CANTHO UNIVERSITY

CHƯƠNG 2:
CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN

Nguyễn Văn Linh

Khoa Công nghệ Thông tin & Truyền thông

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
  - Bằng mảng
  - Bằng con trỏ

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: a1, a2, ..., an
- Nếu
  - n=0: danh sách rỗng
  - n>0: phần tử đầu tiên là a1, 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 dánh 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.

CÁC PHÉP TOÁN TRÊN DANH SÁCH

Tên phép toán	Ý nghĩa của phép toán
Make_Null_List(L)	Khởi tạo danh sách L rỗng
Empty_List(L)	Kiểm tra xem danh sách L có rỗng hay không
First(L)	Trả về vị trí của phần tử đầu tiên trong danh sách L
End(L)	Trả về vị trí sau vị trí cuối cùng trong danh sách L
Next(P,L)	Trả về vị trí sau vị trí P trong danh sách L
Previous(P,L)	Trả về vị trí trước vị trí P trong danh sách L

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 End(L)
Insert_List(X,P,L)	Xen phần tử có giá trị X vào danh sách L tại vị trí P
Delete_List(P,L)	Xóa phần tử tại vị trí P trong danh sách L

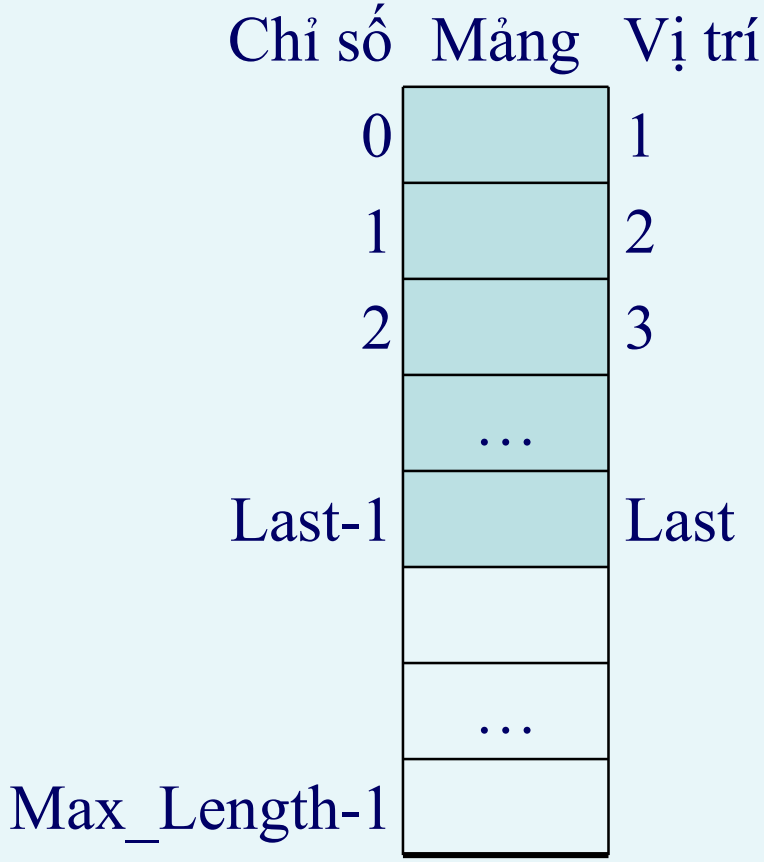
VÍ DỤ

```
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!= End(L))
    {       q=Next(p,L);       //vị trí phần tử đứng ngay sau phần tử p
            while (q!=End(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);
    }
}
```

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)

MÔ HÌNH



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

KHAI BÁO

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

KHỞI TẠO DANH SÁCH RỖNG

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

```
void Make_Null_List(List &L) {
    L.Last=0;
}
```

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

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 End(List L)
{
  return L.Last+1;
}
```

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;
}
```

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;
}
```

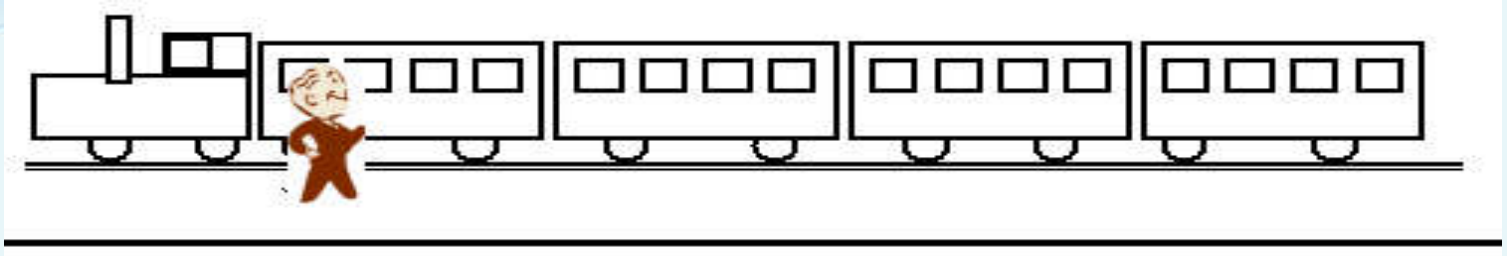
XÁC ĐỊNH GIÁ TRỊ TẠI VỊ TRÍ P

- 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

```
Element_Type 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 (End(L))
  - Gợi ý: Sử dụng các hàm First, End, Retrive, Next, biến Found

TÌM PHẦN TỬ X TRONG DANH SÁCH L (2)

```
Position Locate(Element_Type X, List L){
   Position P = First(L), E=End(L);
   int Found = 0;
   while ((P != E) && (!Found))
        if (Retrieve(P,L) == X) Found = 1;
        else P = Next(P, L);
   return P;
}
```

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 tham chiếu)
- Thuật toán: Có 2 trường hợp xẩy ra:
  - Nếu danh sách đầy (Last = Max_Length) thì không thể xen thêm

CANTHO UNIVERSITY

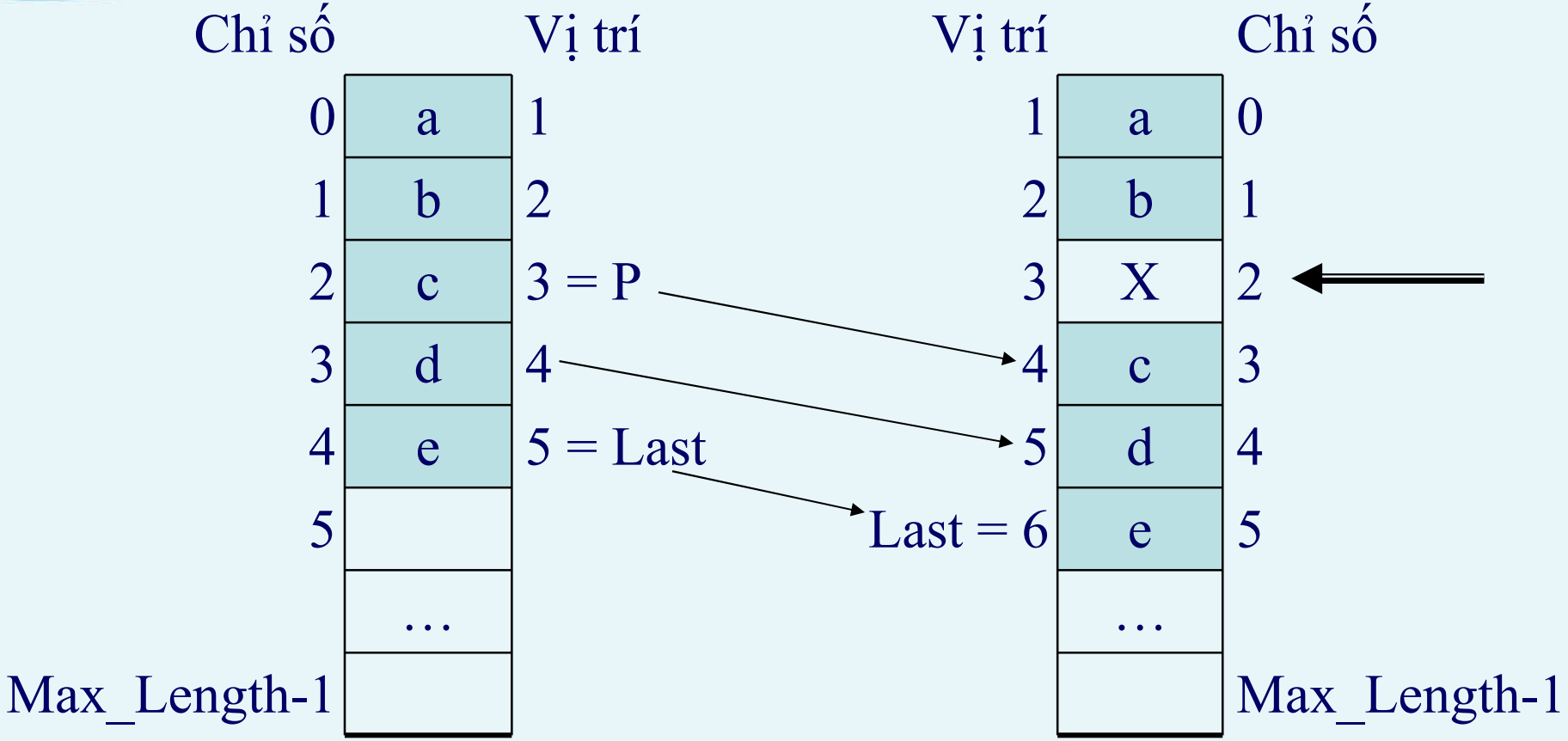
XEN PHẦN TỬ X VÀO DANH SÁCH L TẠI VỊ TRÍ P (2)

– 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

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

Ví dụ xen X vào L tại vị trí P=3



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

```
void Insert_List(Element_Type X, Position P, List &L){
   if(Full_List(L))
      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;
           }
  }
```

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 tham chiếu)
- Thuật toán: Có 2 trường hợp xẩy ra:
  - Danh sách L rỗng: Không thể xóa

