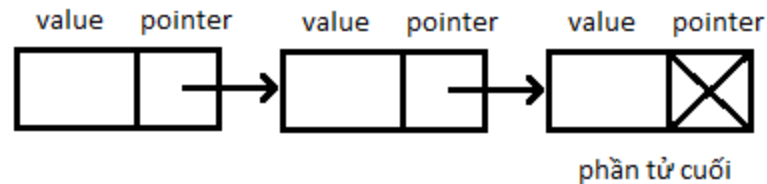
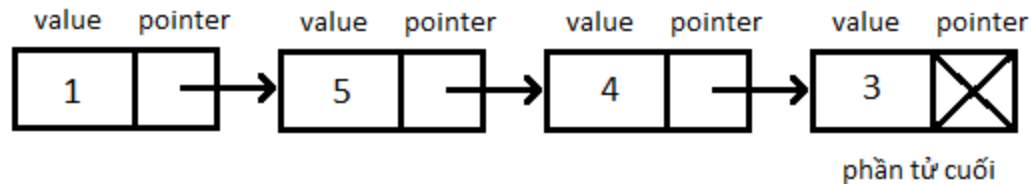


CHƯƠNG 8: Lists, Stacks, Queues

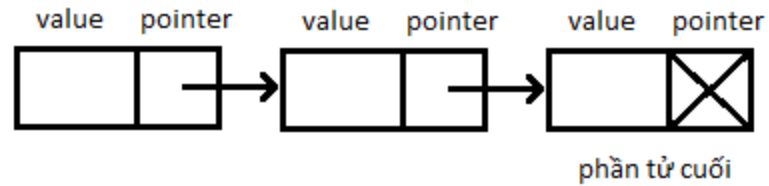
8.1 Lists (danh sách):



Ví dụ : Danh sách các phần tử 1, 5, 4, 3.



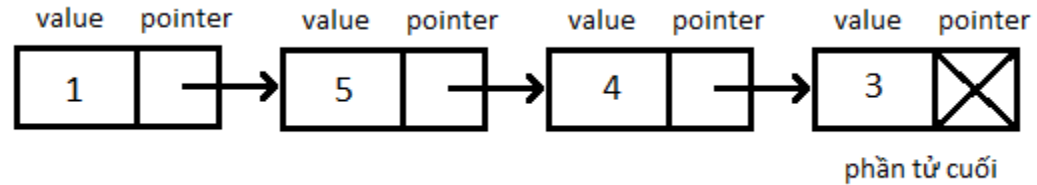
8.1 Lists (danh sách):



8.1.1 Khai báo :

```
struct LIST {  
    T value;  
    LIST *pointer ;  
};
```

Ví dụ :



```
struct LIST {  
    int value;
```

```
    LIST *pointer ;
```

```
};
```

```
void main( )
```

```
{ LIST *dx, *p;
```

```
    1. dx = NULL;
```

```
    2. p = (LIST*)malloc(sizeof(LIST));
```

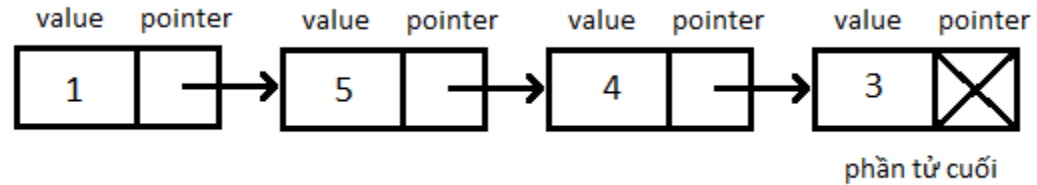
```
    3. p -> value = 3; p -> pointer = dx; dx = p;
```

```
    4. p = (LIST*)malloc(sizeof(LIST));
```

```
    5. p -> value = 4; p -> pointer = dx; dx = p;
```

```
}
```

Ví dụ :



```
struct LIST {  
    int value;
```

```
    LIST *pointer ;
```

```
};
```

```
void main( )
```

```
{
```

```
...
```

```
6. p = (LIST*)malloc(sizeof(LIST));
```

```
7. p -> value = 5; p -> pointer = dx; dx = p;
```

```
8. p = (LIST*)malloc(sizeof(LIST));
```

```
9. p -> value = 1; p -> pointer = dx; dx = p;
```

```
}
```

