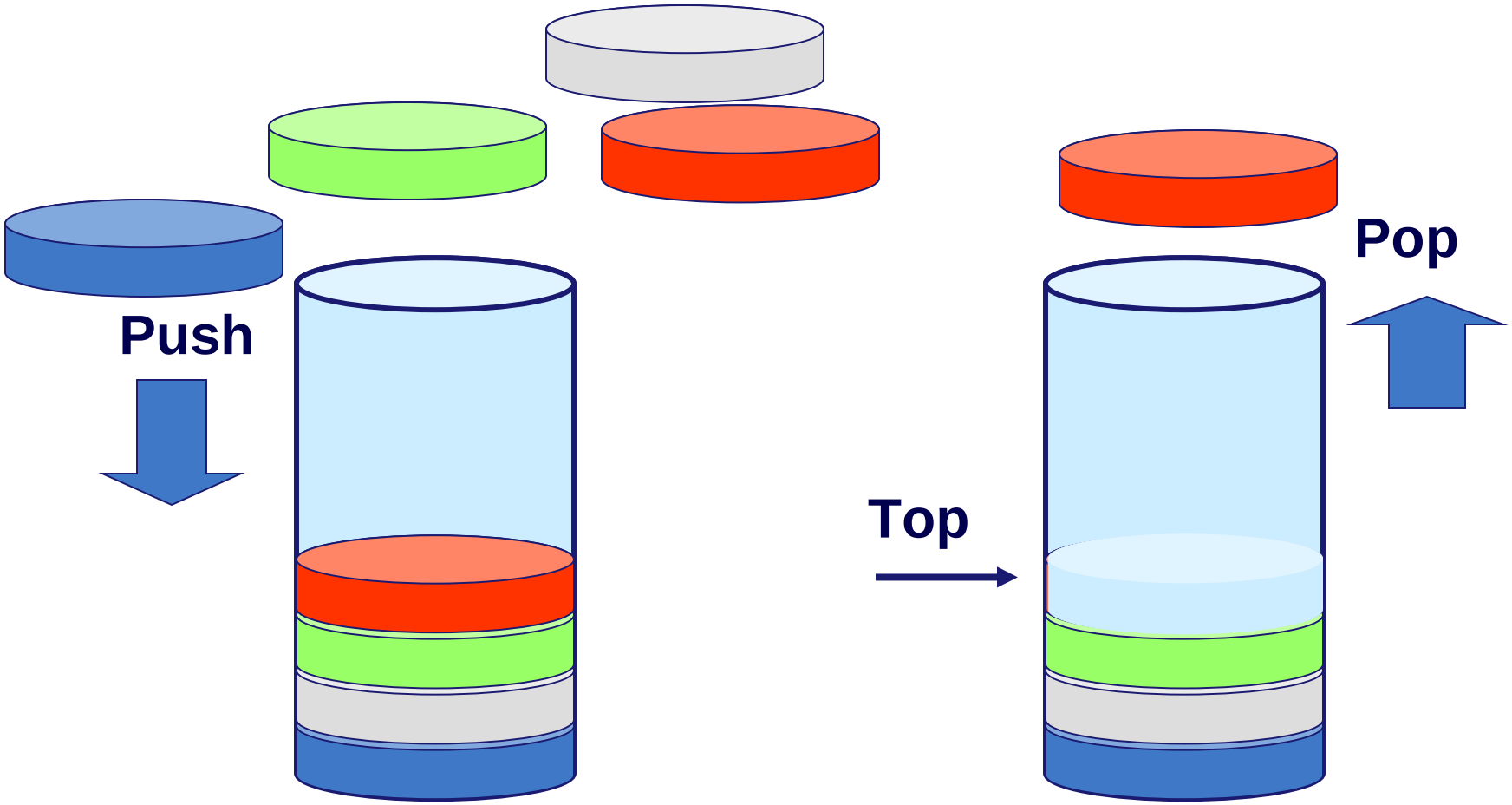
Output: 1 2 4 5 9 18

- Nhập một dãy số nguyên từ bàn phím, và cho biết số lần xuất hiện của từng số trong dãy số

Input: 1 3 2 9 4 3 2 9 8 1 1 3 2 9 1

Ouput: (1,4) (2,3) (3,3) (4,1) (8,1) (9,3)

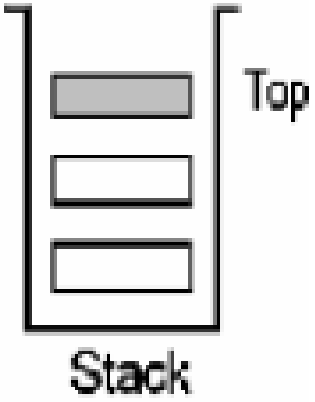
Tổ chức Stack theo kiểu danh sách đặc

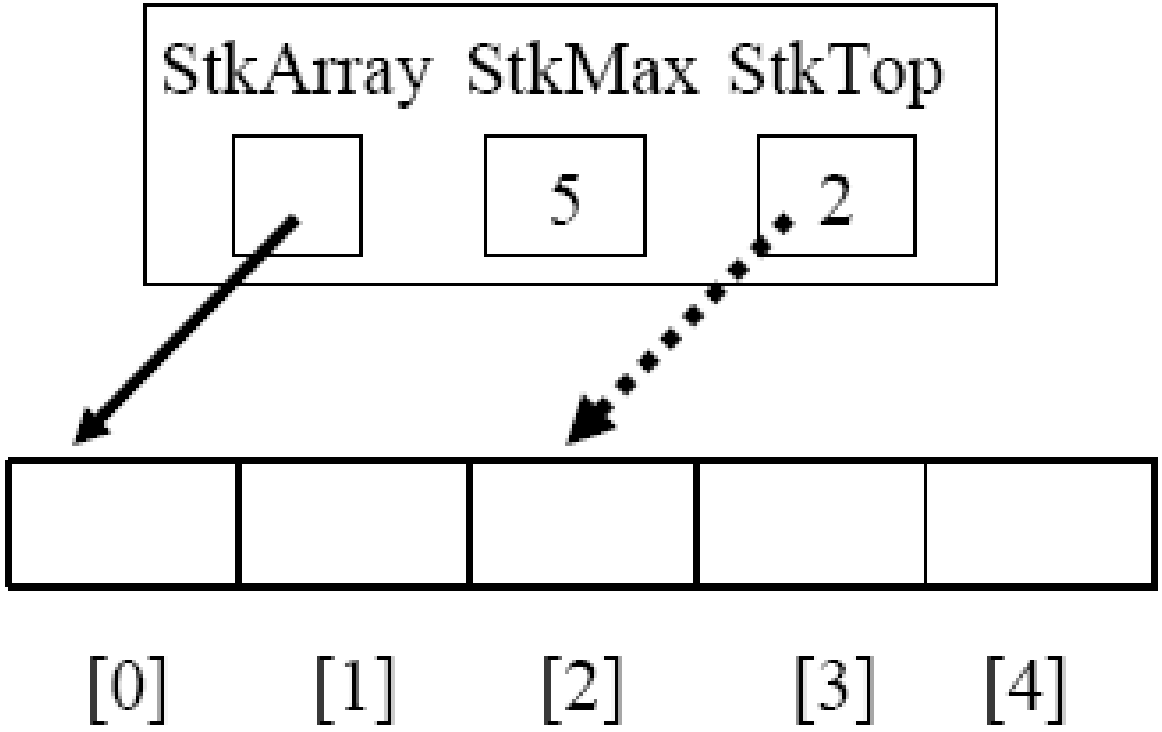


Tổ chức Stack theo kiểu danh sách đặc

- Cấu trúc của STACK
  - Dùng 1 mảng (StkArray) để chứa các phần tử
  - Dùng 1 số nguyên (StkMax) để lưu số phần tử tối đa trong Stack
  - Dùng 1 số nguyên (StkTop) để lưu chỉ số đỉnh của STACK

