

 **KỸ THUẬT LẬP TRÌNH**  
(PROGRAMMING TECHNIQUES)

**Chương 1:**

# TỔNG QUAN KỸ THUẬT LẬP TRÌNH

Khoa Công nghệ thông tin  
Bộ môn Công nghệ phần mềm  
-2022-

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

1

**Nội dung**

---

1. Tổng quan kỹ thuật lập trình
2. Một số kỹ thuật tinh chỉnh mã nguồn
3. Xử lý ngoại lệ
4. Thao tác trên tập tin
5. Quản lý bộ nhớ, con trỏ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

2

**Nội dung**

---

1. Tổng quan kỹ thuật lập trình
2. Một số kỹ thuật tinh chỉnh mã nguồn
3. Xử lý ngoại lệ
4. Thao tác trên tập tin
5. Quản lý bộ nhớ, con trỏ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

3

**1. Tổng quan về kỹ thuật lập trình**

---

Kỹ thuật thực thi một giải pháp phần mềm (CTDL + GT) dựa trên nền tảng một phương pháp luận (methodology) và một hoặc nhiều ngôn ngữ lập trình phù hợp với yêu cầu đặc thù của ứng dụng.

Kỹ thuật lập trình:

- = Tư tưởng thiết kế + Kỹ thuật mã hóa
- = Cấu trúc dữ liệu + Giải thuật + Ngôn ngữ lập trình

*Algorithms + Data Structures = Programs  
(Niklaus Wirth)*

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

4

**Một số tiêu chí chương trình “tốt”**

---

**Tin cậy:**

- Chương trình chạy đúng.

**Hiệu suất:**

- Chương trình nhỏ gọn (sử dụng ít bộ nhớ).
- Tốc độ nhanh (sử dụng ít thời gian).

**Hiệu quả:**

- Thời gian lập trình ngắn.
- Khả năng bảo trì dễ dàng.
- Giá trị “tài sử dụng code” lớn.
- Sử dụng đơn giản, thân thiện.
- Nhiều chức năng tiện ích.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

5

**Ví dụ**

---

SV hãy đề xuất các giải thuật có thể có để giải quyết bài toán sau:

Viết hàm tính giai thừa của một số nguyên  $n$  khi cho trước nguyên mẫu hàm như sau:

`long tinhGiaiThua (int n);`



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

6

## Ví dụ

**Cài đặt**

```

Phương pháp đệ quy
long tinhGiaiThua(int n)
{
    if (n <= 1)
        return 1;
    return n * tinhGiaiThua(n-1);
}

Phương pháp lặp xác định
long tinhGiaiThua(int n)
{
    long kq = 1;
    for(int i = 2; i <= n; i++)
        kq *= i;
    return kq;
}

Phương pháp lặp không xác định
long tinhGiaiThua(int n)
{
    long kq = 1;
    while (n > 1)
        kq *= n--;
    return kq;
}

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 7

## Làm thế nào để lập trình tốt?

**Học cách tư duy và phương pháp lập trình:**

- Tư duy logic, có cấu trúc, hướng đối tượng, tổng quát.
- Hiểu về cấu trúc dữ liệu và giải thuật.

**Hiểu về máy tính:**

- Tương tác giữa CPU, chương trình và bộ nhớ
- Cơ chế quản lý bộ nhớ

**Nắm vững ngôn ngữ lập trình:**

- Biết rõ các ưu điểm và hạn chế của ngôn ngữ
- Kỹ năng lập trình (đọc, viết)

**Tự rèn luyện + Phát triển sự sáng tạo**

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 8

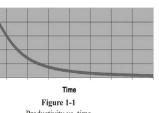
## Nội dung

1. Tổng quan kỹ thuật lập trình
2. Một số kỹ thuật tinh chỉnh mã nguồn
3. Xử lý ngoại lệ
4. Thao tác trên tập tin
5. Quản lý bộ nhớ, con trỏ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 9

## 2. Một số kỹ thuật tinh chỉnh mã nguồn

Vấn đề xảy ra khi viết một chương trình chạy **đúng**, nhưng mã nguồn **lộn xộn**?



Code bẩn (Messy Code) sẽ làm cho công việc chậm lại một cách đáng kể. Lúc đầu, team thực hiện rất nhanh khi bắt đầu dự án. Mỗi lần thay đổi, lại phá vỡ vài phần khác của Code. Mỗi lần thêm hoặc sửa, hệ thống lại "lộn xộn" xoáy vào nhau", ... Qua thời gian thì sự lộn xộn đó lớn dần lên, và không có cách nào dọn sạch nó. Năng suất theo đó giảm dần và tiệm cận về 0 (Năng suất giảm → Management thêm nhân viên → Nhân viên mới không hiểu hệ thống → Làm và làm → Áp lực → Kết quả không đạt yêu cầu).

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 10

## 2. Một số kỹ thuật tinh chỉnh mã nguồn

### Một số qui tắc trong phong cách lập trình

- 2.1. Quy ước đặt tên
- 2.2. Quy tắc trình bày tổng thể chương trình
- 2.3. Quy tắc trình bày dòng lệnh
- 2.4. Quy tắc chú thích chương trình

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 11

## 2.1. Quy ước đặt tên

### 2.1.1. Đặt tên có ý nghĩa (tránh quá ngắn hoặc quá dài).

- Tên biến, tên hàm → viết thường
- Tên hằng số → VIẾT HOA
- Tên lớp (class), cấu trúc (struct) → Viết Hoa Ký Tự Đầu.

Ví dụ:

```

int x;
#define MAX_SIZE 100
const float PI = 3.14;
class NhanVien

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 12

## Khuyến nghị

### Tên biến:

- Viết thường ký tự đầu tiên và viết hoa các ký tự đầu từ tiếp theo.
- Đặt tên bằng danh từ hoặc cụm danh từ.

### Tên hàm:

- Viết thường ký tự đầu tiên và viết hoa các ký tự đầu từ tiếp theo.
- Đặt tên bằng động từ hoặc cụm động từ, có động từ chính thể hiện ý nghĩa của hàm

### Tên lớp/cấu trúc

- Viết hoa các ký tự đầu từ.
- Đặt tên bằng danh từ hoặc cụm danh từ.

### Tên hàng số

- Viết hoa tất cả các ký tự
- Đặt tên bằng danh từ hoặc cụm danh từ.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

13

## 2.1. Quy ước đặt tên

Hai chuẩn quy ước đặt tên phổ biến:

- Lạc Đà (CamelCase notation)

- Hung-ga-ri (Hungarian notation)

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

14

## Một số quy ước

**Cú pháp Hungary:** thêm tiền tố chứa kiểu dữ liệu vào tên biến.

Tiền tố	Kiểu dữ liệu	Ví dụ minh họa
b	bool	bool bEmpty, bChecked ;
c	char	char cInChar, cOutChar ;
str/s	String	string strFirstName, strIn, strOut ;
i/n	integer	int iCount, nNumElement ;
li	long integer	long liPerson, liStars ;
f	float	float fPercent ;
d	double	double dMiles, dFraction ;
if	Input file stream	ifstream ifinFile ;
of	Output file stream	ofstream ofOutFile ;
S	Struct	struct sPoint{...} ;
C	Class	class CStudent, CPerson

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

15

## 2.1. Quy ước đặt tên

**Cú pháp CamelCase (Lạc đà):** Viết Hoa Ký Tự Đầu Từ

Phân loại:



lowerCamelCase



UpperCamelCase

Ví dụ:

- isEmpty
- hoTen

- IsEmpty
- HoTen

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

16

## 2.1. Quy ước đặt tên

**Cú pháp Hung-ga-ri:**

Tiền tố: viết thường + Phần còn lại: Hoa Kí Tự Đầu (thông tin).

Phân loại:

**Hướng hệ thống**  
(Systems Hungarian notation)

=> kiểu dữ liệu

- arru8NumberList: biến là một mảng các số nguyên 8 bit không dấu ("arru8" – array of unsigned 8-bit integers)
- szName: biến là một chuỗi có ký tự kết thúc ('\0' – zero-terminated string)

**Hướng ứng dụng**  
(Apps Hungarian notation)

=> mục đích sử dụng

- rwPosition: biến thể hiện một dòng ("rw" – row)
- strName: biến thể hiện một chuỗi chứa một tên ("str" – string).

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

17

## 2.1. Quy ước đặt tên

**Tên Hàm**

- Viết theo camelCase (hoặc viết thường, các từ cách nhau " ")
- Là động từ - phần ánh hưởng chức năng hàm thực hiện/giá trị trả về.