CANTHO UNIVERSITY

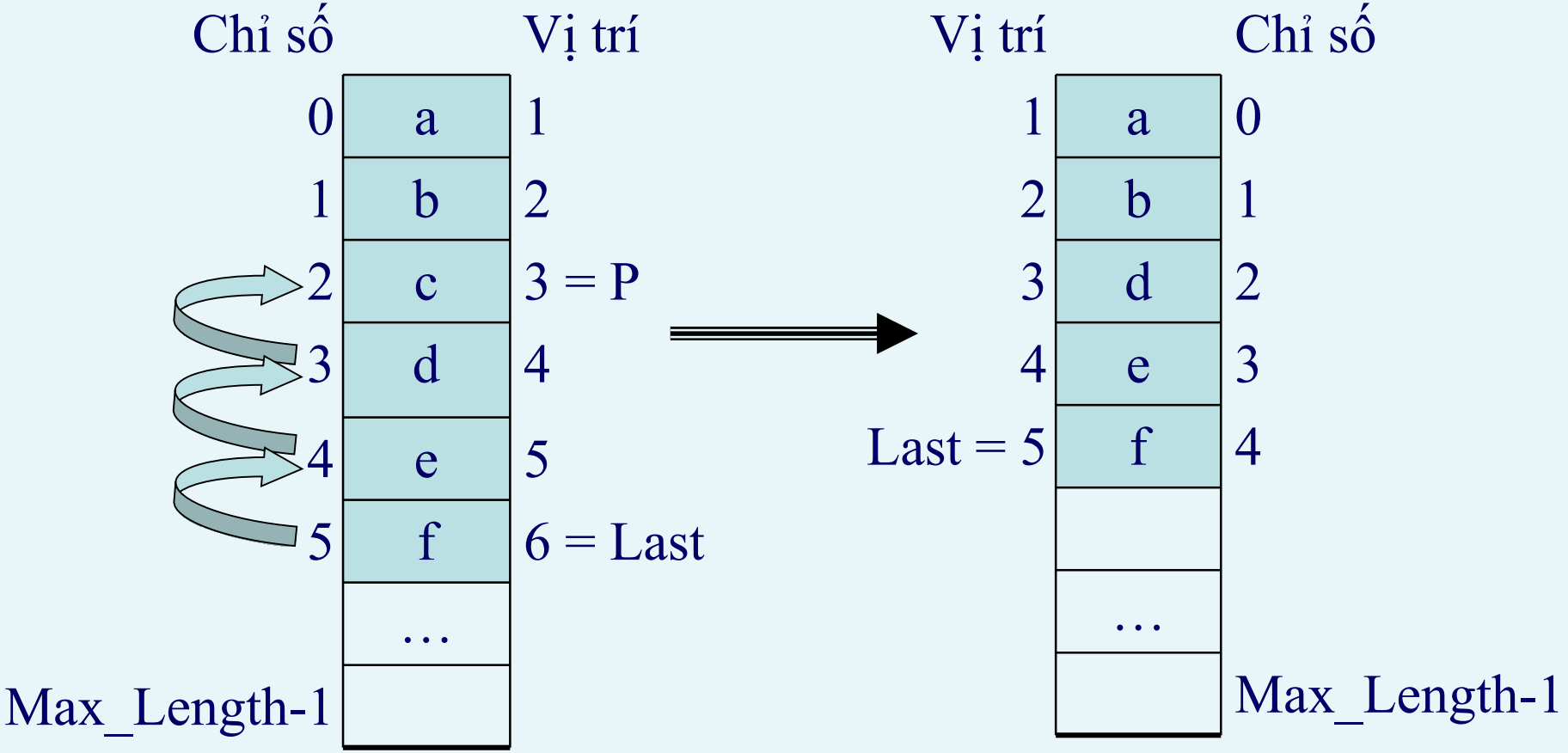
XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (2)

– Dánh sách L không rỗng:

- 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)



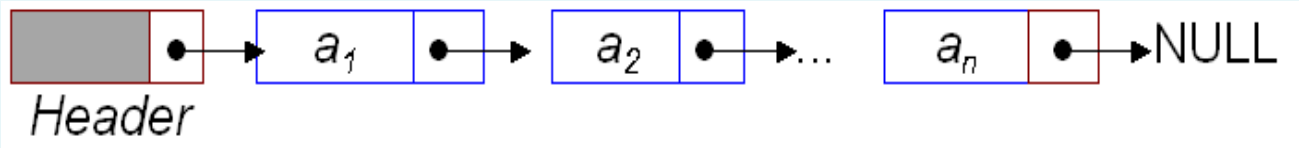
XÓA PHẦN TỬ TẠI VỊ TRÍ P TRONG DANH SÁCH L (4)

```
void Delete_List(Position P, List &L){
        if(Empty_List(L))
            printf("\nDanh sach rong !");
        else{
                for(int i=P; i<=L.Last-1; i++)
                    L.Elements[i-1] = L.Elements[i];
                L.Last--;
            }
}
```

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.

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,…, a_{n-1} là vị trí của a_n và a_n là vị trí End.

KHAI BÁO

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

KHỞI TẠO DANH SÁCH RỖNG

- Input: Danh sách L
- Output: Danh sách L rỗng (truyền tham chiếu)
- Thuật toán:
  - Cấp phát vùng nhớ cho L
  - Đặt trường Next của L bằng NULL

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

KIỂM TRA DANH SÁCH RỖNG

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

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

XÁC ĐỊNH VỊ TRÍ ĐẦU TIÊN

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

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

CANTHO UNIVERSITY

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

- 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->Next == NULL
  - Trả về P

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

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

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->Next

```
Position Next(Position P, List L){
    return P->Next;
}
```

- Chú ý phân biệt 2 chữ Next

VỊ TRÍ TRƯỚC VỊ TRÍ P (1)

- 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->Next == P
  - Trả về Q

VỊ TRÍ TRƯỚC VỊ TRÍ P (2)

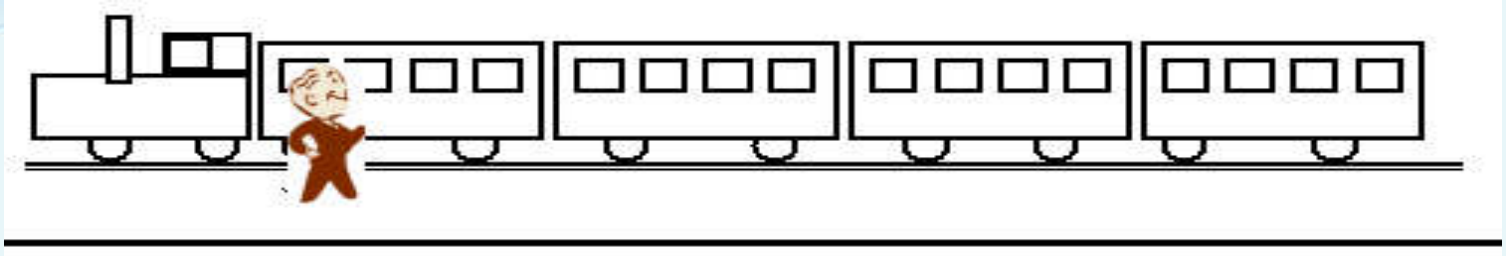
```
Position Previous(Position P, List L){
    Position Q = First(L);
   while (Q->Next !=P) Q = Q->Next;
   return Q;
}
```

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 phần tử của P-> Next

```
Element_Type Retrieve(Position P, List L)
{
    return P->Next->Element;
}
```

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 End(L)

TÌM PHẦN TỬ X TRONG DANH SÁCH L (2)

```
Position Locate(Element_Type X, List L){
   Position P = First(L), E=End(L);
   int Found = 0;
   while ((P != E) && (!Found))
        if (Retrieve(P,L) == X) Found = 1;
        else P = Next(P, L);
   return P;
}
```

```
Position Locate(Element_Type 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;
}
```

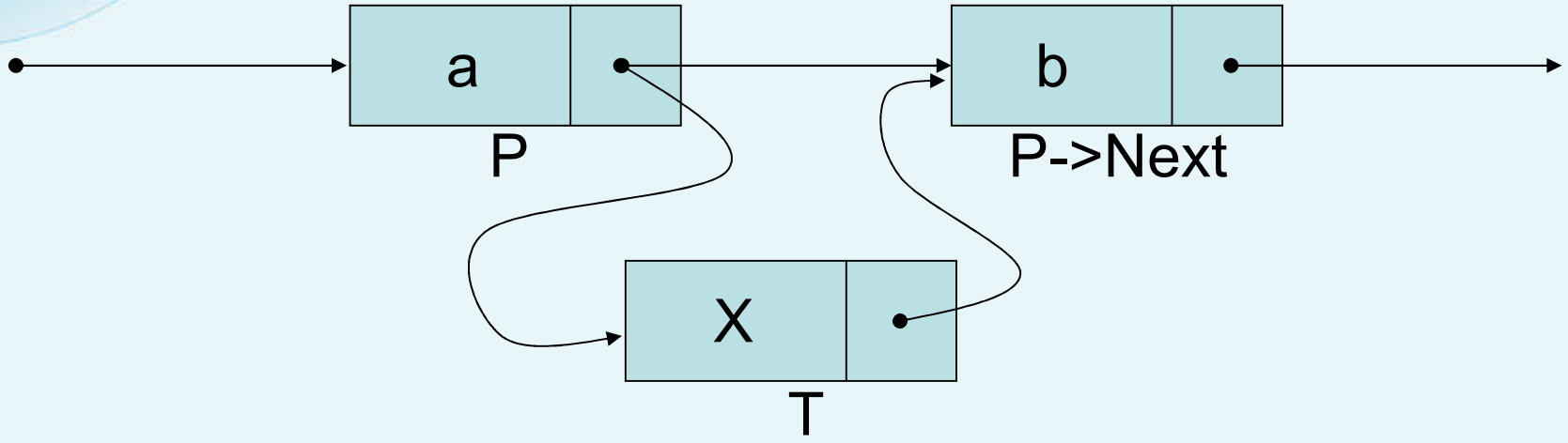
CANTHO UNIVERSITY

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

- Input: Phần tử X, vị trí P, danh sách L
- Output: Danh sách L sau khi đã xen X (truyền tham chiếu)

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.

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

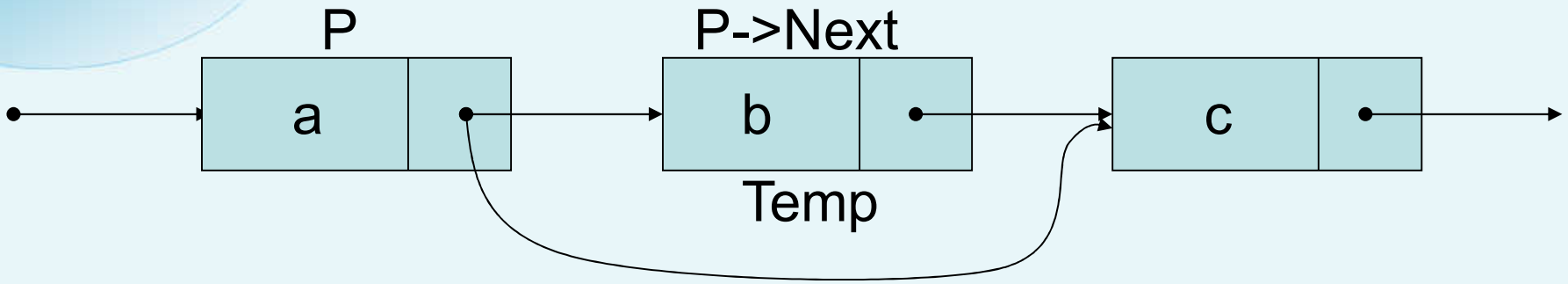
```
void Insert_List(Element_Type X, Position P, List &L)
    {    Position T;
         T=(Node*)malloc(sizeof(Node)); /* 1 */
         T->Element=X;                  /* 2 */
         T->Next=P->Next;               /* 3 */
         P->Next=T;                     /* 4 */
    }
```

CANTHO UNIVERSITY

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

- Input: Vị trí P, danh sách L
- Output: Danh sách L sau khi đã xóa (truyền tham chiếu)

