

# Chương 1

## CẤU TRÚC DỮ LIỆU CƠ BẢN

1

## Mở đầu

**Kiến thức** cần thiết khi tìm hiểu về chương 1, một số CTDL cơ bản:




- CTDL là gì? Giải thuật là gì? Kiểu dữ liệu cơ bản, dữ liệu lưu trữ trong máy tính; Kiểu dữ liệu trong ngôn ngữ C++;
- Các kiến thức về cơ sở lập trình & kỹ thuật lập trình.

**Kỹ năng** cần có:

- Có thể sử dụng Visual Studio 2010
- Có thể lập trình C++

2

## Mục tiêu dạy học

-  Cung cấp *kiến thức* về các CTDL và các thuật toán trên **danh sách đặc**, **danh sách liên kết**, và **danh sách hạn chế** (stack, queue).
-  Rèn luyện và nâng cao các *kỹ năng* lập trình, áp dụng các CTDL và các thuật toán trên danh sách đặc và danh sách liên kết, danh sách hạn chế, giải quyết các bài toán ứng dụng
-  Có khả năng sử dụng cấu trúc dữ liệu danh sách phù hợp, giải quyết các bài toán ứng dụng.

3

## Nội dung chính

- 1.1 Danh sách đặc
- 1.2 Danh sách liên kết
  - Danh sách liên kết đơn
  - Danh sách liên kết kép
- 1.3 Danh sách hạn chế
  - Ngăn xếp
  - Hàng đợi
- 1.4 Tổng kết chương 1
- 1.5 Bài tập chương 1


Tài liệu tham khảo

4

## 1.1 DANH SÁCH ĐẶC (LIST)

5

## 1.1 – DANH SÁCH ĐẶC ĐỊNH NGHĨA

 Danh sách đặc là một danh sách mà các phần tử trong danh sách có *cùng kiểu dữ liệu*, và được *cấp phát liên tục* trong bộ nhớ.

6

## 1.1 – DANH SÁCH ĐẶC KHAI BÁO CẤU TRÚC

```
# define MAX 100
int a[MAX];
int n; // n là tổng số phần tử hiện có trong danh sách, 0 <= n <= MAX
```

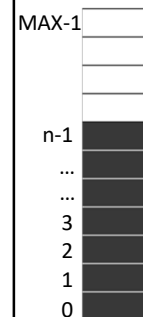
### Nhận xét:

- Trong một danh sách đặc, các phần tử được cấp phát liên tục. Do đó, tổng số phần tử tối đa trong một danh sách phải được biết trước (giá trị MAX là kích cỡ tối đa của mảng a được khai báo trước).
- Thuật toán tìm/truy xuất phần tử trong danh sách tại vị trí i (a[i]) dễ dàng.
- Thuật toán chèn phần tử vào vị trí i, hay xóa phần tử tại vị trí i ( $0 \leq i \leq n-1$ ), phức tạp vì phải thực hiện nhiều phép gán.

7

## 1.1 – DANH SÁCH ĐẶC

### Ví dụ 1.1:



'MAX' là độ dài tối đa của danh sách đặc  
0, 1, 2, 3...: là chỉ số từng phần tử trong danh sách

a[0] là biến chứa giá trị/dữ liệu của danh sách tại vùng có chỉ số 0

a[1] là biến chứa giá trị/dữ liệu của danh sách tại vùng có chỉ số 1

.....

a[n-1] là biến chứa giá trị/dữ liệu của danh sách tại vùng có chỉ số n-1

Hiện đang lưu trữ n phần tử

8

## 1.1 – DANH SÁCH ĐẶC

### CÁC THAO TÁC TRÊN DANH SÁCH ĐẶC

- 📖 Nhập danh sách từ bàn phím
- 📖 Xuất danh sách (ra ngoài màn hình)
- 📖 Tìm một phần tử trong danh sách
- 📖 Chèn/ thêm một phần tử mới vào danh sách tại vị trí i
- 📖 Xóa một phần tử tại vị trí i trong danh sách

9

## 1.1 – DANH SÁCH ĐẶC

### HÀM NHẬP DANH SÁCH ĐẶC

```
void input (int a[], int n)
{
    for (int i=0; i<n; i++)
    {
        cout<<"nhap a["<<i<<"] = ";
        cin>>a[i];
    }
}
```

10

## 1.1 – DANH SÁCH ĐẶC

### HÀM XUẤT DANH SÁCH ĐẶC

```
void output (int a[], int n)
{
    for (int i=0; i<n; i++)
        cout<<a[i]<<endl;
}
```

11

## 1.1 – DANH SÁCH ĐẶC

### TÌM KIẾM PHẦN TỬ TRONG DANH SÁCH

```
int search (int a[], int n, int x)
{
    int i = 0;
    while ( (i<n) && (a[i] != x) )
        i++;
    if (i==n)
        return -1; // tìm không thấy x trong danh sách, return -1
    return i; // tìm thấy x trong danh sách, return i là vị trí tìm thấy x
}
```

12

## 1.1 – DANH SÁCH ĐẶC

**Ví dụ 1.2:** có danh sách gồm  $n = 7$  phần tử được lưu trữ trong một danh sách đặc tại các vị trí từ **0 đến 6** (gồm các phần tử  $a[0], a[1], \dots, a[6]$ ):

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
10	50	20	70	30	60	40

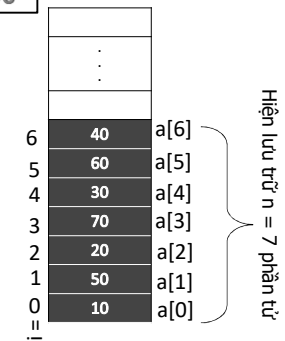
**Yêu cầu 1:** Tìm kiếm giá trị của  $x = 30$  trong danh sách.

13

## 1.1 – DANH SÁCH ĐẶC

Minh họa thuật giải tìm kiếm  **$x = 30$**

```
int search (int a[], int n, int x)
{
    int i = 0;
    while ( (i < n) && (a[i] != x) )
        i++;
    if (i == n) return -1;
    return i;
}
```



14

## 1.1 – DANH SÁCH ĐẶC

**Bước 1:**  $i = 0$

Xét điều kiện while ( $i < n$  &&  $a[i] != x$ )

$i = 0, n = 7$ :

$i < n \rightarrow \text{True}$

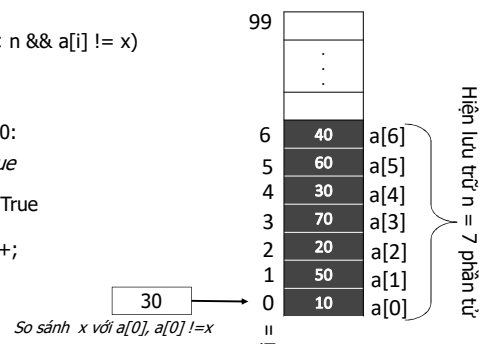
$a[i] = a[0] = 10, x = 30$ :

$a[i] != x \rightarrow \text{True}$

$(i < n \text{ \&\& } a[i] != x) \rightarrow \text{True}$

Thực hiện lệnh tăng  $i++$ ;

( $i$  tăng lên 1)



15

## 1.1 – DANH SÁCH ĐẶC

**Bước 2:**  $i = 1$

Xét điều kiện while ( $i < n$  &&  $a[i] != x$ )

$i = 1, n = 7$ :

$i < n \rightarrow \text{True}$

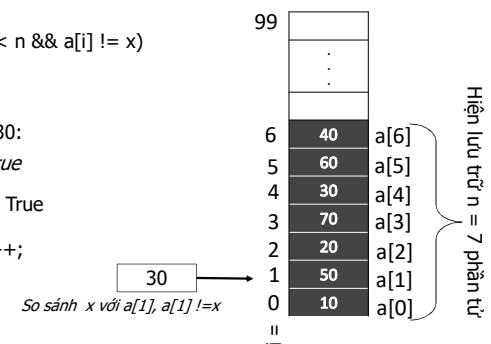
$a[i] = a[1] = 50, x = 30$ :

$a[i] != x \rightarrow \text{True}$

$(i < n \text{ \&\& } a[i] != x) \rightarrow \text{True}$

Thực hiện lệnh tăng  $i++$ ;

( $i$  tăng lên 1)



16

## 1.1 – DANH SÁCH ĐẶC

**Bước 3:**  $i = 2$

Xét điều kiện while ( $i < n \ \&\& \ a[i] \neq x$ )

$i = 2, n = 7$ :