Tổ chức Stack theo kiểu danh sách đặc

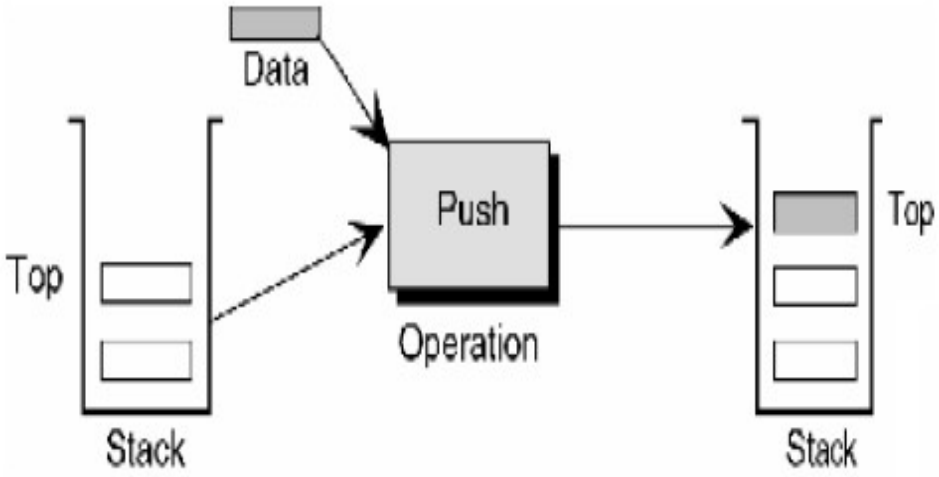




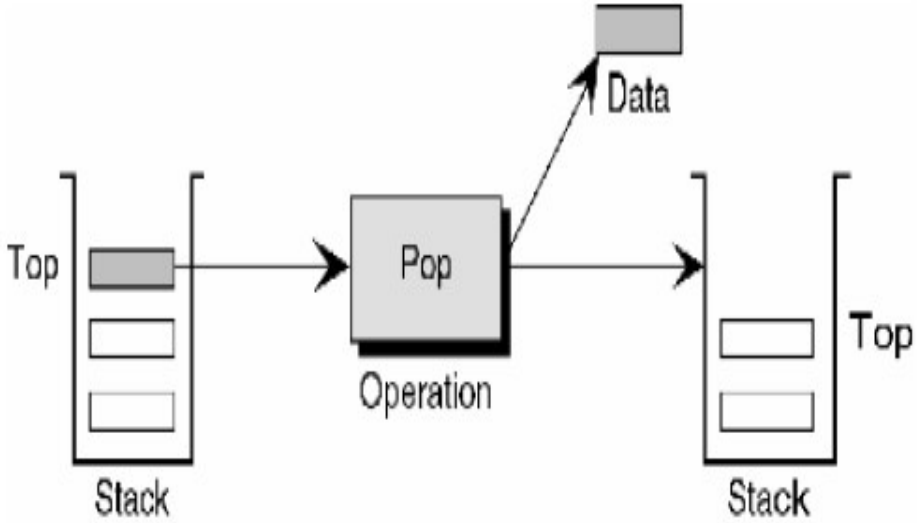
Tổ chức Stack theo kiểu danh sách đặc

- Các thao tác cơ bản trên Stack:
  - Stack: khởi tạo Stack rỗng
  - IsEmpty: kiểm tra Stack rỗng ?
  - IsFull: kiểm tra Stack đầy ?
  - Push: thêm 1 phần tử vào đỉnh Stack, có thể làm Stack đầy
  - Pop: lấy ra 1 phần tử từ đỉnh Stack, có thể làm Stack rỗng

Tổ chức Stack theo kiểu danh sách đặc



Thao tác Push



Thao tác Pop

Khao báo lớp Stack

- **Thao tác "Khởi tạo Stack rỗng"**

```
class Cstack{
    private int [] StkArr;
    private int StkTop;
    private int StkMax;
    public Cstack(int size) {
        StkArr = new int[size];
         StkMax = size;
        StkTop = -1; // Stack rỗng
    }
```

Kiểm tra Stack rỗng

```
boolean IsEmpty()
{
   if (StkTop == -1) return true; // Stack rỗng
   return false; // Stack không rỗng
}
```

Kiểm tra Stack đầy

```
boolean IsFull()
{
   if (StkTop == StkMax-1)
   return true; // Stack đầy
   return false; // Stack chưa đầy
}
```

Thêm một phần tử vào Stack

```
boolean Push(int newitem)
    {
    if (IsFull())
    return false; // Stack đầy, không thêm vào
      được
    StkTop++;
    StkArr[StkTop] = newitem;
    return  true; // Thêm thành công
    }
```