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

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

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

}
```

BÀI TẬP

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

- Nhập vào một danh sách các số nguyên - hàm void Read_List (List &L)
- Hiển thị danh sách vừa nhập ra màn hình – Hàm void Print_List(List L)
- Xóa phần tử đầu tiên có nội dung X ra khỏi danh sách – hàm void Delete(Element_Type X, List &L)
- Viết hàm main để kiểm chứng các hàm trên

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 End thực hiện nhanh.
  - Nhược điểm: Sử dụng bộ nhớ không tối ưu, bị giới hạn số phần tử; Các hàm Insert_List, Delete_List, 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, Insert_List, Delete_List thực hiện nhanh
  - Nhược điểm: Các hàm End, Previous, Locate thực hiện chậm.

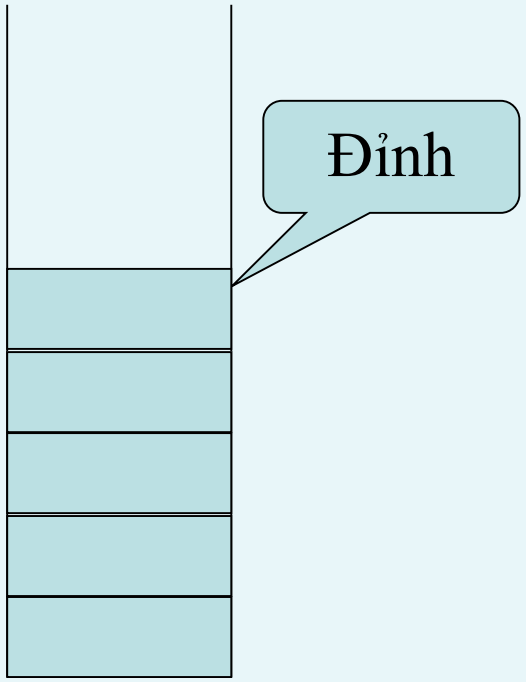
CANTHO UNIVERSITY

NGĂN XẾP (STACK)

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

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)



CÁC PHÉP TOÁN

Phép toán	Ý nghĩa của phép toán
Make_Null_Stack(S)	Tạo một ngăn xếp S rỗng
Empty_Stack(S)	Kiểm tra xem ngăn xếp S có rỗng hay không
Full_Stack(S)	Kiểm tra xem ngăn xếp S có đầy hay không
Push(X,S)	Thêm phần tử X vào đỉnh ngăn xếp S
Pop(S)	Xóa phần tử tại đỉnh ngăn xếp S
Top(S)	Trả về phần tử trên đỉnh ngăn xếp S

CÀI ĐẶT NGĂN XẾP BẰNG DANH SÁCH (1)

- #include <ALISTLIB.CPP>
- Khai báo

  typedef List Stack;
- Tạo ngăn xếp rỗng

  void Make_Null_Stack(Stack &S)

  { Make_Null_List(S);}
- Kiểm tra ngăn xếp rỗng

  int Empty_Stack(Stack S)

  { return Empty_List(S);}
- Thêm phần tử vào ngăn xếp

  void Push(Element_Type X, Stack &S)

  { Insert_List (X, First (S), S);}

CÀI ĐẶT NGĂN XẾP BẰNG DANH SÁCH (2)

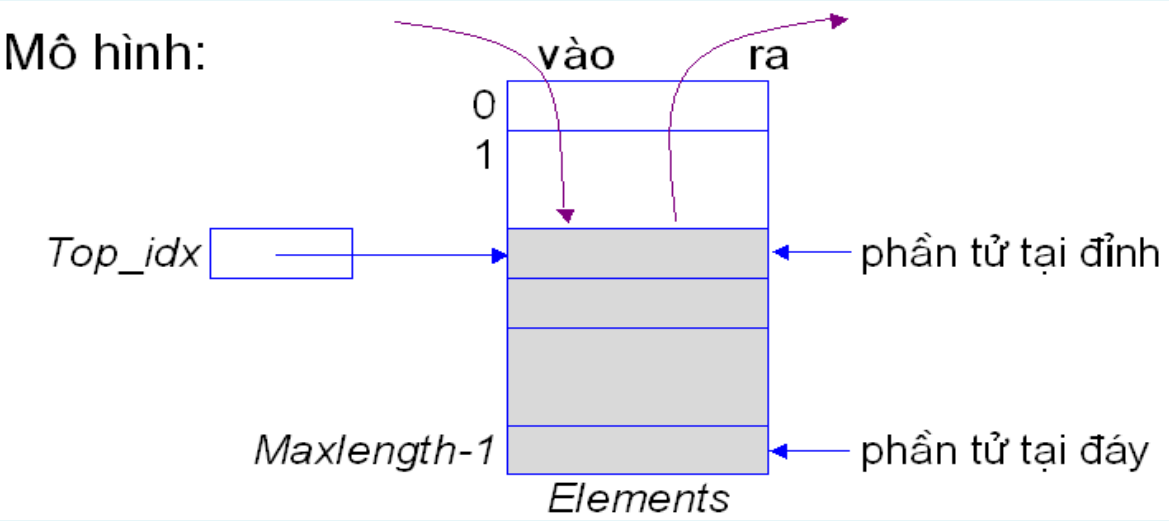
- Xóa phần tử ra khỏi ngăn xếp

```
void Pop (Stack &S)
{  Delete_List (First(S), S);}
```

- Xác định giá trị của phần tử tại đỉnh ngăn xếp

```
Element_Type Top(Stack S) {
    return Retrieve(First(S),S);
}
```

CÀI ĐẶT NGĂN XẾP BẰNG MẢNG (1)

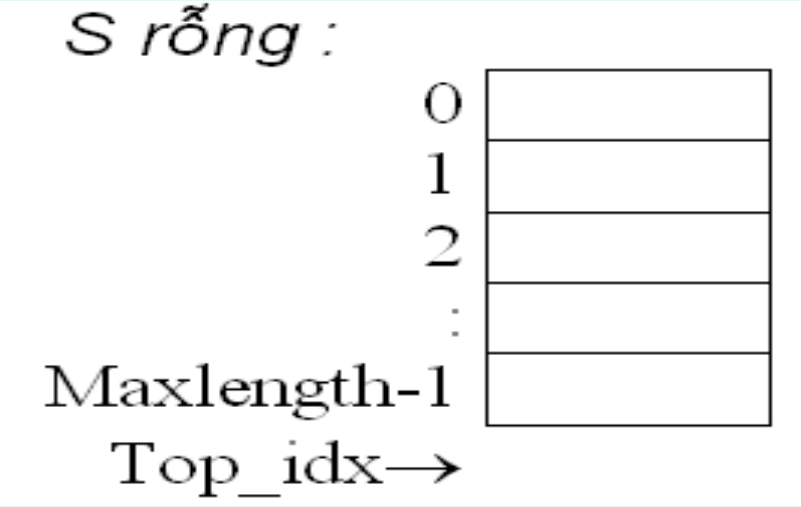


- Khai báo

```
#define Max_Length … //độ dài của mảng
typedef … Element_Type;//kiểu phần tử của ngăn xếp
typedef struct {
    Element_Type Elements[Max_Length]; //Lưu nội dung của các phần tử
    int Top_Idx; //Chỉ số của đỉnh ngăn xếp
}Stack;
```

CANTHO UNIVERSITY

KHỞI TẠO NGĂN XẾP RỖNG



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

```
void Make_Null_Stack(Stack &S)
        {
                S.Top_Idx=Max_Length;
        }
```

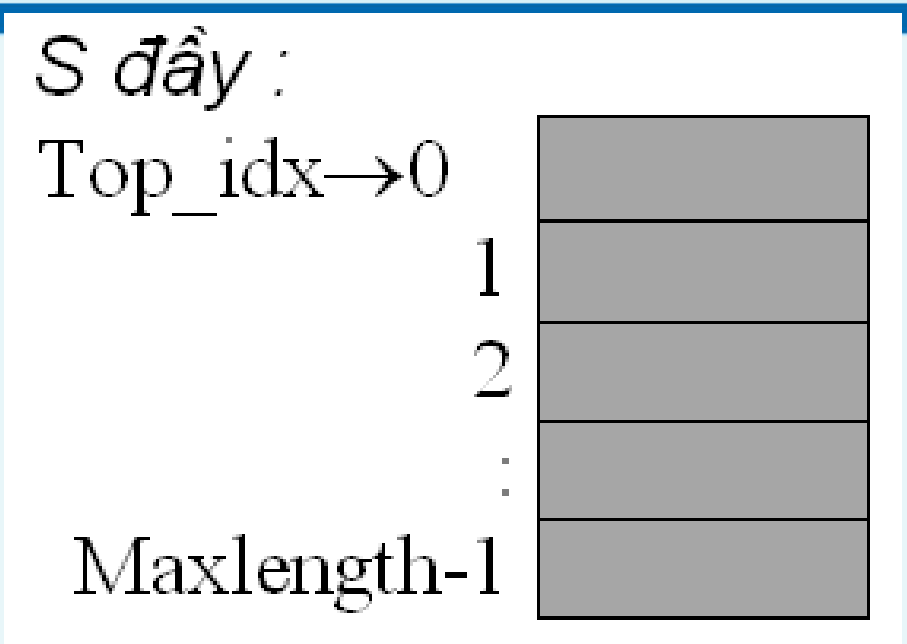
KIỂM TRA NGĂN XẾP RỖNG?

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

```
int Empty_Stack(Stack S)
{
        return S.Top_Idx==Max_Length;
}
```

CANTHO UNIVERSITY

KIỂM TRA NGĂN XẾP ĐẦY?

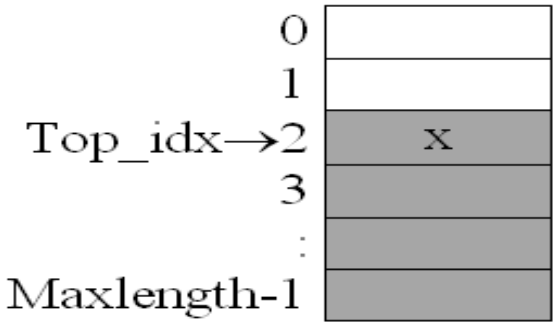


- Ta kiểm tra xem Top_Idx có bằng 0 hay không?