$i < n \rightarrow \text{True}$

$a[i] = a[2] = 20, x = 30$ :

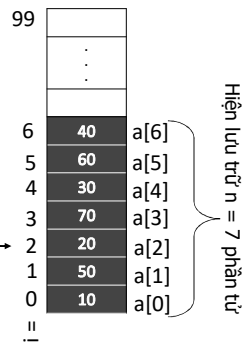
$a[i] \neq x \rightarrow \text{True}$

$(i < n \ \&\& \ a[i] \neq x) \rightarrow \text{True}$

Thực hiện lệnh tăng  $i++$ ;

( $i$  tăng lên 1)

30  
So sánh  $x$  với  $a[2]$ ,  
 $a[2] \neq x$



17

## 1.1 – DANH SÁCH ĐẶC

**Bước 4:**  $i = 3$

Xét điều kiện while ( $i < n \ \&\& \ a[i] \neq x$ )

$i = 3, n = 7$ :

$i < n \rightarrow \text{True}$

$a[i] = a[3] = 70, x = 30$ :

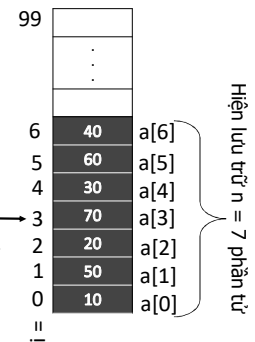
$a[i] \neq x \rightarrow \text{True}$

$(i < n \ \&\& \ a[i] \neq x) \rightarrow \text{True}$

Thực hiện lệnh tăng  $i++$ ;

( $i$  tăng lên 1)

30  
So sánh  $x$  với  $a[3]$ ,  
 $a[3] \neq x$



18

## 1.1 – DANH SÁCH ĐẶC

**Bước 5:**  $i = 4$

Xét điều kiện while ( $i < n \ \&\& \ a[i] \neq x$ )

$i = 4, n = 7$ :

$i < n \rightarrow \text{True}$

$a[i] = a[4] = 30, x = 30$ :

$a[i] \neq x \rightarrow \text{False}$

$(i < n \ \&\& \ a[i] \neq x) \rightarrow \text{False}$

Vòng lặp while sẽ kết thúc;

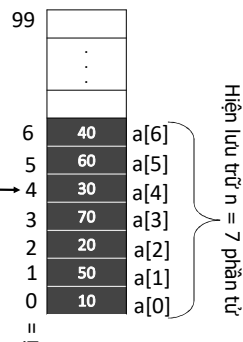
Vì  $i = 4 < n = 7$

$\Rightarrow$  return giá trị  $i$ ;

STOP

$i = 4$

30  
So sánh  $x$  với  $a[4]$ ,  
 $a[4] == x$



19

## 1.1 – DANH SÁCH ĐẶC

**Ví dụ 1.3:** có danh sách gồm  $n = 7$  phần tử được lưu trữ trong một danh sách đặc tại các vị trí từ 0 đến 6 (gồm các phần tử  $a[0], a[1], \dots, a[6]$ ):

$a[0]$	$a[1]$	$a[2]$	$a[3]$	$a[4]$	$a[5]$	$a[6]$
10	50	20	70	30	60	40

**Yêu cầu 2:** Tìm kiếm giá trị của  $x = 5$  trong danh sách.

20

## 1.1 – DANH SÁCH ĐẶC

**i = 0, 1, 2, 3, 4, 5, 6**

$\Rightarrow (i < n) \ \&\& \ (a[i] \neq x) = \text{True}$

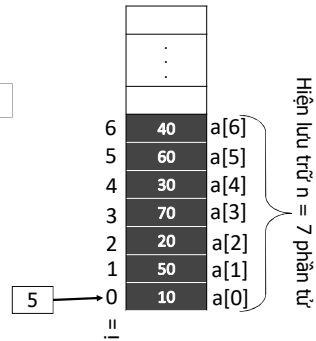
**i = 7**

$\Rightarrow (i < n) \ \&\& \ (a[i] \neq x) = \text{False}$

**i = n**

$\Rightarrow \text{return } -1$

$\Rightarrow$  (không tìm thấy giá trị  $x = 5$  trong danh sách)



21

## 1.1 – DANH SÁCH ĐẶC CHÈN/THÊM MỘT PHẦN TỬ TẠI VỊ TRÍ 'i'

Trong danh sách đặc, quá trình chèn/ thêm một phần tử tại vị trí  $i$  ( $i$  đi từ 0 đến  $n-1$ ) trong danh sách đặc tương đối phức tạp và tốn nhiều thời gian.

**Ý tưởng để chèn một giá trị  $x$  vào vị trí  $i$  ta làm như sau:**

Bước 1: Dời các đoạn phần tử từ  $i$  đến  $n-1$  ra phía sau một vị trí.

Chuyển giá trị  $a[n-1]$  qua  $a[n]$ ; ( $a[n] = a[n-1]$ )

Chuyển giá trị  $a[n-2]$  qua  $a[n-1]$ ; ( $a[n-1] = a[n-2]$ )

....

Chuyển giá trị  $a[i-1]$  qua  $a[i]$ ; ( $a[i] = a[i-1]$ );

Bước 2: Chèn giá trị  $x$  vào vị trí  $i$ ,  $a[i] = x$ ;

Bước 3: Tăng  $n$  lên 1 giá trị;

22

## 1.1 – DANH SÁCH ĐẶC

**Ví dụ 1.5:** Dựa trên danh sách đặc như bên hình, hãy thực hiện xóa phần tử tại vị trí  $i = 3$ .

Bước 1: Dời đoạn các phần tử sau vị trí  $i$  lên phía trước một vị trí (nhỏ dời trước).

chuyển giá trị  $a[i+1]$  qua  $a[i]$ ;

( $a[3] = a[4] \Rightarrow a[3] = 30$ );

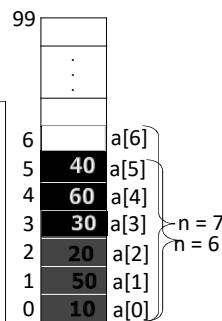
chuyển giá trị  $a[i+2]$  qua  $a[i+1]$ ;

( $a[4] = a[5] \Rightarrow a[4] = 60$ );

chuyển giá trị  $a[n-1]$  qua  $a[n-2]$ ;

( $a[5] = a[6] \Rightarrow a[5] = 40$ );

Bước 2: Giảm  $n$  một giá trị;



23

## 1.1 – DANH SÁCH ĐẶC

### THUẬT TOÁN XÓA MỘT PHẦN TỬ TẠI VỊ TRÍ 'i'

Bước 1: Dời đoạn các phần tử sau vị trí  $i$  lên phía trước một vị trí (nhỏ dời trước).

chuyển giá trị  $a[i+1]$  qua  $a[i]$ ; ( $a[3] = a[4] \Rightarrow a[3] = 30$ );

chuyển giá trị  $a[i+2]$  qua  $a[i+1]$ ; ( $a[4] = a[5] \Rightarrow a[4] = 60$ );

chuyển giá trị  $a[n-1]$  qua  $a[n-2]$ ; ( $a[5] = a[6] \Rightarrow a[5] = 40$ );

Bước 2: Giảm  $n$  một giá trị;

// Code dời giá trị phần tử từ  $n-1$  đến  $i$   
for (int j=i; j<n-1; j++)  
     $a[j] = a[j+1]$ ;

// Code gán giảm  $n$  một giá trị  
 $n--$ ;

24

## 1.1 – DANH SÁCH ĐẶC

### THUẬT TOÁN XÓA MỘT PHẦN TỬ TẠI VỊ TRÍ 'i'

```
// i là vị trí cần xóa
int Delete(int a[], int &n, int i)
{
    if (i >= 0 && i < n)
    {
        for (int j=i; j<n-1; j++)
            a[j] = a[j+1];
        n--;
        return 1;
    }
    return 0;
}
```


25

## 1.2 DANH SÁCH LIÊN KẾT (LINKED LIST)

26

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### ĐỊNH NGHĨA

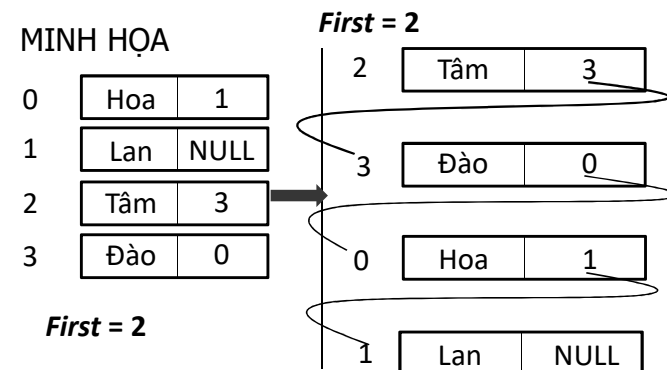
 Danh sách liên kết đơn là một *danh sách* mà các phần tử được cấp phát *rời rạc* nhau, và cố định trong bộ nhớ. Mỗi Phần tử trong danh sách gồm có 2 thành phần:

- Phần 1: vùng thông tin chứa giá trị cần quản lý
- Phần 2: vùng liên kết, chứa địa chỉ bộ nhớ của phần tử kế tiếp

27

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### MINH HỌA

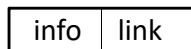


28

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### KHAI BÁO CẤU TRÚC

- 📖 Cấu trúc của một phần tử trong danh sách liên kết gồm 2 vùng (phần):
- info (chứa thông tin của phần tử)
  - link (chứa địa chỉ vùng nhớ của *phần tử tiếp theo*)



<pre>// cấu trúc 1 node struct Node {     int info;     Node *link; };</pre>	<pre>// khai báo danh sách, first sẽ chứa giá trị là địa chỉ ô nhớ của phần tử đầu tiên trong Danh sách Node * first;</pre>
--	---

29

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### CÁCH THAO TÁC CƠ BẢN

- 📖 Khởi tạo danh sách (tạo danh sách rỗng).
- 📖 Duyệt danh sách liên kết (xuất giá trị từng phần tử trong danh sách liên kết ra màn hình).
- 📖 Tìm kiếm một phần tử trong danh sách.
- 📖 Thêm một phần tử vào danh sách: *thêm đầu, thêm cuối, thêm sau một node q.*
- 📖 Xóa một phần tử ra khỏi danh sách: *xóa đầu, xóa cuối, và xóa sau một node q.*

30

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### KHỞI TẠO DANH SÁCH

- 📖 Danh sách rỗng là danh sách không có phần tử nào. Do đó phần tử đầu tiên cũng không tồn tại, nên ta có thể gán giá trị NULL cho biến first (con trỏ first);

```
void init()
{
    first = NULL;
}
// Chú ý: con trỏ first được khai báo toàn cục trong chương trình
```

31

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT ĐƠN

### DUYỆT DANH SÁCH

- 📖 Duyệt danh sách là việc ta truy xuất được giá trị/thông tin của từng phần tử trong danh sách liên kết (bắt đầu từ first)

```
void Process_list()
{
    Node *p;
    p = first;
    while (p!=NULL)
    {
        cout<<p-> info<<endl;
        p=p->link;
    }
}
```

32



1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

TÌM KIẾM MỘT PHẦN TỬ 'x'

```

Node * Search (int x)
{
    Node *p= first;
    while ( (p!=NULL) && (p->info != x) )
        p=p->link;
    return p;
}
// hàm trả về NULL nếu không tìm thấy, trả về địa chỉ của node p nếu tìm thấy.

```

33

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

THÊM MỘT PHẦN TỬ VÀO DANH SÁCH

- Thêm vào đầu danh sách
- Thêm vào cuối danh sách
- Thêm sau node q.

34

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

Thêm đầu <sup>first</sup>  
Danh sách

X = 30 → p

**Bước 1:** cấp phát phần tử mới p (có địa chỉ được quản lý bởi biến p)

**Bước 2:** Gán giá trị X (X=30) cho vùng info của p

**Bước 3:** Gán *giá trị địa chỉ bộ nhớ* của first cho vùng link của p (p->link = first);

**Bước 4:** Gán giá trị địa chỉ của p cho biến first.

```

Node *p = new Node;
p->info = x;
p->link = first;
first = p;

```

35

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

THỦ TỤC THÊM ĐẦU DANH SÁCH

```

void Insert_first(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->link = first;
    first = p;
}

```

```

Node *p = new Node;
p->info = x;
p->link = first;
first = p;

```

36

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

Thêm cuối danh sách

**Bước 1:** cấp phát phần tử mới (có địa chỉ được quản lý bởi biến p)

```
Node *p = new Node;
```

**Bước 2:** Gán giá trị x cho info của p và gán giá trị NULL cho vùng link của p

```
p->info = x;
p->link = NULL;
```

**Bước 3:** Tìm phần tử cuối danh sách (nếu có), và gán địa chỉ bộ nhớ của phần tử cuối cùng cho biến q

```
Node *q = first;
while (q -> link != NULL)
    q = q->link;
```

**Bước 4:** Gán địa chỉ bộ nhớ của p cho vùng link của q

```
q->link = p;
```

37

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**THỦ TỤC THÊM PHẦN TỬ 'x' VÀO CUỐI DS**

```
void Insert_last(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->link = NULL;
    if (first == NULL) //không có phần tử cuối cùng
        first = p;
    else
    {
        Node *q = first;
        while (q->link != NULL)
            q = q->link;
        q->link = p;
    }
}
```

```
Node *p = new Node;

p->info = x;
p->link = NULL;

Node *q = first;
while (q -> link != NULL)
    q = q->link;

q->link = p;
```

38

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**XÓA MỘT PHẦN TỬ**

- Xóa đầu
- Xóa cuối
- Xóa sau node q.

39

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**Xóa đầu**

**Bước 1:** Gán giá trị bộ nhớ của phần tử đầu danh sách cho biến p

```
Node *p = first;
```

**Bước 2 :** Gán địa chỉ bộ nhớ của phần tử thứ 2 trong danh sách cho biến first (first = first->link)

```
first = first->link;
```

**Bước 3 :** Thu hồi vùng nhớ phần tử đầu danh sách (delete p);

```
delete p;
```

40

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**THỦ TỤC XÓA PHẦN TỬ ĐẦU TRONG DSLK**

<pre>int Delete_first() {     if (first != NULL) // danh sách khác rỗng     {         Node *p = first;         first = first-&gt;link;         delete p;         return 1;     }     return 0; }</pre>	<div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">node *p = first;</div> <div style="border: 1px solid black; padding: 2px; margin-bottom: 5px;">first = first-&gt;link;</div> <div style="border: 1px solid black; padding: 2px;">delete p;</div>
--	--

41

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

Xóa cuối  
Danh sách

first      p      q      NULL

<p><b>Bước 1:</b> Tìm địa chỉ bộ nhớ của phần tử cuối danh sách (p), và địa chỉ của phần tử áp cuối danh sách (q) (nếu có).</p> <p><b>Bước 2:</b> Gán giá trị NULL cho vùng link của q.</p> <p><b>Bước 3:</b> Thu hồi vùng nhớ của phần tử cuối p;</p>	<pre>Node *p, * q; p = first; if (p!=NULL) while (p-&gt;link !=NULL) {     q=p;     p=p-&gt;link; } q-&gt;link =NULL; delete p;</pre>
--	---

42

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**THỦ TỤC XÓA PHẦN TỬ CUỐI TRONG DSLK**

<pre>int Delete_last() {     if (first != NULL)     {         Node *p, * q;         p = first; q = NULL;         if (p!=NULL)             while (p-&gt;link !=NULL)             {                 q=p; p=p-&gt;link;             }         if (p!=first) // p là đầu thì không tồn tại q;             q-&gt;link = NULL;         else first = NULL;         delete p;         return 1;     }     return 0; }</pre>	<pre>Node *p, * q; p = first; if (p!=NULL) while (p-&gt;link !=NULL) {     q=p;     p=p-&gt;link; } q-&gt;link =NULL; delete p;</pre>
---	---

43

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT ĐƠN**

**MỘT SỐ LƯU Ý**

- Phải khởi tạo danh sách trước khi bắt đầu làm việc với danh sách liên kết đơn.
- Khi xóa một phần tử ra khỏi danh sách phải thực hiện toán tử delete (để giải phóng vùng nhớ cho hệ thống).
- Mọi thao tác trên danh sách liên kết đơn, chỉ cần lưu node đầu tiên “first” (node này sẽ cho phép ta truy xuất được các node tiếp theo).