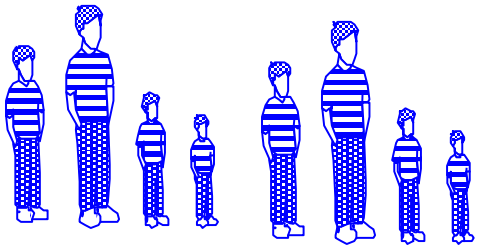
Lấy một phần tử ra khỏi Stack

- Thao tác “Pop”: lấy ra 1 phần tử từ đỉnh Stack

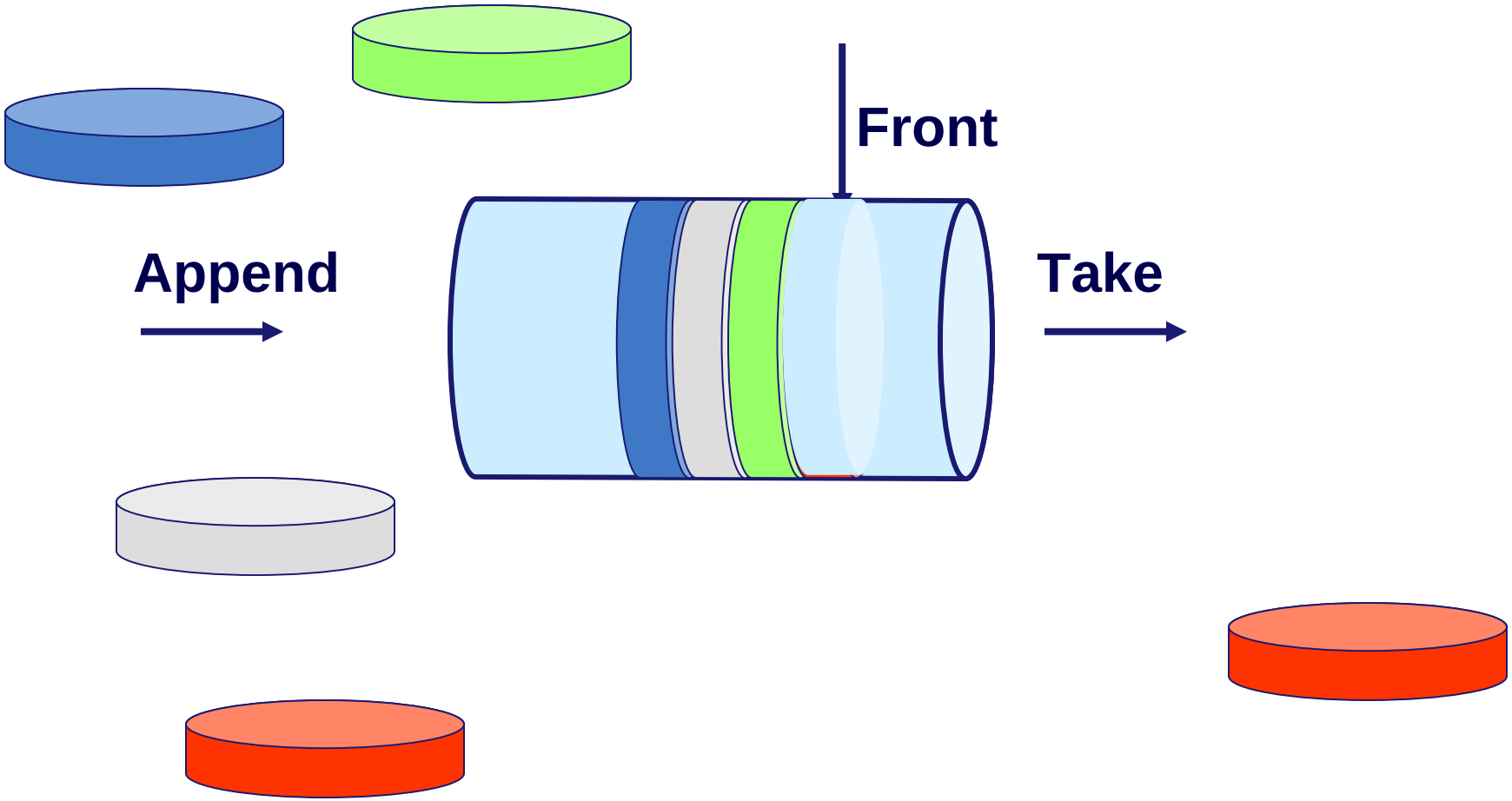
```
boolean Pop(int outitem)
    {
    if (IsEmpty())
    Return false; // Stack rỗng, không lấy ra được
    outitem = StkArr[StkTop];
    StkTop--;
    return true; // Lấy ra thành công
    }
```

Tổ chức Queue theo kiểu danh sách đặc

- Queue là 1 cấu trúc dữ liệu:
  - Gồm nhiều phần tử có thứ tự
  - Hoạt động theo cơ chế “Vào trước – Ra trước” (FIFO – First In, First Out)



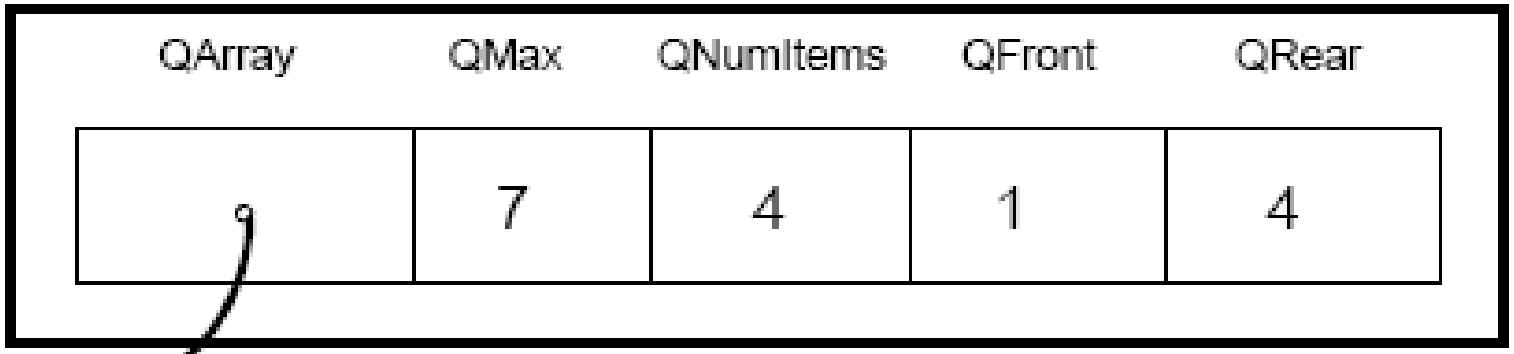
Tổ chức Queue theo kiểu danh sách đặc

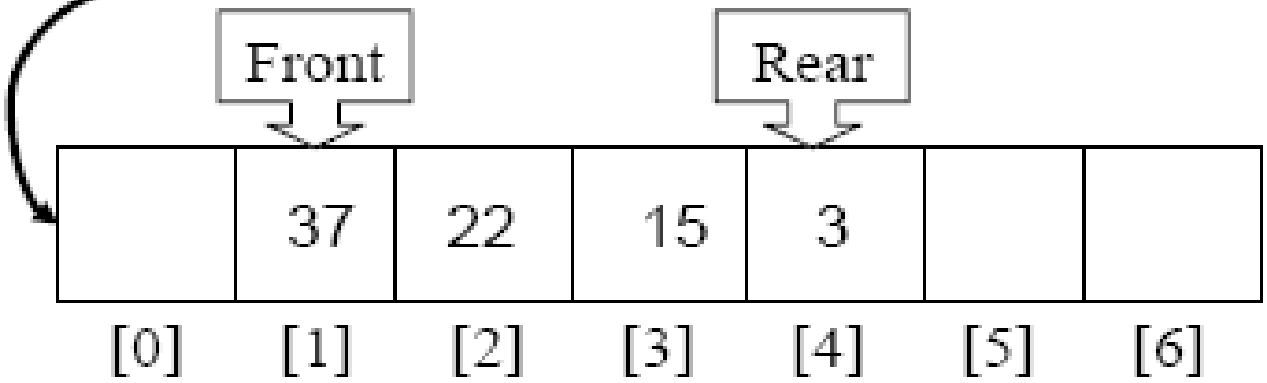


Tổ chức Queue theo kiểu danh sách đặc

- Cấu trúc của Queue
  - Dùng 1 mảng (QArray) để chứa các phần tử
  - Dùng 1 số nguyên (QMax) để lưu số phần tử tối đa trong hàng đợi
  - Dùng 2 số nguyên (QFront, QRear) để xác định vị trí Đầu, Cuối hàng đợi
  - Dùng 1 số nguyên (QNumItems) để lưu số phần tử hiện có trong hàng đợi

Tổ chức Queue theo kiểu danh sách đặc





Tổ chức Queue theo kiểu danh sách đặc

- Các thao tác trên Queue
  - Queue: khởi tạo Queue rỗng
  - IsEmpty: kiểm tra Queue rỗng ?
  - IsFull: kiểm tra Queue đầy ?
  - Append: thêm 1 phần tử vào cuối Queue, có thể làm Queue đầy
  - Take: lấy ra 1 phần tử ở đầu Queue, có thể làm Queue rỗng

