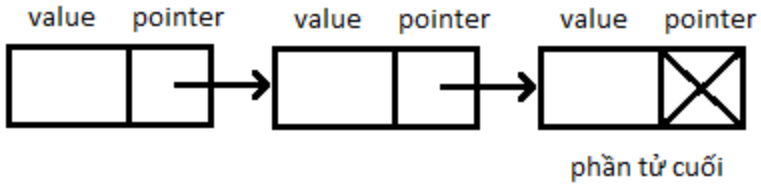
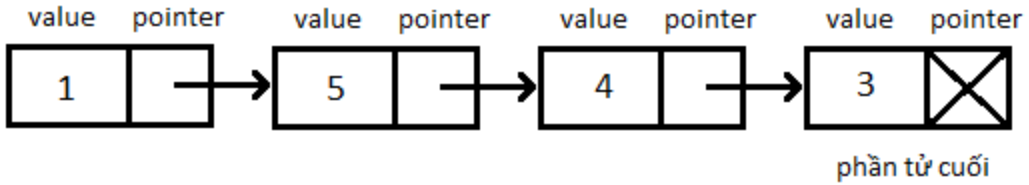
CHƯƠNG 8: Lists, Stacks, Queues

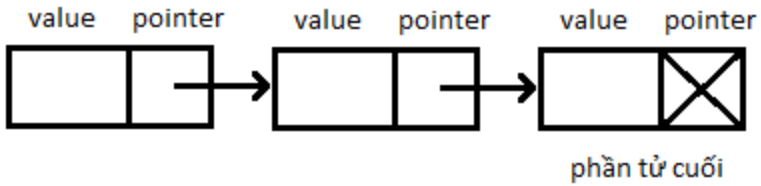
8.1 Lists (danh sách):



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



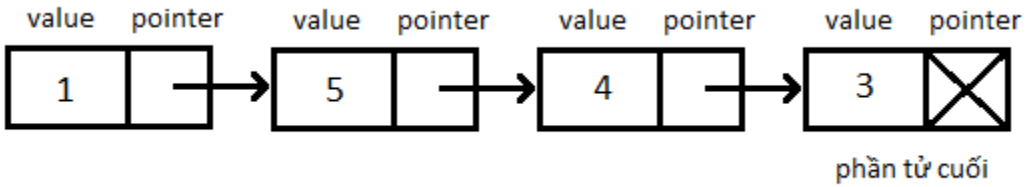
8.1 Lists (danh sách):



8.1.1 Khai báo :

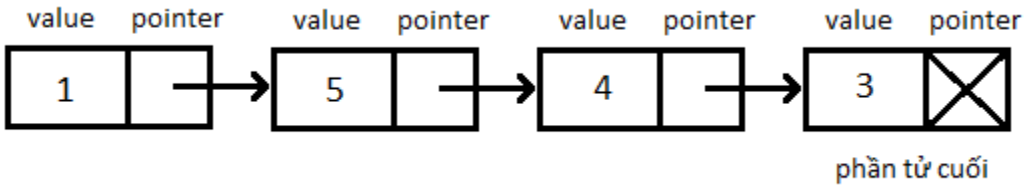
```
struct  LIST {
        T   value;
        LIST  *pointer ;
};
```

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)
{
```

```
}
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;
}
```

Bài tập : Chuyển đoạn chương trình trên thành chương trình con.

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 initL(LIST **L)
{
        *L=NULL;
}
```

5. Thêm vào đầu danh sách :

LIST *p;

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

p->value = Giá trị ;

p->pointer= dx;

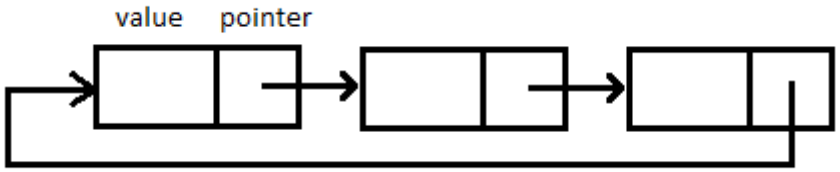
dx = p;

```
int ThemDau(LIST **L, int x)
{
}
```

6. Xóa phần tử đầu danh sách :

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

8.2 Danh sách vòng:



- Danh sách vòng không có phần tử đầu hay phân tử cuối. Tuy nhiên ta chỉ định một phân tử là phần 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.