8.1.2 Các thao tác :

1. Tạo danh sách :

Thuật toán :

dx = NULL;

while (*tiếp tục*) {

 Nhập x;

 p = (LIST*)malloc(sizeof(LIST));

 p -> value = x ; p -> pointer = dx; dx = p;

}

1. **Tạo danh sách** : Chuyển thuật toán thành chương trình con

```
void TaoDS(LIST **L)
```

```
{
```

```
    LIST* p;    int i, x;
```

```
    *L = NULL; i=1;
```

```
    while (i <= 3) {
```

```
        printf("Nhap value :"); scanf("%d", &x);
```

```
        p = (LIST*)malloc(sizeof(LIST));
```

```
        p -> value = x ; p -> pointer = *L; *L = p;
```

```
        i++;
```

```
    }// end while
```

```
}
```

```
int main(int argc, char* argv[])
```

```
{ LIST *dx, *p;
```

```
    TaoDS(&dx); ... }
```

2. Duyệt danh sách :

```
p=dx;  
while (p!=NULL) {  
    Xử lý p->value; // Ví dụ viết p->value  
    p=p->pointer;  
}
```

Chương trình con :

```
void DuyệtDS (LIST *L)  
{ LIST *p;  
    p=L;  
    while (p!=NULL) {  
        Xử lý p->value; // Ví dụ viết p->value  
        p=p->pointer;  
    }  
}
```

3. Giải phóng danh sách :

```
void GiaiPhongDS(LIST **L)
```

```
{    LIST *p, *q;  
    p=*L;  
    while (p!=NULL) {  
        q=p->pointer;  
        free(p);  
        p=q;  
    }  
    *L=NULL;  
}
```


4. Khởi tạo danh sách rỗng :

```
void KhoiTaoDS(LIST **L)
{
    *L=NULL;
}
```

5. Thêm vào đầu danh sách :

LIST *p;

p = (LIST*)malloc(sizeof(LIST));

p->value = Giá trị ;

p->pointer= dx;

dx = p;

```
int ThemDauDS(LIST **L, int x)
{
    LIST *p;
    p = (LIST*)malloc(sizeof(LIST));
    if(p==NULL) return 0;
    p->value = x;
    p->pointer=*L;
    *L = p;
    return 1;
}
```

6. Xóa phần tử đầu danh sách :

```
void XoaDauDS(LIST **L)
```

```
{  
    LIST *p;  
    p = *L;  
    if (p!=NULL) {  
        *L = p->pointer; // *L = (*L)->pointer;  
        free(p);  
    }  
}
```

7. Phần tử đầu danh sách :

```
int PhanTuDauDS(LIST *L, int *x)
{
    if (L!=NULL) {
        *x=L->value;
        return 1;
    }
    return 0;
}
```

Ví dụ :

```
int main()
```

```
{ LIST *dx;
```

```
    KhoiTaoDS(&dx);
```

```
    ThemDauDS(&dx, 1);
```

```
    ThemDauDS(&dx, 2);
```

```
    ThemDauDS(&dx, 3);
```

```
    ThemDauDS(&dx, 4);
```

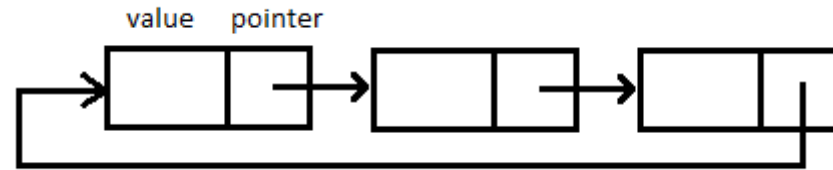
```
    DuyetDS(dx);
```

```
    XoaDauDS(&dx);
```

```
    DuyetDS(dx);
```

```
}
```

8.2 Danh sách vòng:



- Danh sách vòng không có phần tử đầu hay phần tử cuối. Tuy nhiên, ta chỉ định một phân tử làm phân tử mốc (có địa chỉ chứa trong dx),
- Các thao tác tương tự như Danh sách vừa học (Bài tập).