Khai báo lớp Queue

```
// Giả sử Queue chứa các phần tử kiểu nguyên (int)
Class Queue {
  private int [] QArray;
  private int QMax;
  private int QNumItems;
  private int QFront;
  private int QRear;
```

Khai báo lớp Queue

```
// Khởi tạo Queue chứa các phần tử kiểu nguyên (int)
public Queue(int size) {
    QArray = new int[size];
     QMax = size;
     QFront = Qrear=  -1; // Queue rỗng
    QNumItems = 0; // chưa có phần tử nào trong Queue

  }
```

Kiểm tra Queue rỗng, đầy

- Thao tác “Kiểm tra Queue rỗng”

```
boolean IsEmpty()
{
if (QNumItems==0)
return true; // Queue rỗng
return false; // Queue không rỗng
}
```

- Thao tác “Kiểm tra Queue đầy”

```
boolean IsFull()
{
if (QNumItems == QMax)
return true; // Queue đầy
return false; // Queue không đầy
}
```

Thêm 1 phần tử vào Queue

- Thao tác: thêm 1 phần tử vào cuối Queue

```
boolean Append(int newitem)
    {
    if (IsFull()) return false; //Queue đầy, không thêm vào được
    QRear++;
    QArray[q.QRear] = newitem; // thêm phần tử vào cuối Queue
    QNumItems++;
    return true; // Thêm thành công
    }
```

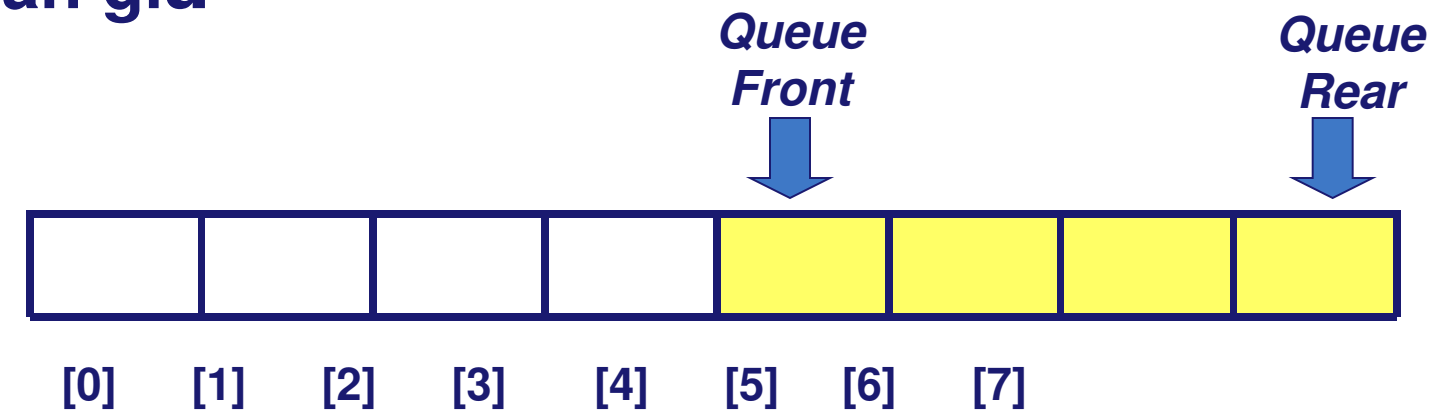
Lấy một phần tử ra khỏi Queue

- Thao tác Take: lấy ra 1 phần tử ở đầu Queue

```
boolean Take(int  itemout)
 {
 if (IsEmpty()) return  false; // Queue rỗng, không lấy ra được
 itemout = QArray[QFront]; // lấy phần tử đầu ra
 QFront++;
 QNumItems--;
 if (QFront==QMax) // nếu đi hết mảng …
 QFront = QRear = -1 ; // … quay trở về đầu mảng
 return true; // lấy thành công
 }
```

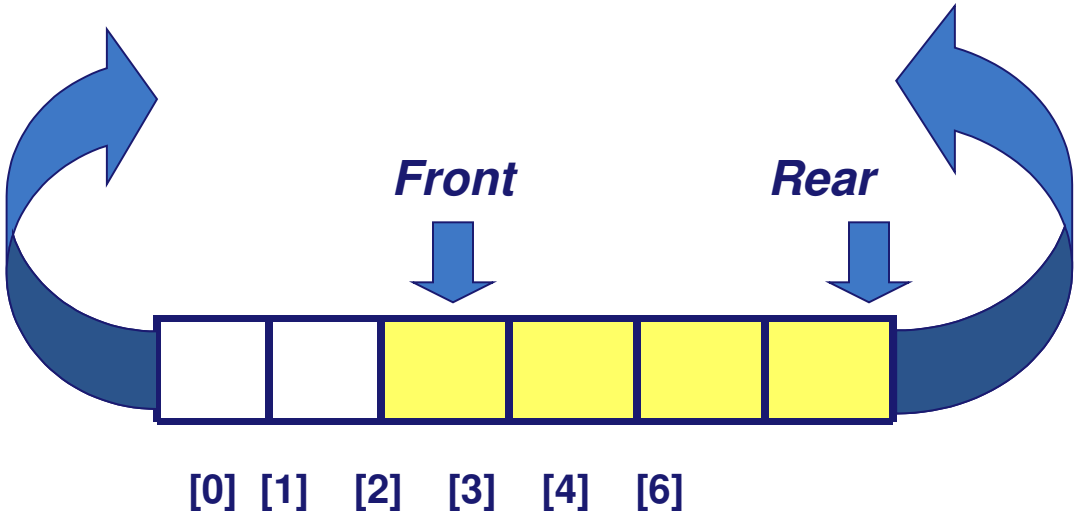
Hạn chế của Queue theo kiểu danh sách đặc

- Khi thêm nhiều phần tử, sẽ làm "tràn" mảng "Tràn giả"