44

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP

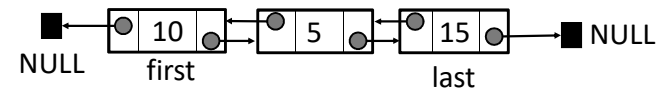
### ĐỊNH NGHĨA

- 📖 Danh sách liên kết kép là một danh sách liên kết mà mỗi phần tử trong danh sách bao gồm 3 Thành phần
- Vùng chứa thông tin (info)
  - Vùng liên kết (next) trỏ đến phần tử đứng liền sau nó
  - Vùng liên kết (previous) trỏ đến phần tử đứng liền trước nó

45

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP

Ví dụ 2.6:

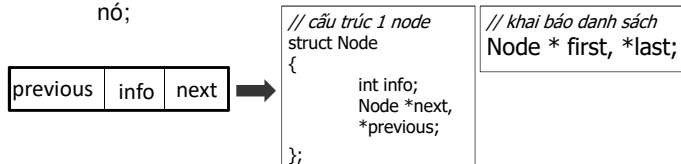


46

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP

### KHAI BÁO CẤU TRÚC

- 📖 Cấu trúc một phần tử trong danh sách liên kết đôi gồm 3 vùng (phần):
- info (chứa thông tin của phần tử)
  - next chứa địa chỉ vùng nhớ phần tử phía sau nó;
  - previous chứa địa chỉ vùng nhớ phần tử phía trước nó;



47


## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP

### CÁC THAO TÁC CƠ BẢN

- 📖 Khởi tạo danh sách (tạo danh sách rỗng).
- 📖 Duyệt danh sách (xuất giá trị từng phần tử trong danh sách liên kết ra màn hình).
- 📖 Tìm kiếm một phần tử trong danh sách.
- 📖 Thêm một phần tử vào danh sách: *thêm đầu, thêm cuối, thêm sau một node q.*
- 📖 Xóa một phần tử ra khỏi danh sách: *xóa đầu, xóa cuối, và xóa sau một node q.*

48


## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP KHỞI TẠO DANH SÁCH LIÊN KẾT ĐÔI

 Danh sách rỗng là danh sách không có phần tử nào. Do đó ta gán giá trị NULL cho biến first và last.

```
void init()
{
    first = NULL;
    last = NULL;
}
// Chú ý: con trỏ first, last được khai báo toàn cục trong chương trình
```

49

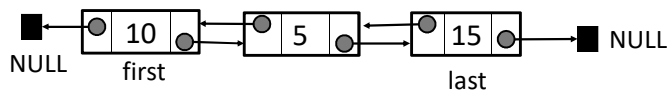
## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP DUYỆT DANH SÁCH

 Thuật toán duyệt danh sách liên kết đôi cũng tương tự kỹ thuật duyệt trong danh sách liên kết đơn. Tuy nhiên danh sách liên kết đôi hỗ trợ con trỏ previous cho phép duyệt từ cuối lên đầu.

<pre>// duyệt từ đầu về cuối void Process_list() {     Node *p;     p = first;     while (p!=NULL)     {         cout&lt;&lt;p-&gt;info&lt;&lt;endl;         p=p-&gt;next;     } }</pre>	<pre>// duyệt từ cuối lên đầu void Process_list_begin_last() {     Node *p;     p = last;     while (p!=NULL)     {         cout&lt;&lt;p-&gt;info&lt;&lt;endl;         p=p-&gt;previous;     } }</pre>
--	---

50

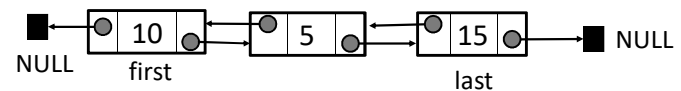
## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP TÌM KIẾM PHẦN TỬ 'x'



```
// duyệt từ đầu về cuối
Node * Search (int x)
{
    Node *p;
    p = first;
    while ( (p!=NULL) && (p->info != x) )
        p=p->next;
    return p;
}
// hàm trả về NULL nếu không tìm thấy, trả về địa chỉ của node p nếu tìm thấy.
```

51

## 1.2 – DANH SÁCH LIÊN KẾT DANH SÁCH LIÊN KẾT KÉP TÌM KIẾM PHẦN TỬ 'x'



```
// duyệt từ cuối lên đầu
Node * Search_list_begin_last(int x)
{
    Node *p;
    p = last;
    while ( (p!=NULL) && (p->info != x) )
        p=p->previous;
    return p;
}
// hàm trả về NULL nếu không tìm thấy, trả về địa chỉ của node p nếu tìm thấy.
```

52

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THÊM MỘT PHẦN TỬ VÀO DANH SÁCH**

Thêm vào đầu danh sách

Thêm vào cuối danh sách

Thêm sau node q.

53

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**Thêm đầu**  
Danh sách

X = 30

NULL

first

last

NULL

p

**Bước 1:** cấp phát phần tử mới (có địa chỉ được quản lý bởi biến p)

**Bước 2:** Gán giá trị X (X=30) cho p->info, và gán con trỏ previous của p về NULL (p->previous = NULL)

**Bước 3:** Gán **giá trị địa chỉ bộ nhớ** của first cho vùng next của p

**Bước 4:** Hiệu chỉnh giá trị vùng liên kết previous của first thành giá trị địa chỉ của p (first->previous = p)

**Bước 5:** Cập nhật lại giá trị của biến first là địa chỉ của phần tử vừa cấp phát p

Node \*p = new Node;

p->info = x;

p-> previous = NULL;

p->next = first;

first->previous = p;

first = p;

54

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THỦ TỤC THÊM ĐẦU**

```
void Insert_first(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->next= first;
    p->previous = NULL;
    if(first != NULL)
        first->previous = p;
    else // danh sách rỗng, trước khi thêm p
        last = p; // sau khi thêm phần tử p, danh sách có 1 phần tử
    first = p;
}
```

Node \*p = new Node;

p->info = x;

p-> previous = NULL;

p->next = first;

first->previous = p;

first = p;

55

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**Thêm cuối danh sách**

NULL

first

last

NULL

p

X = 30

**Bước 1:** Cấp phát phần tử mới (có địa chỉ được quản lý bởi biến p)

**Bước 2:** Gán giá trị X (X = 30), và gán con trỏ next của p về NULL (p->next = NULL)

**Bước 3:** Hiệu chỉnh giá trị vùng liên kết previous của p thành giá trị địa chỉ của last (p->previous = last)

**Bước 4:** Gán **giá trị địa chỉ bộ nhớ** của p cho vùng next của last

**Bước 5:** Gán giá trị địa chỉ của p cho last (last = p).

Node \*p = new Node;

p->info = x;

p-> next = NULL;

p->previous = last;

last->next = p;

last = p;

56

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
THỦ TỤC THÊM CUỐI DANH SÁCH

```

void Insert_last(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->next = NULL;
    p->previous = last;
    if(last != NULL)
        last->next = p;
    else // danh sách rỗng, trước khi thêm p
        first = p; // sau khi thêm phần tử p, danh
        sách có 1 phần tử
    last = p;
}

```

Node \*p = new Node;  
p->info = x;  
p->next = NULL;  
p->previous = last;  
last->next = p;  
last = p;

57

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
XÓA PHẦN TỬ TRONG DANH SÁCH

- Xóa đầu
- Xóa cuối
- Xóa sau node q.

58

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
XÓA PHẦN TỬ ĐẦU TRONG DANH SÁCH

59

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THUẬT GIẢI XÓA ĐẦU PHẦN TỬ ĐẦU**

- Bước 1:** Khai báo biến p lưu trữ địa chỉ của phần tử đầu tiên ( ) giá trị biến bằng giá trị biến first)

Node \*p = first;
- Bước 2:** Cập nhật giá trị biến first lưu trữ địa chỉ phần tử liền sau phần tử first (first = first->next).

first = first->next;
- Bước 3:** Nếu first khác NULL thì cho previous của first trở về NULL (first -> previous = NULL), ngược lại cho last = NULL;

if (first != NULL)
 first->previous = NULL;
else
 last = NULL;
- Bước 4:** Giải phóng vùng nhớ tại p (delete p);

delete p;

60

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THUẬT GIẢI XÓA ĐẦU PHẦN TỬ ĐẦU**

```

int Delete_first()
{
    if (first != NULL)
    {
        Node *p = first;
        first = first->next;
        delete p;
        if (first != NULL) // trường hợp ds khác
            first->previous = NULL;
        else
            last = NULL; // trường hợp
            ds không còn phần tử nào
        return 1;
    }
    return 0;
}

```

Node \*p = first;  
  
first = first->next;  
  
if (first != NULL)  
 first->previous = NULL;  
else  
 last = NULL;  
  
delete p;

61

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**XÓA PHẦN TỬ CUỐI DANH SÁCH**

62

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THUẬT GIẢI XÓA PHẦN TỬ ĐẦU DANH SÁCH**

Bước 1: Gán giá trị bộ nhớ của phần tử cuối danh sách cho biến p;

Bước 2: Gán địa chỉ bộ nhớ của phần tử kế cuối trong danh sách cho biến last (last = last->previous).