```
int Full_Stack(Stack S)
{
        return S.Top_Idx==0;
}
```

TRẢ VỀ PHẦN TỬ TẠI ĐỈNH NGĂN XẾP

Ví dụ :



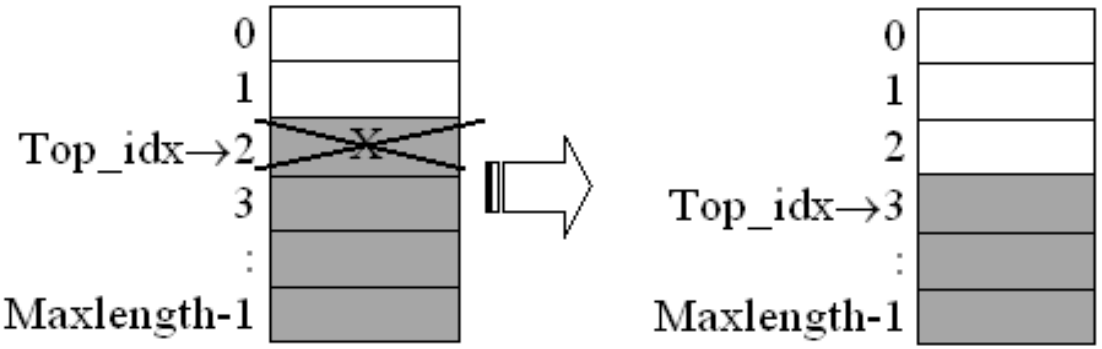
Kết quả của phép toán trên ngăn xếp là x

- 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 Top_Idx

```
Element_Type Top(Stack S) {
        return S.Elements[S.Top_Idx];
 }
```

XÓA PHẦN TỬ TẠI ĐỈNH NGĂN XẾP

Ví dụ :

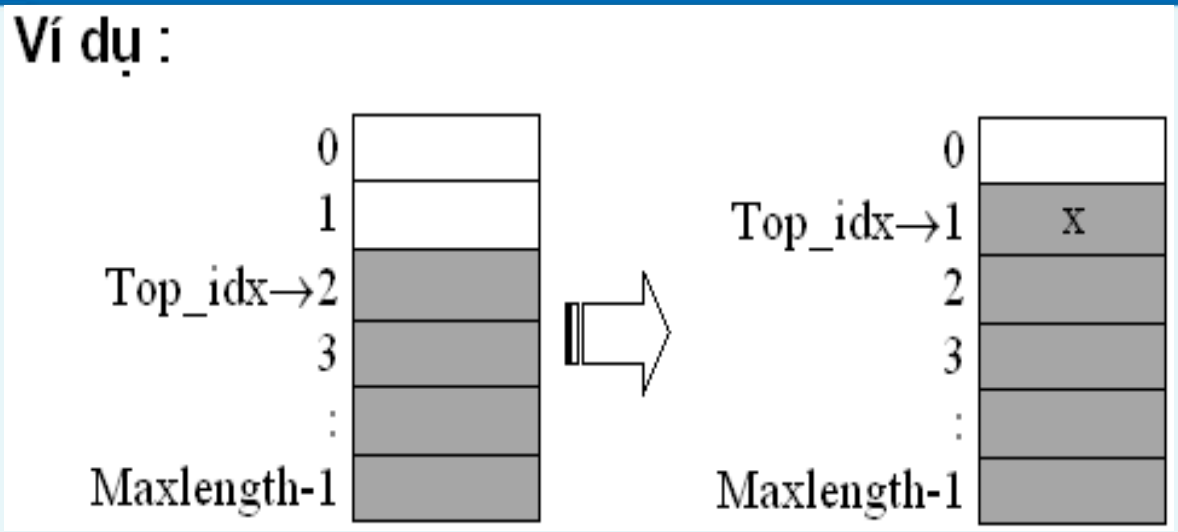


- Input: Ngăn xếp S
- Output: không có giá trị trả về, nhưng ngăn xếp S đã được xóa phần tử tại đỉnh phải được truyền ra ngoài
- Thuật toán: Tăng Top_Idx lên 1 đơn vị

```
void Pop(Stack &S) {
        S.Top_Idx++;
  }
```

THÊM PHẦN TỬ X VÀO NGĂN XẾP

Ví dụ :



- 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 Top_Idx xuống 1 đơn vị rồi đưa giá trị x vào ô có chỉ số Top_idx

```
void Push(Element_Type X, Stack &S){
    S.Top_Idx--;
    S.Elements[S.Top_Idx]=X;
  }
```

BÀI TẬP

Viết hàm **void Print_Binary(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 Print_Binary(int n) {
   if (n==0) printf("0");
   else {
        Stack S;
        Make_Null_Stack(S);
        while (n!=0) {
             Push(n%2, S);
             n=n/2;    }
        while (!Empty_Stack(S)) {
            printf("%d", Top(S));
            Pop(S);  }
   }
}
```

CANTHO UNIVERSITY

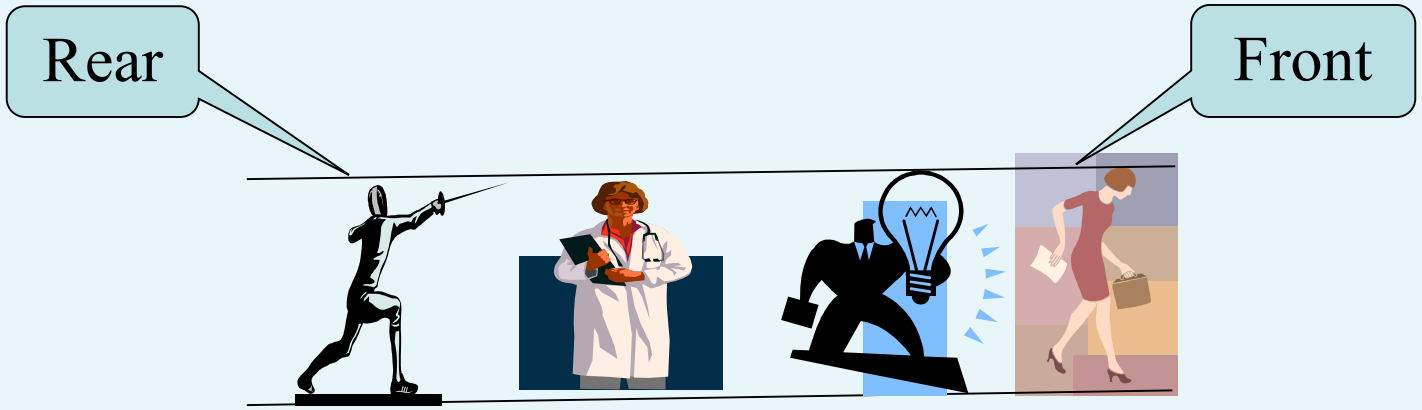
HÀNG ĐỢI (QUEUE)

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

CANTHO UNIVERSITY

ĐỊNH NGHĨA HÀNG ĐỢI

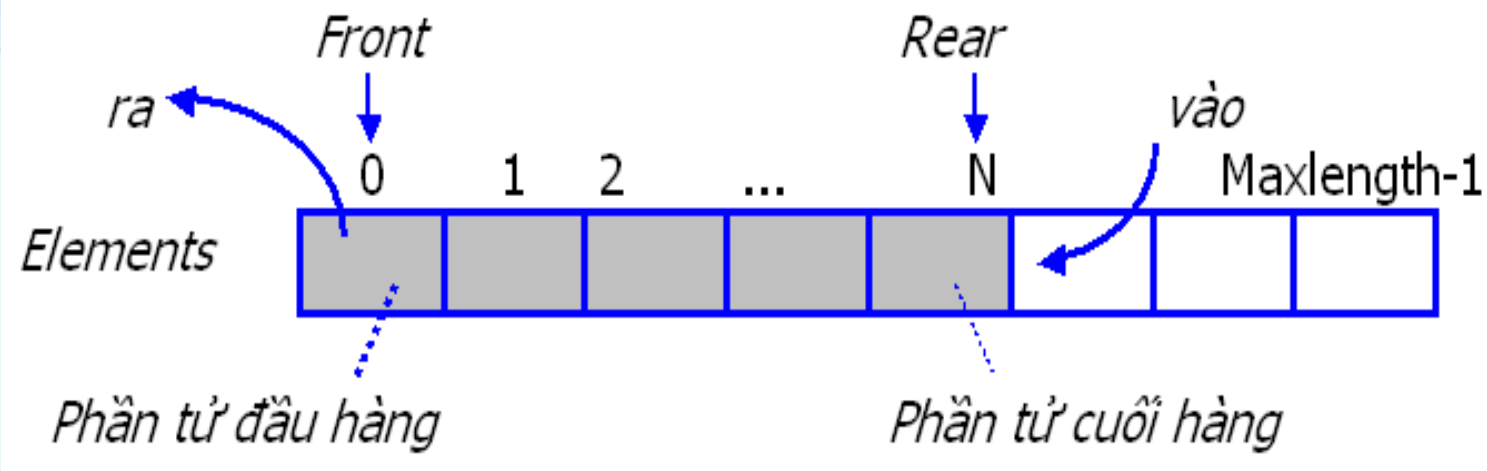
- Là một danh sách đặc biệt, mà phép thêm vào chỉ thực hiện ở 1 đầu, gọi là cuối hàng (Rear), còn phép loại bỏ thì thực hiện ở đầu kia của danh sách, gọi là đầu hàng (Front)
- Cách làm việc theo dạng FIFO (First In First Out)



CÁC PHÉP TOÁN TRÊN HÀNG

Phép toán	Diễn giải
Make_Null_Queue(Q)	Tạo hàng Q rỗng
Empty_Queue(Q)	Kiểm tra xem hàng Q có rỗng?
Full_Queue(Q)	Kiểm tra xem hàng Q có đầy?
En_Queue(X,Q)	Thêm phần tử X vào cuối hàng Q
De_Queue(Q)	Xóa phần tử tại đầu hàng Q
Front(Q)	Trả về giá trị của phần tử tại đầu hàng Q

CÀI ĐẶT HÀNG BẰNG MẢNG DI CHUYỂN TỊNH 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 = Max_Length
- Hàng tràn khi Rear = Max_Length - 1

KHAI BÁO

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

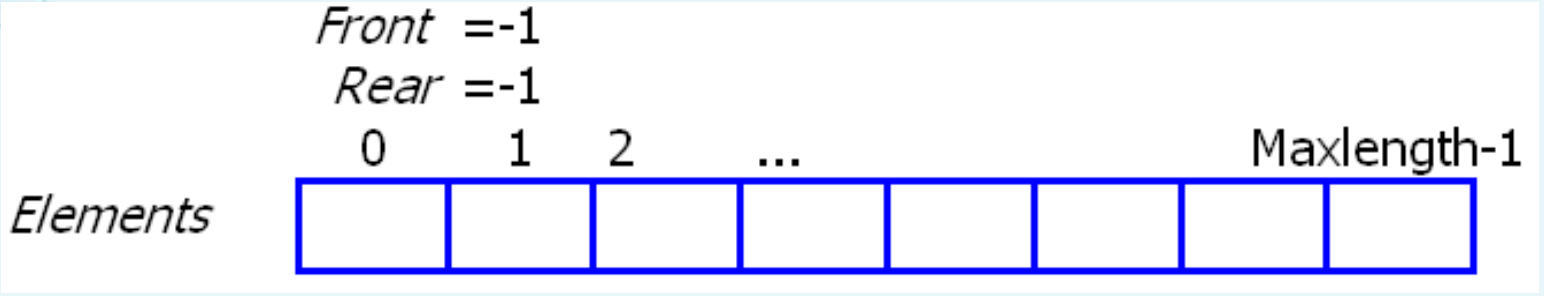
KHỞI TẠO HÀNG Q RỖNG

- Front và Rear không trỏ đến vị trí hợp lệ nào
- Ta cho Front = Rear = -1

```
void Make_Null_Queue(Queue &Q){
    Q.Front = -1;
    Q.Rear = -1;
}
```

