



# CHƯƠNG 2: CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN

Nguyễn Văn Linh

Khoa Công nghệ Thông tin & Truyền thông

[nvlinh@ctu.edu.vn](mailto:nvlinh@ctu.edu.vn)



CANTHO UNIVERSITY

# MỤC TIÊU

- Hiểu được khái niệm về các kiểu dữ liệu trừu tượng cơ bản: danh sách, ngăn xếp và hàng đợi.
- Vận dụng được các cấu trúc dữ liệu mảng và con trỏ để cài đặt danh sách, ngăn xếp và hàng đợi.
- Vận dụng được các kiểu dữ liệu trừu tượng danh sách, ngăn xếp và hàng đợi để giải một số bài toán thực tế.



CANTHO UNIVERSITY

# KIỂU DỮ LIỆU TRỪU TƯỢNG DANH SÁCH (LIST)

- Khái niệm danh sách
- Các phép toán trên danh sách
- Cài đặt danh sách
  - Bảng mảng
  - Bảng con trỏ



CANTHO UNIVERSITY

# KHÁI NIỆM DANH SÁCH

- Là tập hợp hữu hạn các phần tử có thứ tự tuyến tính và có cùng một kiểu.
- DS thường được biểu diễn dưới dạng:  $a_1, a_2, \dots, a_n$
- Nếu
  - $n=0$ : danh sách rỗng
  - $n>0$ : phần tử đầu tiên là  $a_1$ , phần tử cuối cùng là  $a_n$
- Độ dài của danh sách: số phần tử của danh sách
- Mỗi phần tử trong danh sách có một vị trí.
- Thứ tự tuyến tính của các phần tử trong danh sách là thứ tự theo vị trí xuất hiện của chúng. Ta nói  $a_i$  đứng trước  $a_{i+1}$  ( $i=1..n-1$ ). Theo đó danh sách có 2 phần tử  $a, b$  khác danh sách  $b, a$ .



CANTHO UNIVERSITY

# CÁC PHÉP TOÁN TRÊN DANH SÁCH

Tên phép toán	Ý nghĩa của phép toán
<code>makenullList(L)</code>	Khởi tạo danh sách L rỗng
<code>emptyList(L)</code>	Kiểm tra xem danh sách L có rỗng hay không
<code>first(L)</code>	Trả về vị trí của phần tử đầu tiên trong danh sách L
<code>endList(L)</code>	Trả về vị trí sau vị trí cuối cùng trong danh sách L
<code>Next(P,L)</code>	Trả về vị trí sau vị trí P trong danh sách L
<code>previous(P,L)</code>	Trả về vị trí trước vị trí P trong danh sách L



CANTHO UNIVERSITY

# CÁC PHÉP TOÁN TRÊN DANH SÁCH (tt)

Tên phép toán	Ý nghĩa của phép toán
retrieve(P,L)	Trả về giá trị của phần tử tại vị trí P trong danh sách L
locate(X,L)	Tìm X trong danh sách L. Nếu tìm thấy thì trả về vị trí của phần tử đầu tiên có giá trị X trong danh sách L. Nếu không tìm thấy thì trả về endList(L)
insertList(X,P,L)	Xen phần tử có giá trị X vào danh sách L tại vị trí P
deleteList(P,L)	Xóa phần tử tại vị trí P trong danh sách L



CANTHO UNIVERSITY

# VÍ DỤ

```
void SORT(List *L)
{
    Position p,q, e;           //kiểu vị trí của các phần tử trong danh sách
    p= first(L); //vị trí phần tử đầu tiên trong danh sách
    while (p!= endList(L))
    {
        q=Next(p,L); //vị trí phần tử đứng ngay sau phần tử p
        while (q!=endList(L))
        {
            if (retrieve(p,L) > retrieve(q,L))
                swap(p,q,L); // hoán đổi nội dung 2 phần tử
            q=Next(q,L);
        }
        p=Next(p,L);
    }
}
```



CANTHO UNIVERSITY

# CÀI ĐẶT DANH SÁCH BẰNG MẢNG

- Sử dụng một mảng để biểu diễn cho một danh sách.
- Các phần tử của mảng lưu trữ các phần tử của danh sách, bắt đầu từ phần tử đầu đầu tiên.
- Ta phải ước lượng số phần tử tối đa của danh sách để khai báo độ dài của mảng.
- Ta phải lưu trữ độ dài hiện tại của danh sách (Last)





CANTHO UNIVERSITY

# MÔ HÌNH

Chỉ số	Mảng	Vị trí
0		1
1		2
2		3
	...	
Last-1		Last
	...	
MaxLength-1		

- $Vị\ trí = Chỉ\ số + 1$
- Last là độ dài của danh sách
- Danh sách rỗng:  $Last == 0$
- Danh sách đầy:  $Last == MaxLength$
- Thêm phần tử: Tăng Last
- Xóa phần tử: Giảm Last



CANTHO UNIVERSITY

# KHAI BÁO

```
#define MaxLength ...  
    //Độ dài tối đa của danh sách  
typedef ... ElementType;  
    //kiểu của phần tử trong danh sách  
typedef int Position;  
    //kiểu vị trí của các phần tử  
typedef struct {  
    ElementType Elements[MaxLength];  
    //mảng chứa các phần tử của danh sách  
    Position Last; //giữ độ dài danh sách  
} List;
```



CANTHO UNIVERSITY

# KHỞI TẠO DANH SÁCH RỖNG

- Input: Danh sách L
- Output: Danh sách L rỗng (truyền địa chỉ)
- Thuật toán: Cho độ dài danh sách bằng 0

```
void makenullList(List *L) {  
    (*L).Last=0;}
```



CANTHO UNIVERSITY

# KIỂM TRA DANH SÁCH RỖNG

- Input: Danh sách L
- Output: Số nguyên 1 hoặc 0
- Thuật toán: Kiểm tra xem độ dài của danh sách có bằng 0 hay không?

```
int emptyList(List L){  
    return L.Last==0;  
}
```



CANTHO UNIVERSITY

# XÁC ĐỊNH VỊ TRÍ ĐẦU TIÊN

- Input: Danh sách L
- Output: Vị trí đầu tiên của danh sách L
- Thuật toán: Trả về 1

```
Position first(List L){  
    return 1;  
}
```



CANTHO UNIVERSITY

# XÁC ĐỊNH VỊ TRÍ SAU VỊ TRÍ CUỐI CÙNG

- Input: Danh sách L
- Output: Vị trí sau vị trí cuối cùng trong danh sách L
- Thuật toán: Trả về Last+1  
Position endList(List L){  
    return L.Last+1;  
}



CANTHO UNIVERSITY

# VỊ TRÍ SAU VỊ TRÍ P

- Input: Vị trí P, danh sách L
- Output: Vị trí sau vị trí P trong ds L
- Thuật toán: Trả về P+1

```
Position Next(Position P, List L){  
    return P+1;  
}
```



CANTHO UNIVERSITY

# VỊ TRÍ TRƯỚC VỊ TRÍ P

- Input: Vị trí P, danh sách L
- Output: Vị trí trước vị trí P trong ds L
- Thuật toán: Trả về P-1

```
Position previous(Position P, List L){  
    return P-1;  
}
```





CANTHO UNIVERSITY

# XÁC ĐỊNH GIÁ TRỊ TẠI VỊ TRÍ P

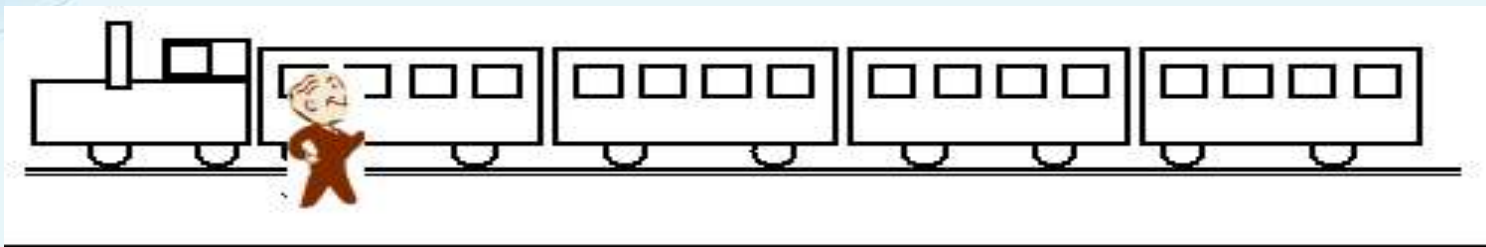
- Input: Vị trí P, danh sách L
- Output: Giá trị của phần tử tại vị trí P trong ds L
- Thuật toán: Trả về giá trị tại phần tử mảng Elements có chỉ số P-1

```
ElementType retrieve(Position P, List L){  
    return L.Elements[P-1];  
}
```



CANTHO UNIVERSITY

# TÌM PHẦN TỬ X TRONG DANH SÁCH L (1)



- Input: Phần tử X, danh sách L
- Output: Vị trí của X trong ds L
- Thuật toán:
  - Tiến hành tìm từ đầu danh sách cho đến khi tìm thấy hoặc hết danh sách
  - Nếu tìm thấy thì trả về vị trí đầu tiên của X
  - Nếu không tìm thấy thì trả về (endList(L))
  - Gợi ý: Sử dụng các hàm first, endList, Retrive, Next, biến Found



CANTHO UNIVERSITY

## TÌM PHẦN TỬ X TRONG DANH SÁCH L (2)

```
Position locate(ElementType X, List L){  
    Position P = first(L) , E=endList(L);  
    int Found = 0;  
    while ((P != E) && (!Found))  
        if (retrieve(P,L) == X) Found = 1;  
        else P = Next(P, L);  
    return P;  
}
```



CANTHO UNIVERSITY

## TÌM PHẦN TỬ X TRONG DANH SÁCH L (2)

```
Position locate(ElementType X, List L){  
    Position P, E=endList(L);  
    for (P= first(L); P!=E, P=Next(P,L))  
        if (retrieve(P,L) == X) return 1;  
    return 0;  
}
```



CANTHO UNIVERSITY

# XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (1)

- Input: Phần tử X, vị trí P, danh sách L
- Output: Hàm kh có giá trị trả về; Nhưng danh sách L sau khi đã xen X phải được truyền trở lại cho TSTT (truyền địa chỉ)
- Thuật toán: Có 2 trường hợp xảy ra:
  - Nếu danh sách đầy ( $Last = MaxLength$ ) thì không thể xen thêm



CANTHO UNIVERSITY

# XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (2)

– Danh sách không đầy:

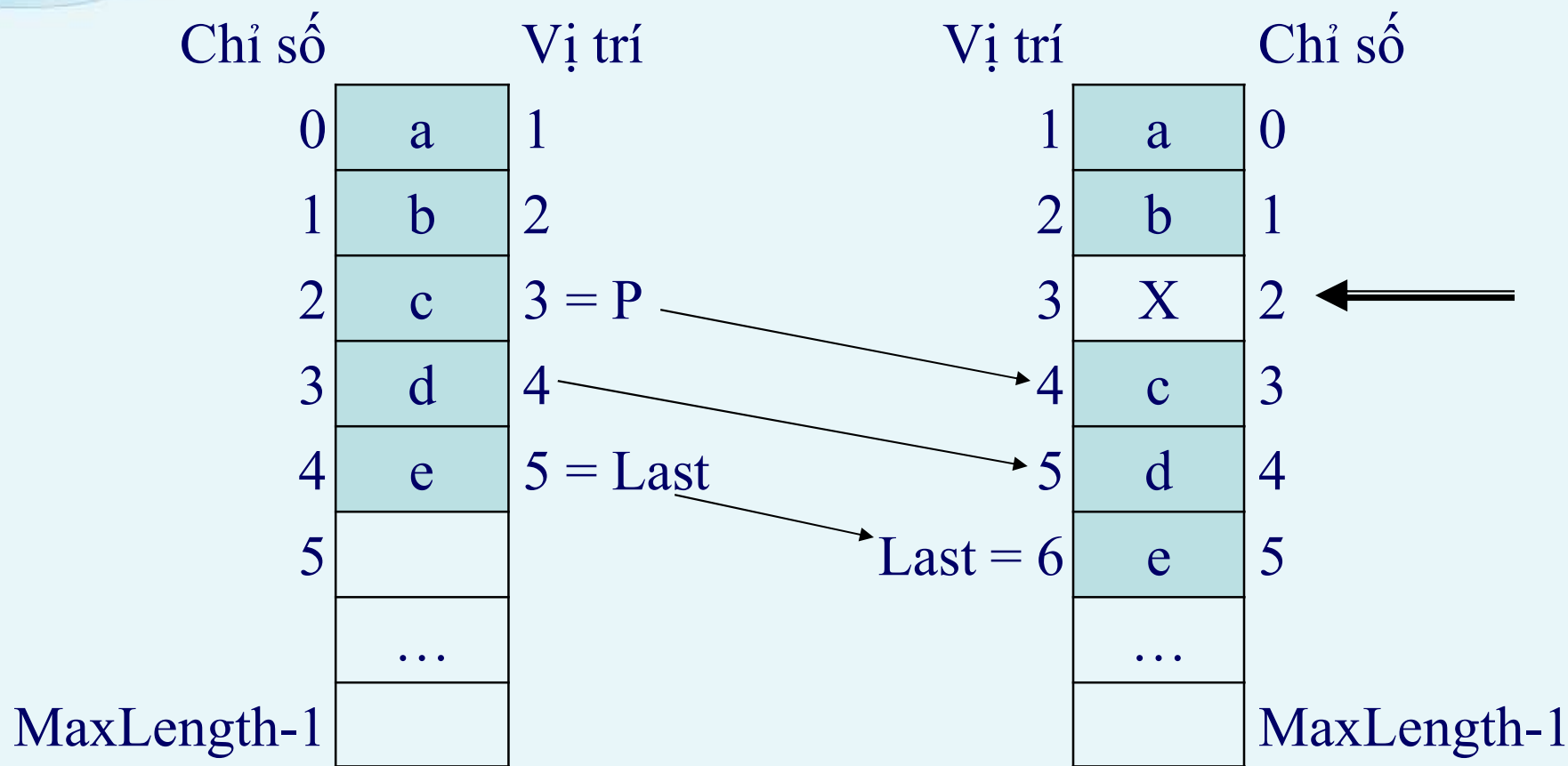
- Dời các phần tử từ vị trí Last đến vị trí P ra SAU một vị trí
- Đưa X vào vị trí P (phần tử mảng có chỉ số P-1)
- Tăng Last lên 1 đơn vị



