CHƯƠNG 2
CÁC KIỂU DỮ LIỆU TRỪU TƯỢNG CƠ BẢN (BASIC ABSTRACT DATA TYPES)

Nguyễn Công Danh

1

NỘI DUNG SẼ HỌC

- DANH SÁCH
- NGĂN XẾP
- HÀNG ĐỢI

2

DANH SÁCH

- KHÁI NIỆM VỀ DANH SÁCH
- CÁC PHÉP TOÁN
- CÀI ĐẶT
  - DÙNG MẢNG (DS ĐẶC)
  - DÙNG CON TRỎ (DS LIÊN KẾT)

3

KHÁI NIỆM VỀ DANH SÁCH

- Là tập hợp hữu hạn các phần tử có cùng kiểu
- Kiểu chung được gọi là kiểu phần tử (element type)
- Ta thường biểu diễn dạng: a1, a2, a3, ..., 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
- Các phần tử trong danh sách có thứ tự tuyến tính theo vị trí xuất hiện. Ta nói a_i đứng trước a_{i+1} (i=1..n-1)

4

CÁC PHÉP TOÁN (1)

T³n phÐp to¸n	C«ng dông
ENDLIST(L)	Trả về vị trí sau phần tử cuối trong ds L
MAKENULL_LIST(*L*)	Khởi tạo một danh sách L rỗng
EMPTY_LIST(*L*)	Kiểm tra xem danh sách L có rỗng hay không
FULL_LIST(*L*)	Kiểm tra xem danh sách L có đầy hay không
INSERT_LIST(*X,P,L*)	Xen phần tử có nội dung X vào danh sách L tại vị trí P
DELETE_LIST(*P,L*)	Xóa phần tử tại vị trí P trong danh sách L
LOCATE_LIST(*X,L*)	Trả về kết quả là vị trí của phần tử có nội dung X trong danh sách L Nếu không tìm thấy: trả về ENDLIST(L)

5

CÁC PHÉP TOÁN (2)

RETRIEVE(*P,L*)	Trả về nội dung phần tử thứ P trong danh sách L
NEXT(*P,L*)	Trả về phần tử đứng sau phần tử thứ P trong danh sách L
PREVIOUS(*P,L*)	Trả về phần tử đứng trước phần tử thứ P trong danh sách L
FIRST(*L*)	Trả về kết quả là vị trí của phần tử đầu danh sách, ENDLIST(L) nếu danh sách rỗng
PRINT_LIST(*L*)	Hiển thị các phần tử trong danh sách L theo thứ tự xuất hiện

6

VÍ DỤ

Dùng các phép toán trừu tượng trên danh sách, viết chương trình con nhận vào 1 danh sách rồi sắp xếp danh sách theo thứ tự tăng dần

```
void SORT(LIST L)
{       Position p,q;       //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);  // hoán đổi nội dung 2 phần tử
                  q=NEXT(q,L);
        }
        p=NEXT(p,L);
   }
}
```

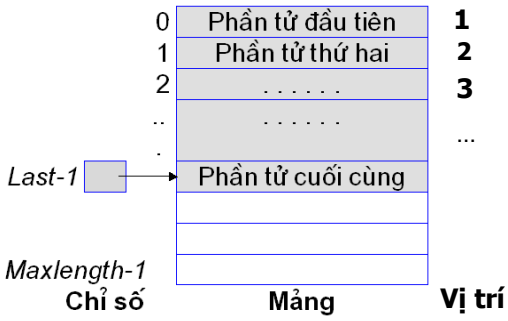
7

CÀI ĐẶT DANH SÁCH BẰNG MẢNG (DS ĐẶC)

- Dùng 1 mảng để lưu trữ liên tiếp các phần tử, bắt đầu từ vị trí đầu tiên
- Ta phải ước lượng số phần tử tối đa của danh sách
- Ta phải lưu trữ độ dài hiện tại của danh sách (Last)

8

MÔ HÌNH



- *Ta định nghĩa vị trí của một phần tử trong danh sách là "**chỉ số của mảng tại vị trí lưu trữ phần tử đó + 1**"*

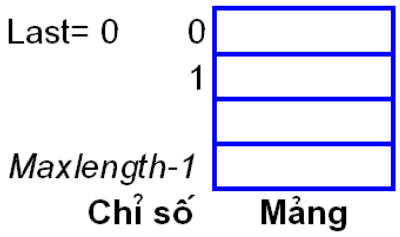
9

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 cuả các phần tử
typedef struct {
     ElementType Elements[MaxLength];
    //mảng chứa các phần tử của danh sách
     Position Last; //giữ độ dài danh sách
 } List;
List L;
```

10

KHỞI TẠO DANH SÁCH RỖNG



- Cho độ dài danh sách bằng 0