CANTHO UNIVERSITY

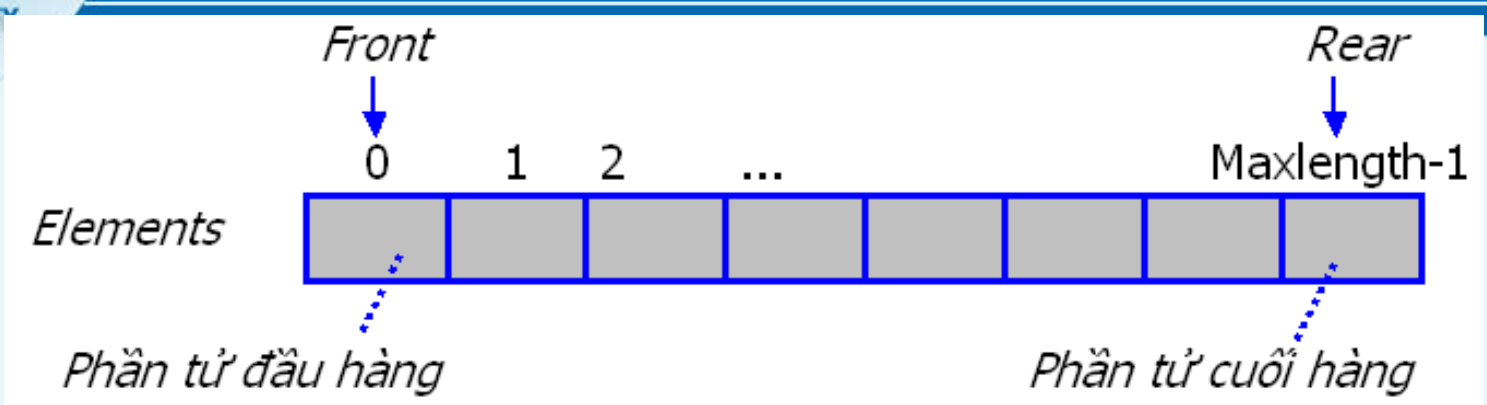
KIỂM TRA HÀNG RỖNG



- Hàng rỗng khi Front = -1

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

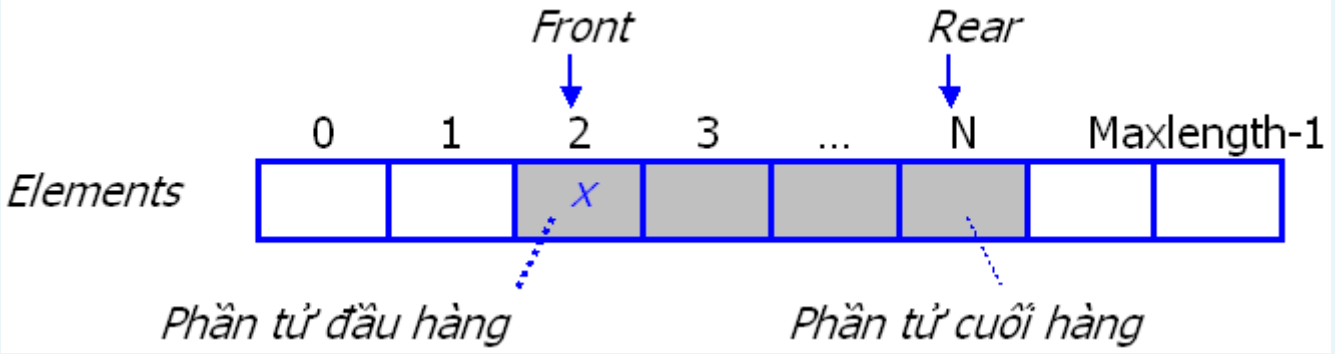
KIỂM TRA HÀNG ĐẦY



- Hàng đầy khi số phần tử hiện có trong hàng = Max_Length

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

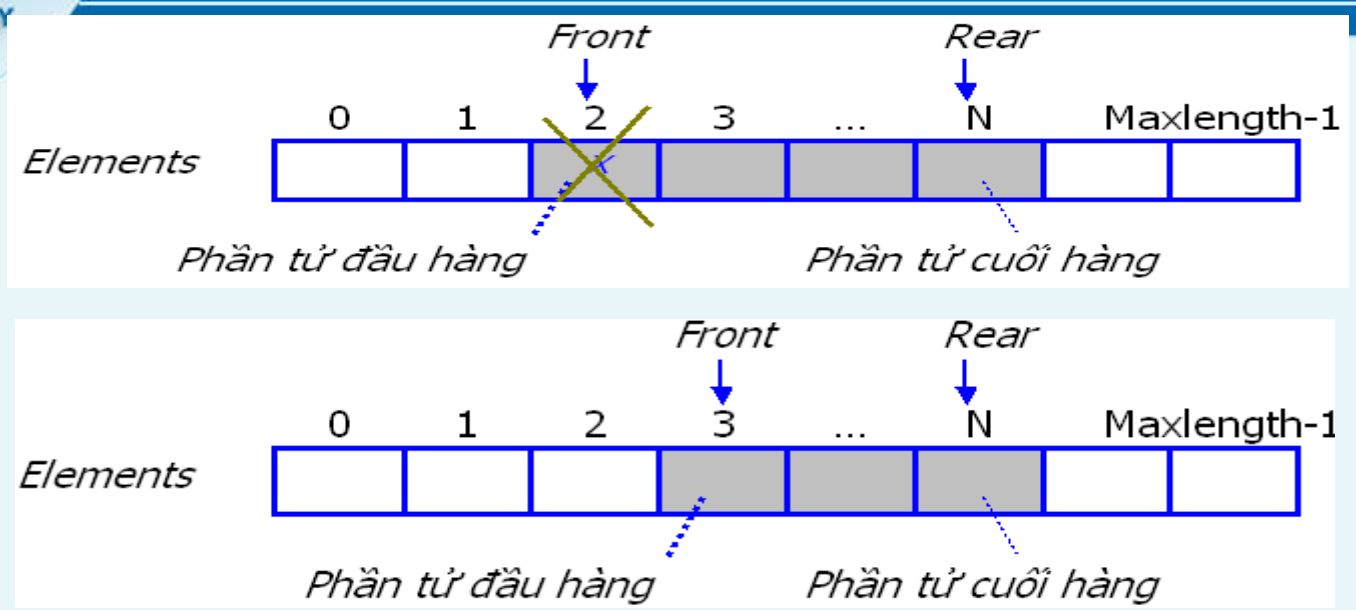
TRẢ VỀ PHẦN TỬ ĐẦU HÀNG



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

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

XÓA PHẦN TỬ TẠI ĐẦU HÀNG (1)



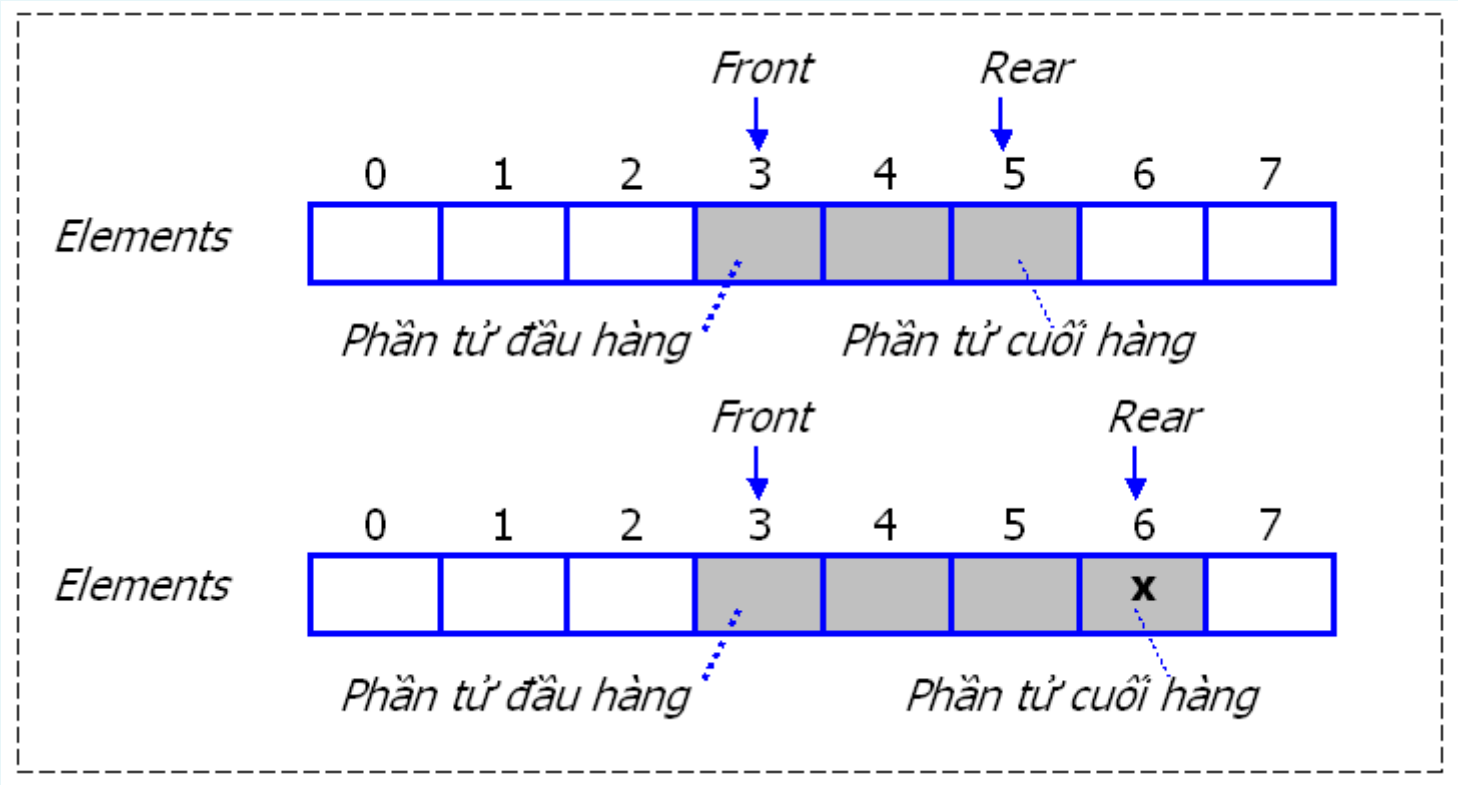
- Thuật toán:
  - Nếu hàng chỉ có một phần tử thì khởi tạo lại hàng rỗng
  - Ngược lại, tăng Front lên 1 đơn vị

XÓA PHẦN TỬ TẠI ĐẦU HÀNG (2)

```
void De_Queue(Queue &Q) {
        if (Q.Front == Q.Rear)
                Make_Null_Queue(Q);
        else    Q.Front++;
}
```