CANTHO UNIVERSITY

# XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (3)

Ví dụ xen X vào L tại vị trí P=3





CANTHO UNIVERSITY

# XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (4)

```
void insertList(ElementType X, Position P, List *L){  
    if ((*L).Last==MaxLength)  
        printf("\nDanh sach day!");  
    else {  
        for(Position i =(*L).Last; i>=P; i--)  
            (*L).Elements[i] = (*L).Elements[i-1];  
        (*L).Last++;  
        (*L).Elements[P-1] = X;  
    }  
}
```





CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (1)

- Input: Vị trí P, danh sách L
- Output: Hàm không có giá trị trả về, nhưng danh sách L sau khi đã xóa phải được truyền lại cho TSTT (truyền địa chỉ)
- Thuật toán:



CANTHO UNIVERSITY

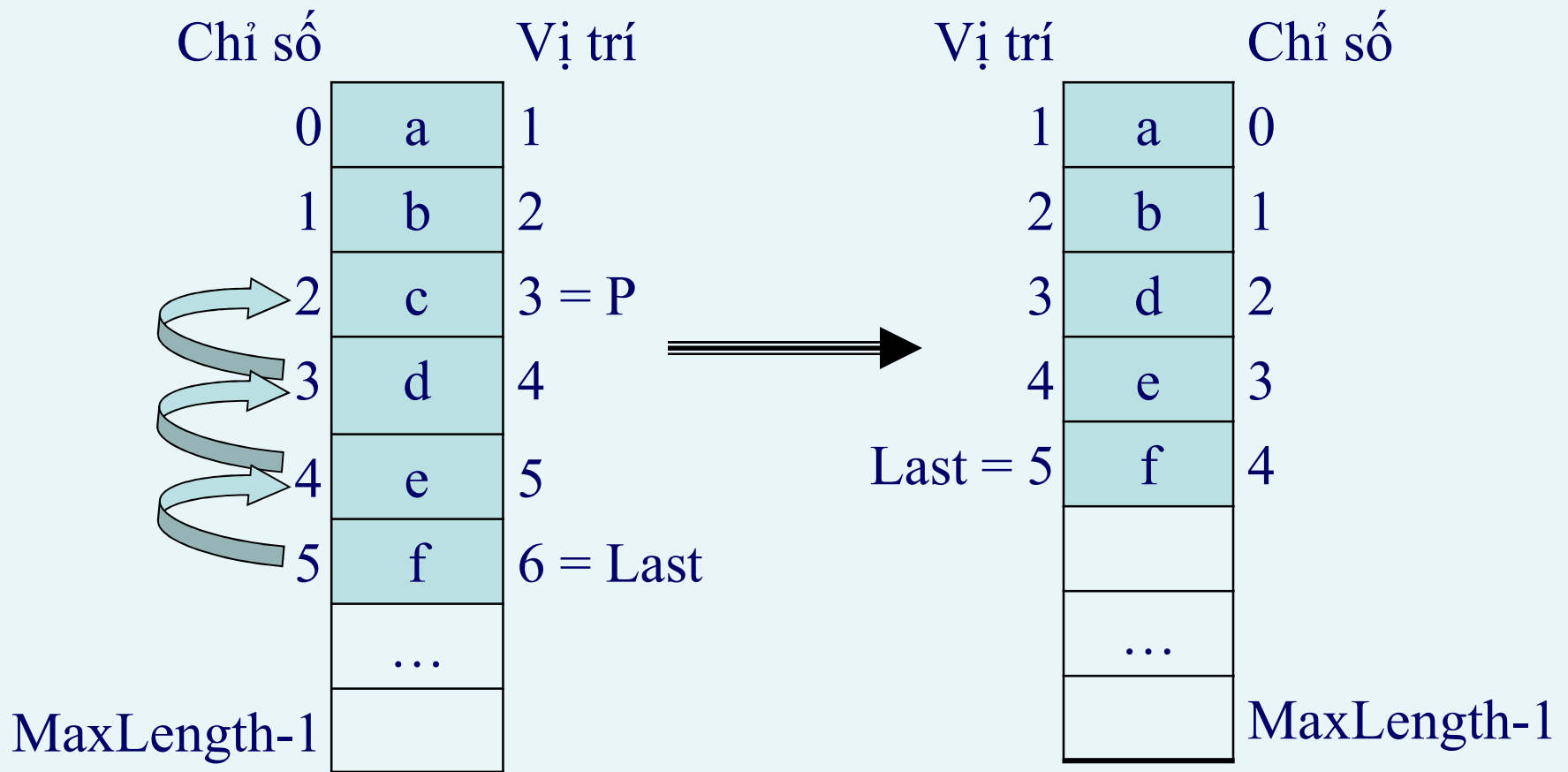
## XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (2)

- Dời các phần tử từ vị trí  $P+1$  đến Last ra trước một vị trí (Dời các phần tử mảng có chỉ số từ  $P$  đến Last-1 ra trước).
- Giảm Last một đơn vị



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (3)





CANTHO UNIVERSITY

## XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (4)

```
void deleteList(Position P, List *L){  
    for(int i=P; i<=(*L).Last-1; i++)  
        (*L).Elements[i-1] = (*L).Elements[i];  
    (*L).Last--;  
}
```



CANTHO UNIVERSITY

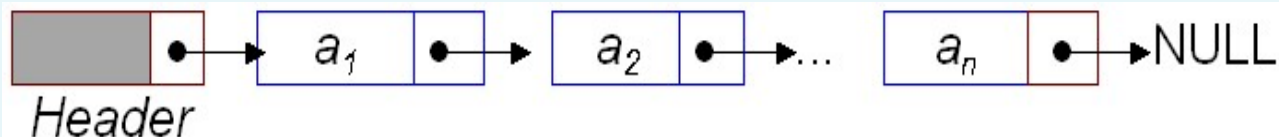
## CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

- DSLK là một dãy các node được kết nối với nhau bằng con trỏ.
- Mỗi node bao gồm 2 phần: Một phần để lưu trữ dữ liệu và một phần là con trỏ (Next) để trỏ tới node kế tiếp.
- Node đầu tiên là đầu (Header) của DSLK.
- Sử dụng danh sách liên kết để biểu diễn một danh sách.
- Mỗi node trong DSLK (trừ node đầu tiên) lưu trữ một phần tử của danh sách.



CANTHO UNIVERSITY

# MÔ HÌNH



- Các phần tử  $a_i$  của danh sách được lưu trong phần dữ liệu của các node trong DSLK.
- Vị trí của một phần tử trong danh sách là địa chỉ của node lưu trữ phần tử đó. Địa chỉ này được lưu trong phần Next của node đứng trước.
- Có thể hiểu nôm na: Vị trí của một phần tử là node đứng trước phần tử đó.
- Theo đó thì Header là vị trí của  $a_1$ ,  $a_1$  là vị trí của  $a_2, \dots, a_{n-1}$  là vị trí của  $a_n$  và  $a_n$  là vị trí endList.



CANTHO UNIVERSITY

# KHAI BÁO

```
typedef ... ElementType;  
typedef struct Node {  
    ElementType Element;  
    struct Node* Next;  
};  
typedef struct Node* Position;  
typedef Position List; // Danh sách là Vị trí
```



CANTHO UNIVERSITY

# KHỞI TẠO DANH SÁCH RỖNG

- Tên hàm: `makenullList`
- Input: Danh sách L
- Output: Danh sách L rỗng (truyền địa chỉ)
- Thuật toán:
  - Cấp phát vùng nhớ cho L
  - Đặt trường Next của L bằng NULL





CANTHO UNIVERSITY

```
void makenullList(List *L) {  
    (*L) = (struct Node*)malloc(sizeof(struct Node));  
    (*L) ->Next = NULL;  
}
```



CANTHO UNIVERSITY

# KIỂM TRA DANH SÁCH RỖNG

- Tên hàm: emptyList
- Input: Danh sách L
- Output: Số nguyên 1 hoặc 0
- Thuật toán: Xem trường Next của L có bằng NULL hay không?



CANTHO UNIVERSITY

```
int emptyList(List L) {  
    return (L->Next==NULL);  
}
```



CANTHO UNIVERSITY

# XÁC ĐỊNH VỊ TRÍ ĐẦU TIÊN

- Tên hàm: first
- Input: Danh sách L
- Output: Vị trí đầu tiên của danh sách L
- Thuật toán: Trả về L



CANTHO UNIVERSITY

```
Position first(List L){  
    return L;  
}
```



CANTHO UNIVERSITY

# XÁC ĐỊNH VỊ TRÍ SAU VỊ TRÍ CUỐI CÙNG (1)

- Tên hàm: endList
- Input: Danh sách L
- Output: Vị trí sau vị trí cuối cùng trong danh sách L
- Thuật toán:
  - Đặt vị trí P vào đầu danh sách L.
  - Di chuyển P ra sau cho tới khi  $P \rightarrow \text{Next} == \text{NULL}$
  - Trả về P



CANTHO UNIVERSITY

## XÁC ĐỊNH VỊ TRÍ SAU VỊ TRÍ CUỐI CÙNG (2)

```
Position endList(List L) {  
    Position P = first(L);  
    while (P->Next!=NULL)  
        P=P->Next;  
    return P;  
}
```



CANTHO UNIVERSITY

# VỊ TRÍ SAU VỊ TRÍ P

- Tên hàm: Next
- Input: Vị trí P, danh sách L
- Output: Vị trí sau vị trí P trong ds L
- Thuật toán: Trả về P->Next  
Position Next(Position P, List L){  
    return P->Next;  
}  
• Chú ý phân biệt 2 chữ Next





CANTHO UNIVERSITY

# VỊ TRÍ TRƯỚC VỊ TRÍ P (1)

- Tên hàm: previous
- Input: Vị trí P, danh sách L
- Output: Vị trí trước vị trí P trong ds L
- Thuật toán:
  - Đặt Q vào đầu danh sách L.
  - Di chuyển Q ra sau cho đến khi  $Q \rightarrow \text{Next} == P$
  - Trả về Q



CANTHO UNIVERSITY

## VỊ TRÍ TRƯỚC VỊ TRÍ P (2)

```
Position previous(Position P, List L){  
    Position Q = first(L);  
    while (Q->Next != P) Q = Q->Next;  
    return Q;  
}
```



CANTHO UNIVERSITY

# XÁC ĐỊNH GIÁ TRỊ TẠI VỊ TRÍ P

- Tên hàm: retrieve
- Input: Vị trí P, danh sách L
- Output: Giá trị của phần tử tại vị trí P trong ds L
- Thuật toán: Trả về phần tử của  $P \rightarrow \text{Next}$



CANTHO UNIVERSITY

ElementType retrieve(Position P, List L)

{

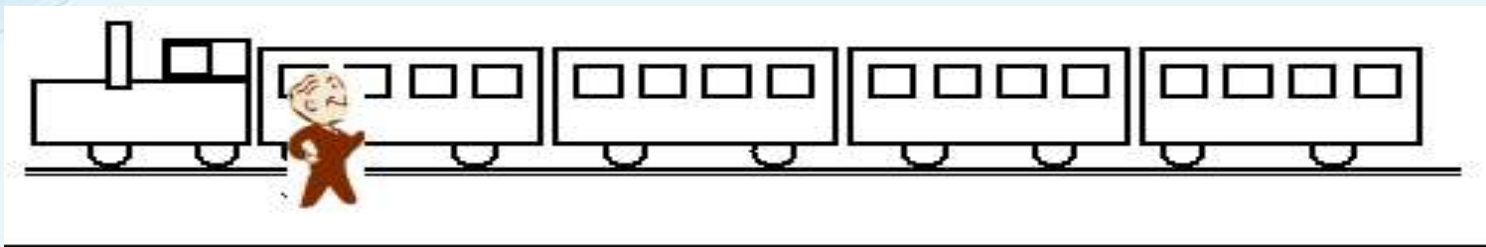
    return P->Next->Element;

}



CANTHO UNIVERSITY

# TÌM PHẦN TỬ X TRONG DANH SÁCH L (1)



- Tên hàm: locate
- Input: Phần tử X, danh sách L
- Output: Vị trí của X trong ds L
- Thuật toán:
  - Tiến hành tìm từ đầu danh sách cho đến khi tìm thấy hoặc hết danh sách
  - Nếu tìm thấy thì trả về vị trí đầu tiên của X
  - Nếu không tìm thấy thì trả về endList(L)



CANTHO UNIVERSITY

# TÌM PHẦN TỬ X TRONG DANH SÁCH L (2)

```
Position locate(ElementType X, List L){  
    Position P = first(L), E=endList(L);  
    int Found = 0;  
    while ((P != E) && (!Found))  
        if (retrieve(P,L) == X) Found = 1;  
        else P = Next(P, L);  
    return P;  
}
```



```
Position locate(ElementType X, List L){  
    Position P =L ;  
    int Found = 0;  
    while ((P->Next != NULL) && (!Found))  
        if ( P->Next->Element== X) Found = 1;  
        else P = P->Next ;  
    return P;  
}
```