8.3 Stack :

- Stack là một tập hợp có tính chất *vào trước ra sau* (*First-In-Last-Out (FILO)*) .
- Sử dụng danh sách làm Stack.

Các thao tác :

- **init(S)** : Khởi tạo stack rỗng,
- **push(S, x)** : Thêm x vào (đỉnh) stack S. Hàm push trả về giá trị 0 nếu không thể thêm x vào S (S bị tràn), ngược lại trả về 1,
- **pop(S, &x)** : Lấy phần tử ở đỉnh S chứa vào x. Hàm pop trả về giá trị 0 nếu S rỗng, ngược lại trả về 1.

8.3 Stack :

- **init :**

```
typedef LIST *STACK;
```

```
void initS(STACK *S)
```

```
{
```

```
    *S=NULL;
```

```
}
```

- **push** : Thêm x vào (đỉnh) stack S. Hàm push trả về giá trị 0 nếu không thể thêm x vào S (S bị tràn), ngược lại trả về 1,

```
int pushS(STACK *S, int x)
```

```
{
```

```
    return ThemDauDS(S,x);
```

```
}
```

➤ **int ThemDauDS(LIST **L, **int** x)**

8.3 Stack :

typedef LIST *STACK;

- pop :


int popS(STACK *S, int *x)

{

if(*S!=NULL) {
 *x = (*S)->value;
 XoaDauDS(S);
 return 1;
}

return 0;

}



```
if (PhanTuDauDS(*S,x))  
{  
    XoaDauDS(S);  
    return 1;  
}
```

8.3 Stack :

Ví dụ :

main()

```
{    STACK stack ;
    int x;
    initS(&stack);
    pushS(&stack, 1);
    pushS(&stack, 2);
    pushS(&stack, 3);
    printf("Cac phan tu :\n");
    while (popS(&stack, &x)==1) {
        printf("%d ,",x);
    }
}
```

Dùng Stack khử đệ qui:

Ví dụ : Tính giai thừa n!

Dạng toán : $GT(n) = GT(n-1) * n$

Dạng hàm tính giai thừa dạng đệ qui :

```
int GT(int n)
{
    if (n == 1) return 1;
    else return GT(n-1)*n;
}
```

Dùng Stack khử đệ qui:

Ví dụ 1 : Tính giai thừa n!

Khử đệ qui :

STACK stack ;

int KQ, n;

n=3;

init(&stack);

while (n > 1)

{

push(&stack, n); n=n-1;

}

KQ = n ;

while (pop(&stack, &n)==1)

KQ=KQ*n;

Dạng hàm tính giai thừa dạng đệ qui :

```
int GT(int n)
```

```
{
```

```
    if (n == 1) return 1;
```

```
    else return GT(n-1)*n;
```

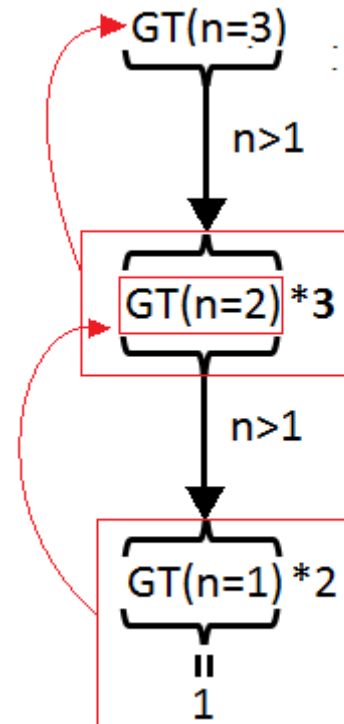
```
}
```

```
void main()
```

```
{
```

```
    ketqua=GT(3);
```

```
}
```



Dùng Stack khử đệ qui:

Ví dụ 2 : $F(n) = F(n-2) + F(n-1)$, $F(0)=1$, $F(1)=2$, $n \geq 0$

Khử đệ qui :

STACK stack ;

F=0; n=4;

init(&stack);

push(&stack,0);

do

{

while (n > 1){

 push(&stack, n-1);

 n=n-2;