```
void MakeNull_List(List *L)
{
      L->Last=0;
}
```

11

KIỂM TRA DANH SÁCH RỖNG DS ĐẦY

- Xem độ dài danh sách có bằng 0 không?

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

12

XEN PHẦN TỬ X VÀO VỊ TRÍ P (1)

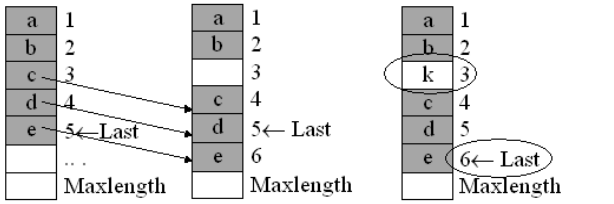
- Xen phần tử x='k' vào vị trí p=3 trong danh sách L (chỉ số 2 trong mảng)

- **Nếu L có dạng :**

⇨ Trường hợp này mảng đầy => báo lỗi

Chỉ số mảng	Nội dung	Vị trí trong dsách
0	a	1
1	b	2
2	c	3
3	d	4
4	g	:
5	h	Maxlength ←Last

- **Nếu danh sách L có dạng :**



1. Dời các phần tử từ vị trí 3 đến cuối danh sách ra sau một vị trí
2. Độ dài danh sách tăng 1
3. Đưa k vào vị trí 3

13

XEN PHẦN TỬ X VÀO VỊ TRÍ P (2)

- Tóm lại, để chèn x vào vị trí p của L, ta làm như sau:
  - Nếu mảng đầy thì thông báo lỗi
  - Ngược lại, nếu vị trí p không hợp lệ thì báo lỗi
  - Ngược lại:
    - Dời các phần tử từ vị trí p đến cuối danh sách ra sau một vị trí
    - Đưa phần tử mới x vào tại vị trí p
    - Độ dài danh sách tăng 1

14

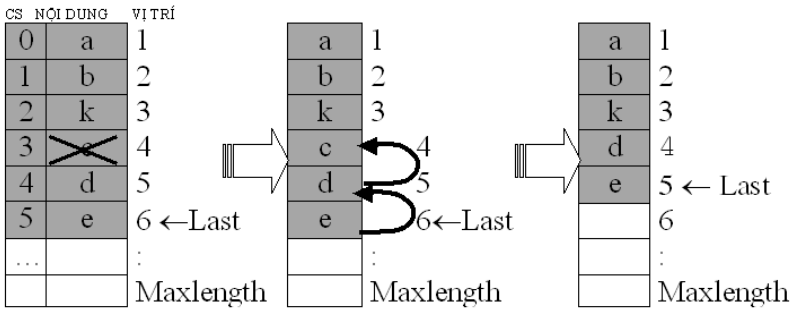
XEN PHẦN TỬ X VÀO VỊ TRÍ P (3)

```
void Insert_List(ElementType X,Position P, List *L){
    if (L->Last==MaxLength)
       printf("Danh sach day");
  else if ((P<1) || (P>L->Last+1))
       printf("Vi tri khong hop le");
   else {
     Position Q;
    /*Dời các phần tử từ vị trí p đến cuối dsách ra
       trước 1 vị trí*/
     for(Q=(L->Last-1)+1;Q>P-1;Q--)
        L->Elements[Q]=L->Elements[Q-1];
     //Đưa x vào vị trí p
     L->Elements[P-1]=X;
     //Tăng độ dài danh sách lên 1
     L->Last++;
    }
}
```

15

XÓA MỘT PHẦN TỬ TẠI VỊ TRÍ P TRONG DS (1)

- Ví dụ: Xóa phần tử vị trí p=4 của L



16

XÓA MỘT PHẦN TỬ TẠI VỊ TRÍ P TRONG DS (2)

- Nếu p là một vị trí không hợp lệ thì thông báo lỗi
- Ngược lại:
  - Di dời các phần tử từ vị trí p+1 đến cuối danh sách ra trước một vị trí
  - Độ dài của danh sách giảm 1

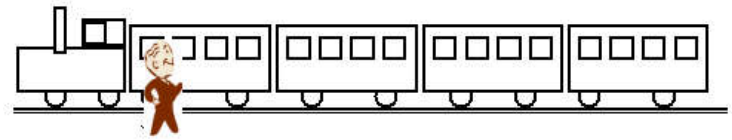
17

XÓA MỘT PHẦN TỬ TẠI VỊ TRÍ P TRONG DS (3)

```
void Delete_List(Position P,List *L)
{ if ((P<1) || (P>L->Last))
       printf("Vi tri khong hop le");
  else if (EmptyList(*L))
       printf("Danh sach rong!");
  else
    {Position Q;
   /*Dời các phần tử từ vị trí p+1   đến cuối
        danh sách ra trước 1 vị trí*/
       for(Q=P-1;Q<L->Last-1;Q++)
          L->Elements[Q]=L->Elements[Q+1];
       L->Last--;
      }
 }
```

18

TÌM KIẾM PHẦN TỬ X TRONG DS(1)



- Để tìm phần tử x trong danh sách ta tiến hành tìm từ đầu danh sách cho đến khi tìm gặp
- Nếu gặp thì vị trí của phần tử đầu tiên tìm thấy được trả về
- Nếu không tìm gặp thì trả về vị trí Last+1(EndList)

19

TÌM KIẾM PHẦN TỬ X TRONG DS(1)

```
Position Locate(ElementType X, List L)
{Position P;
  int Found = 0;
  P = First(L); //vị trí phần tử đầu tiên

/*trong khi chưa tìm thấy và chưa kết thúc
  danh sách thì xét phần tử kế tiếp*/

  while ((P != EndList(L)) && (Found == 0))
     if (Retrieve(P,L) == X) Found = 1;
     else P = Next(P, L);
  return P;
}
```

20

ĐÁNH GIÁ GIẢI THUẬT TÌM KIẾM

- Thời gian tìm kiếm
  - nhanh nhất (tốt nhất) là khi nào, x ở đâu?
  - xấu nhất khi nào?
- Độ phức tạp của giải thuật thường được xác định là trong trường hợp xấu nhất O(n)

21

CÁC PHÉP TOÁN KHÁC (1)

- Xác định vị trí sau phần tử cuối trong danh sách

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

- Xác định vị trí đầu tiên trong danh sách

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

22

CÁC PHÉP TOÁN KHÁC (2)

- Xác định nội dung phần tử tại vị trí P trong dsách

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

- Xác định vị trí kế tiếp trong danh sách

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

23

BÀI TẬP

- Vận dụng các phép toán trên danh sách đặc để viết chương trình nhập vào một danh sách các số nguyên và hiển thị danh sách vừa nhập ra màn hình.
- Thêm phần tử có nội dung x vào danh sách tại ví trí p (trong đó x và p được nhập từ bàn phím).
- Xóa phần tử đầu tiên có nội dung x (nhập từ bàn phím) ra khỏi danh sách.

24

NHẬP DANH SÁCH TỪ BÀN PHÍM