CANTHO UNIVERSITY

# XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (1)

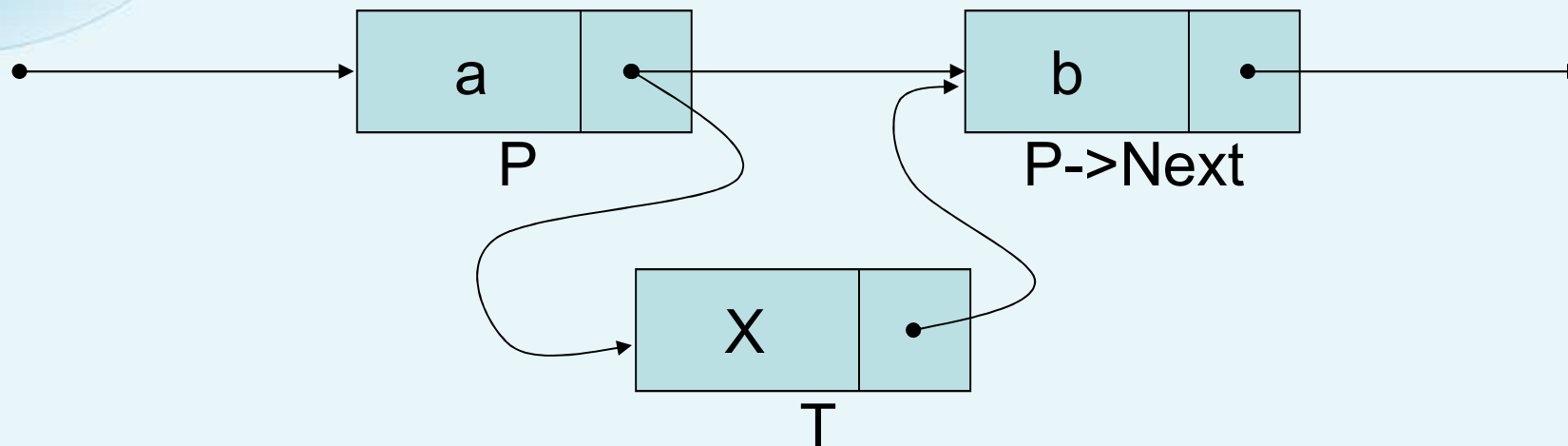
- Tên hàm: insertList
- Input: Phần tử X, vị trí P, danh sách L
- Output: Danh sách L sau khi đã xen X (truyền địa chỉ)





CANTHO UNIVERSITY

## XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (2)



Thuật toán:

- Cấp phát ô nhớ cho vị trí T
- Đặt phần Element của T bằng X.
- Đặt phần Next của T bằng phần Next của P
- Đặt phần Next của P bằng T.



CANTHO UNIVERSITY

## XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (3)

```
void insertList(ElementType X, Position P, List *L)
{
    Position T;
    T=(Node*)malloc(sizeof(Node)); /* 1 */
    T->Element=X;                  /* 2 */
    T->Next=P->Next;                /* 3 */
    P->Next=T;                      /* 4 */
}
```



CANTHO UNIVERSITY

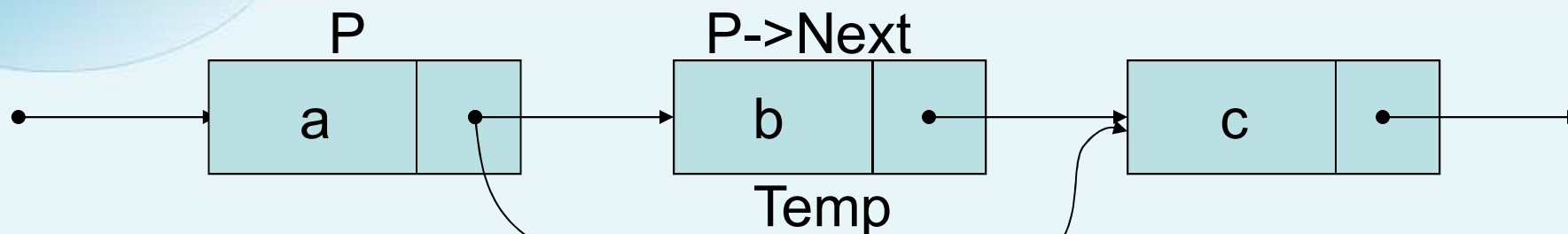
# XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (1)

- Tên hàm: deleteList
- Input: Vị trí P, danh sách L
- Output: Danh sách L sau khi đã xóa (truyền địa chỉ)



CANTHO UNIVERSITY

## XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (2)



Thuật toán:

- Đặt Temp bằng P->Next
- Đặt P->Next bằng Temp->Next
- Giải phóng Temp



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (3)

```
void deleteList(Position P, List *L) {  
    Position Temp;  
    Temp=P->Next;           /* 1 */  
    P->Next=Temp->Next;      /* 2 */  
    free(Temp);              /* 3 */  
}
```



CANTHO UNIVERSITY

## Vận dụng các phép toán trên danh sách để viết các hàm:

- Nhập vào một danh sách các số nguyên - hàm `void readList (List *L)`
- Hiển thị danh sách vừa nhập ra màn hình – Hàm `void printList(List L)`
- Xóa phần tử đầu tiên có nội dung X ra khỏi danh sách – hàm

**`void Delete(ElementType X, List *L)`**

- Xóa tất cả các phần tử có nội dung X ra khỏi danh sách – hàm

**`void deleteAll(ElementType X, List *L)`**

Viết hàm main để kiểm chứng các hàm trên



CANTHO UNIVERSITY

## Bài tập (tiếp)

- Viết hàm

`void insertOrder_List(ElementType X, List *L)`

Giả sử DS L đã có thứ tự (nhỏ đến lớn)

Xen X vào DS L sao cho vẫn có thứ tự

Ví dụ: DS đã có là 3 5 6 8 9

Xen 7 sẽ được 3 5 6 7 8 9

Xen 1 sẽ được 1 3 5 6 8 9

Xen 12 sẽ được 3 5 6 8 9 12



CANTHO UNIVERSITY

# Ham doc du lieu tu ban phim, dua vao danh sach L

```
void readList(List *L){  
    int n, i;  
    ElementType X;  
    printf("Cho biet so phan tu cua danh sach: ");  
    scanf("%d",&n);  
    for (i=1; i<=n; i++){  
        printf("Nhap phan tu thu %d= ",  
            scanf("%d", &X);  
            insertList(X, endList(*L),*L);  
    }  
}
```





CANTHO UNIVERSITY

# In các phần tử trong ds L ra màn hình

```
void printList(List L) {  
    Position P=first(L), E=endList(L);  
    while (P!=E) {  
        printf("%5d", retrieve(P,L));  
        P=Next(P,L);  
    }  
}
```



CANTHO UNIVERSITY

# Tính tổng tất cả các phần tử trong DS

```
int Sum_List(List L){  
    Position P=first(L), E=endList(L);  
    int S=0;  
    while (P!=E) {  
        S = S + retrieve(P,L);  
        P=Next(P,L);  
    }  
    return S;  
}
```



CANTHO UNIVERSITY

## ham main

```
// gia su phan cai dat de trong  
D://CT0302/alistlib.cpp  
#include "D://CT0302/alistlib.cpp"  
int main(){  
    List L;  
    makenullList(&L);  
    readList(&L);  
    printList(L);  
    return 0;  
}
```



CANTHO UNIVERSITY

# SO SÁNH 2 PHƯƠNG PHÁP CÀI ĐẶT DS

- Bạn hãy phân tích ưu và khuyết điểm của
  - Danh sách đặc (pp cài đặt ds bằng mảng)
  - Danh sách liên kết (pp cài đặt ds bằng con trỏ)
- Bạn nên chọn pp cài đặt nào cho ứng dụng của mình?



CANTHO UNIVERSITY

# SO SÁNH 2 PP CÀI ĐẶT

- Cài đặt bằng mảng
  - Ưu điểm: Các hàm first, Next, previous, retrieve và endList thực hiện nhanh.
  - Nhược điểm: Sử dụng bộ nhớ không tối ưu, bị giới hạn số phần tử; Các hàm insertList, deleteList, locate thực hiện chậm.
- Cài đặt bằng con trỏ
  - Ưu điểm: Sử dụng bộ nhớ tối ưu, không bị giới hạn số phần tử; Các hàm first, Next, retrieve, insertList, deleteList thực hiện nhanh
  - Nhược điểm: Các hàm endList, previous, locate thực hiện chậm.



CANTHO UNIVERSITY

# NGĂN XẾP (STACK)

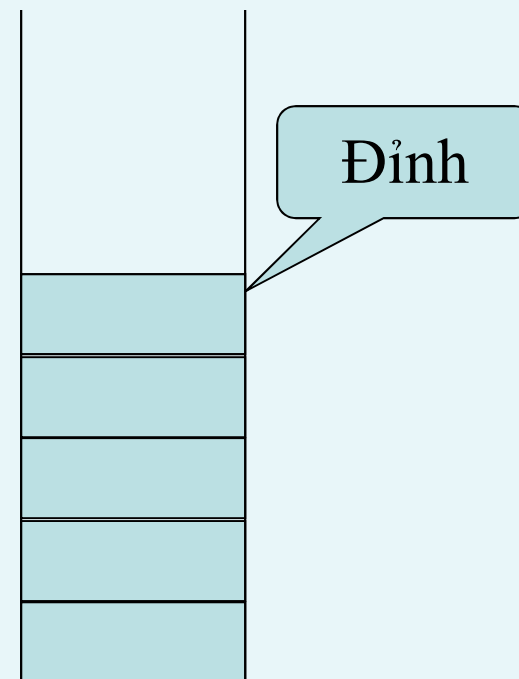
- ĐỊNH NGHĨA
- CÁC PHÉP TOÁN
- CÀI ĐẶT
  - CÀI ĐẶT BẰNG DANH SÁCH
  - CÀI ĐẶT BẰNG MẢNG



CANTHO UNIVERSITY

# ĐỊNH NGHĨA

- Là một danh sách đặc biệt mà việc thêm và xóa phần tử chỉ thực hiện tại một đầu của danh sách. Đầu này được gọi là đỉnh của ngăn xếp
- Cách làm việc theo dạng FILO (first In Last Out) hay LIFO (Last In first Out)





CANTHO UNIVERSITY

# CÁC PHÉP TOÁN

Phép toán	Ý nghĩa của phép toán
<code>makenullStack(S)</code>	Tạo một ngăn xếp S rỗng
<code>emptyStack(S)</code>	Kiểm tra xem ngăn xếp S có rỗng hay không
<code>fullStack(S)</code>	Kiểm tra xem ngăn xếp S có đầy hay không
<code>push(X,S)</code>	Thêm phần tử X vào đỉnh ngăn xếp S
<code>pop(S)</code>	Xóa phần tử tại đỉnh ngăn xếp S
<code>top(S)</code>	Trả về phần tử trên đỉnh ngăn xếp S





CANTHO UNIVERSITY

# CÀI ĐẶT NGĂN XẾP BẰNG DANH SÁCH (1)

- `#include <ALISTLIB.CPP>`
- `//#include <PLISTLIB.CPP>`
- Khai báo  
`typedef List Stack;`
- Tạo ngăn xếp rỗng  
`void makenullStack(Stack *S)  
{ makenullList(S);}`
- Kiểm tra ngăn xếp rỗng  
`int emptyStack(Stack S)  
{ return emptyList(S);}`
- Thêm phần tử vào ngăn xếp  
`void push(ElementType X, Stack *S)  
{ insertList (X, first (*S), S);}`



CANTHO UNIVERSITY

# CÀI ĐẶT NGĂN XẾP BẰNG DANH SÁCH (2)

- Xóa phần tử ra khỏi ngăn xếp  

```
void pop (Stack *S)  
{ deleteList (first(*S), S);}
```
- Xác định giá trị của phần tử tại đỉnh ngăn xếp  

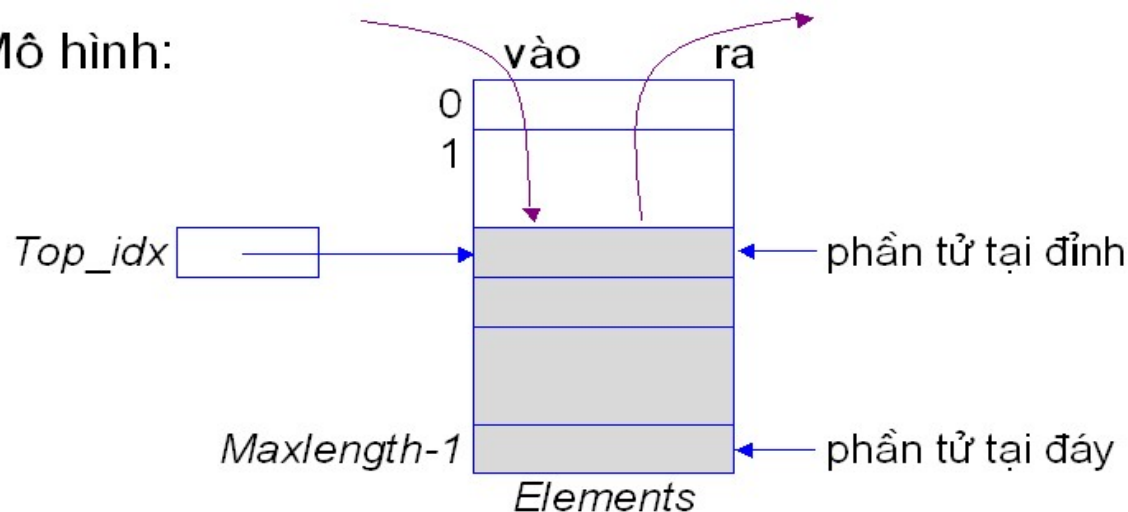
```
ElementType Top(Stack S) {  
    return retrieve(first(S),S);  
}
```



CANTHO UNIVERSITY

# CÀI ĐẶT NGĂN XẾP BẰNG MẢNG (1)

Mô hình:



- Khai báo

```
#define MaxLength ... //độ dài của mảng
```

```
typedef ... ElementType; //kiểu phần tử của ngăn xếp
```

```
typedef struct {
```

```
    ElementType Elements[MaxLength]; //Lưu nội dung của các phần tử
```

```
    int TopIdx; //Chỉ số của đỉnh ngăn xếp
```

```
} Stack;
```



CANTHO UNIVERSITY

# KHỞI TẠO NGĂN XẾP RỖNG

*S rỗng :*

0	
1	
2	
:	
Maxlength-1	
Top_idx→	

- Tạo ngăn xếp S rỗng bằng cách cho chỉ số đỉnh ngăn xếp bằng MaxLength



CANTHO UNIVERSITY

```
void makenullStack(Stack *S)
{
    (*S).TopIdx=MaxLength;
    // S->TopIdx = MaxLength;
}
```



CANTHO UNIVERSITY

# KIỂM TRA NGĂN XẾP RỖNG?

- Ta kiểm tra xem chỉ số của đỉnh ngăn xếp có bằng MaxLength không?