}

if(n==0)F=F+1; if(n==1)F=F+2;

while (pop(&stack, &n)==1) { if (n==1) F=F+2;else break;}

}while (n!=0);

Dạng hàm tính giai thừa dạng đệ qui :

```
int F(int n)
```

```
{
```

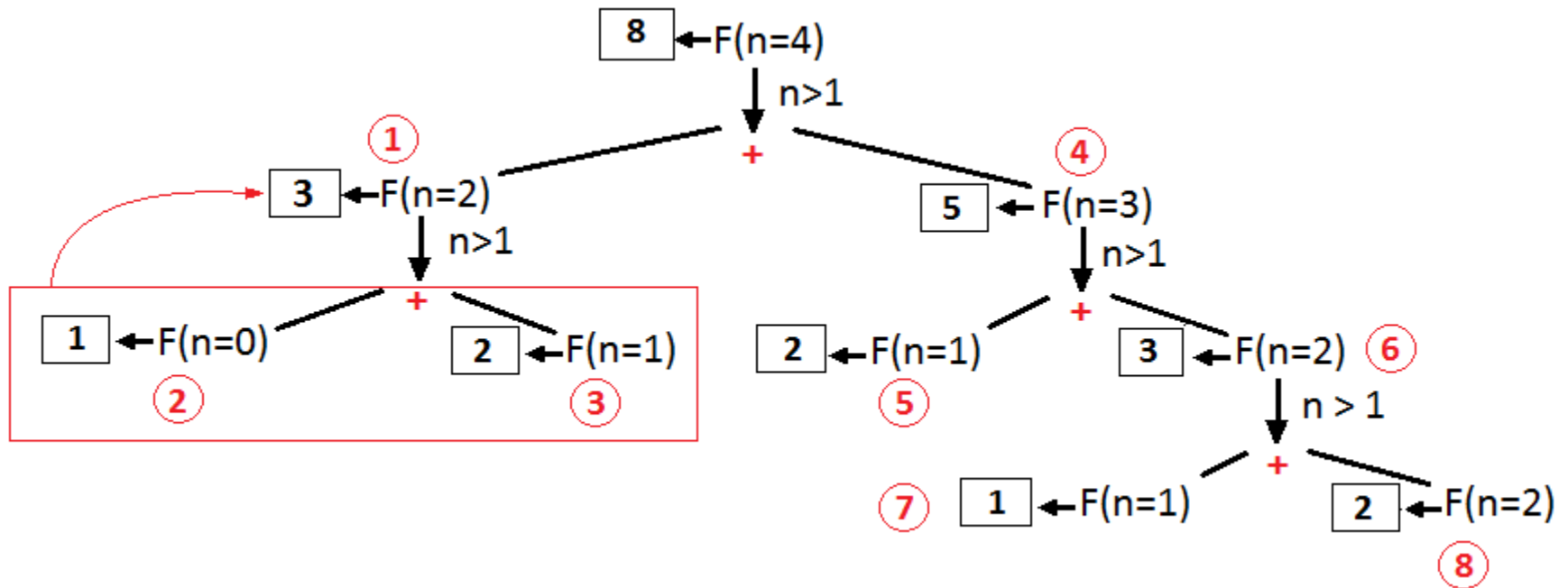
```
    if (n == 0) return 1;
```

```
    if (n == 1) return 2;
```

```
    return F(n-2) + F(n-1);
```

```
}
```