```
void ReadList(List *L)
{   int i,N;
    ElementType X;
    MakeNullList(L);
    printf("So phan tu danh sach N= ");scanf("%d",&N);
    for(i=1;i<=N;i++)
     { printf("Phan tu thu %d: ",i);scanf("%d",&X);
       InsertList(X,EndList(*L),L);
      }
  }
```

25

HIỂN THỊ DANH SÁCH RA MÀN HÌNH

```
void PrintList(List L)
{  Position P;
   P = First(L);
   while (P != EndList(L))
   {     printf("%d ",Retrieve(P,L));
         P = Next(P, L);
   }
   printf("\n");
}
```

26

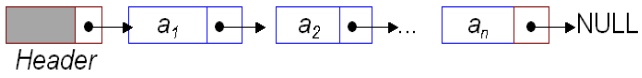
CHƯƠNG TRÌNH CHÍNH

```
int main()
{List L;
ElementType X;
Position P;
ReadList(&L);
printf("Danh sach vua nhap: ");
PrintList(L); //In danh sach len man hinh
printf("Phantu can them: ");scanf("%d",&X);
printf("Vi tri can them: ");scanf("%d",&P);
InsertList(X,P,&L);
printf("Dsach sau khi them phan tu la: ");
PrintList(L);
printf("Noi dung phan tu can xoa: ");
scanf("%d",&X);
P=Locate(X,L);
DeleteList(P,&L);
printf("Danh sach sau khi xoa %d la: ",X);
PrintList(L);
return 0;
}
```

27

CÀI ĐẶT DANH SÁCH BẰNG CON TRỎ

- Mô hình



- Nối kết các phần tử liên tiếp nhau bằng con trỏ
  - Phần tử a_i trỏ tới phần tử a_{i+1}
  - Phần tử a_n trỏ tới phần tử đặc biệt NULL
  - Phần tử header trỏ tới phần tử đầu tiên a_1
- Khai báo

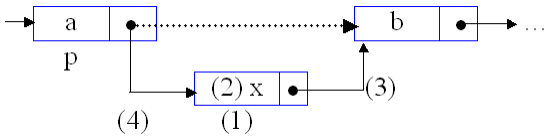
```
typedef ... ElementType; //kiểu của phần tử trong danh sách
typedef struct Node
  { ElementType Element;    //Chứa nội dung của phần tử
    Node* Next;              //con trỏ chỉ đến phần tử kế tiếp
    };
typedef Node* Position;      //Kiểu vị trí
typedef Position List;
```

28

KHỞI TẠO DANH SÁCH RỖNG

29

XEN MỘT PHẦN TỬ VÀO DANH SÁCH

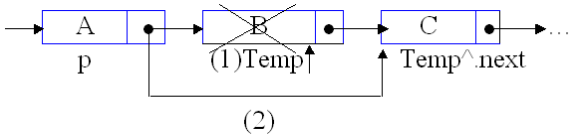


- Để xen phần tử x vào vị trí p của L, ta làm như sau:
  - Cấp phát 1 ô mới để lưu trữ phần tử x
  - Nối kết lại các con trỏ để đưa ô mới này vào vị trí p

```
void Insert_List(ElementType X,Position P, List *L)
{ Position T;
     T=(Node*)malloc(sizeof(Node));
     T->Element=X;
     T->Next=P->Next;
     P->Next=T;
 }
```

31

XÓA MỘT PHẦN TỬ KHỎI DANH SÁCH(1)



=>Muốn xóa phần tử ở vị trí p trong danh sách ta cần nối kết lại các con trỏ bằng cách cho p trỏ tới phần tử đứng sau phần tử thứ p

```
void Delete_List(Position P, List *L)
{ Position Temp;
   if (P->Next!=NULL)
   {  Temp=P->Next;
      /*giữ ô chứa phần tử bị xoá để thu hồi vùng nhớ*/
      P->Next=Temp->Next;
      /*nối kết con trỏ trỏ tới phần tử ke tiep*/
      free(Temp); //thu hồi vùng nhớ
    }
}
```

32

XÓA MỘT PHẦN TỬ KHỎI DANH SÁCH(2)

```
void Delete_List(Position P, List *L)
{
   Position Temp;
   if (P->Next!=NULL)
    {
        Temp=P->Next;
        /*giữ ô chứa phần tử bị xoá để thu hồi vùng nhớ*/
        P->Next=Temp->Next;
        /*nối kết con trỏ trỏ tới phần tử ke tiep*/
        free(Temp); //thu hồi vùng nhớ
   }
}
```

33

TÌM KIẾM MỘT PHẦN TỬ TRONG DANH SÁCH

- Để tìm phần tử x trong danh sách ta tìm từ đầu danh sách:
  - Nếu tìm gặp thì vị trí phần tử đầu tiên bằng x được trả về (p) và ta có p->next->element = x
  - Nếu không gặp thì vị trí ENDLIST được trả về

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

34

XÁC ĐỊNH NỘI DUNG PHẦN TỬ