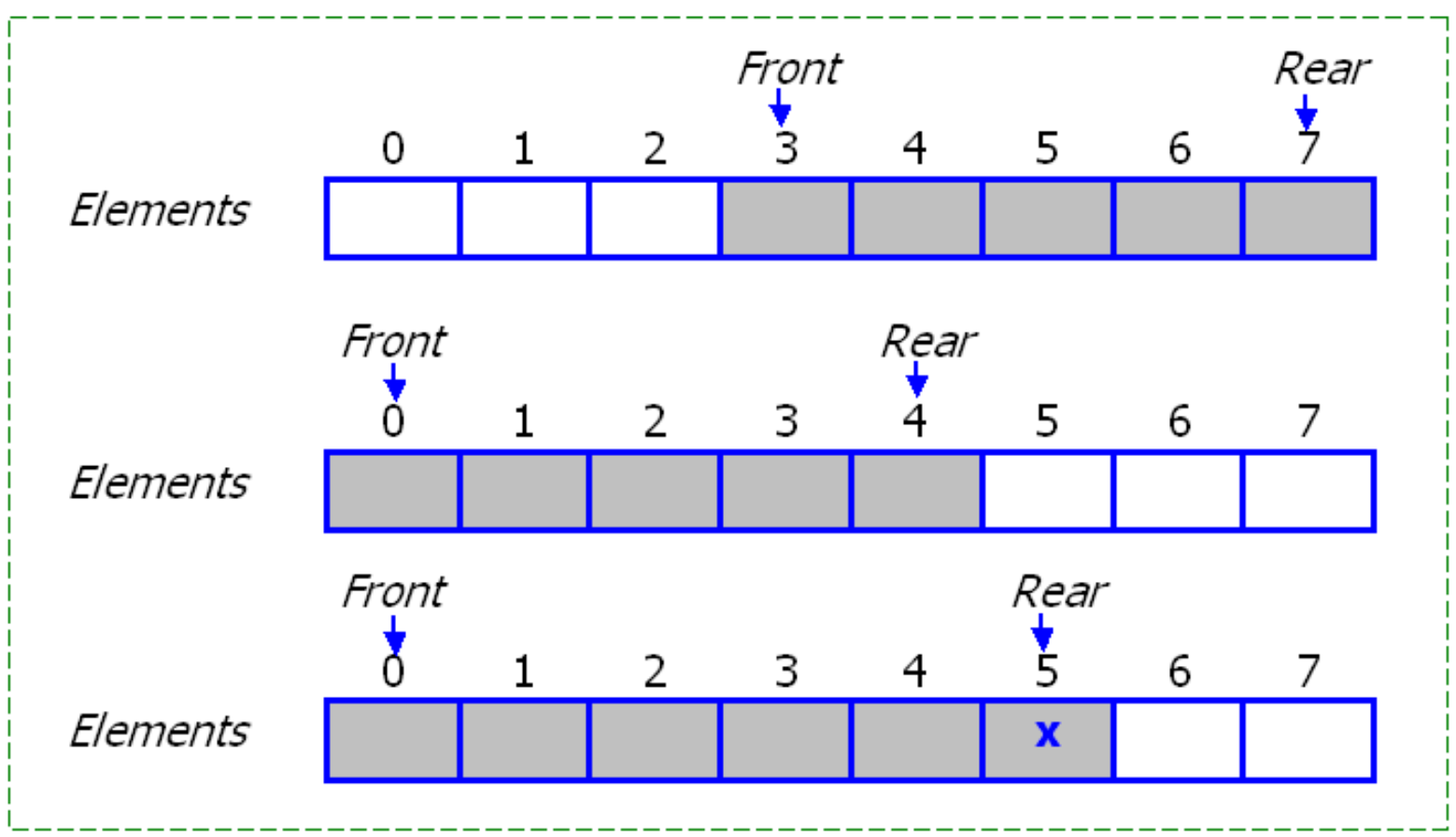
THÊM PHẦN TỬ X VÀO CUỐI HÀNG (1)

- Trường hợp bình thường



THÊM PHẦN TỬ X VÀO CUỐI HÀNG (2)

– Trường hợp hàng bị tràn



THÊM PHẦN TỬ X VÀO CUỐI HÀNG (3)
Hàm En_Queue(X,Q)

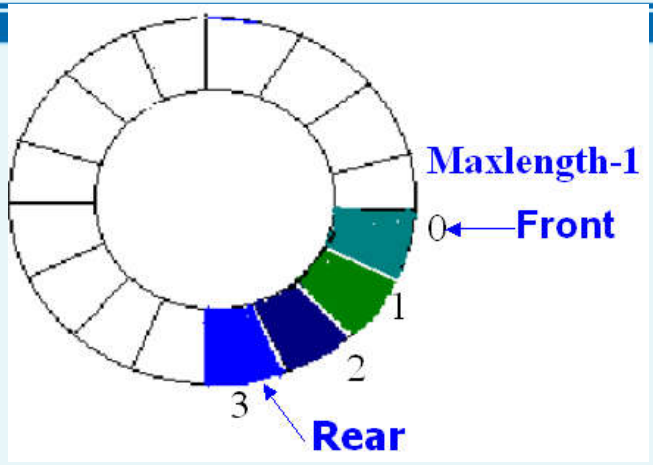
- Thuật toán:
  - Nếu hàng rỗng thì đặt Front = 0
  - Nếu hàng tràn thì phải dời tất cả phần tử lên Front vị trí. Xác định lại Front và Rear mới:
    - Rear = Rear-Front
    - Front =0
  - Tăng Rear 1 đơn vị và đưa giá trị X vào ô có chỉ số Rear mới này

THÊM PHẦN TỬ X VÀO CUỐI HÀNG (4)

```
void En_Queue(Element_Type X,Queue &Q){
    if (Empty_Queue(Q)) Q.Front = 0;
    if (Q.Rear == Max_Length-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;
}
```

CÀI ĐẶT HÀNG BẰNG MẢNG VÒNG

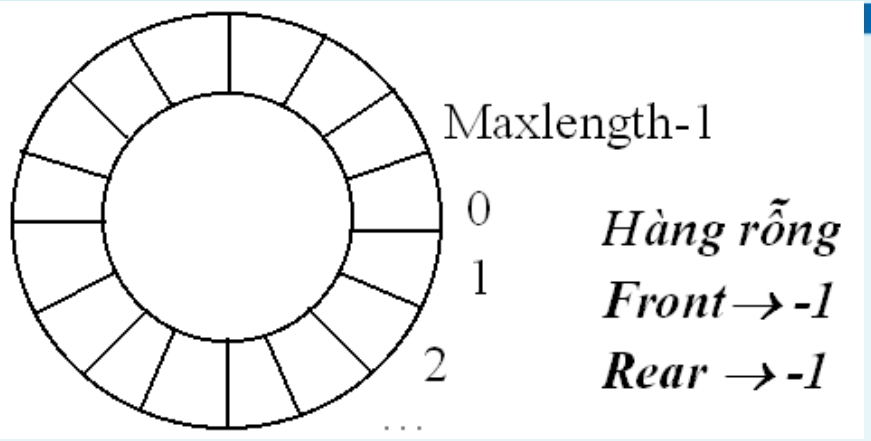
- Mô hình



- Khai báo

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

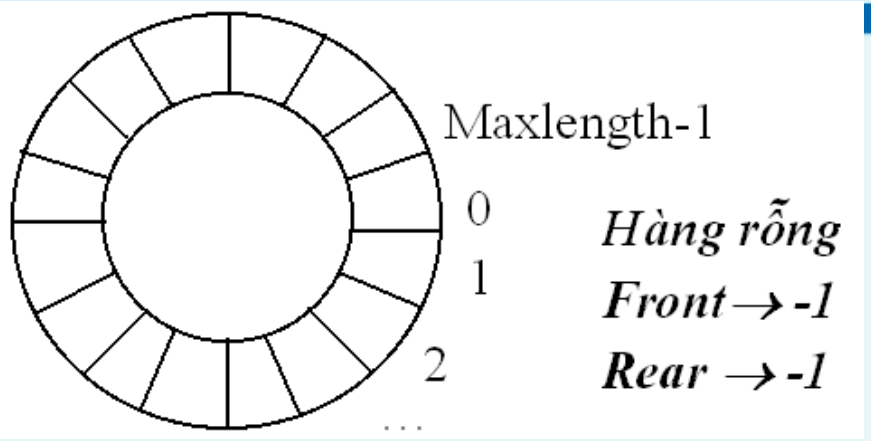
KHỞI TẠO HÀNG RỖNG



- Front và Rear không trỏ đến vị trí hợp lệ nào
- Ta cho Front = Rear = -1
- **void** Make_Null_Queue(Queue &Q)

```
{
        Q.Front = -1;
        Q.Rear = -1;
}
```