Bước 3: Nếu last khác NULL thì cho vùng next của last trở về NULL (last->next= NULL), ngược lại last = NULL;

Bước 4: Giải phóng vùng nhớ tại p (delete p);

Node \*p = last;  
  
last = last->previous;  
  
if (last!=NULL)  
 last->next = NULL;  
else  
 first = NULL;  
  
delete p;

63

1.2 – DANH SÁCH LIÊN KẾT  
**DANH SÁCH LIÊN KẾT KÉP**  
**THUẬT GIẢI XÓA PHẦN TỬ ĐẦU DANH SÁCH**

```

int Delete_last()
{
    if (last != NULL)
    {
        Node *p = last;
        last = last->previous;
        if (last != NULL) // trường hợp ds khác rỗng nào
            last->next = NULL;
        else
            first = NULL;
        delete p;
        return 1;
    }
    return 0;
}

```

Node \*p = last;  
  
last = last->previous;  
  
if (last!=NULL)  
 last->next = NULL;  
else  
 first = NULL;  
  
delete p;

64



1.3

DANH SÁCH HẠN CHẾ  
(Stack & Queue)




65

1.3 – DANH SÁCH HẠN CHẾ

**NGĂN XẾP (STACK)**

ĐỊNH NGHĨA NGĂN XẾP - STACK

 Ngăn xếp (Stack) là một danh sách các phần tử được quản lý theo thứ tự như sau: Phần tử được *thêm vào* ngăn xếp *sau*, sẽ *được lấy ra* (xóa) khỏi ngăn xếp *trước*


Last In First Out - LIFO


66


1.3 – DANH SÁCH HẠN CHẾ


**NGĂN XẾP (STACK)**


Các thao tác trên Stack

 `init`: khởi tạo Stack rỗng

 `Push`: cho phần tử vào Stack

 `Pop`: lấy phần tử từ trong Stack ra ngoài

 `isFull`: kiểm tra Stack đầy chưa

 `isEmpty`: kiểm tra Stack có rỗng không

67

1.3 – DANH SÁCH HẠN CHẾ

**NGĂN XẾP (STACK)**

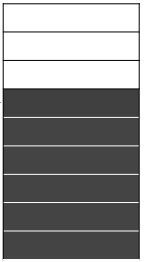
CÀI ĐẶT STACK BẰNG DANH SÁCH ĐẶC

**KHAI BÁO CẤU TRÚC**

```
# define MAX 100
int a[MAX];
int sp; // đỉnh stack
```

Vị trí xác định vị trí thêm vào, hay lấy phần tử trong stack ra.

MAX -1



sp

...

...

2

1

0

68

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### KHỞI TẠO STACK RỖNG

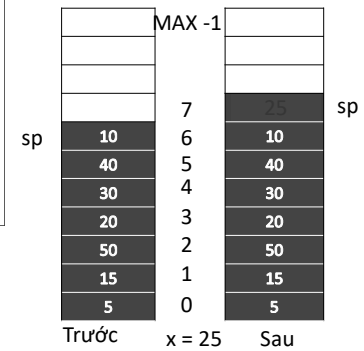
```
void init(int a[], int &sp)
{
    sp = -1;
}
```

69

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### THÊM PHẦN TỬ VÀO STACK (push)

```
int Push(int a[], int &sp, int x)
{
    if (sp < MAX - 1)
    {
        a[++sp] = x;
        return 1;
    }
    return 0;
}
```

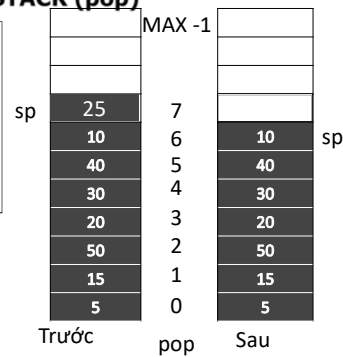


70

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### LẤY MỘT PHẦN TỬ TRONG STACK (pop)

```
int Pop(int a[], int &sp, int &x)
{
    if (sp != -1) // khi stack khác rỗng
    {
        x = a[sp--];
        return 1;
    }
    return 0;
}
```

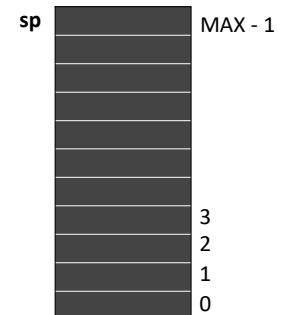


71

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### HÀM KIỂM TRA STACK ĐẦY (isFull)

```
int isFull(int a[], int sp)
{
    if (sp == MAX - 1)
        return 1;
    return 0;
}
```



Nếu ngăn sắp đầy trả về 1;  
Nếu ngăn sắp chưa đầy trả về 0;

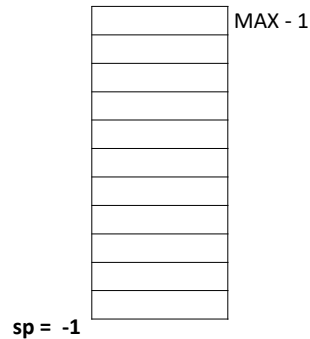
72

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### HÀM KIỂM TRA STACK RỖNG (isEmpty)

```
int isEmpty(int a[], int sp)
{
    if (sp == -1)
        return 1;
    return 0;
}
```

Nếu ngăn xếp rỗng trả về 1;  
Nếu ngăn xếp không rỗng trả về 0;



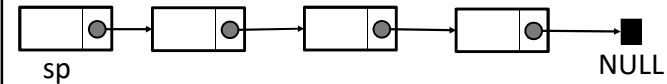
73

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### CÀI ĐẶT STACK BẰNG DANH SÁCH LIÊN KẾT

##### KHAI BÁO CẤU TRÚC

```
// cấu trúc 1 node
struct Node
{
    int info;
    Node *link;
};
Node * sp; // lưu trữ địa chỉ phần tử đầu tiên của Stack
```



74

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### KHỞI TẠO STACK RỖNG

```
void init()
{
    sp = NULL;
}
```

75

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### THÊM PHẦN TỬ VÀO STACK

- Bước 1: cấp phát phần tử mới (có địa chỉ do p quản lý)
- Bước 2: Gán giá trị x cho vùng infor của phần tử vừa cấp phát p.
- Bước 3: Gán **giá trị địa chỉ bộ nhớ** của phần tử đầu danh sách cho vùng link của p
- Bước 4: Gán giá trị địa chỉ của p cho sp

```
Node *p = new Node;
```

```
p->info = x;
```

```
p->link = sp;
```

```
sp = p;
```

76

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### THÊM PHẦN TỬ VÀO STACK

```
void Push(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->link = sp;
    sp = p;
}
```

Node \*p = new Node;

p->info = x;

p->link = sp;

sp = p;

77

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### THUẬT GIẢI LẤY MỘT PHẦN TỬ TRONG STACK

Bước 1: Gán giá trị địa chỉ bộ nhớ của phần tử đầu danh sách cho biến p

Bước 2: Gán giá trị vùng info của p cho biến x;

Bước 3: Gán địa chỉ bộ nhớ của phần tử thứ 2 trong danh sách (nếu có) cho biến sp

Bước 4: Thu hồi vùng nhớ phần tử đầu danh sách

Node \*p = sp;

x= p->info;

sp= sp->link;

delete p;

78

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### THỦ TỤC LẤY MỘT PHẦN TỬ TRONG STACK

```
int Pop(int &x)
{
    if (sp!= NULL)
    {
        Node *p = sp;
        x = p -> info;
        sp= p ->link;
        delete p;
        return 1;
    }
    return 0;
}
```

Node \*p = sp;

x= p->info;

sp= p->link;

delete p;

79

### 1.3 – DANH SÁCH HẠN CHẾ NGĂN XẾP (STACK)

#### KIỂM TRA STACK RỖNG

```
int isEmpty()
{
    if (sp == NULL)
        return 1;
    return 0;
}
```

80

### 1.3 – DANH SÁCH HẠN CHẾ **HÀNG ĐỘI (QUEUE)**

#### ĐỊNH NGHĨA

📖 Hàng đợi (Queue) là danh sách chứa các phần tử được quản lý theo thứ tự sau: Phần tử được *thêm* vào trước, sẽ *được lấy ra* (xóa) *trước*