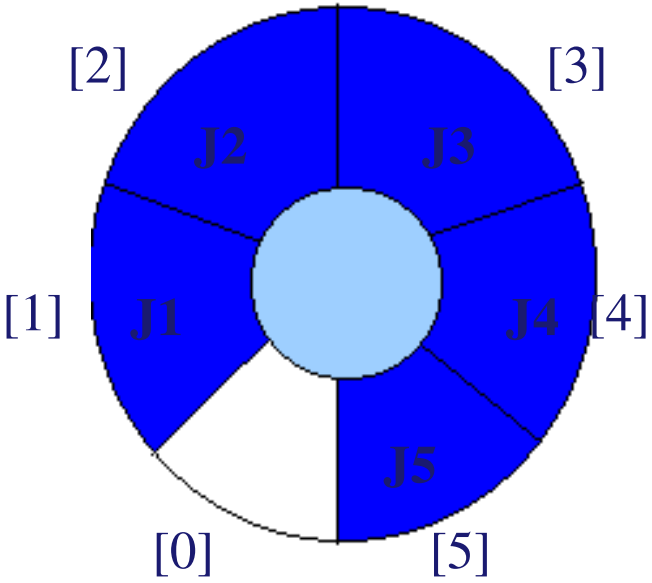


Giải pháp

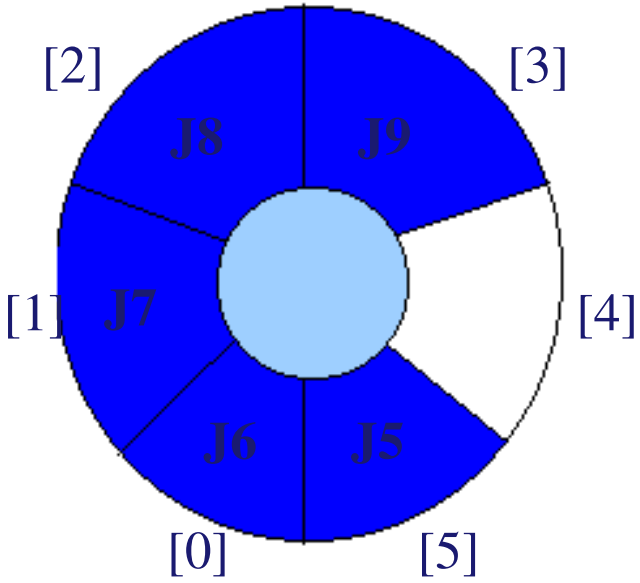
- Giải pháp cho tình huống “tràn giả”: xử lý mảng như là 1 mảng vòng tròn



Tổ chức Queue theo kiểu danh sách đặc



front =0
rear = 5



front =4
rear =3

Bài tập

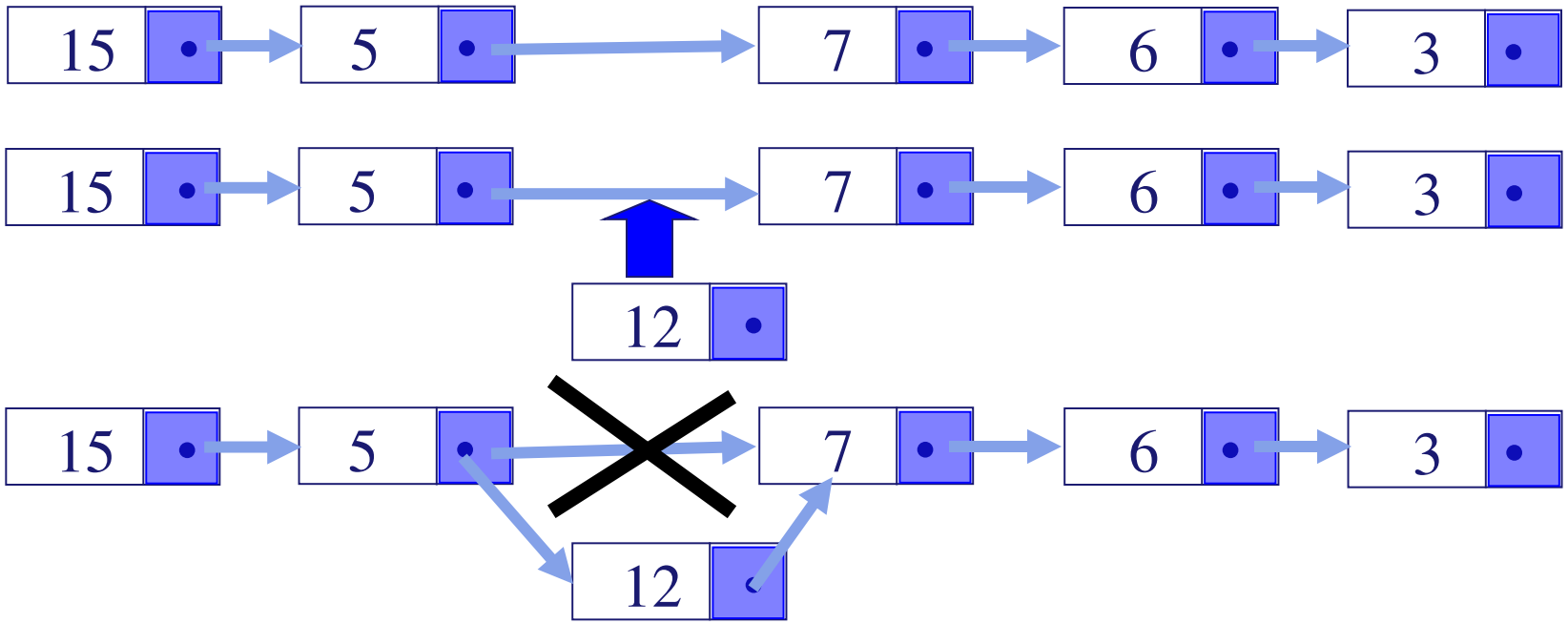
- Viết lại các giải thuật, bổ sung, lấy phần tử cho QUEUE nối vòng

DANH SÁCH LIÊN KẾT

- Đặt vấn đề:
  - Nếu muốn chèn vào 1 phần tử vào mảng ?
    - Chi phí là O(n)
  - Muốn xóa một phần tử trong mảng ?
    - Chi phí O(n)

DANH SÁCH LIÊN KẾT

- Ta tách rời các phần tử của mảng, và kết nối chúng lại với nhau bằng một “móc xích”



DANH SÁCH LIÊN KẾT

- Một dãy tuần tự các nút (Node)
- Giữa hai nút có một tham chiếu
- Các nút không cần phải lưu trữ liên tiếp nhau trong bộ nhớ
- Có thể mở rộng tuỳ ý (chỉ giới hạn bởi dung lượng bộ nhớ)
- Thao tác Chèn/Xóa không cần phải dịch chuyển phần tử
- Quản lý danh sách bởi con trỏ đầu pHead
- Có thể truy xuất đến các phần tử khác thông qua con trỏ liên kết

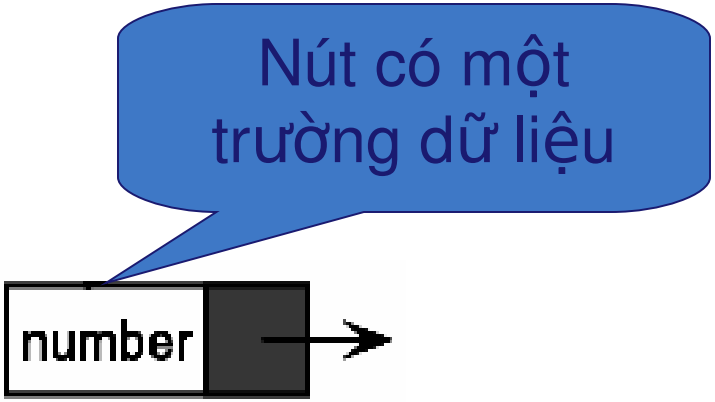
DANH SÁCH LIÊN KẾT

- Cấu tạo nút
  - Tạo lập bằng cách cấp phát bộ nhớ động
  - Mỗi nút có 2 thông tin:
    - Dữ liệu (data)
    - Tham chiếu liên kết đến phần tử kế tiếp trong danh sách (Next pointer link)