Cây đệ qui $F(n=4)$

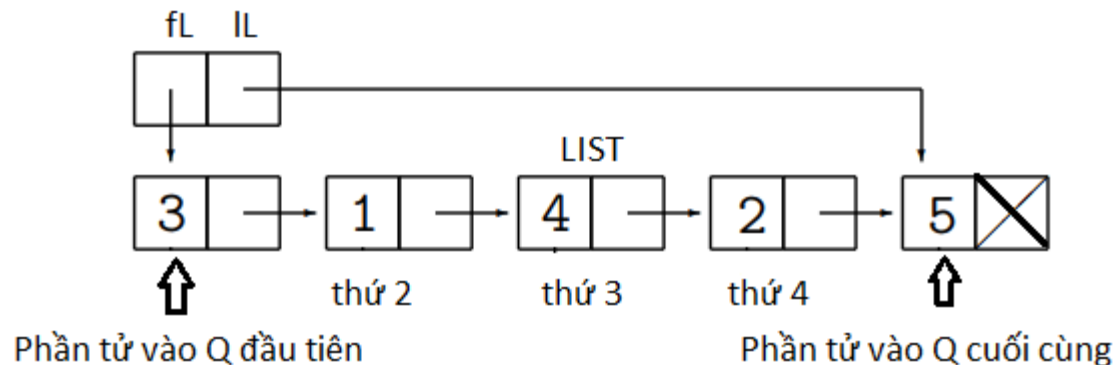


8.4 Queue :

- Queue là một tập hợp có tính chất *vào trước ra trước (First-In-First-Out (FIFO))*

Các thao tác :

- **initQ(Q)** : Khởi tạo queue Q rỗng,
- **pushQ(Q, x)** : Thêm x vào queue Q. Hàm push trả về giá trị 0 nếu không thể thêm x vào Q (Q bị tràn), ngược lại trả về 1,
- **popQ(Q, &x)** : Lấy phần tử ở đỉnh Q chứa vào x. Hàm pop trả về giá trị 0 nếu Q rỗng, ngược lại trả về 1.



8.4 Queue :

```
struct QUEUE {  
    LIST *fL, *lL;  
};
```

Các thao tác :

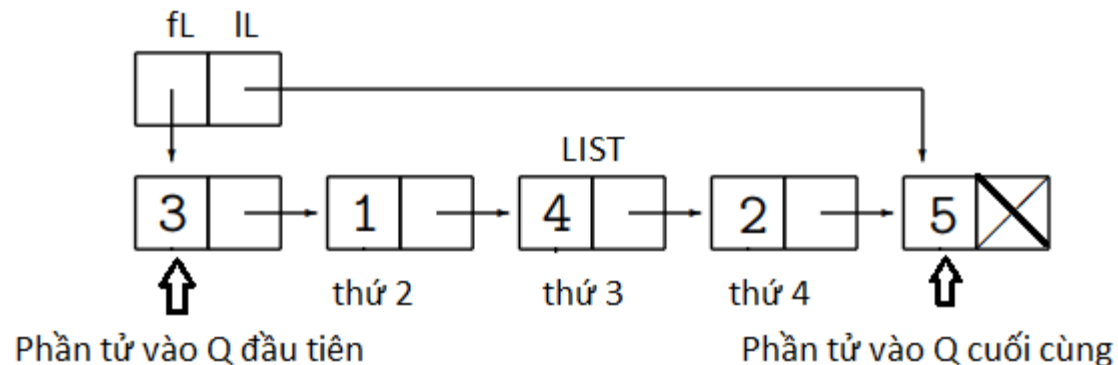
- **initQ(Q)** : Khởi tạo queue Q rỗng,

```
void initQ( QUEUE *Q)
```

```
{
```

...

```
}
```



8.4 Queue :

```
struct QUEUE {  
    LIST *fL, *lL;  
};
```

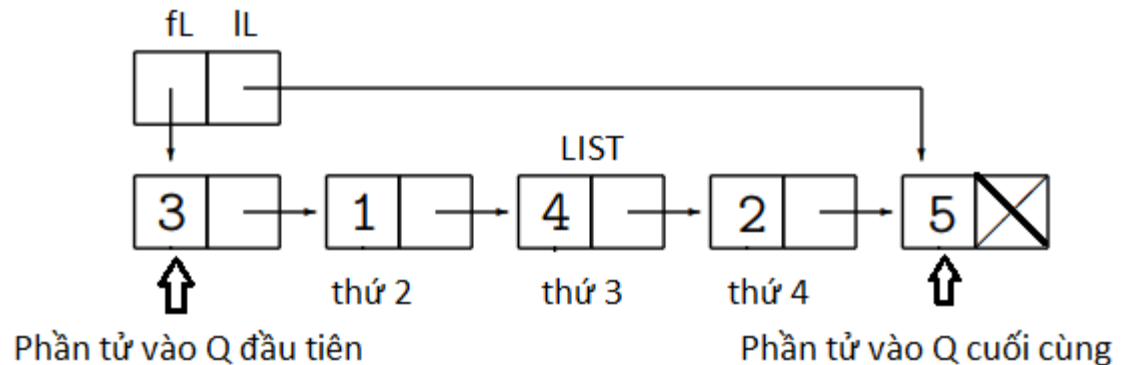
- **pushQ(Q, x)** : Thêm x vào queue Q. Hàm push trả về giá trị 0 nếu không thể thêm x vào Q (Q bị tràn), ngược lại trả về 1,

```
int pushQ( QUEUE *Q, int x)
```

```
{
```