First In First Out - FIFO

81

### 1.3 – DANH SÁCH HẠN CHẾ **HÀNG ĐỘI (QUEUE)**

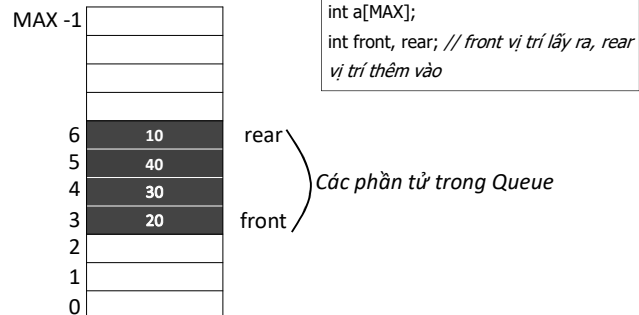
#### CÁC THAO TÁC TÊN QUEUE

- 📖 init: khởi tạo Queue rỗng.
- 📖 Push: cho phần tử vào Queue
- 📖 Pop: lấy phần tử từ trong Queue
- 📖 isFull: kiểm tra Queue đầy chưa
- 📖 isEmpty: kiểm tra Queue có rỗng không

82

### 1.3 – DANH SÁCH HẠN CHẾ **HÀNG ĐỘI (QUEUE)**

#### CÀI ĐẶT QUEUE BẰNG DANH SÁCH ĐẶC **KHAI BÁO CẤU TRÚC**



83

### 1.3 – DANH SÁCH HẠN CHẾ **HÀNG ĐỘI (QUEUE)**

#### KHỞI TẠO HÀNG ĐỘI RỖNG

```
void init(int a[], int &front, int &rear)
{
    front = -1;
    rear = -1;
}
```

84

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**  
HÀNG ĐỢI BỊ TRÀN

Hàng đợi bị tràn là trạng thái hàng đợi chưa bị đầy (các phần tử trong hàng chưa lấp đầy danh sách đặc), tuy nhiên giá trị rear = với giá trị MAX-1. Để khắc phục hàng đợi bị tràn có hai phương pháp:

- Phương pháp tịnh tiến
- Phương pháp vòng

rear

10

99

MAX -1

40

98

30

97

20

96

50

95

front

·

·

·

2

1

0

Trước (bị tràn)

**TRƯỜNG HỢP BỊ TRÀN**

85

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**  
XỬ LÝ HÀNG ĐỢI TRÀN BẰNG **PHƯƠNG PHÁP TỊNH TIẾN**

Thêm vào/hay lấy phần tử ra ta luôn cập nhật lại trạng thái của hàng đợi cho front chuyển về vị trí đầu queue (front=0)

**Chú ý:** chỉ xét nếu hàng đợi có nhiều hơn 1 phần tử (và front đang ở vị trí khác 0).

86

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**  
THÊM MỘT PHẦN TỬ VÀO HÀNG ĐỢI BẰNG PHƯƠNG PHÁP TỊNH TIẾN

MAX -1

7

6

5

4

3

2

1

0

Trước

10

40

30

20

50

rear

front

·

·

·

2

1

0

Trước (bị tràn)

25

10

40

30

20

50

rear

front

·

·

·

2

1

0

Trước (bị tràn)

**TRƯỜNG HỢP BÌNH THƯỜNG**

87

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**  
THÊM MỘT PHẦN TỬ VÀO HÀNG ĐỢI BẰNG PHƯƠNG PHÁP TỊNH TIẾN

MAX -1

99

98

97

96

95

·

·

·

2

1

0

Trước (bị tràn)

10

40

30

20

50

rear

front

·

·

·

2

1

0

Trước (bị tràn)

10

40

30

20

50

rear

front

·

·

·

2

1

0

Trước (bị tràn)

**TRƯỜNG HỢP BỊ TRÀN**

88

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

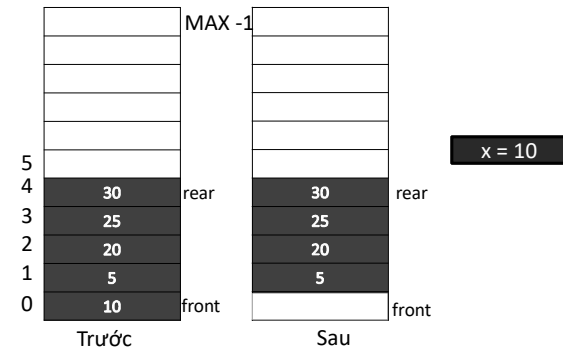
#### THỦ TỤC THÊM MỘT PHẦN TỬ BẰNG PP TỊNH TIẾN

```
int Push(int a[], int &front, int &rear, int x)
{
    if (rear - front == MAX - 1) // hàng đợi đầy
        return 0;
    else
    {
        if (front == -1) // hàng đợi rỗng
            front = 0;
        if (rear == MAX - 1) // hàng đợi bị tràn
        {
            for(int i = front; i <= rear; i++)
                // thực hiện dời tịnh tiến các phần tử trong hàng
                a[i-front] = a[i];
            rear = MAX - 1 - front;
            // rear nhận giá trị mới, sau khi tịnh tiến
            front = 0; // front nhận giá trị mới, sau khi tịnh tiến
        }
        a[++rear] = x;
        return 1;
    }
}
```

89

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### LẤY PHẦN TỬ BẰNG PHƯƠNG PHÁP TỊNH TIẾN



90

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### CÀI ĐẶT THUẬT GIẢI LẤY PHẦN TỬ BẰNG PP TỊNH TIẾN

```
int Pop(int a[], int &front, int &rear, int &x)
{
    if (front == -1) // hàng đợi rỗng
        return 0;
    else
    {
        x = a[front++];
        if (front > rear)
            // trường hợp hàng đợi có 1 phần tử, sau khi xóa hàng rỗng
        {
            front = -1;
            rear = -1;
        }
        return 1;
    }
}
```

91

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### XỬ LÝ HÀNG ĐỢI TRÀN BẰNG PHƯƠNG PHÁP VÒNG THÊM PHẦN TỬ

Khi thêm phần tử nếu như rear trong danh sách = MAX-1, và danh sách vẫn còn trống thì khi ta thực hiện thêm phần tử mới, ta sẽ duy chuyển rear về vị trí 0 (vị trí bắt đầu của danh sách)

92

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**

Thêm một phần tử vào hàng đợi bằng phương pháp vòng

**THÊM X = 50**

99	10	rear	10	MAX -1 rear
98	40		40	
97	30		30	
96	20	front	20	front
95				
.	.		.	
.	.		.	
2				
1				
0			50	

-1  
TRẠNG THÁI CHƯA BỊ TRÀN

XỬ LÝ TRÀN (PP VÒNG)

93

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**

THỦ TỤC THÊM MỘT PHẦN TỬ VÀO HÀNG ĐỢI  
**BẰNG PHƯƠNG PHÁP VÒNG**

```

int Pust(int a[], int &front, int &rear, int x)
{
    if ( ( rear - front == MAX - 1 ) || ( rear - front == -1 ) ) // hàng đợi bị đầy
        return 0;
    else
    {
        if (front == -1) // xét hàng đợi rỗng
            front = 0; // khi đó rear = -1;
        if (rear == max - 1) // hàng đợi bị tràn
            rear = -1; // hàng đợi tràn, vòng giá trị rear xuống -1
        a[++rear] = x; // tăng rear lên 1 giá trị, và thêm phần tử mới tại rear
        return 1;
    }
}

```

94

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**

THỦ TỤC LẤY MỘT PHẦN TỬ VÀO HÀNG ĐỢI  
**BẰNG PHƯƠNG PHÁP VÒNG**

```

int Pop(int a[], int &x)
{
    if (front != -1)
    {
        x = a[front];
        if (front == rear)
        {
            front = -1;
            rear = -1;
        }
        else
        {
            front++;
            if (front > max - 1) front = 0;
        }
        return 1;
    }
    return 0;
}

```

95

1.3 – DANH SÁCH HẠN CHẾ  
**HÀNG ĐỢI (QUEUE)**

CÀI ĐẶT BẰNG DANH SÁCH LIÊN KẾT  
**KHAI BÁO CẤU TRÚC**

```

// cấu trúc 1 node
struct Node
{
    int info;
    Node *link;
};
Node * front, *rear // front vị trí xóa, rear vị trí thêm

```

96



### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)





#### KHOẸ TẠO QUEUE RỖNG

```
void init()
{
    front= NULL;
    rear= NULL;
}
```

97

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### THUẬT GIẢI THÊM PHẦN TỬ VÀO QUEUE