DANH SÁCH LIÊN KẾT

- Cấu tạo nút : Gồm 2 thành phần

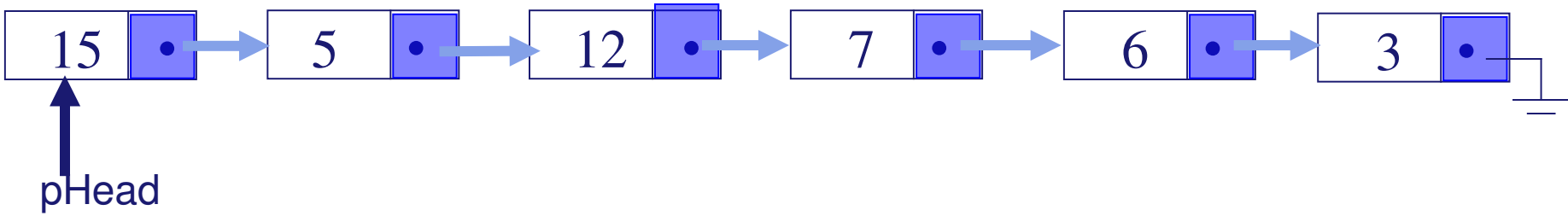
```
public class Node {
    public Object Element;
    public Node Link;
    public Node() {
        Element = null;
        Link = null;
            }
  public Node(Object theElement) {
        Element = theElement;
        Link = null;
            }
    }
```

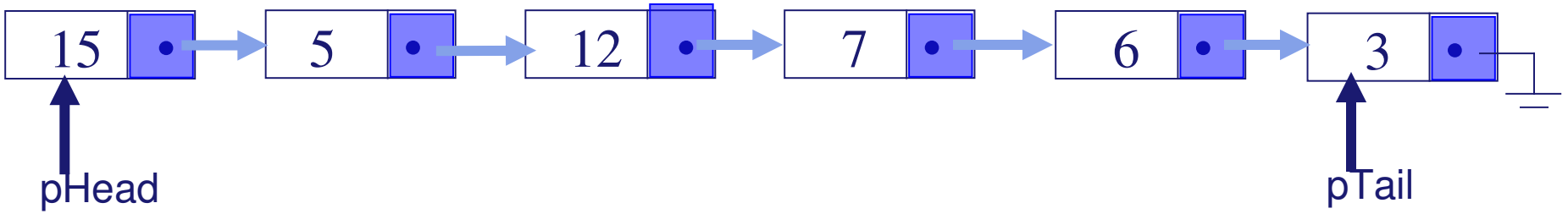


Cấu tạo của danh sách liên kết

- Quản lý danh sách qua con trỏ đầu pHead, có thể thêm con tro cuối pTail
- Có hai cách tổ chức danh sách
  - pHead, pTail là một nút của danh sách
  - pHead, pTail không phải là nút mà chỉ là con trỏ trỏ đến nút đầu và nút cuối của danh sách

Cấu tạo của danh sách liên kết





ƯU VÀ NHƯỢC CỦA DANH SÁCH LIÊN KẾT

- Ưu điểm
  - Tận dụng được không gian nhớ để lưu trữ
    - Các nút không cần lưu trữ kế tiếp nhau
    - Có thể mở rộng kích thước tùy ý (phụ thuộc bộ nhớ)
  - Việc thêm vào hay loại bỏ được tiến hành dễ dàng O(1)
  - Dễ dàng kết nối hay phân rã danh sách
- Nhược điểm
  - Truy xuất tuần tự từng phần tử

Các loại danh sách liên kết

- Danh sách liên kết đơn (Single-Linked list)
  - Mỗi nút chỉ có 1 con trỏ liên kết (pNext)
- Danh sách liên kết đôi (Double-Linked list)
  - Mỗi nút có 2 con trỏ liên kết (pPrev, pNext)
- Danh sách đa liên kết (Multi-Linked list)
  - Mỗi nút có nhiều hơn 2 con trỏ liên kết
- Danh sách liên kết vòng (Circular-Linked list)
  - Liên kết ở nút cuối cùng của danh sách chỉ đến nút đầu tiên trong danh sách

Danh sách liên kết đơn

- Mỗi nút, bao gồm hai phần,
  - Phần Data: chứa dữ liệu, có thể nhiều hơn 1 trường
  - Phần next: chỉ có duy nhất một liên kết đến nút kế tiếp
  - Phần tử cuối cùng có liên kết NULL

MẢNG & DANH SÁCH LIÊN KẾT

- Mảng
  - Phải biết trước số phần tử
  - Lưu trữ tuần tự
  - Khi chèn và xóa phải dịch chuyển các phần tử
  - Truy xuất qua chỉ mục
- Danh sách liên kết
  - Số phân tử tùy biến
  - Sử dụng con trỏ
  - Khi chèn/xóa chỉ cần thay đổi con trỏ liên kết
  - Truy xuất tuần tự

Các thao tác trên danh sách liên kết đơn

- Tạo lập danh sách rỗng
- Kiểm tra danh sách rỗng
- Đếm số phần tử trong danh sách
- Thêm 1 nút vào danh sách
- Xóa 1 nút khỏi danh sách
- Duyệt danh sách
- Tìm 1 phần tử trong danh sách

Tạo lập danh sách rỗng

```
public class LinkedList {
   protected Node header;
   public LinkedList() {
   header = new Node("header");
    }
...
}
```

Tạo lập danh sách rỗng

- Tạo lập danh sách rỗng

```
void Init_List( Node pHead)
    {
      pHead = NULL;
    }
```

Các thao tác trên danh sách liên kết đơn

- Kiểm tra danh sách rỗng

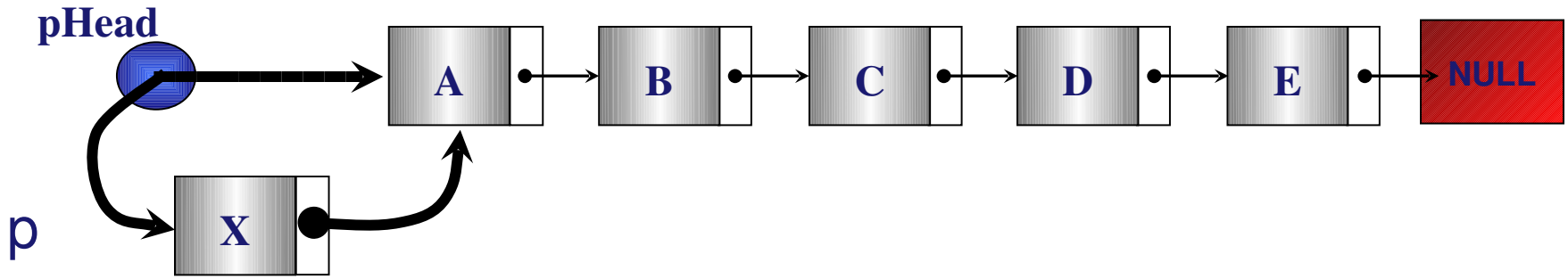
```
boolean IsEmptyList( pHead)
   { return (pHead ==NULL); }
```

- Đếm số phần tử trong danh sách

```
int CountNode(Node pHead){
      int count = 0;
      Node p = pHead;
    while (p != NULL)
          { cout++ ; p = p.Link; }
        return cout;
      }
```