CANTHO UNIVERSITY

```
int emptyStack(Stack S)
{
    return S.TopIdx==MaxLength;
}
```



CANTHO UNIVERSITY

# KIỂM TRA NGĂN XẾP ĐẦY?

*S* đây :

Top\_idx  $\rightarrow$  0

1

2

:

Maxlength-1



- Ta kiểm tra xem TopIdx có bằng 0 hay không?





CANTHO UNIVERSITY

```
int fullStack(Stack S)
{
    return S.TopIdx==0;
}
```



CANTHO UNIVERSITY

# TRẢ VỀ PHẦN TỬ TẠI ĐỈNH NGĂN XẾP

**Ví dụ :**

	0	
	1	
Top_idx →	2	x
	3	
	:	
Maxlength-1		

Kết quả của phép toán trên ngăn xếp là x

- Tên hàm: top
- Input: Ngăn xếp S
- Output: Giá trị tại đỉnh ngăn xếp
- Thuật toán: Trả về giá trị được lưu trữ tại ô có chỉ số là TopIdx



CANTHO UNIVERSITY

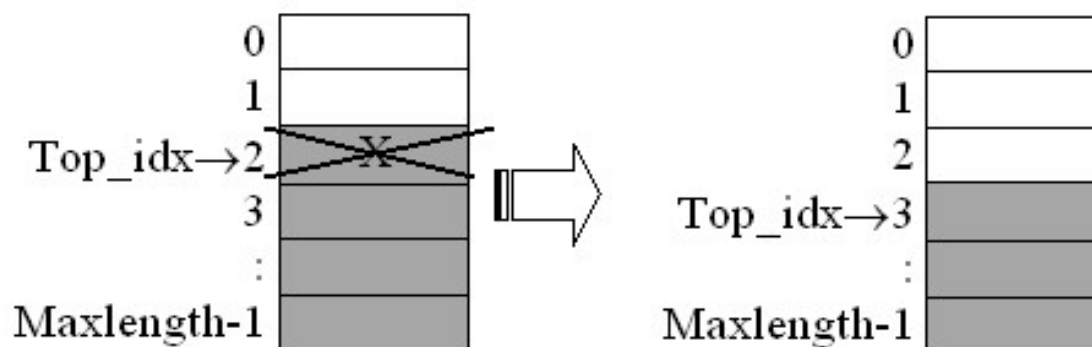
```
ElementType top(Stack S) {  
    return S.Elements[S.TopIdx];  
}
```



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI ĐỈNH NGĂN XẾP

Ví dụ :



- Tên hàm: pop
- Input: Ngăn xếp S
- Output: không có giá trị trả về, nhưng ngăn xếp S đã được xóa phần tử tại đỉnh phải được truyền ra ngoài (truyền địa chỉ)
- Thuật toán: Tăng TopIdx lên 1 đơn vị



CANTHO UNIVERSITY

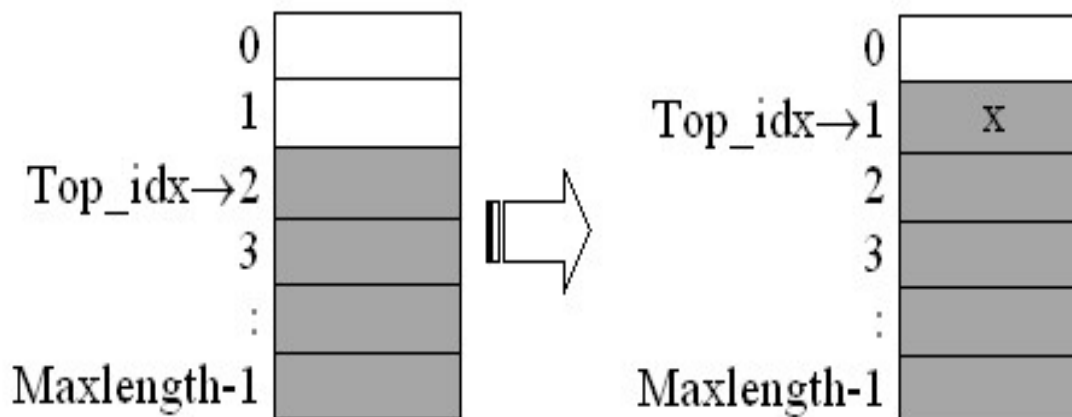
```
void pop(Stack *S) {  
    (*S).TopIdx++;  
}
```



CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO NGĂN XẾP

Ví dụ :



- Tên hàm: push
- Input: Phần tử X, Ngăn xếp S.
- Output: Kh có giá trị trả về nhưng ngăn xếp S sau khi thêm phần tử X phải được truyền ra ngoài
- Thuật toán: Giảm TopIdx xuống 1 đơn vị rồi đưa giá trị x vào ô có chỉ số Topidx



```
void push(ElementType X, Stack *S){  
    if (fullStack(*S)) printf (“Stack day”);  
    else {  
        (*S).TopIdx--;  
        (*S).Elements[(*S).TopIdx]=X;  
    }  
}
```



CANTHO UNIVERSITY

# BÀI TẬP

Viết hàm **void printBinary(int n)**, nhận vào 1 số nguyên không âm n. In ra biểu diễn nhị phân của số n (phải sử dụng các phép toán trên ngăn xếp)





```
void printBinary(int n) {  
    if (n==0) printf("0");  
    else {  
        Stack S;  
        makenullStack(S);  
        while (n!=0) {  
            push(n%2, S);  
            n=n/2;    }  
        while (!emptyStack(S)) {  
            printf("%d", Top(S));  
            pop(S); }  
    }  
}
```



CANTHO UNIVERSITY

# HÀNG ĐỢI (QUEUE)

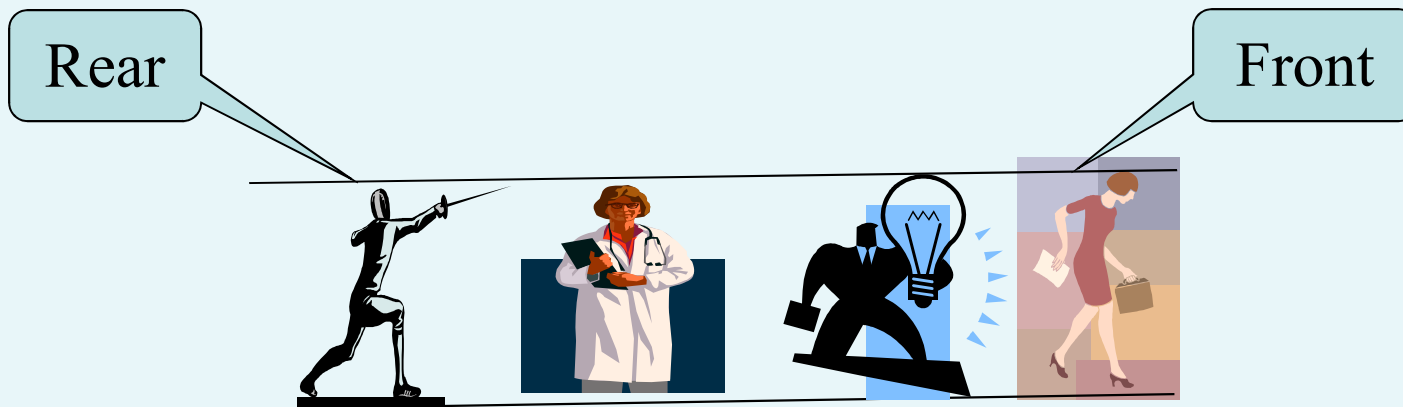
- ĐỊNH NGHĨA
- CÁC PHÉP TOÁN
- CÀI ĐẶT HÀNG ĐỢI
  - DÙNG MẢNG DI CHUYỂN TÍNH TIẾN
  - DÙNG MẢNG VÒNG
  - DÙNG DANH SÁCH LIÊN KẾT



CANTHO UNIVERSITY

# ĐỊNH NGHĨA HÀNG ĐỢI

- Là một danh sách đặc biệt, mà phép thêm vào chỉ thực hiện ở 1 đầu, gọi là cuối hàng (Rear), còn phép loại bỏ thì thực hiện ở đầu kia của danh sách, gọi là đầu hàng (Front)
- Cách làm việc theo dạng FIFO (first In first Out)





CANTHO UNIVERSITY

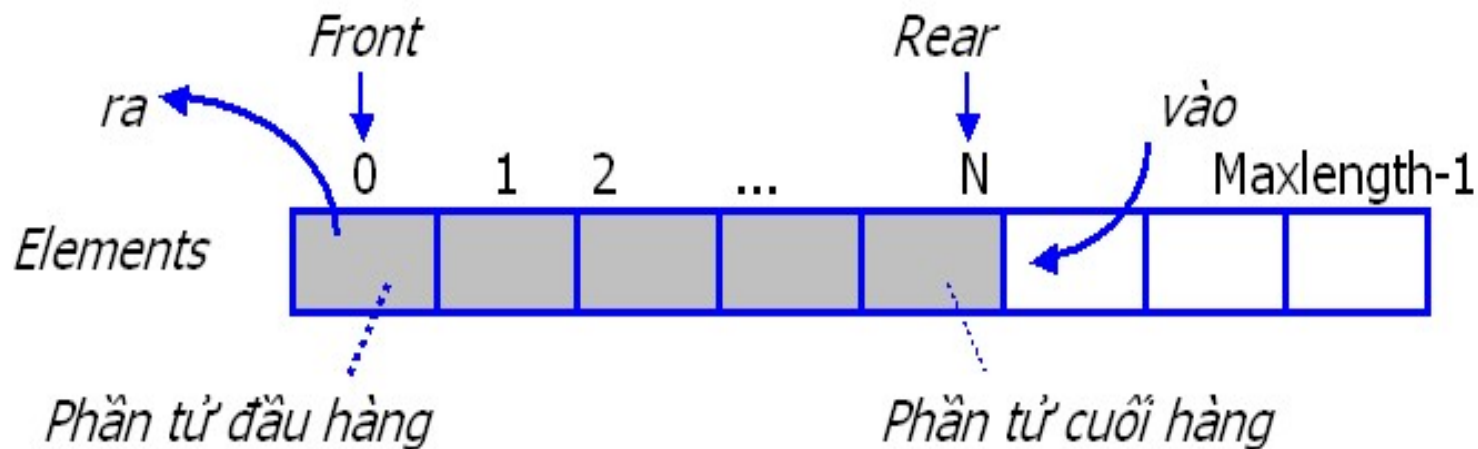
# CÁC PHÉP TOÁN TRÊN HÀNG

Phép toán	Diễn giải
<code>makeNullQueue(Q)</code>	Tạo hàng Q rỗng
<code>emptyQueue(Q)</code>	Kiểm tra xem hàng Q có rỗng?
<code>fullQueue(Q)</code>	Kiểm tra xem hàng Q có đầy?
<code>enqueue(X, Q)</code>	Thêm phần tử X vào cuối hàng Q
<code>dequeue(Q)</code>	Xóa phần tử tại đầu hàng Q
<code>front(Q)</code>	Trả về giá trị của phần tử tại đầu hàng Q



CANTHO UNIVERSITY

# CÀI ĐẶT HÀNG BẰNG MẢNG DI CHUYỂN TÍNH TIẾN



- Số phần tử của hàng =  $Rear - Front + 1$
- Đặc biệt khi hàng chỉ có một phần tử thì  $Rear = Front$
- Xóa phần tử tại đầu hàng thì tăng  $Front$  1 đơn vị
- Thêm phần tử vào cuối hàng thì tăng  $Rear$  lên một đơn vị
- Hàng rỗng khi  $Front = Rear = -1$
- Hàng đầy khi  $Rear - Front + 1 = MaxLength$
- Hàng tràn khi  $Rear = MaxLength - 1$



CANTHO UNIVERSITY

# KHAI BÁO

```
#define MaxLength ....  
    //chiều dài tối đa của mảng  
typedef ... ElementType;  
    //Kiểu dữ liệu của các phần tử trong hàng  
typedef struct  
{  
    ElementType Elements[MaxLength];  
    //Lưu trữ nội dung các phần tử  
    int Front, Rear;  
    //chỉ số đầu và cuối hàng  
} Queue;
```