```
ElementType Retrieve(Position P, List L)
{
  if (P->Next!=NULL)
      return P->Next->Element;
 }
```

35

XÁC ĐỊNH VỊ TRÍ PHẦN TỬ

- Vị trí phần tử đầu tiên

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

- Vị trí sau phần tử cuối cùng

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

- Vị trí phần tử kế tiếp

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

36

IN DANH SÁCH RA MÀN HÌNH

```
void PrintList(List L)
{
    Position P;
    P = First(L);
    while (P != EndList(L))
    {
        printf("%d ",Retrieve(P,L));
        P = Next(P, L);
    }
    printf("\n");
}
```

37

BÀI TẬP

Vận dụng các phép toán trên danh sách liên kết để viết chương trình:

- Nhập vào một danh sách các số nguyên
- Hiển thị danh sách vừa nhập ra màn hình
- Thêm phần tử có nội dung x vào danh sách tại vị trí p (trong đó x và p được nhập từ bàn phím)
- Xóa phần tử đầu tiên có nội dung x (nhập từ bàn phím) ra khỏi danh sách

38

SO SÁNH 2 PHƯƠNG PHÁP CÀI ĐẶT DS

- Bạn hãy phân tích ưu và khuyết điểm của
  - Danh sách đặc
  - Danh sách liên kết
- Bạn nên chọn pp cài đặt nào cho ứng dụng của mình?

39

NGĂN XẾP (STACK)

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

40

ĐỊNH NGHĨA

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

41

CÁC PHÉP TOÁN

Phép toán	Diễn giải
MAKENULL(S)	Tạo một ngăn xếp rỗng (S)
EMPTY(S)	Kiểm tra xem ngăn xếp S có rỗng hay không
FULL(S)	Kiểm tra xem ngăn xếp S có đầy hay không
PUSH(*X,S*)	Thêm phần tử X vào đỉnh ngăn xếp S. Tương đương: INSERT(X,FIRST(S),S)
POP(S)	Xóa phần tử tại đỉnh ngăn xếp S. Tương đương: DELETE(FIRST(S),S)
TOP(S)	Trả về phần tử đầu tiên trên đỉnh ngăn xếp S, tương đương: RETRIEVE(FIRST(S),S)

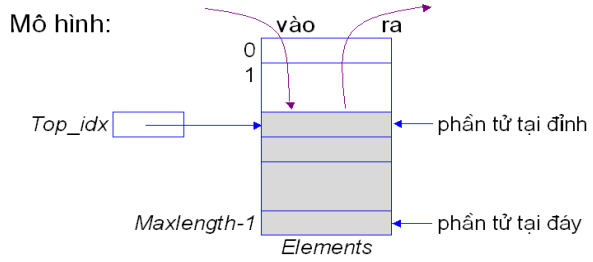
42

CÀI ĐẶT NGĂN XẾP BẰNG DSLK

- Khai báo
  typedef List Stack;
- Tạo ngăn xếp rỗng
  void MakeNull_Stack(Stack *S)
  { MakeNull_List(S);}
- Kiểm tra ngăn xếp rỗng
  int Empty_Stack(Stack S)
  { return Empty_List(S);}
- Thêm phần tử vào ngăn xếp
  void Push(Elementtype X, Stack *S)
  { Insert_List (x, First (*S), &(*S));}
- Xóa phần tử ra khỏi ngăn xếp
  void Pop (Stack *S)
  { Delete_List (First (*S), &(*S));}

43

CÀI ĐẶT NGĂN XẾP BẰNG MẢNG (1)

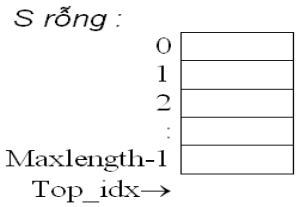


- Khai báo

```
#define … MaxLength //độ dài của mảng
typedef … ElementType;//kiểu phần tử của ngăn xếp
typedef struct {
    //Lưu nội dung của các phần tử
    ElementType Elements[MaxLength];
   int Top_idx; //giữ vị trí đỉnh ngăn xếp
}Stack;
Stack S;
```

44

KHỞI TẠO NGĂN XẾP RỖNG



– Khi ngăn xếp S rỗng ta cho đỉnh ngăn xếp được khởi tạo bằng Maxlength

```
void MakeNull_Stack(Stack *S)
{
      S->Top_idx=MaxLength;
}
```

45

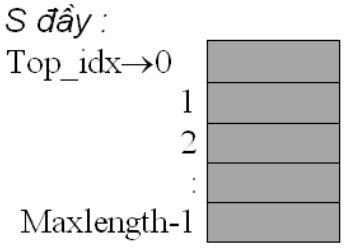
KIỂM TRA NGĂN XẾP RỖNG?

– Ta kiểm tra xem đỉnh ngăn xếp có bằng MaxLength không?