Các thao tác trên danh sách liên kết đơn

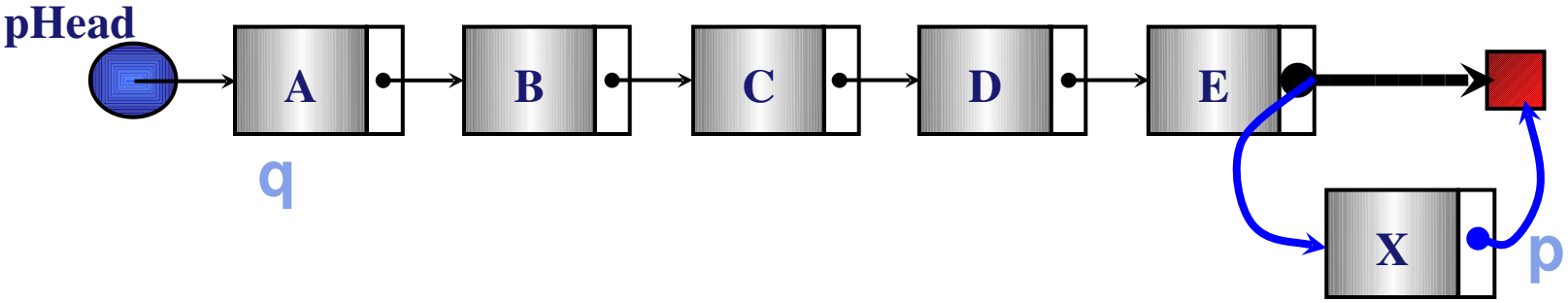
- Thêm một nút p vào đầu danh sách
  - Nếu Danh sách rỗng Thì
    - Gán: pHead = p;
  - Ngược lại
    - p.Link = pHead;
    - pHead = p;



Chèn nút P vào cuối Danh sách

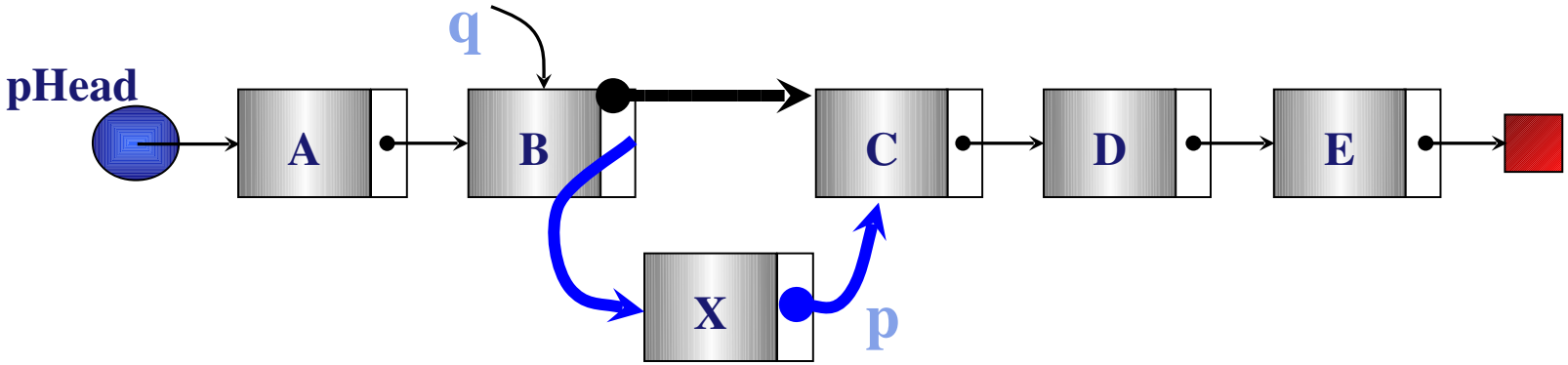
- Chèn nút p vào cuối
  - Nếu Danh sách rỗng Thì
    - Gán: pHead = p;
  - Ngược lại
    - Gán q = pHead;
    - Đi về nút cuối của danh sách

      (while (q.Link != NULL) q = q.Link;)
    - q.Link = p;



Thêm một nút p vào vào sau nút q

- Nếu (q != NULL)
  - P.Link = q.Link;
  - Q.Link = p ;
- Ngược lại: q = p;

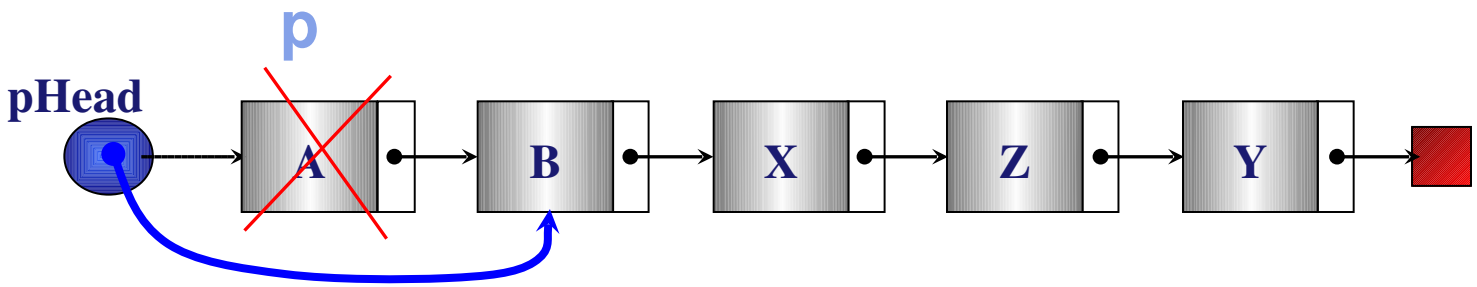


Các thao tác trên danh sách liên kết đơn

- Xóa một nút khỏi danh sách
  - Xóa nút đầu danh sách
  - Xóa một nút đứng sau nút q
  - Xóa một nút có khóa k

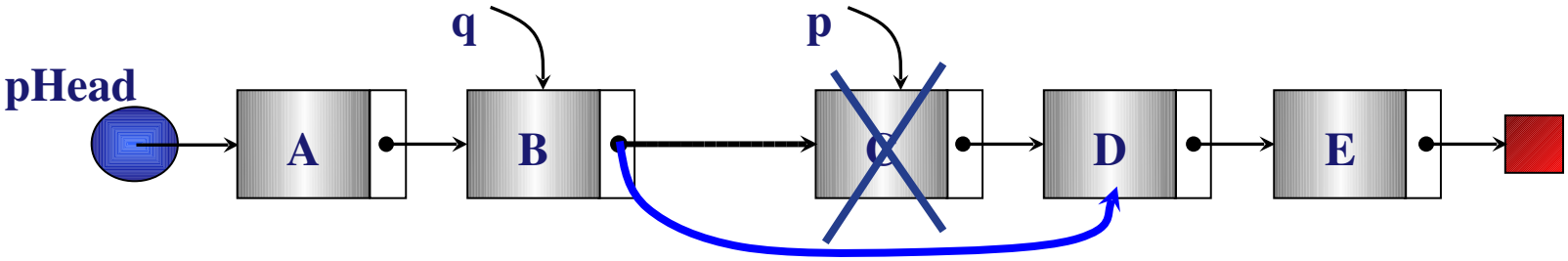
Các thao tác trên danh sách liên kết đơn

- **Xóa nút đầu danh sách**
  - Nếu (pHead != NULL) thì
    - p = pHead;*// p là nút cần xóa*
    - pHead = pHead.Link;