-  Bước 1: Cấp phát phần tử mới (có địa chỉ do p quản lý)
-  Bước 2: Gán giá trị NULL cho vùng link của phần tử p. Và gán giá trị biến x cho vùng info của p
-  Bước 3: Nếu rear khác NULL thì gán **giá trị địa chỉ bộ nhớ** của p cho vùng link của rear, ngược lại gán front = p;
-  Bước 4: Gán giá trị địa chỉ của p cho vùng link của phần tử cuối rear, nếu danh sách khác rỗng

```
Node *p = new Node;
```

```
p->info = x;
p->link = NULL;
```

```
if (rear != NULL)
    rear->link =
    p;
else
    front =p;
```

```
rear = p;
```

98

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### THỦ TỤC THÊM PHẦN TỬ

```
void Push(int x)
{
    Node *p;
    p = new Node;
    p->info = x;
    p->link = NULL;
    if (rear == NULL)
        front =p;
    else
        rear->link = p;
    rear = p;
}
```

```
Node *p = new Node;
```

```
p->info = x;
p->link = NULL;
```





```
if (rear == NULL)
    front =p;
else
    rear->link = p;
```

```
rear = p;
```

99

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### THUẬT GIẢI LẤY PHẦN TỬ TỪ QUEUE

-  Bước 1: Gán giá trị bộ nhớ của phần tử đầu danh sách cho biến p;
-  Bước 2: Gán giá trị vùng info của p cho biến x;
-  Bước 3: Gán địa chỉ bộ nhớ của phần tử thứ 2 trong danh sách (nếu có) cho biến front
-  Bước 4: Thu hồi vùng nhớ phần tử đầu danh sách

```
Node *p = front;
```

```
x = p ->info;
```

```
front= front->link;
if (front == NULL)
    rear = NULL;
```

```
delete p;
```

100

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE) THỦ TỤC LẤY PHẦN TỬ TỪ QUEUE

```
int Pop(int &x)
{
    if (front != NULL)
    {
        Node *p = front;
        x = p -> info;
        front = front -> link;
        if (front == NULL)
            rear = NULL;
        delete p;
        return 1;
    }
    return 0;
}
```

Node \*p = front;

x = p -> info;

front = front -> link;  
if (front == NULL)  
 rear = NULL;

delete p;

101

### 1.3 – DANH SÁCH HẠN CHẾ HÀNG ĐỢI (QUEUE)

#### KIỂM TRA QUEUE RỖNG

```
int isEmpty()
{
    if (front == NULL)
        return 1;
    return 0;
}
```

102

## 1.4 – TỔNG KẾT CHƯƠNG 1

### 1.4 – Tổng kết chương 1

- 📖 **Danh sách đặc (LIST)** là một danh sách mà các phần tử trong danh sách được *cấp phát liên tục* trong bộ nhớ.
- 📖 **Danh sách liên kết (LINKED LIST)** là danh sách mà các phần tử được *cấp phát rời rạc*.
- 📖 **Danh sách liên kết đơn (SINGLY LINKED LIST)**
- 📖 **Danh sách liên kết đôi (DOUBLY LINKED LIST)**

103

104

## 1.4 – Tổng kết chương 1

📖 Danh sách hạn chế:

- 📖 Cấu trúc **ngăn xếp** (STACK) là cấu trúc hoạt động theo cơ chế *LIFO*
- 📖 Cấu trúc **hàng đợi** (QUEUE) là cấu trúc hoạt động theo cơ chế *FIFO*

105



## 1.5- Bài tập rèn luyện CHƯƠNG 1



106

### 1.5 - Bài tập chương 1 CÂU HỎI

- 📖 **Câu 1:** Trong khoa học máy tính, ***danh sách đặc*** được hiểu như thế nào? Cho ví dụ.
- 📖 **Câu 2:** Trong khoa học máy tính, ***danh sách liên kết*** được hiểu như thế nào? Có mấy loại? Cho ví dụ.
- 📖 **Câu 3:** Tại sao nói STACK và QUEUE là danh sách hạn chế? Cho ví dụ?
- 📖 **Câu 4:** Thế nào là LIFO, FIFO? Cho ví dụ.
- 📖 **Câu 6:** Theo bạn, danh sách danh sách liên kết có thể ứng dụng xử lý các vấn đề gì trong máy tính?
- 📖 **Câu 7:** Thế nào là cấu trúc dữ liệu động? Cho ví dụ.

107

### 1.5 - Bài tập chương 1 BÀI THỰC HÀNH SỐ 1

**Bài 1:** Quản lý một danh sách có tối đa 100 phần tử, mỗi phần tử trong danh sách có kiểu int. (Danh sách không có thứ tự)

- 1.1. Khai báo cấu trúc danh sách
- 1.2. Viết thủ tục nhập danh sách
- 1.3. Viết thủ tục xuất danh sách ra màn hình
- 1.4. Viết thủ tục tìm một phần tử trong danh sách. Tính độ phức tạp của thuật toán.
- 1.5. Viết thủ tục thêm một phần tử vào cuối danh sách.
- 1.6. Viết thủ tục xóa phần tử cuối danh sách.
- 1.7. Viết thủ tục xóa phần tử tại vị trí thứ i. Tính độ phức tạp của thuật toán.
- 1.8. Tìm một phần tử trong danh sách. Nếu tìm thấy, xóa phần tử đó. (Tính độ phức tạp của thuật toán) (\*)

108

108

### 1.5 - Bài tập chương 1 BÀI LÀM THÊM

**Bài 2:** Quản lý một danh sách có *thứ tự* tối đa 100 phần tử, mỗi phần tử trong danh sách có kiểu int. (Danh sách đặc)

- 2.1. Khai báo cấu trúc danh sách
- 2.2. Viết thủ tục thêm một phần tử vào danh sách (thêm một phần tử vào vị trí phù hợp trong danh sách đã có thứ tự). Lưu ý: Không xếp thứ tự danh sách.
- 2.3. Viết thủ tục xuất các phần tử danh sách
- 2.4. Viết thủ tục tìm một phần tử trong danh sách (dùng phương pháp tìm kiếm tuần tự). Đánh giá độ phức tạp của thuật toán.
- 2.5. Viết thủ tục tìm một phần tử trong danh sách (dùng phương pháp tìm kiếm nhị phân). Đánh giá độ phức tạp của thuật toán. (\*)
- 2.6. Viết thủ tục tìm một phần tử trong danh sách. Nếu tìm thấy, xóa phần tử này. (\*)

109

109

### 1.5 - Bài tập chương 1 BÀI THỰC HÀNH SỐ 2

**Bài 3:** Quản lý một danh sách có số phần tử khá lớn, biến động. Mỗi phần tử có kiểu int. (Dùng cấu trúc danh sách liên kết đơn)

- 3.1. Khai báo cấu trúc danh sách
- 3.2. Viết thủ tục khởi tạo danh sách rỗng
- 3.3. Viết thủ tục xuất các phần tử trong danh sách.
- 3.4. Viết thủ tục tìm một phần tử trong danh sách.

110

110

### 1.5 - Bài tập chương 1 BÀI THỰC HÀNH SỐ 2 (tiếp)

- 3.5. Viết thủ tục thêm một phần tử vào đầu danh sách
- 3.6. Viết thủ tục xóa một phần tử đầu danh sách.
- 3.7. Viết thủ tục thêm một phần tử vào cuối danh sách
- 3.8. Viết thủ tục xóa một phần tử cuối danh sách.
- 3.9. Viết thủ tục tìm một phần tử trong danh sách. Nếu tìm thấy, hãy xóa phần tử này
- 3.10. danhTử danh sách trên hãy chuyển thành danh sách có thứ tự (\*)

111

111

### 1.5 - Bài tập chương 1 BÀI LÀM THÊM

**Bài 4:** Quản lý một danh sách *có thứ tự*, số phần tử khá lớn, biến động. Mỗi phần tử có kiểu int. (Dùng cấu trúc danh sách liên kết đơn)

- 4.1. Khai báo cấu trúc danh sách
- 4.2. Viết thủ tục khởi tạo danh sách rỗng.
- 4.3. Viết thủ tục thêm một phần tử vào danh sách (thêm một phần tử vào vị trí phù hợp trong danh sách đã có thứ tự). Lưu ý: Không xếp thứ tự danh sách.
- 4.4. Viết thủ tục xuất các phần tử trong danh sách.
- 4.5. Viết thủ tục tìm một phần tử trong danh sách (lưu ý: danh sách đã có thứ tự)
- 4.6. Viết thủ tục tìm một phần tử trong danh sách. Nếu tìm thấy, xóa phần tử này (Lưu ý: danh sách đã có thứ tự)