```
int Empty_Stack(Stack S)
{
      return S.Top_idx==MaxLength;
}
```

46

KIỂM TRA NGĂN XẾP ĐẦY?



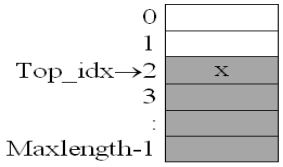
– Ta kiểm tra xem Top_idx có chỉ vào 0 hay không?

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

47

TRẢ VỀ PHẦN TỬ ĐẦU NGĂN XẾP

Ví dụ :



Kết quả của phép toán trên ngăn xếp là x

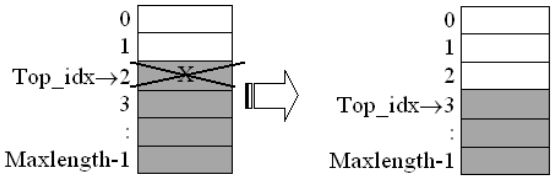
- Giải thuật :
  - Nếu ngăn xếp rỗng thì thông báo lỗi
  - Ngược lại, trả về giá trị được lưu trữ tại ô có chỉ số là Top_idx

```
ElementType Top(Stack S)
{
  if (!Empty_Stack(S))
       return S.Elements[S.Top_idx];
  else printf("Loi! Ngan xep rong");
  }
```

48

XÓA PHẦN TỬ TẠI ĐỈNH NGĂN XẾP

Ví dụ :



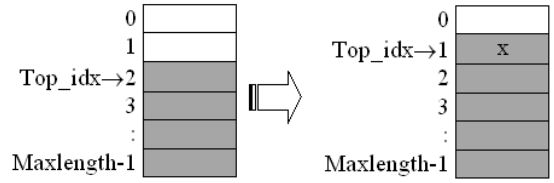
- Giải thuật :
  - Nếu ngăn xếp rỗng thì thông báo lỗi
  - Ngược lại, tăng Top_idx lên 1 đơn vị

```
void Pop(Stack *S)
{
  if (!Empty_Stack(*S))
        S->Top_idx=S->Top_idx+1;
  else printf("Loi! Ngan xep rong!");
}
```

49

THÊM PHẦN TỬ X VÀO NGĂN XẾP

Ví dụ :



- Giải thuật :
  - Nếu ngăn xếp đầy thì thông báo lỗi
  - Ngược lại, giảm Top_idx xuống 1 đơn vị rồi đưa giá trị x vào ô có chỉ số Top_idx

```
void Push(ElementType X, Stack *S){
  if (Full_Stack(*S))
        printf("Loi! Ngan xep day!");
  else{
        S->Top_idx=S->Top_idx-1;
        S->Elements[S->Top_idx]=X;
  }
```

50

BÀI TẬP

- Viết chương trình nhập vào 1 số nguyên n
- Chuyển số nguyên n sang số nhị phân (có sử dụng các phép toán trên ngăn xếp)

51

HÀNG ĐỢI (QUEUE)

- ĐỊNH NGHĨA
- CÁC PHÉP TOÁN
- CÀI ĐẶT HÀNG ĐỢI
  - DÙNG MẢNG DI CHUYỂN TỊNH TIẾN
  - DÙNG MẢNG VÒNG
  - DÙNG DSLK

52

ĐỊNH NGHĨA HÀNG ĐỢI

- Là một dạng 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)

53

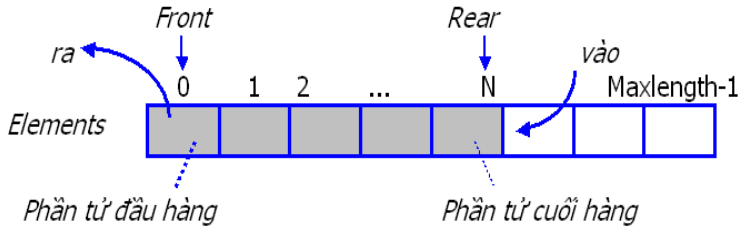
CÁC PHÉP TOÁN

Phép toán	Diễn giải
MAKENULL_QUEUE(*Q*)	Tạo một hàng đợi rỗng (Q)
EMPTY_QUEUE(*Q*)	Kiểm tra xem hàng đợi Q có rỗng không
FULL_QUEUE(*Q*)	KiÓm tra xem hµng ®îi Q cã ®Çy kh«ng
ENQUEUE(*X,Q*)	Thêm phần tử X vào cuối hàng đợi Q
DEQUEUE(*Q*)	Xóa phần tử tại đầu hàng đợi Q
FRONT(*Q*)	Trả về phần tử đầu tiên của hàng đợi Q

54

CÀI ĐẶT HÀNG BẰNG MẢNG DI CHUYỂN TỊNH TIẾN

- Mô hình



55

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;
Queue Q;
```

56

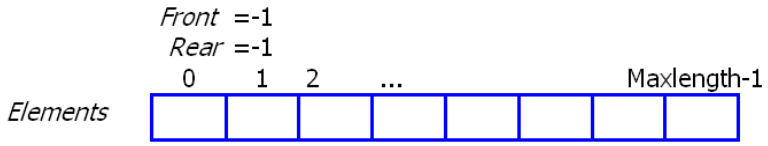
KHỞI TẠO HÀNG Q RỖNG

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

```
void MakeNull_Queue(Queue *Q)
{
    Q->Front=-1;
    Q->Rear=-1;
}
```