CHƯƠNG 8: Lists, Stacks, Queues

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 init(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 push(STACK *S, int x)
{
        return ThemDau(S,x); // ThemDau
}
```

- **int ThemDau(LIST **L, int x)**

8.3 Stack :

typedef LIST *STACK;

- pop :

```
int pop(STACK *S, int *x)
{
        if(*S!=NULL) {
                *x = (*S)->value;
                XoaDau(S);
                return 1;
        }

        return 0;
}
```

8.3 Stack :

Ví dụ :

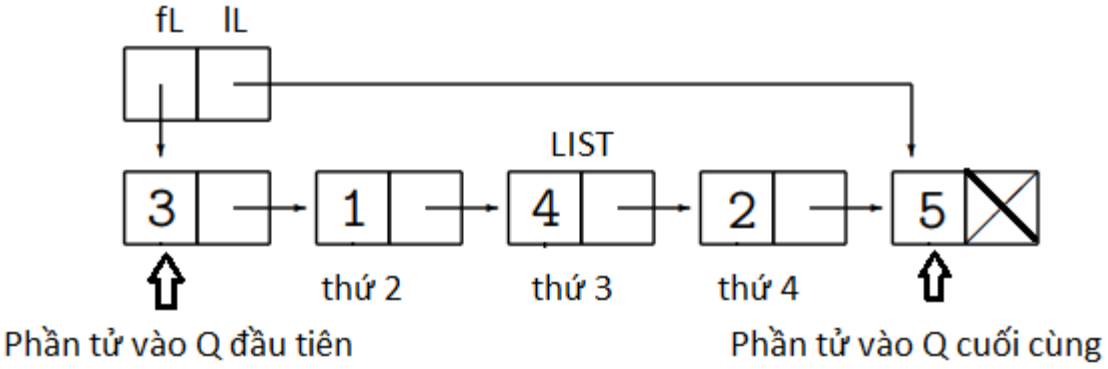
```
main( )
{       STACK stack ;
        int x;
        init(&stack);
        push(&stack, 1);
        push(&stack, 2);
        push(&stack, 3);
        printf("Cac phan tu :\n");
        while (pop(&stack, &x)==1) {
                printf("%d ,",x);
        }
}
```

8.4 Queue :

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

Các thao tác :

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



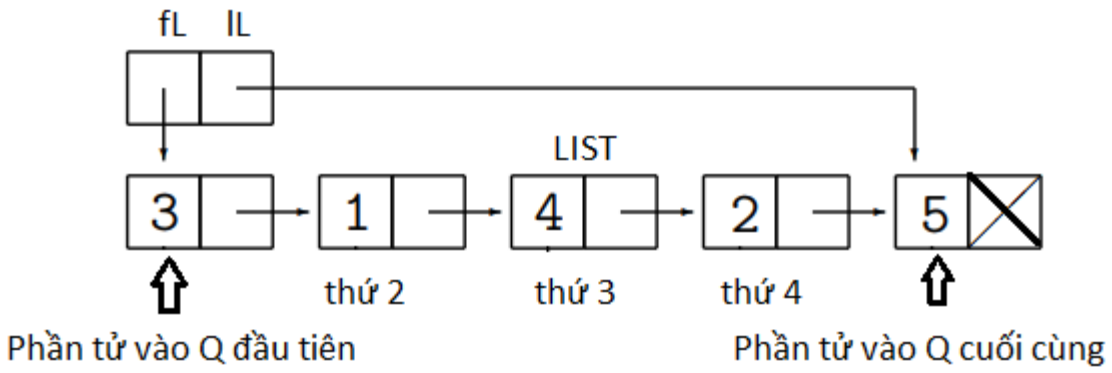
8.4 Queue :

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

Các thao tác :

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