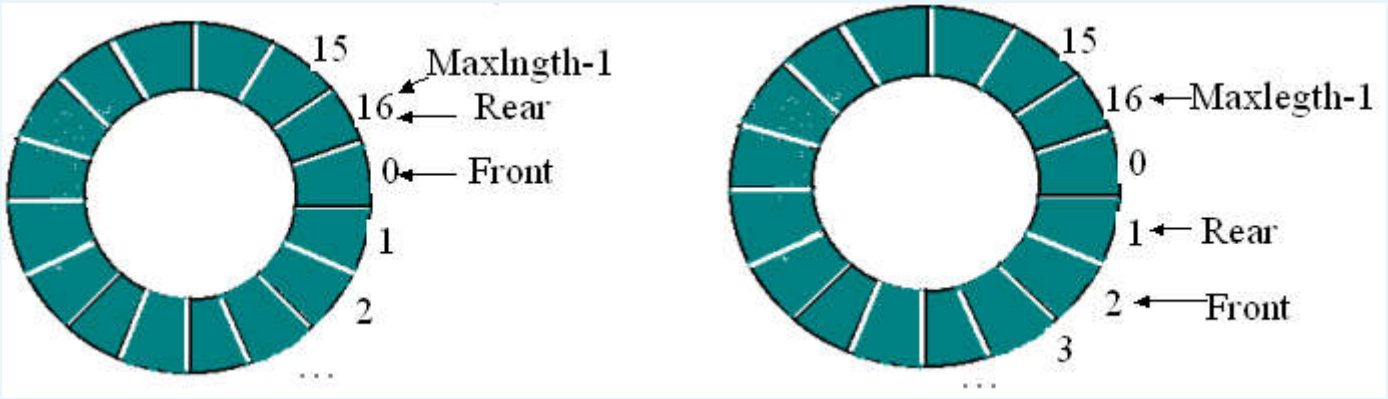
KIỂM TRA HÀNG RỖNG




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

KIỂM TRA HÀNG ĐẦY

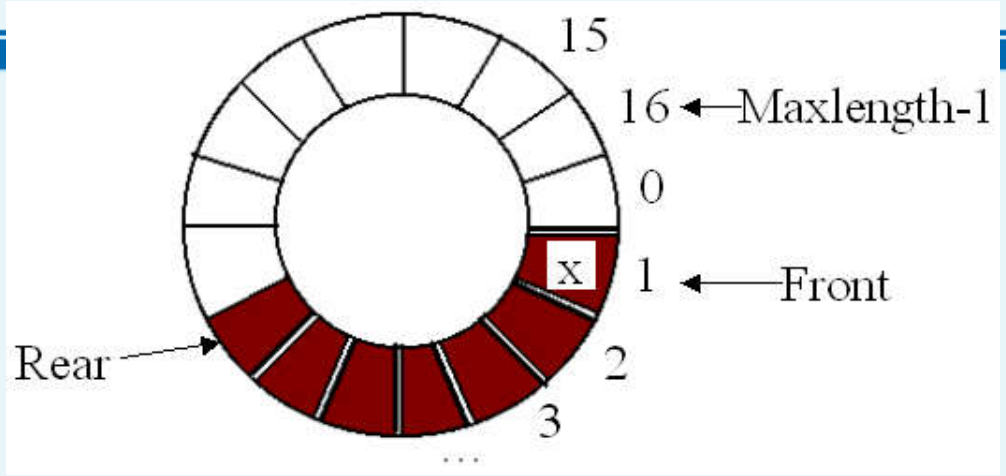
- Ví dụ



- Hàng đầy khi số phần tử hiện có trong hàng "bằng" Max_Length

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

LẤY GIÁ TRỊ PHẦN TỬ ĐẦU HÀNG

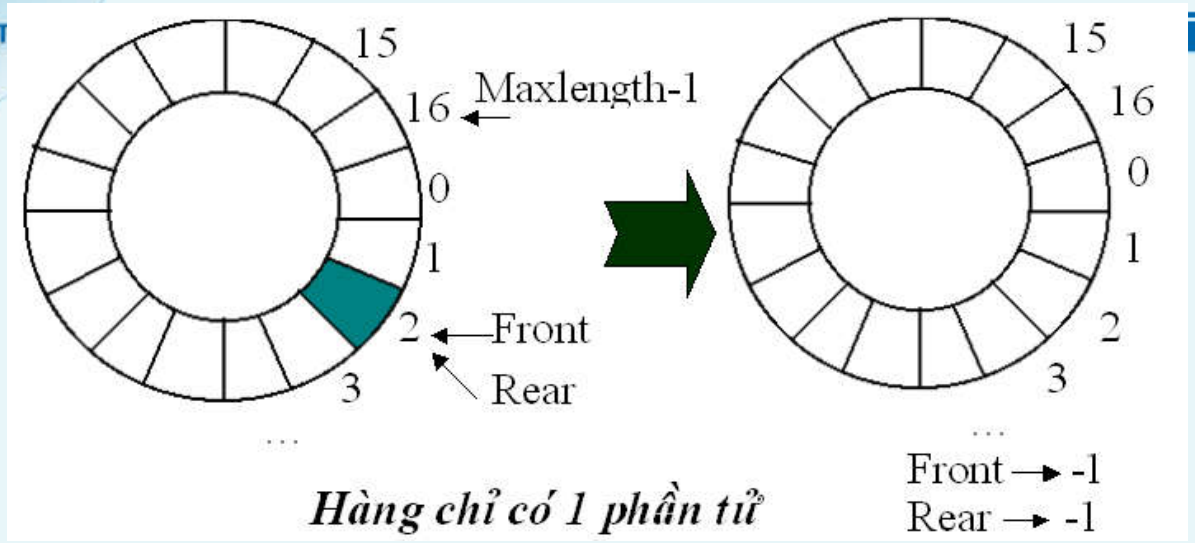


=>Thuật toán: trả về giá trị được lưu trữ tại ô có chỉ số là Front

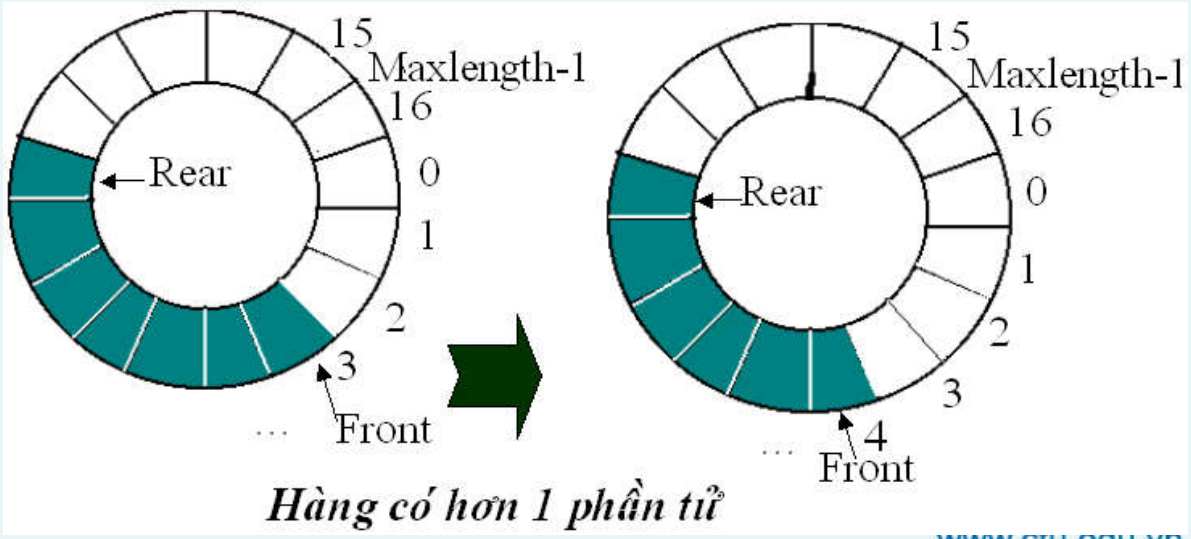
```
Element_Type Front(Queue Q){
    return Q.Elements[Q.Front];
}
```

XÓA PHẦN TỬ TẠI ĐẦU HÀNG (1)

- Các trường hợp có thể:



Hàng chỉ có 1 phần tử



Hàng có hơn 1 phần tử

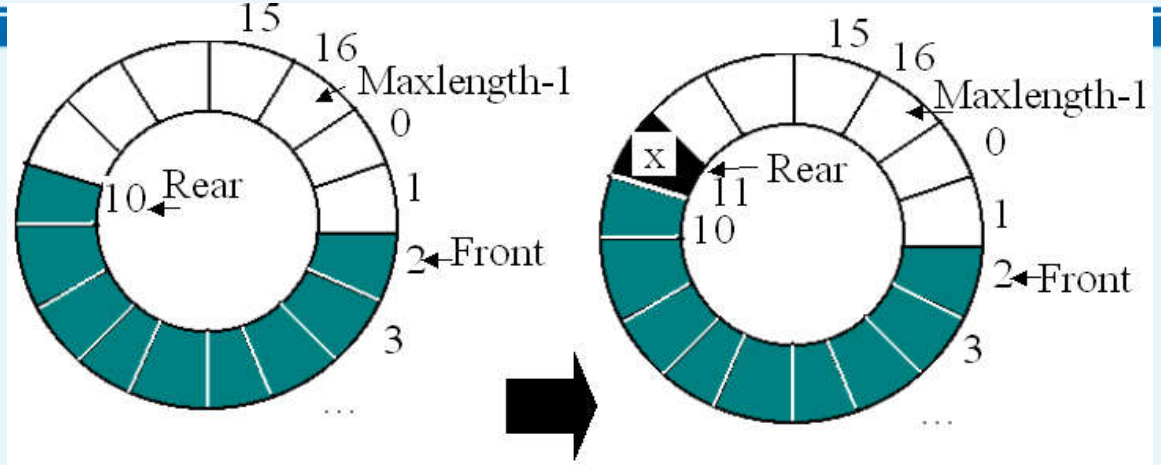
XÓA PHẦN TỬ TẠI ĐẦU HÀNG(2)

- Thuật toán :
  - Nếu Front=Rear tức hàng chỉ còn 1 phần tử thì khởi tạo lại hàng rỗng
  - Ngược lại, "tăng" Front lên 1 đơn vị

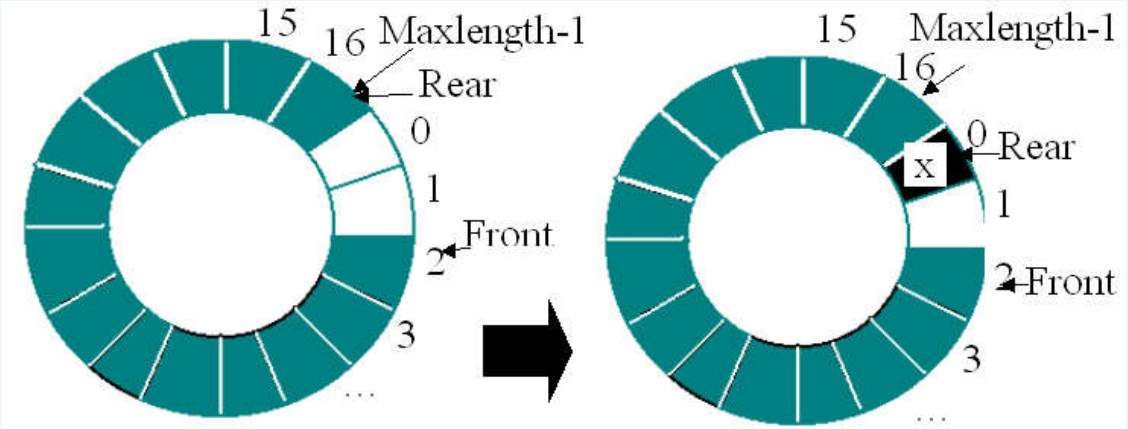
```
void De_Queue(Queue &Q){
  if (Q.Front==Q.Rear) Make_Null_Queue(Q);
  else Q.Front = (Q.Front+1) % Max_Length;
}
```

THÊM PHẦN TỬ X VÀO CUỐI HÀNG (1)

- Các trường hợp có thể:



Trường hợp Rear<Maxlength-1



Trường hợp Rear=Maxlength-1

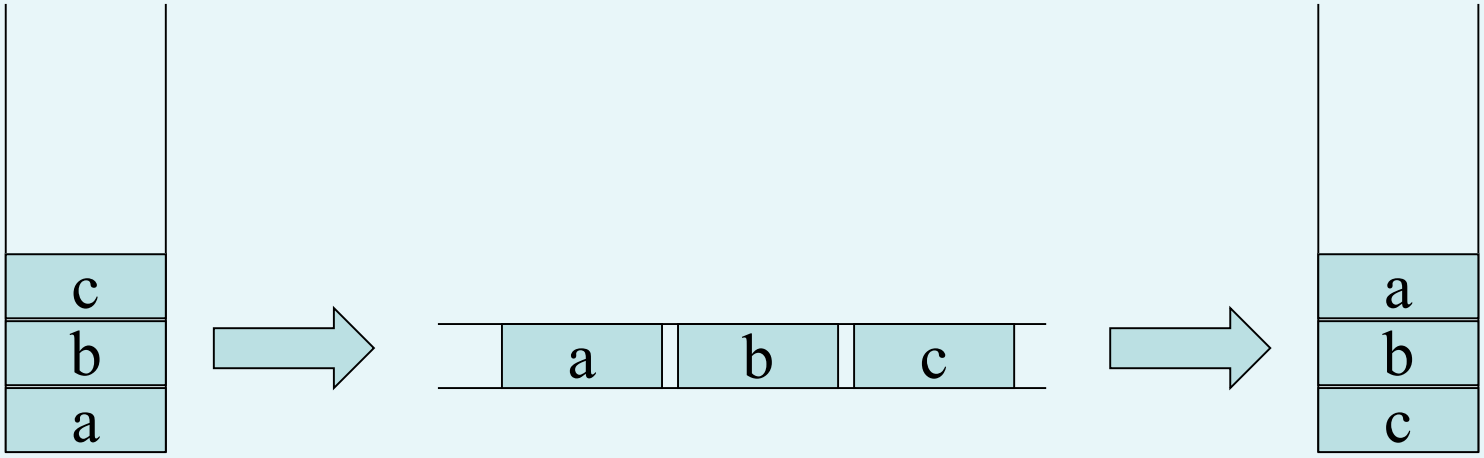
THÊM PHẦN TỬ X VÀO CUỐI HÀNG (2)
Hàm En_Queue(X,Q)