57

KIỂM TRA HÀNG RỖNG

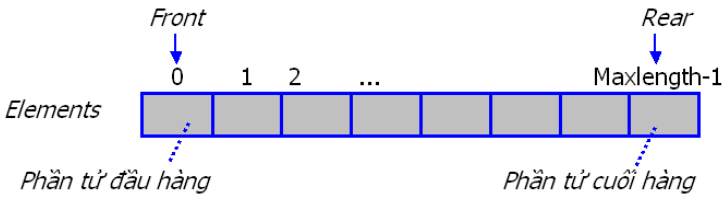


- Hàng rỗng khi front=-1

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

58

KIỂM TRA HÀNG ĐẦY

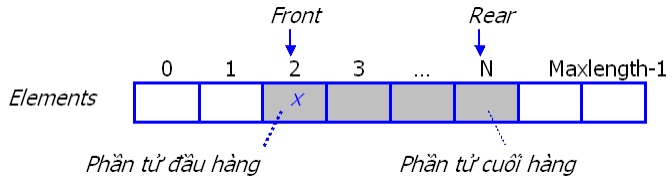


– Hàng đầy khi số phần tử hiện có trong hàng=Maxlength

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

59

TRẢ VỀ PHẦN TỬ ĐẦU HÀNG



Kết quả của phép toán trên là x

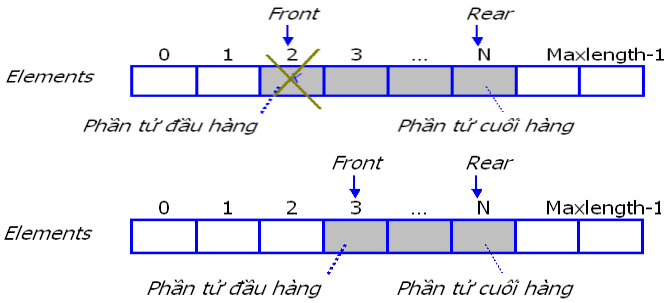
=>Giải thuật:

- Nếu hàng Q rỗng thì thông báo lỗi
- Ngược lại, trả về giá trị được lưu trữ tại ô có chỉ số là Front

```
ElementType Front(Queue Q){
    if Empty_Queue (Q)
       printf ("Hang rong");
    else  return Q.Elements[Q.Front];
}
```

60

XÓA MỘT PHẦN TỬ KHỎI HÀNG(1)



=>Giải thuật:

- Nếu hàng Q rỗng thì thông báo lỗi
- Ngược lại, tăng Front lên 1 đơn vị
  - Nếu Front>Rear tức hàng chỉ còn 1 phần tử thì khởi tạo lại hàng rỗng luôn

61

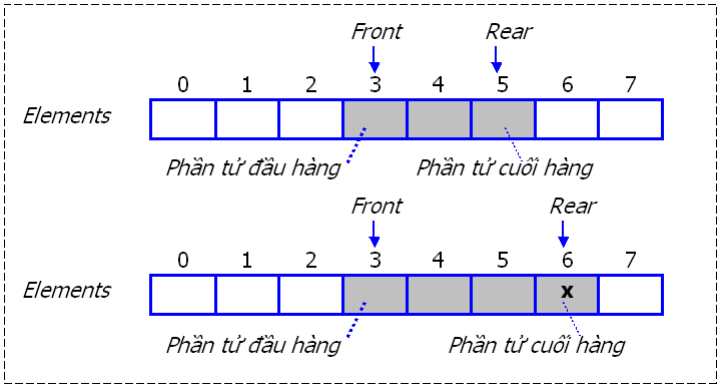
XÓA MỘT PHẦN TỬ KHỎI HÀNG(2)

```
void DeQueue(Queue *Q)
{ if (!Empty_Queue(*Q))
    {Q->Front=Q->Front+1;
     if (Q->Front>Q->Rear)
          MakeNull_Queue(Q);
     //Dat lai hang rong
     }else printf("Loi: Hang rong!");
}
```

62

THÊM MỘT PHẦN TỬ VÀO HÀNG(1)

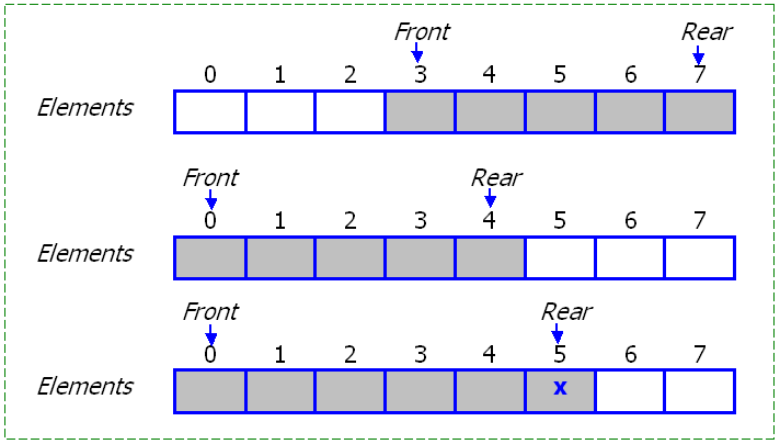
- Trường hợp bình thường



63

THÊM MỘT PHẦN TỬ VÀO HÀNG(2)

– Trường hợp hàng bị tràn



64

THÊM MỘT PHẦN TỬ VÀO HÀNG(3)

=>Giải thuật:

- Nếu hàng đầy thì thông báo lỗi
- Ngược lại, nếu hàng tràn thì phải tịnh tiến tất cả phần tử lên Front-1 vị trí
- Tăng Rear 1 đơn vị và đưa giá trị x vào ô có chỉ số Rear mới này