112

112

1.5 - Bài tập chương 1  
**BÀI THỰC HÀNH SỐ 3**

**Bài 5:** Quản lý một stack có tối đa 100 phần tử, mỗi phần tử trong stack có kiểu int (stack danh sách đặc)

- 5.1. Khai báo cấu trúc stack.
- 5.2. Viết thủ tục khởi tạo stack rỗng.
- 5.3. Viết thủ tục kiểm tra stack rỗng.
- 5.4. Viết thủ tục kiểm tra stack đầy.
- 5.5. Viết thủ tục thêm một phần tử vào stack
- 5.6. Viết thủ tục xóa một phần tử trong stack

113

113

1.5 - Bài tập chương 1  
**BÀI THỰC HÀNH SỐ 3**

**Bài 6:** Sử dụng Stack đã xây dựng, đổi một số hệ thập phân sang hệ nhị phân.

114

114

1.5 - Bài tập chương 1  
**BÀI THỰC HÀNH SỐ 3**

**Bài 7:** Quản lý một queue có tối đa 100 phần tử, mỗi phần tử trong queue có kiểu int (queue danh sách đặc)

- 7.1. Khai báo cấu trúc queue.
- 7.2. Viết thủ tục khởi tạo queue rỗng
- 7.3. Viết thủ tục kiểm tra queue rỗng.
- 7.4. Viết thủ tục kiểm tra queue đầy.
- 7.5. Viết thủ tục thêm một phần tử vào queue
- 7.6. Viết thủ tục xóa một phần tử trong queue

115

115

1.5 - Bài tập chương 1  
**BÀI LÀM THÊM**

**Bài làm thêm của phần thực hành chương 1**

**Bài 8:** Sử dụng stack đã xây dựng, đổi cơ số hệ thập phân sang hệ bất kì.

**Bài 9:** Sử dụng stack đã xây dựng, giải quyết bài toán THÁP HÀ NỘI (\*)

116

116

### 1.5 - Bài tập chương 1 **BÀI THỰC HÀNH SỐ 4**

**Bài 10:** Quản lý một stack có số phần tử khá lớn, biến động. Mỗi phần tử có kiểu int (danh sách liên kết đơn)

- 10.1. Khai báo cấu trúc stack.
- 10.2. Viết thủ tục khởi tạo stack rỗng.
- 10.3. Viết thủ tục kiểm tra stack rỗng.
- 10.4. Viết thủ tục thêm một phần tử vào stack (push).
- 10.5. Viết thủ tục xóa một phần tử trong stack (pop).
- 10.6. Áp dụng stack đã xây dựng, giải bài toán đổi một số từ hệ thập phân sang hệ nhị phân
- 10.7. Áp dụng stack đã xây dựng, giải bài toán THÁP HÀ NỘI (\*)

117

117

### 1.5 - Bài tập chương 1 **BÀI THỰC HÀNH SỐ 4**

**Bài 11:** Quản lý một Queue có số phần tử khá lớn, biến động. Mỗi phần tử có kiểu int (danh sách liên kết đơn)

- 11.1. Khai báo cấu trúc queue.
- 11.2. Viết thủ tục khởi tạo queue rỗng.
- 11.3. Viết thủ tục kiểm tra queue rỗng.
- 11.4. Viết thủ tục thêm một phần tử vào queue.
- 11.5. Viết thủ tục xóa một phần tử trong queue.

118

118

### 1.5 - Bài tập chương 1 **BÀI LÀM THÊM**

**Bài 12:** Quản lý một danh sách có số phần tử khá lớn, biến động. Mỗi phần tử có kiểu int. Thường có nhu cầu truy xuất phần tử đứng trước và phần tử đứng sau phần tử đang truy xuất. (Dùng cấu trúc danh sách liên kết đôi)

- 12.1. Khai báo cấu trúc danh sách.
- 12.2. Viết thủ tục khởi tạo danh sách rỗng.
- 12.3. Xuất các phần tử trong danh sách
- 12.4. Viết thủ tục thêm một phần tử vào đầu danh sách.

119

119

### 1.5 - Bài tập chương 1 **BÀI LÀM THÊM**

- 12.5. Viết thủ tục thêm một phần tử vào cuối danh sách.
- 12.6. Viết thủ tục xóa phần tử đầu danh sách.
- 12.7. Viết thủ tục xóa phần tử cuối danh sách.
- 12.8. Viết thủ tục tìm một phần tử trong danh sách. Nếu tìm thấy, xóa phần tử này.
- 12.9. Viết thủ tục tìm một phần tử có giá trị bằng với giá trị X hoặc gần nhất và lớn hơn phần tử nhập vào; Thêm một phần tử đứng trước phần tử tìm thấy.

120

120

1.5 - Bài tập chương 1  
**BÀI TẬP NÂNG CAO**

**Bài 13:** Dùng cấu trúc danh sách đặc quản lý một đa thức

- 13.1. Khai báo cấu trúc danh sách
- 13.2. Viết thủ tục nhập vào một đa thức
- 13.4. Viết thủ tục xuất đa thức
- 13.5. Viết thủ tục cộng hai đa thức
- 13.6. Viết thủ tục trừ hai đa thức
- 13.7. Viết thủ tục nhân hai đa thức
- 13.8. Viết thủ tục chia hai đa thức

121

121

1.5 - Bài tập chương 1  
**BÀI TẬP NÂNG CAO**

**Bài 14:** Dùng cấu trúc danh sách liên kết quản lý một đa thức

- 14.1. Khai báo cấu trúc danh sách
- 14.2. Viết thủ tục nhập vào một đa thức
- 14.4. Viết thủ tục xuất đa thức
- 14.5. Viết thủ tục cộng hai đa thức
- 14.6. Viết thủ tục trừ hai đa thức
- 14.7. Viết thủ tục nhân hai đa thức
- 14.8. Viết thủ tục chia hai đa thức

122

122

1.5 - Bài tập chương 1  
**BÀI TẬP NÂNG CAO**

**Bài 15:** Dùng cấu trúc danh sách đặc quản lý tập hợp các phần tử số nguyên.

- 15.1. Viết thủ tục xuất các phần tử thuộc tập hợp của hai danh sách
- 15.2. Viết thủ tục xuất các phần tử thuộc tập giao của hai danh sách
- 15.3. Viết thủ tục xuất danh sách phần bù của hai danh sách

123

123

1.5 - Bài tập chương 1  
**BÀI TẬP NÂNG CAO**

**Bài 16:** Dùng cấu trúc danh sách liên kết đơn quản lý tập hợp các phần tử số nguyên.

- 16.1. Viết thủ tục xuất các phần tử thuộc tập hợp của hai danh sách
- 16.2. Viết thủ tục xuất các phần tử thuộc tập hợp giao của hai danh sách
- 16.3. Viết thủ tục xuất danh sách phần bù của của hai danh sách.

124

124

## Hướng dẫn

- Tất cả sinh viên phải trả lời các **câu hỏi** của chương, làm **bài tập thực hành** tại phòng máy (**bài làm thêm** ở nhà, và **bài nâng cao** khuyến khích hoàn tất) và nộp bài qua LMS của trường.
- Câu hỏi chương 1 làm trên file WORD; trong bài làm ghi rõ họ tên, lớp, bài tập chương và các thông tin cần thiết.
- Khuyến khích sử dụng tiếng Anh trong bài tập.

⇒ **Ngày nộp:** trước khi học chương 4.

⇒ **Cách nộp:** sử dụng **github** để nộp bài, sau đó nộp lên LMS của trường.

125

## Tài liệu tham khảo

- **Lê Xuân Trường**, (Chương 1) *Cấu trúc dữ liệu*, NXB Trường Đại học Mở TP-HCM, 2017.
- **Đỗ Xuân Lôi**, *Cấu trúc dữ liệu và giải thuật*, Đại học quốc gia TP-HCM, 2004.
- **Dương Anh Đức**, *Giáo trình cấu trúc dữ liệu & giải thuật (Chương 3)*, 2010, ĐH KHTN TP.HCM
- **Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest, Clifford Stein**, (Chapter 10) *Introduction to Algorithms*, Third Edition, 2009.
- **Adam Drozdek**, (Chapter 3) *Data Structures and Algorithms in C++*, Fourth Edition, CENGAGE Learning, 2013.

126

## KẾT THÚC chương 1



**Trường Đại học Mở TP.HCM**  
Khoa Công Nghệ Thông Tin

127