CANTHO UNIVERSITY

# KHỞI TẠO HÀNG Q RỖNG

- Tên hàm: `makenullQueue`
- Input: Hàng Q
- Output: Không có giá trị trả về, nhưng hàng Q rỗng được tạo ra phải được truyền ra ngoài (dùng truyền địa chỉ)
- Thuật toán: Front và Rear không phải là một chỉ số hợp lệ nào của mảng: ta đặt cả Front và Rear giá trị -1



CANTHO UNIVERSITY

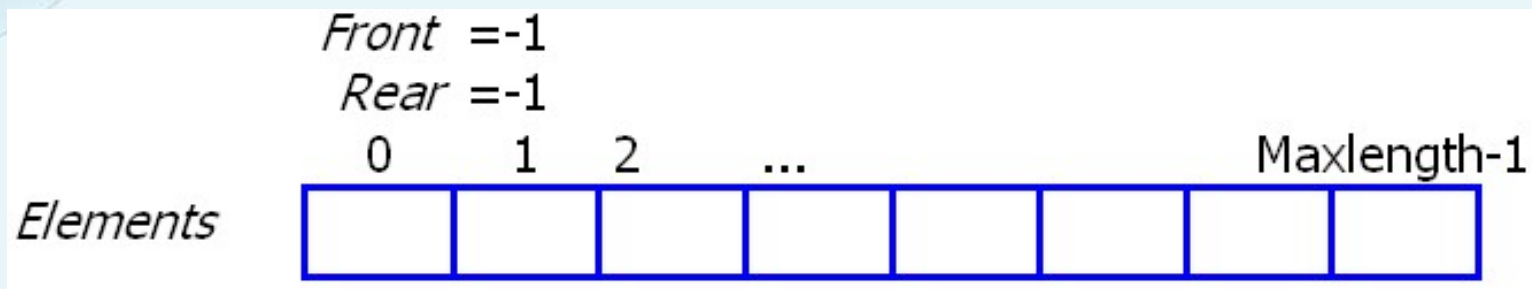
```
void makenullQueue(Queue *Q){  
    (*Q).Front = -1;  
    (*Q).Rear = -1;  
}
```





CANTHO UNIVERSITY

# KIỂM TRA HÀNG RỖNG



- Tên hàm: `emptyQueue`
- Input: Hàng Q;
- Output: Số nguyên (1 hoặc 0)
- Hàng rỗng khi  $Front = -1$



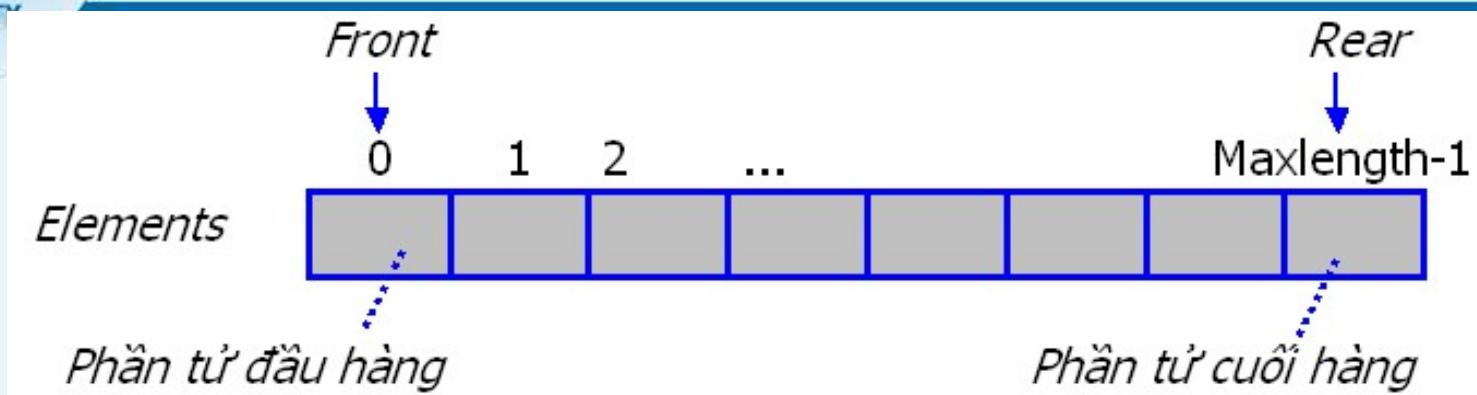
CANTHO UNIVERSITY

```
int emptyQueue(Queue Q)
{
    return (Q.Front == -1);
}
```



CANTHO UNIVERSITY

# KIỂM TRA HÀNG ĐẦY



- Tên hàm: fullQueue;
- Input: Hàng Q
- Output: Số nguyên 1 hoặc 0
- Hàng đầy khi số phần tử hiện có trong hàng = MaxLength



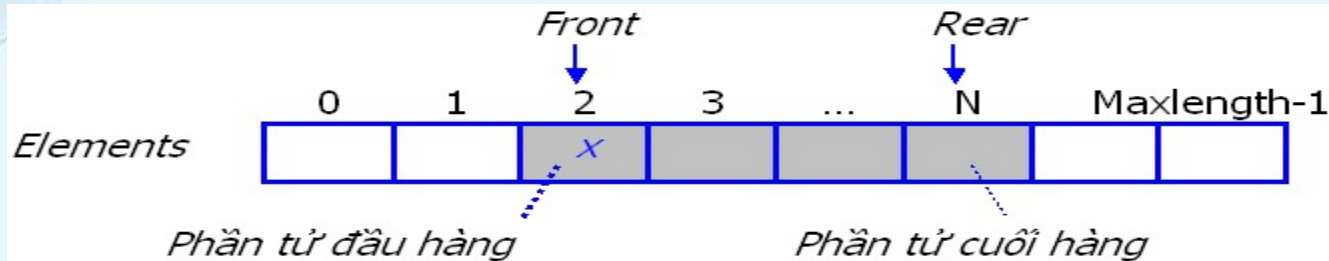
CANTHO UNIVERSITY

```
int fullQueue(Queue Q)
{ return ((Q.Rear - Q.Front + 1) ==
  MaxLength );
}
```



CANTHO UNIVERSITY

# TRẢ VỀ PHẦN TỬ ĐẦU HÀNG



- Tên hàm: front
- Input: Hàng Q
- Output: Giá trị của phần tử tại đầu hàng
- Thuật toán: trả về giá trị được lưu trữ tại ô có chỉ số là Front



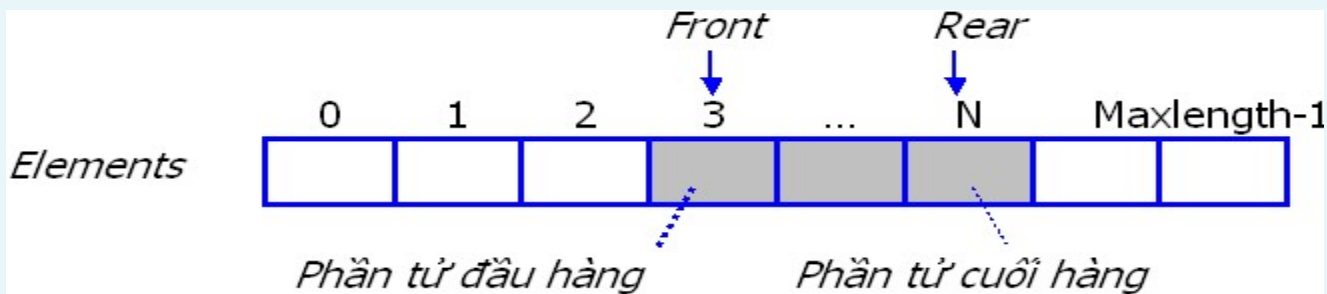
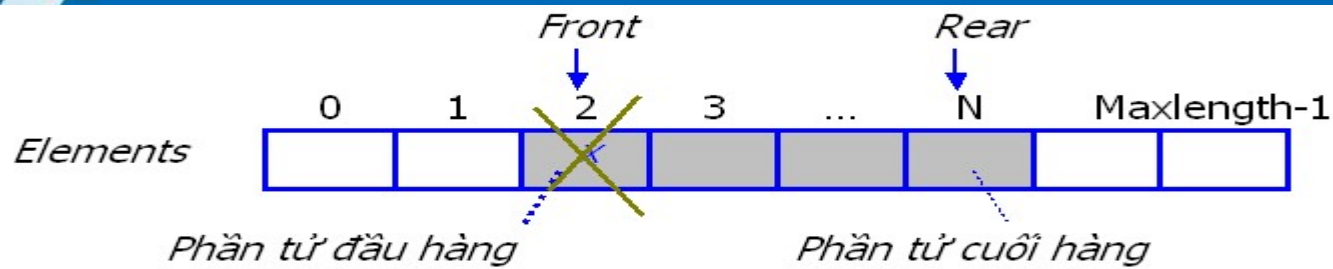
CANTHO UNIVERSITY

```
ElementType front(Queue Q){  
return Q.Elements[Q.Front];  
}
```



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI ĐẦU HÀNG (1)



- Tên hàm: deQueue
- Input: Hàng Q, truyền địa chỉ
- Thuật toán:
  - Nếu hàng chỉ có một phần tử thì khởi tạo lại hàng rỗng
  - Ngược lại, tăng Front lên 1 đơn vị



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI ĐẦU HÀNG (2)

```
void deQueue(Queue *Q) {  
    if ((*Q).Front == (*Q).Rear)  
        makenullQueue(Q);  
    else    (*Q).Front++;  
}
```

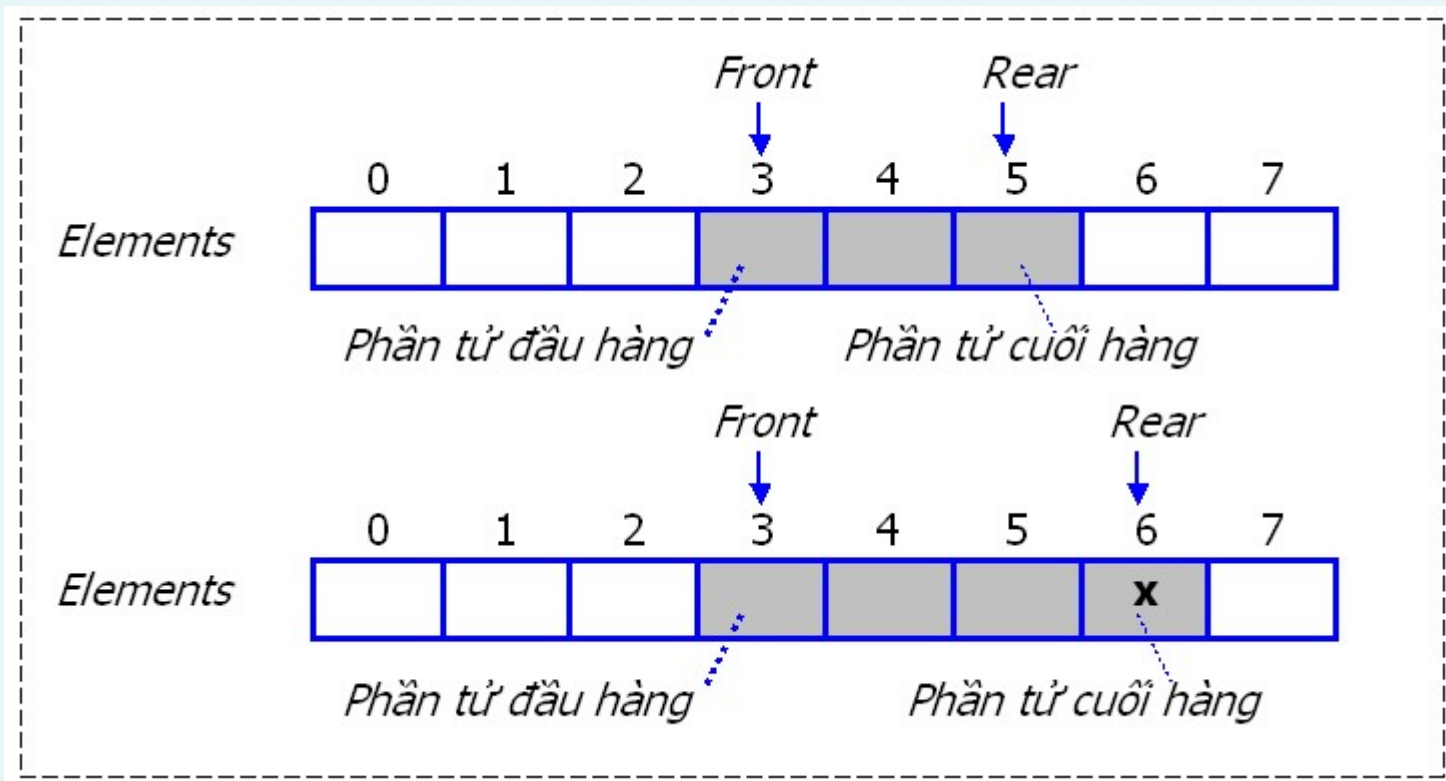




CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO CUỐI HÀNG (1)