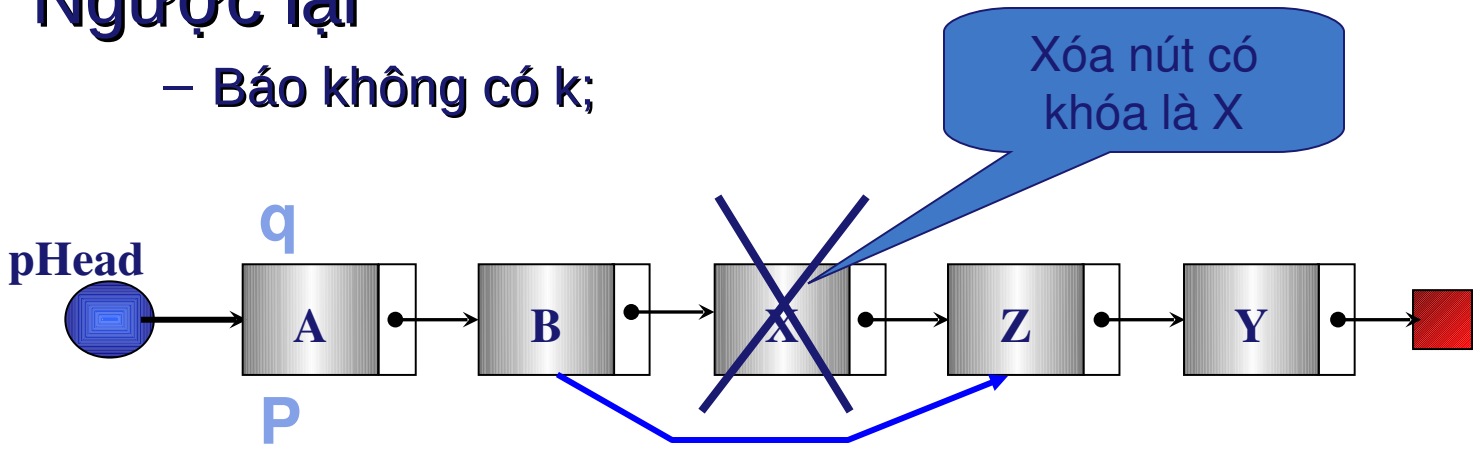
Các thao tác trên danh sách liên kết đơn

- Xóa một nút p, đứng sau q
  - Nếu (q!= NULL) thì
    - p = q.Next; *// p là nút cần xóa*
    - Q.Next = p.Next; *// tách p ra khỏi ds*
    - delete p; *// Giải phóng p*
  - Ngược lại:
    - Xóa nút đầu danh sách



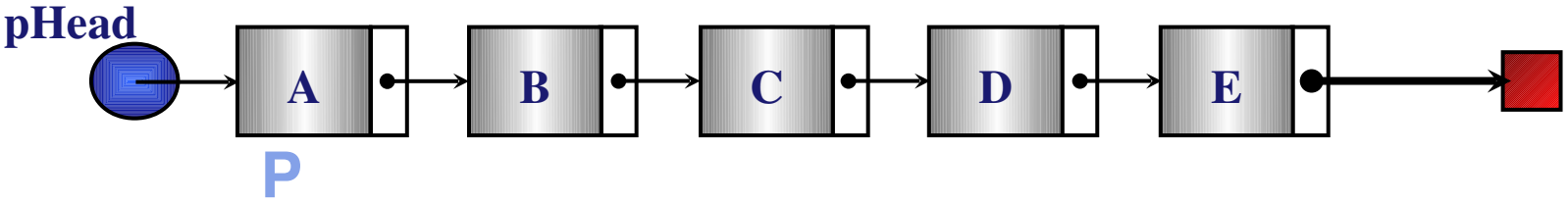
Các thao tác trên danh sách liên kết đơn

- Xóa một nút có khóa k
  - Tìm nút p có khóa k và phần tử q đứng trước nó
  - Nếu (p != NULL) // tìm thấy k
    - xóa p đứng sau nút q;
  - Ngược lại
    - Báo không có k;



Các thao tác trên danh sách liên kết đơn

- Duyệt danh sách
  - p = pHead;
  - Trong khi chưa hết danh sách
    - Xử lý nút p ;
    - p= p.pNext;



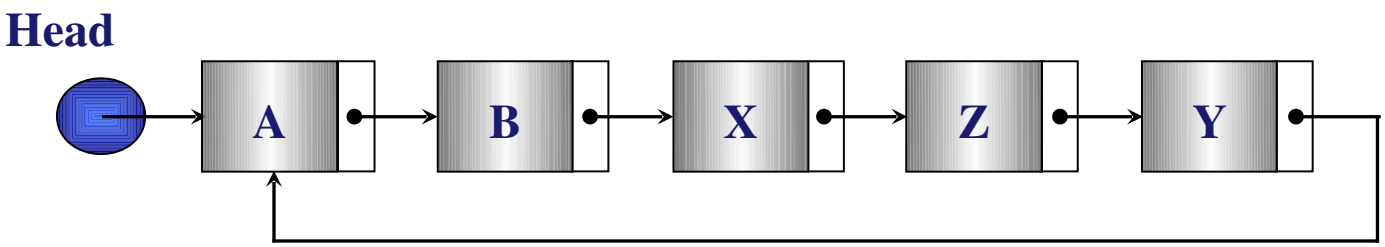
Duyệt danh sách

```
void TraverseList(Node pHead)
 {
  Node p = pHead;
  while (p !=NULL) {
            <Xử lý nút p>;
            p = p.Link; // chuyển sang nút kế
            }
 }
```

Bài tập

- Dùng danh sách liên kết quản lý sinh viên trong lớp (mssv, họ, tên, ngày sinh, điểm tổng kết học kỳ).
- Thực hiện các thao tác: thêm, bớt, sắp xếp sinh viên (theo điểm tổng kết) trong danh sách

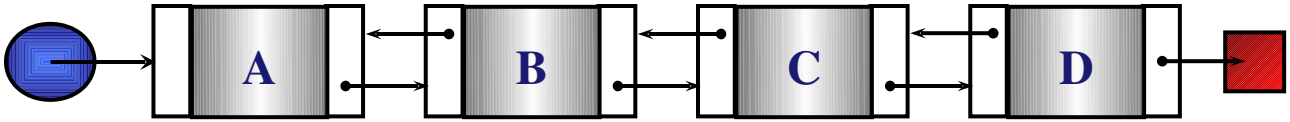
Danh sách liên kết vòng



- Các thao tác trên danh sách liên kết vòng
  - Giải thuật thêm một nút vào đầu danh sách
  - Giải thuật thêm một nút vào danh sách
  - Giải thuật loại một nút ra khỏi danh sách

Danh sách liên kết kép

- Mỗi nút có 2 con trỏ liên kết:



- Khai báo:

Danh sách liên kết kép

- Khi xóa 1 nút, không cần phải duyệt danh sách để tìm phần tử đứng trước
- Được sử dụng đối với các dữ liệu mà ta cần truy xuất theo cả 2 chiều:
- Bài tập: Viết các giải thuật, khởi tạo, bổ sung, tìm kiếm, duyệt, xóa trên danh sách liên kết kép.

Tổ chức STACK và QUEUE bằng danh sách liên kết

- Sinh viên tự cài đặt.
  - Tổ chức ngăn xếp (Stack)
  - Tổ chức hàng đợi (Queue)

Q&A