...

```
}
```



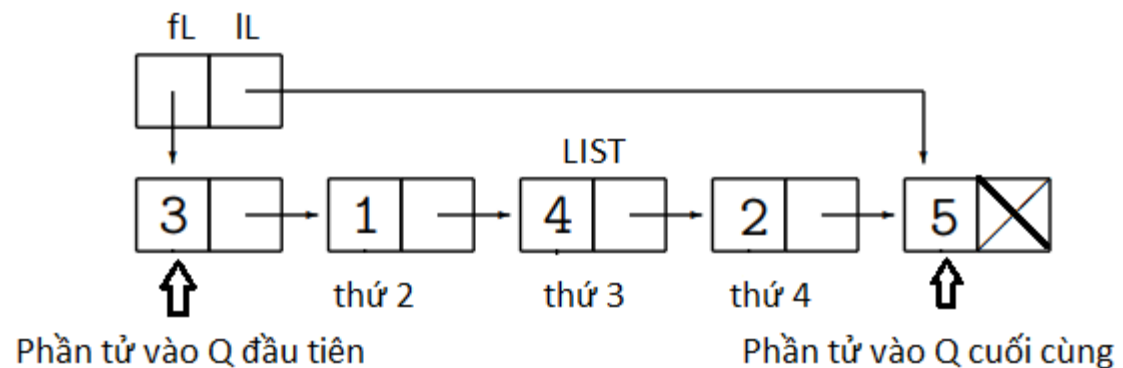
8.4 Queue :

```
struct QUEUE {  
    LIST *fL, *lL;  
};
```

- **popQ(Q, x)** : Lấy phần tử ở đỉnh Q chứa vào x. Hàm pop trả về giá trị 0 nếu Q rỗng, ngược lại trả về 1.

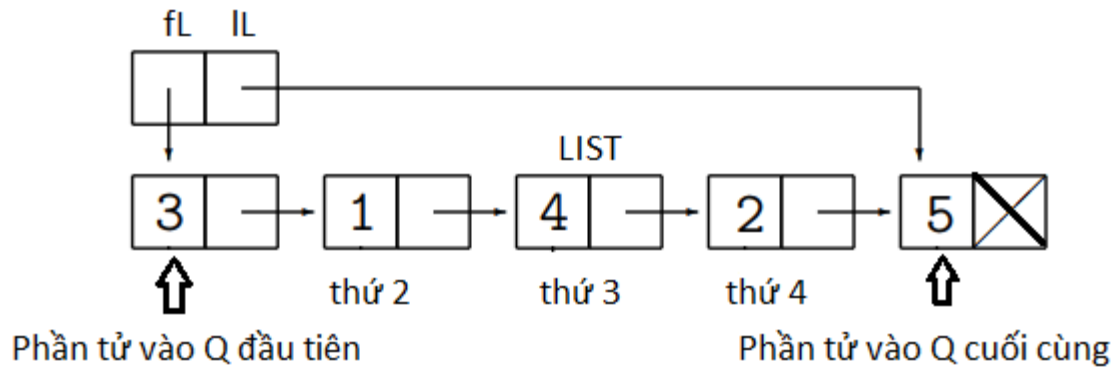
```
int popQ( QUEUE *Q, int *x)
```

```
{  
  
...  
}
```