- Thuật toán :
  - 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 En_Queue(Element_Type X, Queue &Q){
        if (Empty_Queue(Q)) Q. Front=0;
        Q.Rear = (Q.Rear + 1) % Max_Length;
        Q.Elements[Q.Rear] = X;
}
```

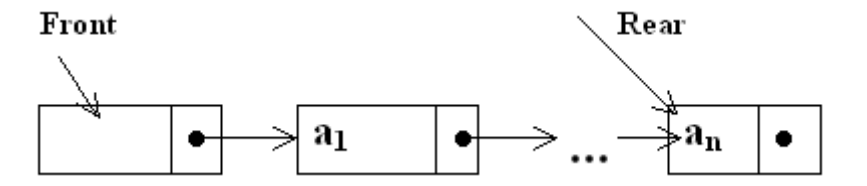
BÀI TẬP

- 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



CÀI ĐẶT HÀNG BẰNG DSLK

- Mô hình

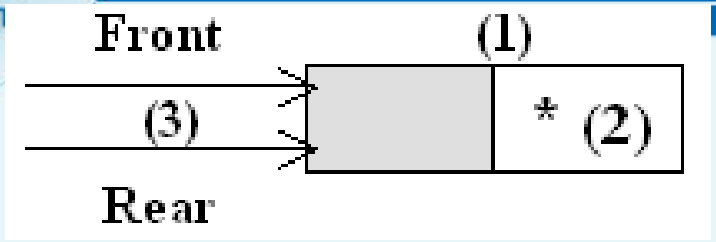


  - Dùng 2 con trỏ Front và Rear để chỉ tới phần tử đầu hàng và cuối hàng

- Khai báo

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

KHỞI TẠO HÀNG Q RỖNG



- Cho Front và Rear cùng trỏ đến HEADER của hàng

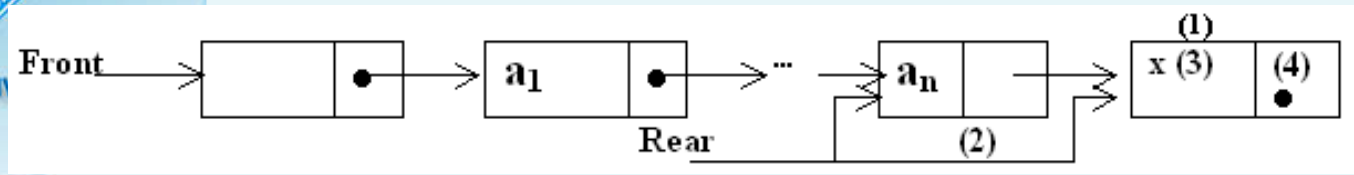
```
void Make_Null_Queue(Queue &Q){
   Position Header;
   Header = (Node*)malloc(sizeof(Node)); //Cấp phát Header
   Header->Next = NULL;
   Q.Front = Header;
   Q.Rear = Header;
}
```

KIỂM TRA HÀNG Q RỖNG

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

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

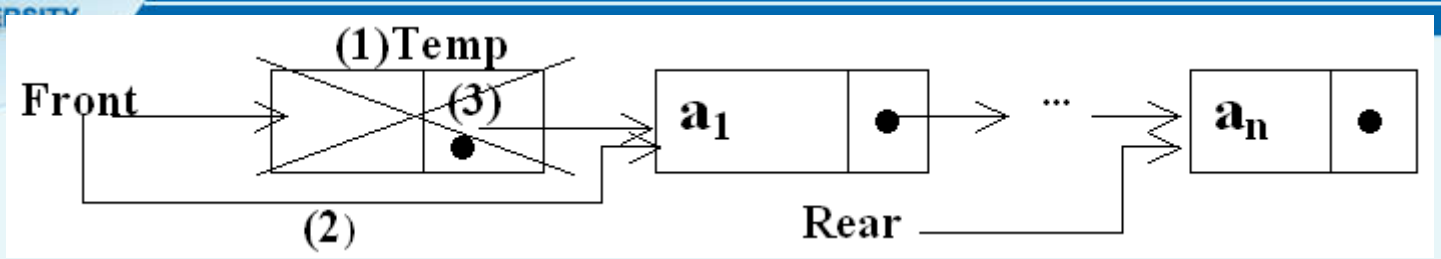
THÊM MỘT PHẦN TỬ X VÀO HÀNG Q



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

```
void En_Queue(Element_Type 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;
}
```

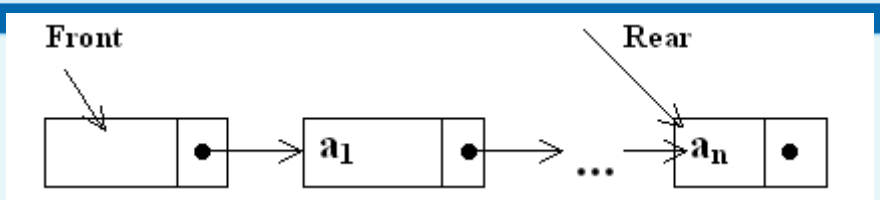
XÓA MỘT PHẦN TỬ KHỎI HÀNG Q



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

```
void De_Queue(Queue &Q){
    if (!Empty_Queue(Q)){
            Position Tempt;
            Tempt=Q.Front;
            Q.Front=Q.Front->Next;
            free(Tempt);
    }else printf("Loi : Hang rong");
 }
```

Xác định giá trị của phần tử tại đầu hàng



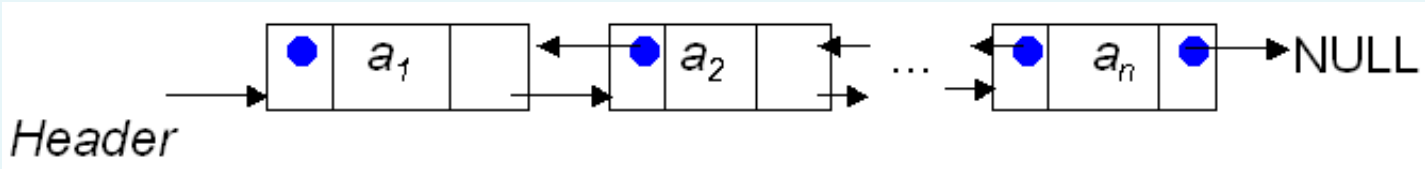

```
Element_Type Front (Queue Q) {
    if (!Empty_Queue(Q))
        returrn (Q.Front->Next->Element);
    else
        printf("Hang rong");
}
```

CÁC ỨNG DỤNG CỦA NGĂN XẾP VÀ HÀNG ĐỢI

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

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 ... Element_Type;//kiểu nội dung của phần tử
  typedef struct Node{
                    Element_Type 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;
```

DANH SÁCH RỖNG

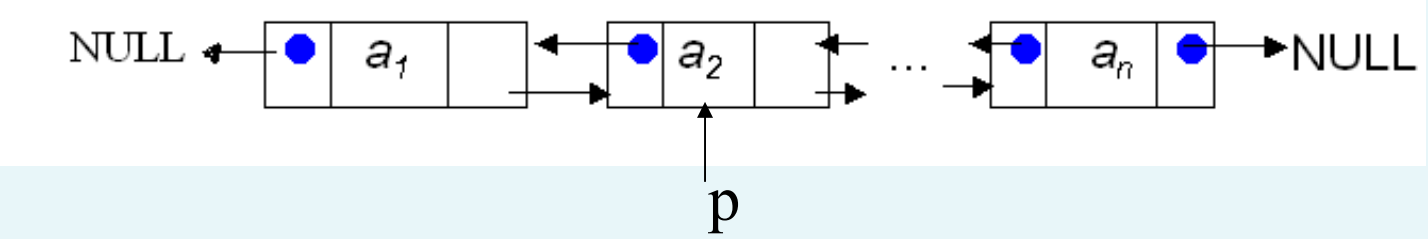
- Tạo danh sách rỗng

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

- Kiểm tra danh sách rỗng

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

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



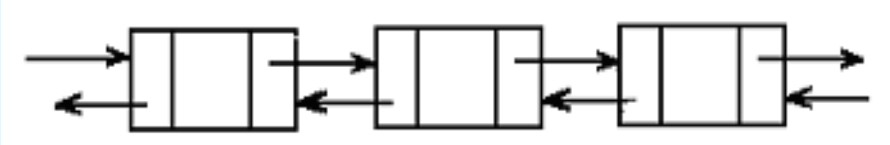
- =>Vị trí của một phần tử là con trỏ trỏ vào ngay chính phần tử đó

- **Element_Type** Retrieve_Double_List (Position P, Double_List DL)

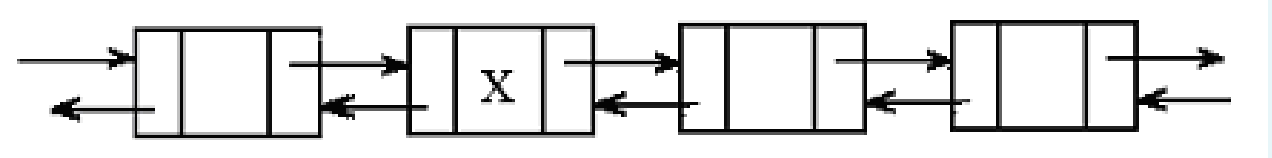
```
{
        return  P->Element;
}
```

THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (1)

- Trước khi thêm



- Sau khi thêm

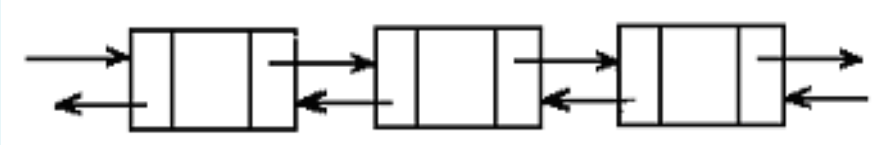


=>Cấp phát một ô nhớ mới chứa phần tử cần thêm

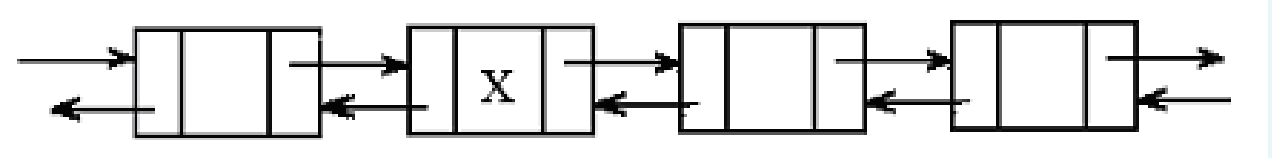
=>Đặt lại các liên kết

THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (2)

- Trước khi thêm



- Sau khi thêm



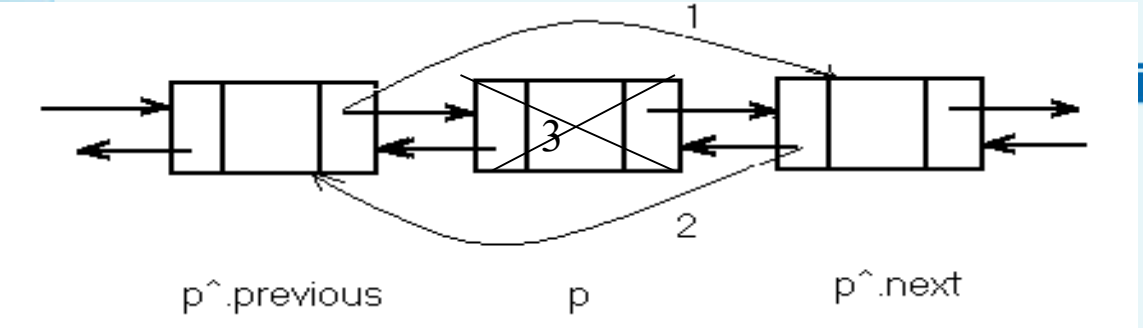
=>Cấp phát một ô nhớ mới chứa phần tử cần thêm

=>Đặt lại các liên kết

THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (2)

```
void Insert_Double_List (Element_Type 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;
        }
 }
```

XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH




```
void Delete_Double_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);
     }
}
```

ƯU ĐIỂM CỦA DSLK KÉP

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

Thank you!