```
void initQ( QUEUE *Q)
{
        . . .
}
```

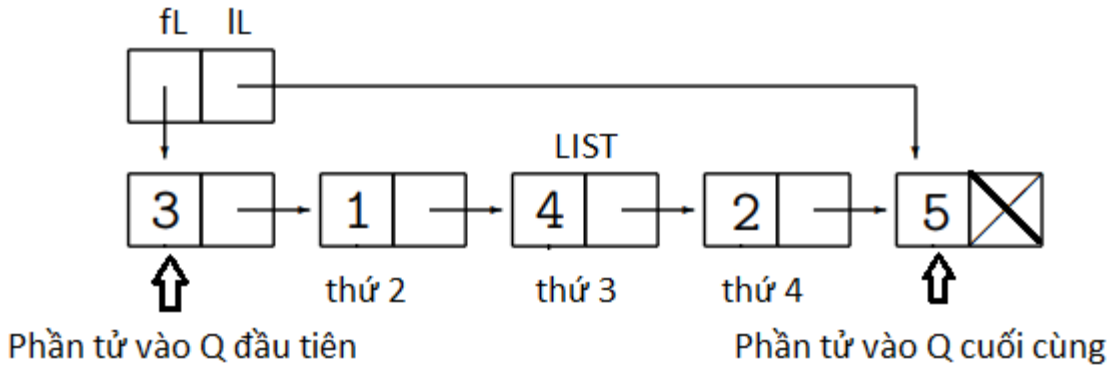


8.4 Queue :

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

- **push(Q, x)** : Thêm x vào (đỉnh) 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 push ( QUEUE *Q, int x)
{
        . . .
}
```