- Trường hợp bình thường

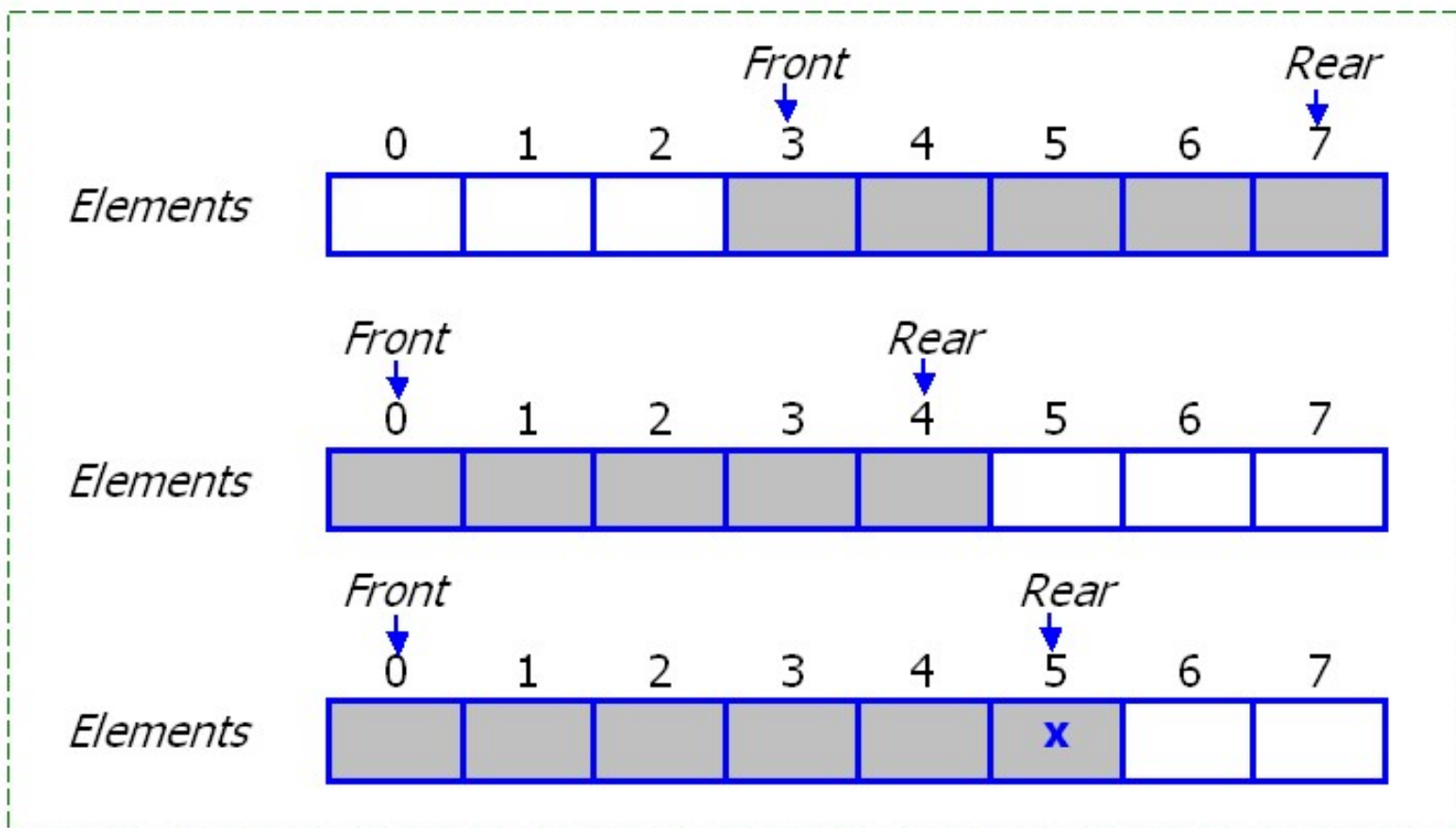




CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO CUỐI HÀNG (2)

– Trường hợp hàng bị tràn





CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO CUỐI HÀNG (3)

## Hàm `enQueue(X,Q)`

- Thuật toán:
  - Nếu hàng rỗng thì đặt  $\text{Front} = 0$
  - Nếu hàng tràn thì phải dời tất cả phần tử lên  $\text{Front}$  vị trí. Xác định lại  $\text{Front}$  và  $\text{Rear}$  mới:
    - $\text{Rear} = \text{Rear} - \text{Front}$
    - $\text{Front} = 0$
  - Tăng  $\text{Rear}$  1 đơn vị và đưa giá trị  $X$  vào ô có chỉ số  $\text{Rear}$  mới này



## THÊM PHẦN TỬ X VÀO CUỐI HÀNG (4)

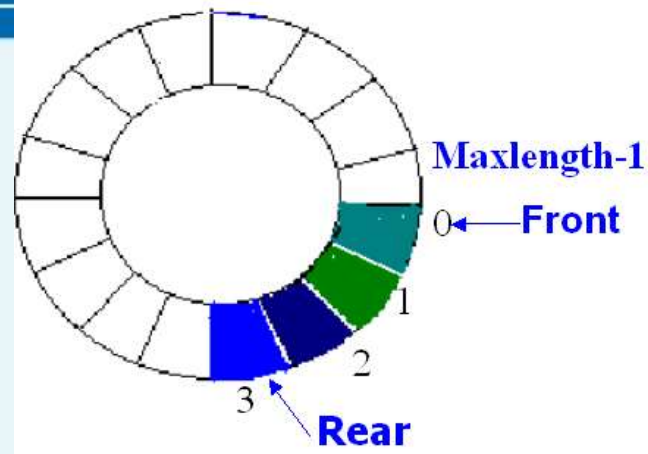
```
void enqueue(ElementType X, Queue *Q){  
    if (fullQueue(*Q)) print("Hang day\n");  
    else { if (emptyQueue(*Q)) (*Q).Front = 0;  
        if ((*Q).Rear == MaxLength-1) {  
            for(int i = (*Q).Front; i <= (*Q).Rear; i++)  
                (*Q).Elements[i-(*Q).Front]= (*Q).Elements[i];  
            (*Q).Rear = (*Q).Rear - (*Q).Front;  
            (*Q).Front = 0; }  
        (*Q).Rear = (*Q).Rear + 1;  
        (*Q).Elements[(*Q).Rear] = X}  
}
```



CANTHO UNIVERSITY

# CÀI ĐẶT HÀNG BẰNG MẢNG VÒNG

- Mô hình



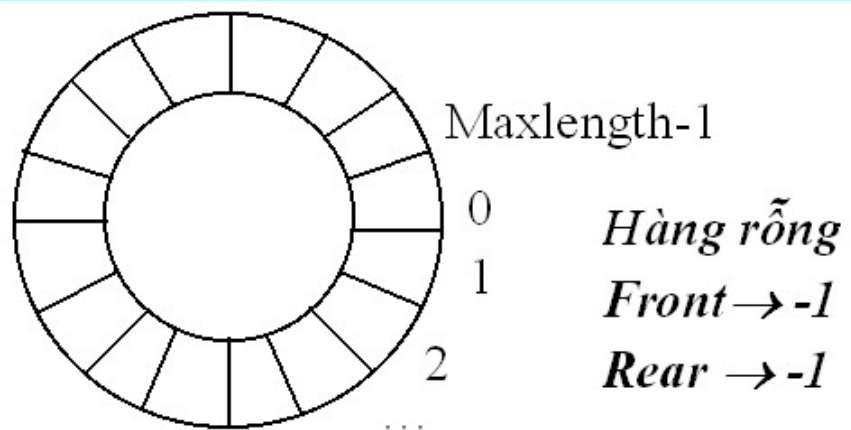
- Khai báo

```
#define MaxLength ...  
//chiều dài tối đa của mảng  
typedef ... ElementType;  
//Kiểu dữ liệu của các phần tử trong hàng  
typedef struct  
{  
    ElementType Elements[MaxLength];  
//Lưu trữ nội dung các phần tử  
    int Front, Rear;  
//chỉ số đầu và đuôi hàng  
} Queue;
```



CANTHO UNIVERSITY

# KHỞI TẠO HÀNG RỖNG

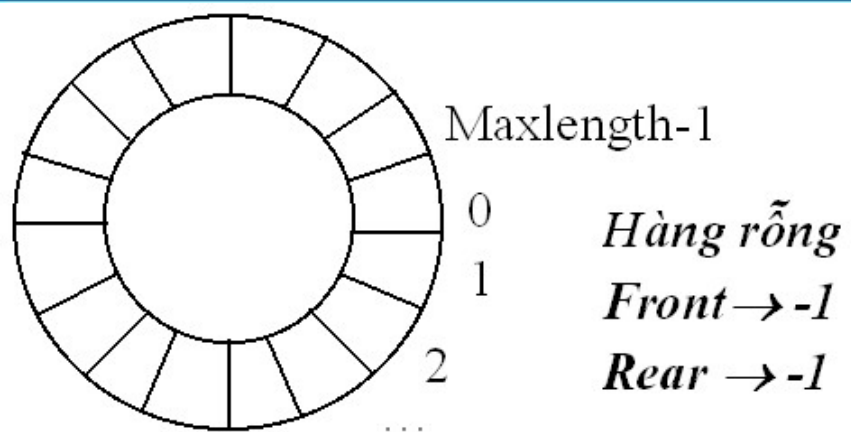


- Front và Rear không trở đến vị trí hợp lệ nào
- Ta cho  $\text{Front} = \text{Rear} = -1$
- **void** makenullQueue(Queue \*Q) {  
    (\*Q).Front = -1;  
    (\*Q).Rear = -1  
}



CANTHO UNIVERSITY

# KIỂM TRA HÀNG RỖNG



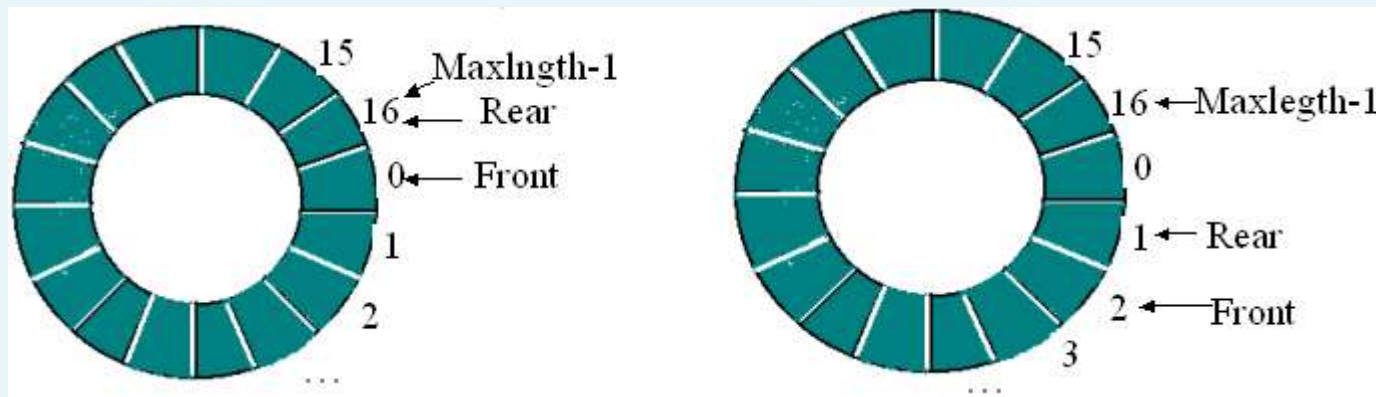
```
int emptyQueue(Queue Q){  
    return Q.Front==-1;  
}
```



CANTHO UNIVERSITY

# KIỂM TRA HÀNG ĐẦY

- Ví dụ



- Hàng đầy khi số phần tử hiện có trong hàng “bằng” MaxLength





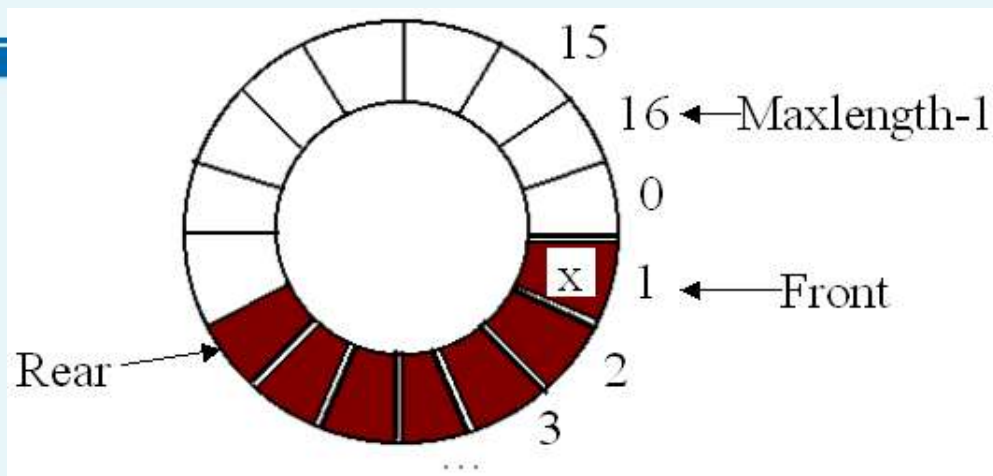
CANTHO UNIVERSITY

```
int fullQueue(Queue Q){  
    return (Q.Rear-Q.Front+1) % MaxLength == 0;  
}
```