65

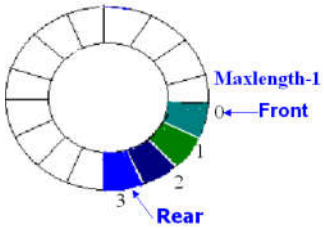
THÊM MỘT PHẦN TỬ VÀO HÀNG(4)

```
void EnQueue(ElementType X,Queue *Q)
{  if (!Full_Queue(*Q))
  { if (Empty_Queue(*Q)) Q->Front=0;
    if (Q->Rear==MaxLength-1)
     {//Di chuyen tinh tien ra truoc Front -1 vi tri
         for(int i=Q->Front;i<=Q->Rear;i++)
               Q->Elements[i-Q->Front]=Q->Elements[i];
       //Xac dinh vi tri Rear moi
        Q->Rear=MaxLength - Q->Front-1;
        Q->Front=0;
       }
      //Tang Rear de luu noi dung moi
      Q->Rear=Q->Rear+1;
      Q->Element[Q->Rear]=X;
  }
  else printf("Loi: Hang day!");}
```

66

CÀI ĐẶT HÀNG BẰNG MẢNG VÒNG

- Mô hình



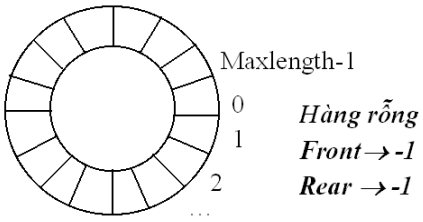
- Khai báo

```
#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;
```

Queue Q;

67

KHỞI TẠO HÀNG RỖNG

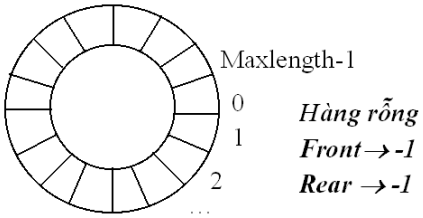


- Front và Rear không trỏ đến vị trí hợp lệ nào
- Ta cho Front=Rear=-1

```
void MakeNull_Queue(Queue *Q)
    {
        Q->Front=-1;
        Q->Rear=-1;
    }
```

68

KIỂM TRA HÀNG RỖNG

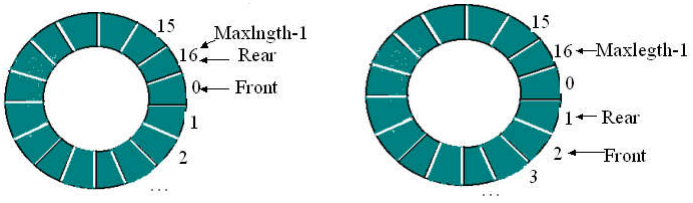



```
int Empty_Queue(Queue Q){
    return Q.Front==-1;
}
```

69

KIỂM TRA HÀNG ĐẦY

- Ví dụ

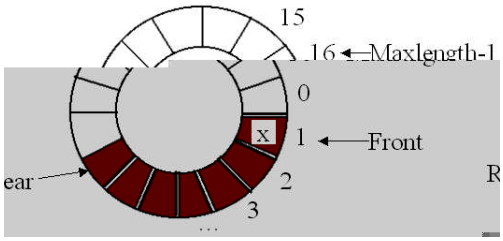


- Hàng đầy khi số phần tử hiện có trong hàng bằng Maxlength

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

70

LẤY GIÁ TRỊ PHẦN TỬ ĐẦU HÀNG



=>Giải thuật

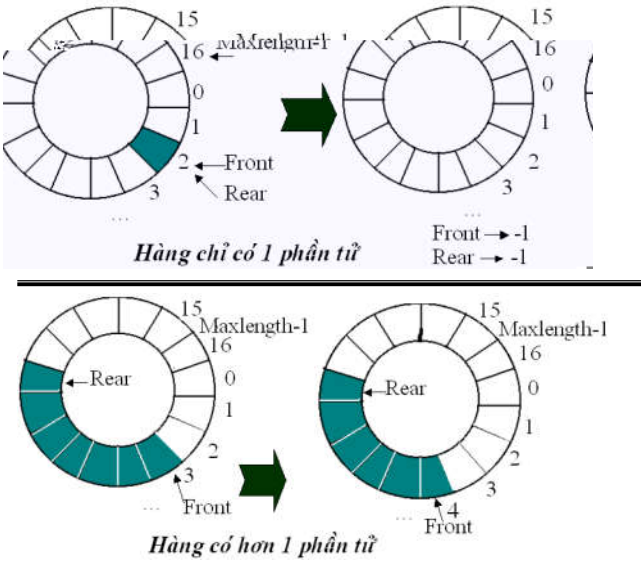
- Nếu hàng Q rỗng thì thông báo lỗi
- Ngược lại, trả về giá trị được lưu trữ tại ô có chỉ số là Front

```
ElementType Front(Queue Q)
 { if (!Empty_Queue (Q))
       return Q.Element[Q.Front];
 }
```

71

XÓA PHẦN TỬ ĐẦU HÀNG(1)

- Các trường hợp có thể:



72

XÓA PHẦN TỬ ĐẦU HÀNG(2)

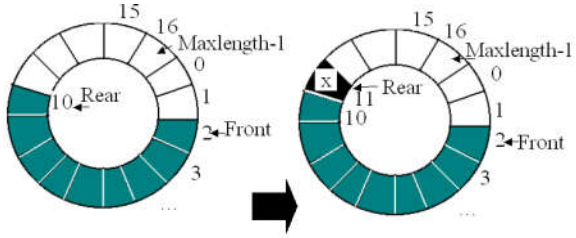
- Giải thuật :
  - Nếu hàng Q rỗng thì thông báo lỗi
  - Ngược lại:
    - Nếu Front=Rear tức hàng chỉ còn 1 phần tử thì khởi tạo lại hàng rỗng
    - Ngược lại, thay đổi giá trị cho Front

```
void DeQueue(Queue *Q){
   if (!Empty_Queue(*Q)){
   //Nếu hàng chỉ chứa một phần tử thì khởi tạo
  hàng lại
  if (Q->Front==Q->Rear) MakeNull_Queue(Q);
  else Q->Front=(Q->Front+1) % MaxLength;
  //tăng Front lên 1 đơn vị
   }else printf("Loi: Hang rong!");
}
```