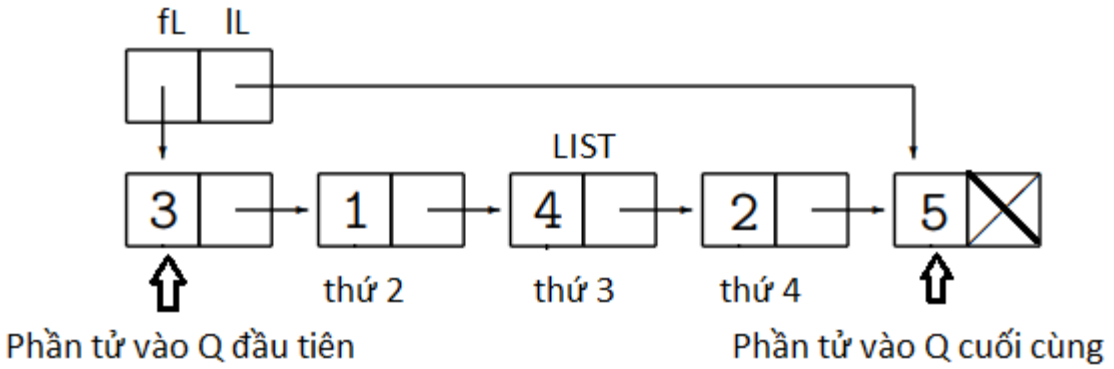


8.4 Queue :

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

- **pop(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 pop ( QUEUE *Q, int *x)
{
        . . .
}
```

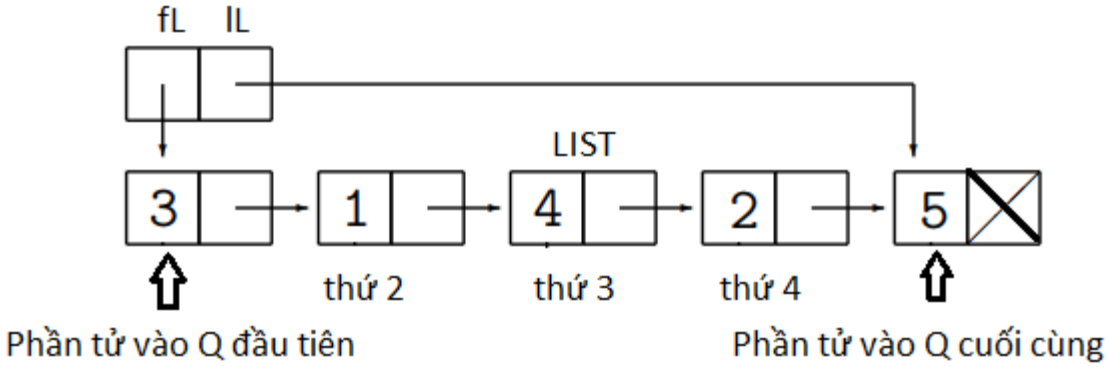


8.4 Queue :

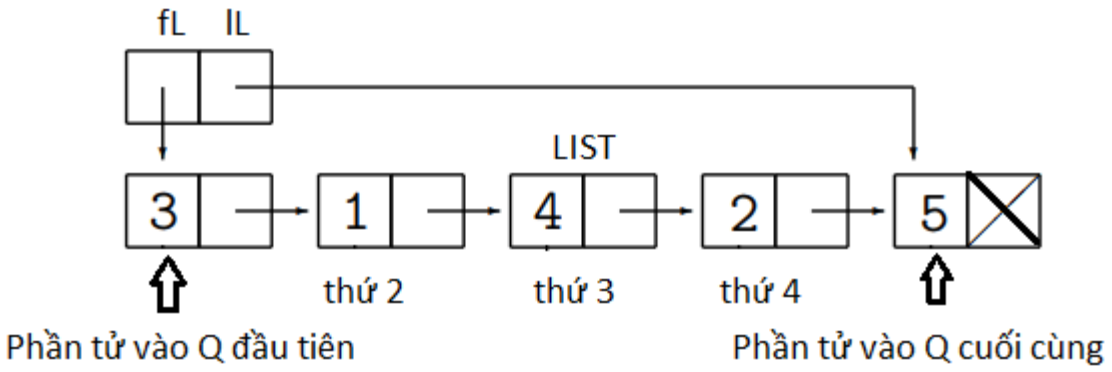
void initQ(QUEUE *Q)

{

}

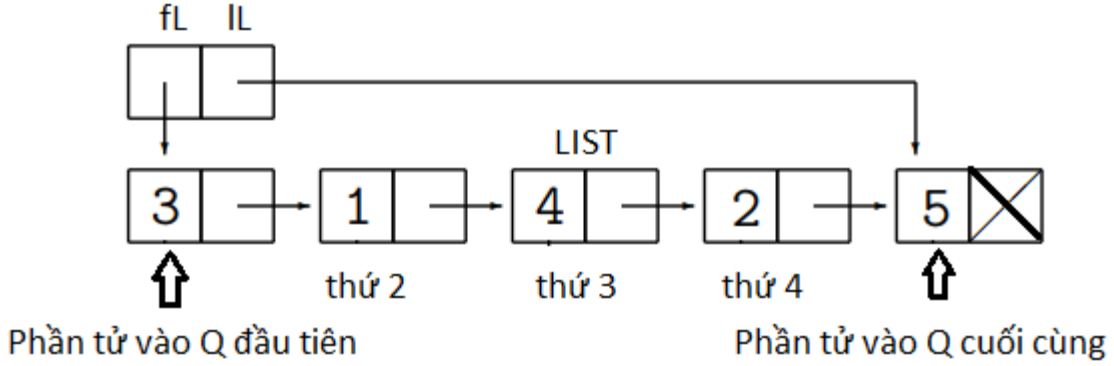


```
int push ( QUEUE *Q, int x)
{       LIST *p;
        p = (LIST*)malloc(sizeof(LIST));
        if (p==NULL) return 0;
        if(Q->fL==NULL){ p->value = x; p->pointer=Q->fL;
                                Q->fL = p; Q->lL = p;
                              }
        else {
        }
        return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



```
int pop(QUEUE *Q, int *x)
{LIST *p;
p = Q->fL;
if (p!=NULL) {
```

```
            return 1;
        }
        return 0;
}
```



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

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

    return 0;
}
```