CANTHO UNIVERSITY

# LẤY GIÁ TRỊ PHẦN TỬ ĐẦU HÀNG



=>Thuật toán: trả về giá trị được lưu trữ tại ô có chỉ số là Front

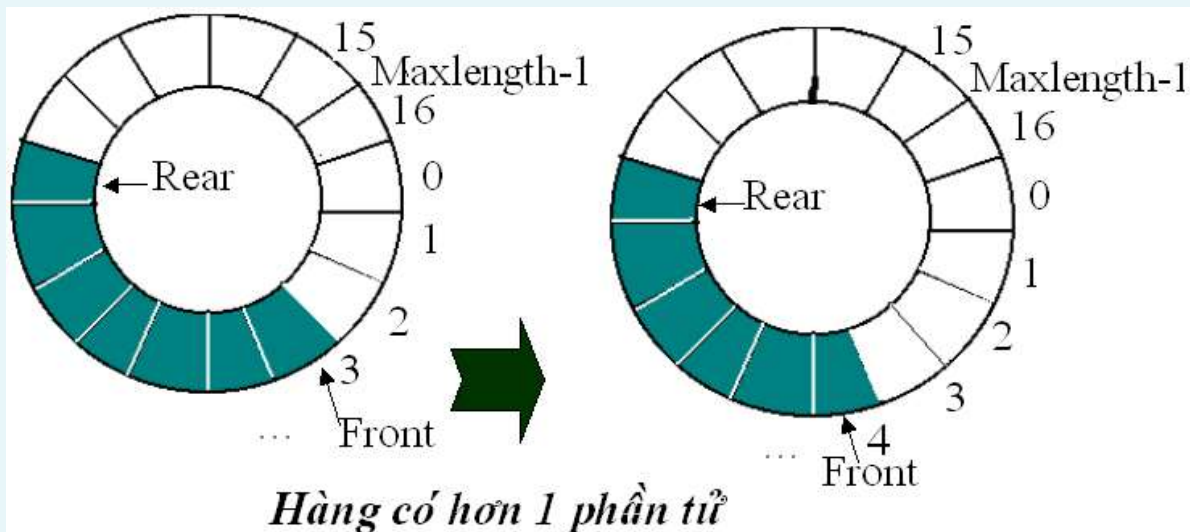
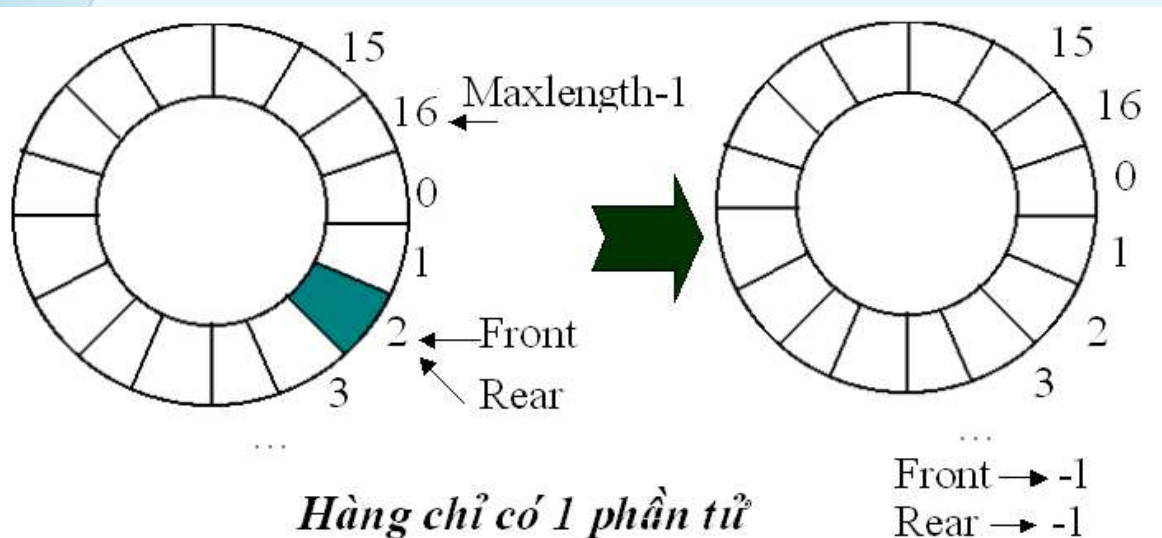
```
ElementType front(Queue Q){  
    return Q.Elements[Q.Front];  
}
```



CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI ĐẦU HÀNG (1)

- Các trường hợp có thể:





CANTHO UNIVERSITY

# XÓA PHẦN TỬ TẠI ĐẦU HÀNG(2)

- Thuật toán :
  - Nếu  $\text{Front} = \text{Rear}$  tức hàng chỉ còn 1 phần tử thì khởi tạo lại hàng rỗng
  - Ngược lại, “tăng” Front lên 1 đơn vị



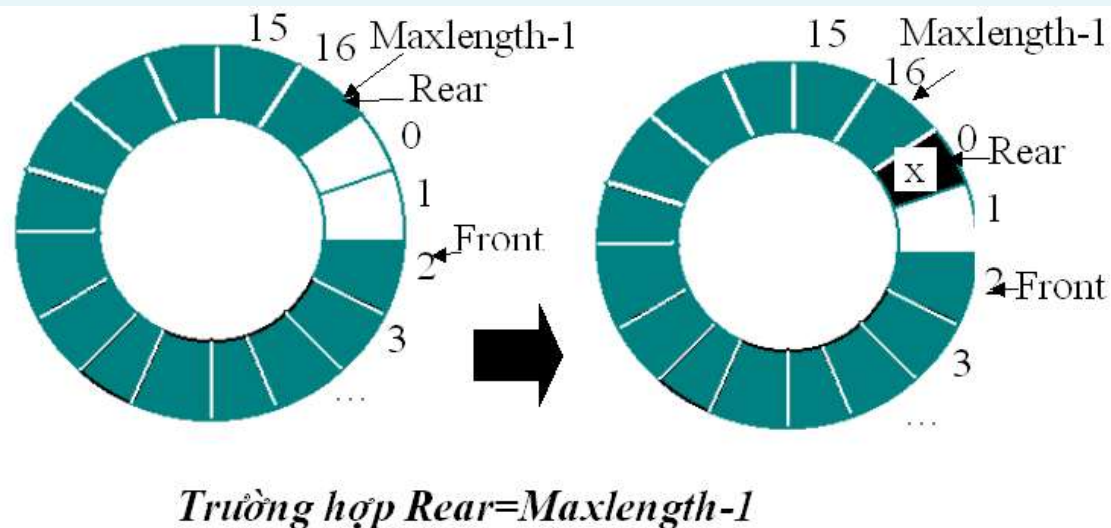
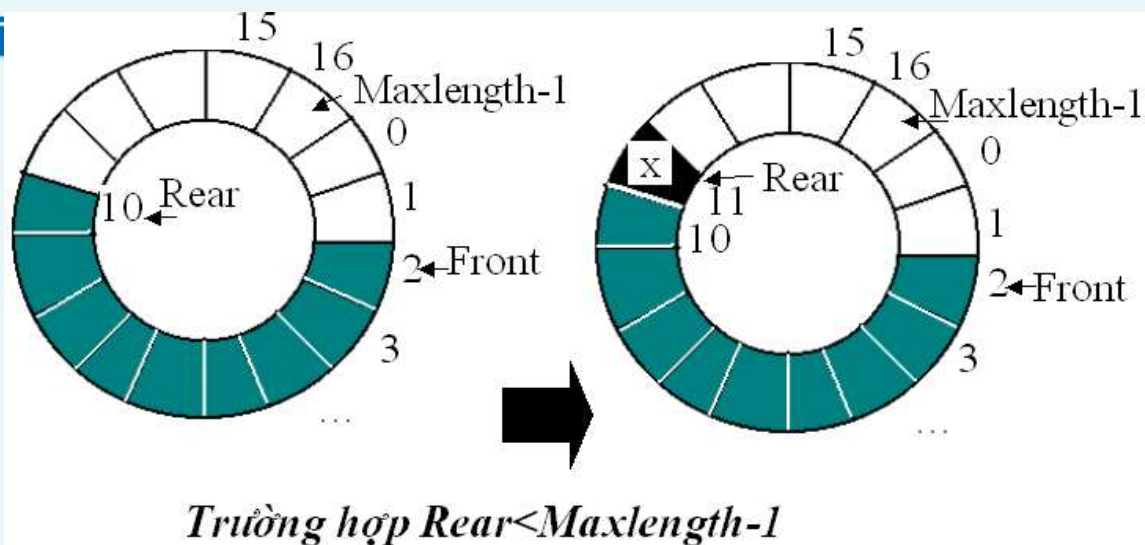
```
void deQueue(Queue *Q){  
    if ((*Q).Front==(*Q).Rear)  
        makenullQueue(Q);  
    else  
        (*Q).Front= ((*Q).Front+1)% MaxLength;  
}
```



CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO CUỐI HÀNG (1)

- Các trường hợp có thể:





CANTHO UNIVERSITY

# THÊM PHẦN TỬ X VÀO CUỐI HÀNG (2)

## Hàm `enQueue(X,Q)`

- Thuật toán :
  - Nếu hàng đầy thì không thêm được
  - Nếu hàng rỗng thì đặt `Front = 0`;
  - “Tăng” `Rear` lên 1 đơn vị và đưa giá trị `X` vào ô có chỉ số `Rear` mới này



```
void enQueue(ElementType X, Queue *Q){  
    if (fullQueue(*Q)) printf("Hang day\n");  
    else {  
        if (emptyQueue(*Q)) (*Q).Front=0;  
        (*Q).Rear=(((*Q).Rear + 1) % MaxLength;  
        (*Q).Elements[(*Q).Rear] = X;  
    }  
}
```

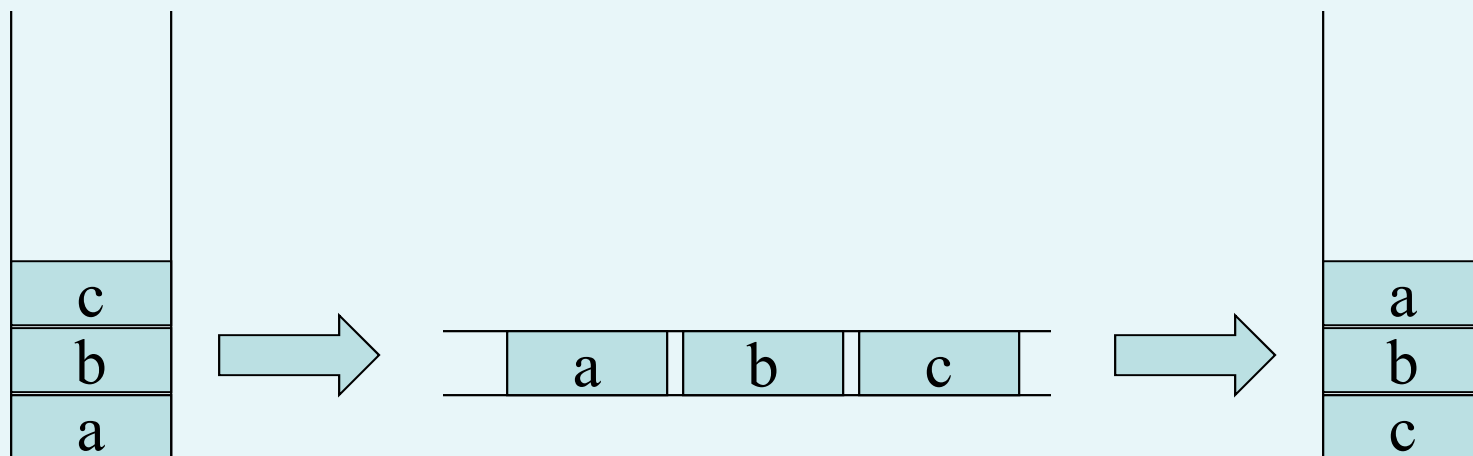




CANTHO UNIVERSITY

# BÀI TẬP

- Viết chương trình nhập vào một ngăn xếp chứa các số nguyên
- Sau đó sử dụng một hàng đợi để đảo ngược thứ tự của các phần tử trong ngăn xếp đó

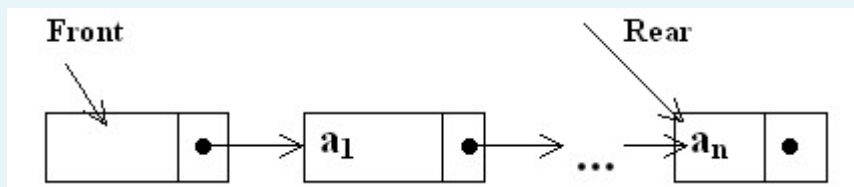




CANTHO UNIVERSITY

# CÀI ĐẶT HÀNG BẰNG DSLK

- Mô hình



- Dùng 2 con trỏ Front và Rear để chỉ tới phần tử đầu hàng và cuối hàng

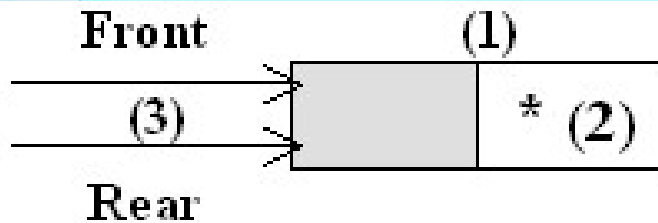
- Khai báo

```
typedef ... ElementType; //kiểu phần tử của hàng  
typedef struct Node{  
    ElementType Element;  
    Node* Next; //Con trỏ chỉ ô kế tiếp  
};  
typedef Node* Position;  
typedef struct{  
    Position Front, Rear; //2 con trỏ  
} Queue;
```



CANTHO UNIVERSITY

# KHỞI TẠO HÀNG Q RỖNG



- Cho Front và Rear cùng trỏ đến HEADER của hàng

```
void makenullQueue(Queue *Q){  
    Position Header;  
    Header = (Node*)malloc(sizeof(Node)); //Cấp phát Header  
    Header->Next = NULL;  
    (*Q).Front = Header;  
    (*Q).Rear = Header;  
}
```