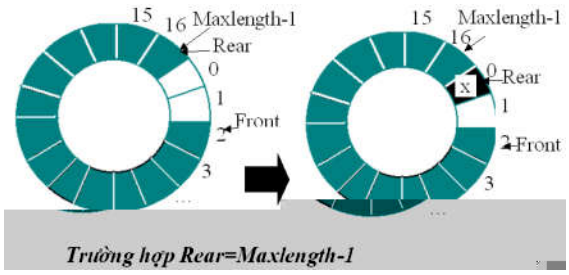
73

THÊM PHẦN TỬ X VÀO HÀNG Q(1)

- Các trường hợp có thể:



Trường hợp Rear<Maxlength-1



Trường hợp Rear=Maxlength-1

74

THÊM PHẦN TỬ X VÀO HÀNG Q(2)

- Giải thuật :
  - Nếu hàng đầy thì thông báo lỗi
  - Ngược lại, thay đổi giá trị Rear và đưa giá trị x vào ô có chỉ số Rear mới này

```
void EnQueue(ElementType X,Queue *Q){
  if (!Full_Queue(*Q)){
   if (Empty_Queue(*Q)) Q->Front=0;
       Q->Rear=(Q->Rear+1) % MaxLength;
       Q->Elements[Q->Rear]=X;
   }else printf("Loi: Hang day!");
}
```

75

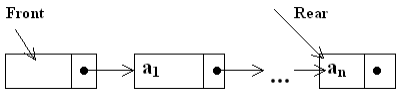
BÀI TẬP

- 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

76

CÀI ĐẶT HÀNG BẰNG DSLK

- Mô hình



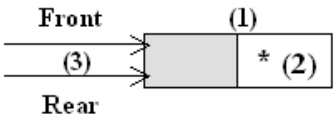
  - Dùng 2 con trỏ Front và Rear để chỉ tới phần tử đầu hàng và cuối hàng

- 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;
Queue Q;
```

77

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;
}
```

78

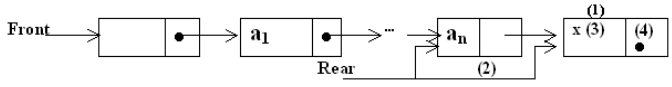
KIỂM TRA HÀNG Q RỖNG

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

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

79

THÊM MỘT PHẦN TỬ X VÀO HÀNG Q



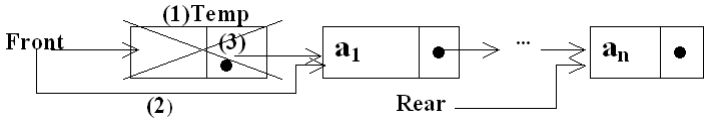
=>Giải thuật:

- 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 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;
}
```

80

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

81

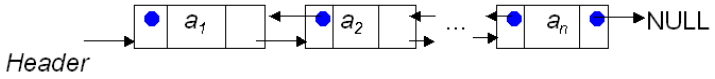
CÁC ỨNG DỤNG CỦA NGĂN XẾP VÀ HÀNG ĐỢI

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

82

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 DoubleList;
```

83

DANH SÁCH RỖNG

- Tạo danh sách rỗng

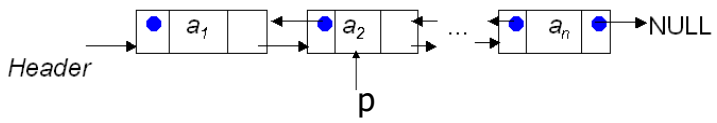
```
void MakeNull_List (DoubleList *DL)
 {
  (*DL)= NULL;
 }
```

- Kiểm tra danh sách rỗng

```
int Empty (DoubleList DL)
 {
  return (DL==NULL);
 }
```

84

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



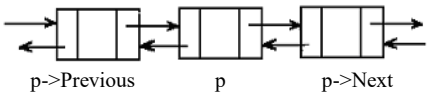
- =>Vị trí của một phần tử là con trỏ trỏ vào ngay chính phần tử đó

- ```
  ElementType Retrieve (Position P, DoubleList DL)
  {
      return  P->Element;
  }
  ```

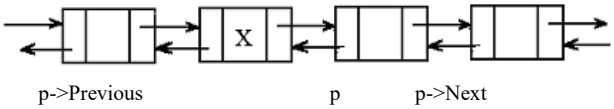
85

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

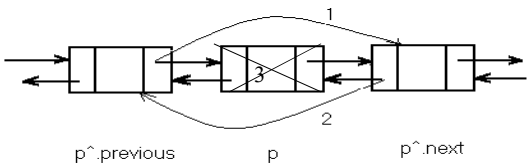
86
  ```

THÊM MỘT PHẦN TỬ VÀO DANH SÁCH (2)

```
• void Insert_List (ElementType X,Position p,
  DoubleList *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;
      }
  }
```

87

XÓA MỘT PHẦN TỬ RA KHỎI DANH SÁCH




```
void Delete_List (Position p, DoubleList *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);
   }
}
```

88

ƯU ĐIỂM CỦA DSLK KÉP

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

89

Hết chương

90