8.4 Queue :

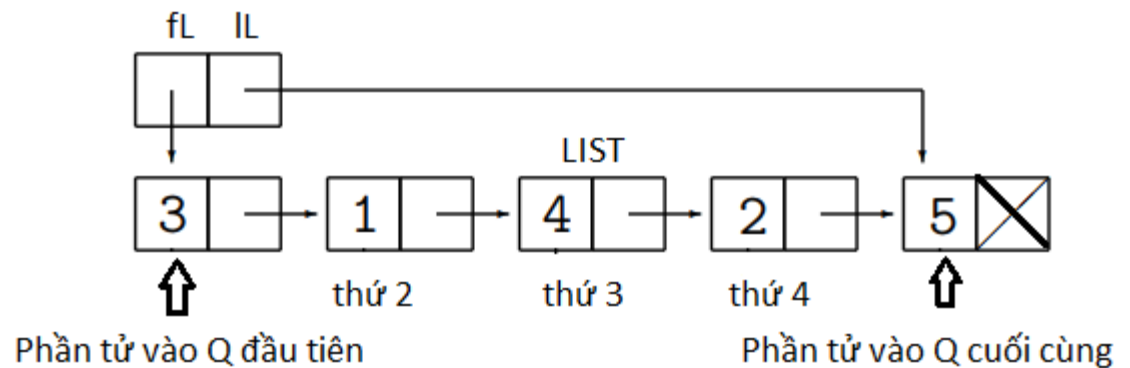
```
void initQ( QUEUE *Q)
{
    Q→fL = NULL;
    Q→lL = NULL;
}
```



```

int pushQ( QUEUE *Q, int x)
{
    LIST *p;
    p = (LIST*)malloc(sizeof(LIST));
    if (p==NULL) return 0;
    if(Q→fL==NULL){ p→value = x; p→pointer=Q→fL;
                    Q→fL = p; Q→lL = p;
                    }
    else {   p→value = x; p → pointer = NULL;
            (Q→lL)→pointer=p;
            Q→lL= p;
        }
    return 1;
}

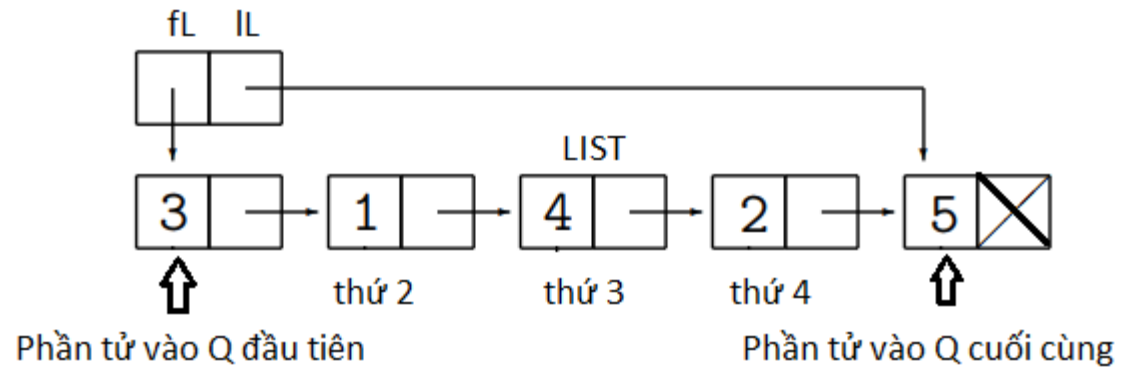
```



```

int popQ(Queue *Q, int *x)
{
    LIST *p;
    p = Q->fL;
    if (p!=NULL) {
        *x=p->value;
        Q->fL = p->pointer;
        if(Q->fL==NULL) Q->IL=NULL;
        free(p);
        return 1;
    }
    return 0;
}

```



```
int main(int argc, char* argv[])
{
    QUEUE Queue; int x;

    initQ(&Queue);
    pushQ(&Queue,1);
    pushQ(&Queue,2);
    pushQ(&Queue,3);
    while(pop(&Queue,&x)==1) printf("%x\n", x);
    pushQ(&Queue,4);
    pushQ(&Queue,5);
    pushQ(&Queue,6);
    while(popQ(&Queue,&x)) printf("%x\n", x);

    return 0;
}
```