CANTHO UNIVERSITY

# KIỂM TRA HÀNG Q RỖNG

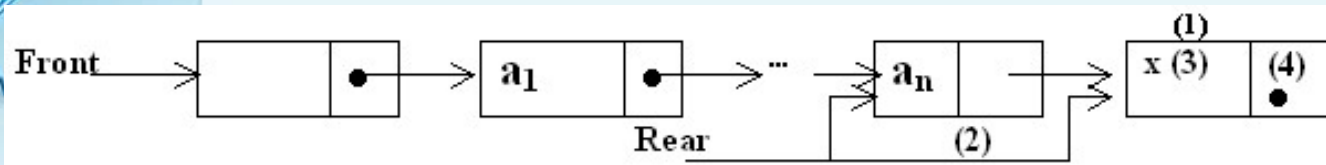
- Kiểm tra xem Front và Rear có cùng chỉ đến 1 ô (HEADER) không?

```
int emptyQueue(Queue Q)
{
    return (Q.Front==Q.Rear);
}
```



CANTHO UNI

# THÊM MỘT PHẦN TỬ X VÀO HÀNG Q



- =>Thuật toán:
  - Thêm 1 phần tử vào hàng ta thêm vào sau Rear 1 ô mới
  - Cho Rear trở đến phần tử mới này
  - Cho trường next của ô mới này trở tới NULL

**void** enqueue(ElementType X, Queue \*Q)

```
{(*Q).Rear->Next=(Node*)malloc(sizeof(Node));
```

```
  (*Q).Rear=(*Q).Rear->Next;
```

```
  //Dat gia tri vao cho Rear
```

```
  (*Q).Rear->Element=X;
```

```
  (*Q).Rear->Next=NULL;
```

```
}
```

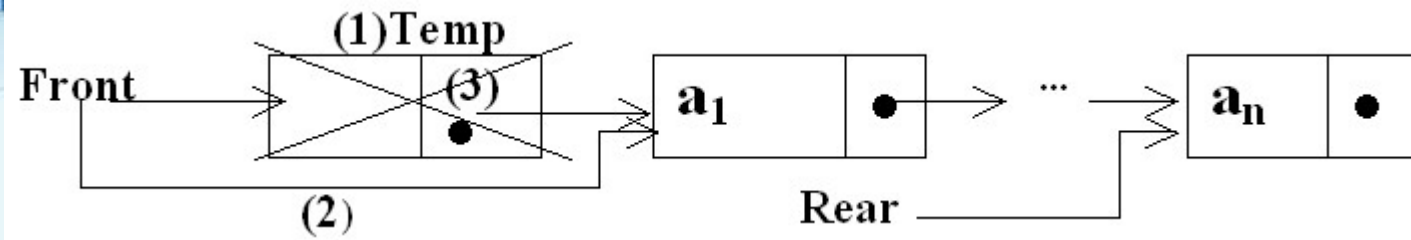


```
void enQueue(ElementType X, Queue *Q){  
    Position H = (Node*)malloc(sizeof(Node));  
    H->Element = X;  
    H->Next = NULL;  
    (*Q).Rear->Next = H;  
    (*Q).Rear = H;  
}
```



CANTHO UNIVERSITY

# XÓA MỘT PHẦN TỬ KHỎI HÀNG Q



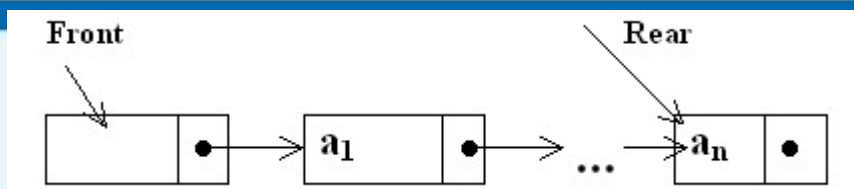
- Để xóa 1 phần tử khỏi hàng ta chỉ cần cho Front trở tới vị trí kế tiếp của nó trong danh sách

```
void deQueue(Queue *Q){  
    Position Tempt;  
    Tempt=(*Q).Front;  
    (*Q).Front=(*Q).Front->Next;  
    free(Tempt);  
}
```



CANTHO UNIVERSITY

## Xác định giá trị của phần tử tại đầu hàng



```
ElementType Front (Queue Q) {  
    if (!emptyQueue(Q))  
        return (Q.Front->Next->Element);  
    else  
        printf("Hang rong");  
}
```





CANTHO UNIVERSITY

# CÁC ỨNG DỤNG CỦA NGĂN XẾP VÀ HÀNG ĐỢI

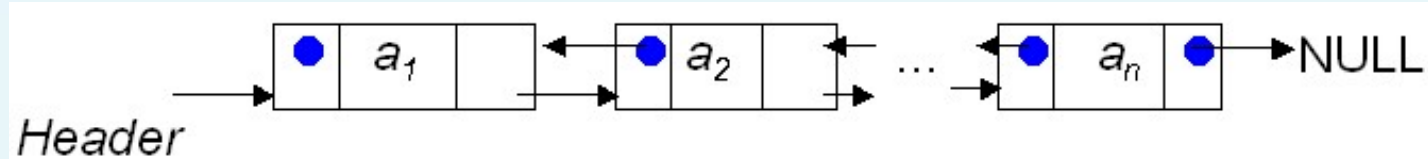
- Bạn hãy liệt kê một số ứng dụng có sử dụng
  - Ngăn xếp
  - Hàng đợi



CANTHO UNIVERSITY

# DANH SÁCH LIÊN KẾT KÉP

- Mô hình



- Trong một phần tử của danh sách, ta dùng hai con trỏ Next và previous để chỉ đến phần tử đứng sau và phần tử đứng trước phần tử đang xét

- Khai báo

```
typedef ... ElementType; //kiểu nội dung của phần tử
typedef struct Node{
    ElementType Element;
    //lưu trữ nội dung phần tử
    Node* Prev;
    Node* Next; //Con trỏ trỏ tới phần tử trước và sau
};
typedef Node* Position;
typedef Position Double_List;
```



CANTHO UNIVERSITY

# DANH SÁCH RỖNG

- Tạo danh sách rỗng

```
void makenullDouble_List (Double_List &DL)
{
    DL= NULL;
}
```

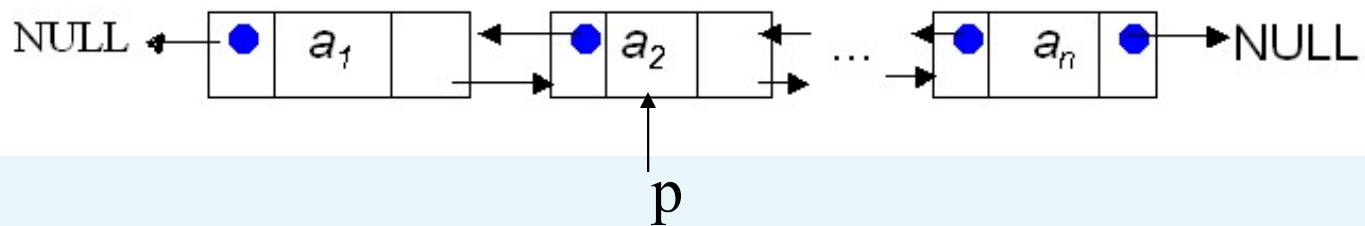
- Kiểm tra danh sách rỗng

```
int emptyDouble_List (Double_List DL)
{
    return (DL==NULL);
}
```



CANTHO UNIVERSITY

## TRẢ VỀ NỘI DUNG PHẦN TỬ VỊ TRÍ P TRONG DANH SÁCH



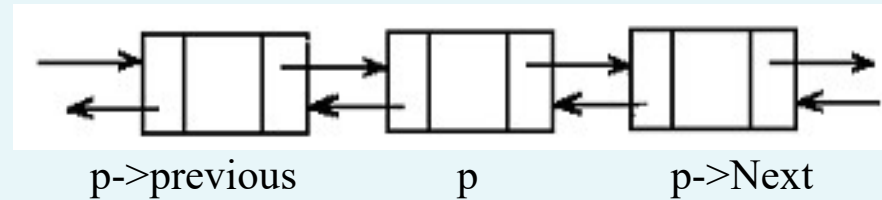
- $\Rightarrow$  Vị trí của một phần tử là con trỏ trỏ vào ngay chính phần tử đó
- **ElementType** retrieve\_Double\_List (Position P, Double\_List DL)  
{  
    **return** P->Element;  
}



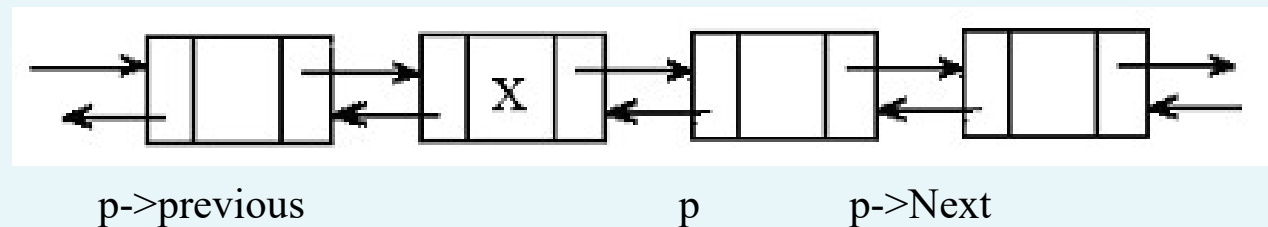
CANTHO UNIVERSITY

## THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (1)

- Trước khi thêm



- Sau khi thêm



=>Cấp phát một ô nhớ mới chứa phần tử cần thêm

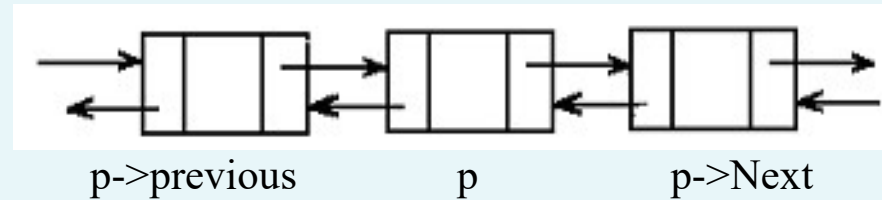
=>Đặt lại các liên kết



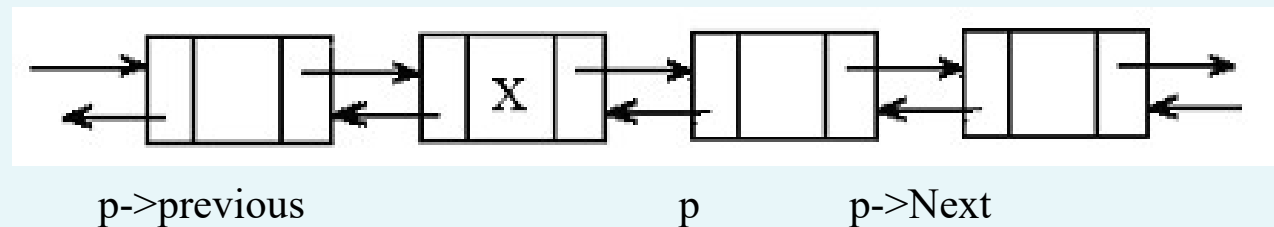
CANTHO UNIVERSITY

## THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (2)

- Trước khi thêm



- Sau khi thêm



=>Cấp phát một ô nhớ mới chứa phần tử cần thêm

=>Đặt lại các liên kết



CANTHO UNIVERSITY

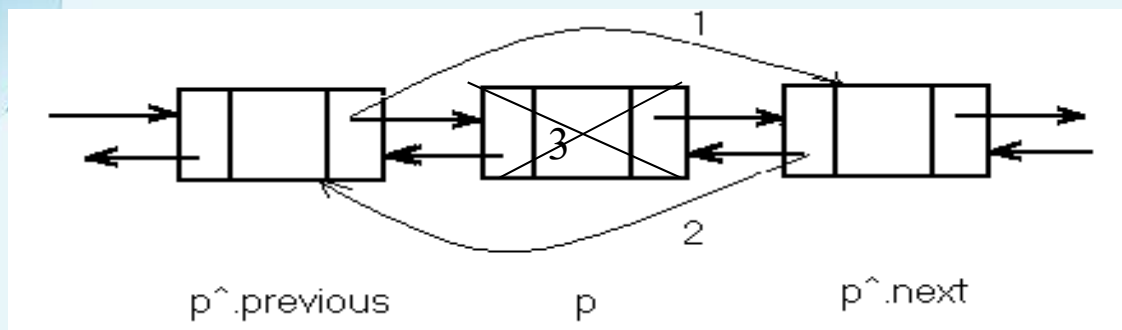
## THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (2)

```
• void insertDouble_List (ElementType X, Position p, Double_List &DL)
• {   if (DL == NULL)
•     {       DL=(Node*)malloc(sizeof(Node));
              DL->Element = X;
              DL->previous =NULL;
              DL->Next =NULL;
        } else {Position temp;
                temp=(Node*)malloc(sizeof(Node));
                temp->Element=X;
                temp->Next=p;
                temp->previous=p->previous;
                if (p->previous!=NULL)
                    p->previous->Next=temp;
                p->previous=temp;
            }
        }
```



CANTHO UNIVERSITY

## XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH



```
void deleteDouble_List (Position p, Double_List &DL)
{ if (DL == NULL)
    printf("Danh sach rong");
else
    {if (p==DL)
        DL= DL->Next;
        //Xóa phần tử đầu tiên của danh sách nên phải thay đổi DL
    else p->previous->Next = p->Next;
    if (p->Next!=NULL)
        p->Next->previous=p->previous;
    free(p);
    }
}
```





CANTHO UNIVERSITY

# ƯU ĐIỂM CỦA DSLK KÉP

- Theo bạn, ưu điểm của việc sử dụng dslk kép là gì?



Thank you!