Ví dụ:

```
int SoLonNhat(int a[], int n); // Không rõ hành động
int TimSoLonNhat(int a[], int n); // Sai quy tắc đặt tên
int timSoLonNhat(int a[], int n); // OK
int tim_so_lon_nhat(int a[], int n); // OK
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

18

## 2.2. Quy tắc trình bày tổng thể chương trình

### Mô-đun hoá:

- Chương trình nên được tách thành nhiều **mô-đun** (đơn thể), mỗi mô-đun thực hiện một công việc **độc lập** với nhau.
- ➡ Chương trình **dễ đọc** và **bảo dưỡng** vì không phải đọc và nhớ nhiều các đoạn lệnh nằm rải rác.

## 2.2. Quy tắc trình bày tổng thể chương trình

### Bổ cục

Chương trình nên thực hiện theo trình tự như sau (đối với C):

1. Khai báo thư viện `#include` trên cùng
2. Khai báo hằng số `#define / const`.
3. Khai báo các biến toàn cục, class hay struct.
4. Khai báo các nguyên mẫu hàm
5. Hàm main(..)
6. Viết các thân hàm

**Lưu ý:** Tránh những thay đổi bất chợt do sử dụng **goto** hay **continue**.

```
// Thông tin chương trình, tác giả...
//...
//-----[Khai báo thư viện
#include<stdio.h>
#include<conio.h>
//-----]

//-----[Khai báo biến toàn cục, hàng số, cấu trúc - struct (nếu có)
//...]

//-----[Khai báo nguyên mẫu hàm - prototype
//...]

//-----[Hàm main
void main()
{
    //...
}
//-----[Thân các hàm con
//...]
```

## 2.2. Quy tắc trình bày tổng thể chương trình

### Hạn chế dùng biến toàn cục:

Khi nhiều hàm cùng sử dụng một biến toàn cục, việc thay đổi giá trị biến toàn cục của một hàm nào đó có thể dẫn đến những thay đổi không mong muốn ở các hàm khác.

Biến toàn cục sẽ làm cho các hàm trong chương trình giảm tính độc lập với nhau.

## 2.3. Quy tắc trình bày dòng lệnh

Câu lệnh phải thể hiện đúng cấu trúc chương trình bằng cách sử dụng hợp lý thụt đầu dòng (tab)

Không nên	Nên
<pre>void docFile (SV a[], int &amp;n) {     ifstream in;     char* filename="filein.txt";     in.open (filename);     in&gt;&gt;n;     for(int i=0; i &lt; n;i++)     {         in&gt;&gt;a[i].Masv;         in&gt;&gt;a[i].hoten;         in&gt;&gt;a[i].diem;     } }</pre>	<pre>void docFile (SV a[], int &amp;n) {     ifstream in;     char* filename = "filein.txt";     in.open (filename);     in &gt;&gt; n;     for (int i = 0 ; i &lt; n ; i++)     {         in &gt;&gt; a[i].Masv;         in &gt;&gt; a[i].hoten;         in &gt;&gt; a[i].diem;     } }</pre>

## 2.3. Quy tắc trình bày dòng lệnh

**Sử dụng khoảng trắng:** chương trình sẽ dễ nhìn hơn

Không nên	Nên
<pre>int iCount = 0 ; for(int i=0;i&lt;n;i++) {     iCount++; } cout&lt;&lt;"Ket qua la."&lt;&lt;iCount;</pre>	<pre>int iCount = 0 ; for (int i = 0 ; i &lt; n ; i++) {     iCount++; } cout &lt;&lt; "Ket qua la." &lt;&lt; iCount;</pre>

**Tránh viết nhiều lệnh trên một dòng.**

Không nên	Nên
<pre>if(a&gt;5){b=a; a++}</pre>	<pre>if( a &gt; 5 ) {     b = a;     a++; }</pre>

## 2.3. Quy tắc trình bày dòng lệnh

### Sử dụng biến:

Các biến không nên được sử dụng lại với nhiều nghĩa khác nhau trong cùng một hàm.

```
for (i = 0; i < n; i++) // n là số lần lặp
...
for (n = 0; n < 10; n++) // n là biến điều khiển
...
```

## 2.3. Quy tắc chú thích

Chú thích (comment) giúp việc **đọc hiểu** và **chỉnh sửa** code dễ dàng hơn.

### Viết chú thích

- Nên chú thích ngắn gọn nhưng đầy đủ, dễ hiểu.
- Chú thích ngắn
  - => Nên đặt ngang với dòng lệnh cần chú thích.
- Chú thích dài hoặc cho cả một đoạn lệnh
  - => Nên đặt trên một dòng riêng, phía trên câu lệnh/khoi lệnh cần chú thích.

## 2.3. Quy tắc chú thích

### Ví dụ:

```
//Ham cap phat 1 node
Node* createNode(int x)
{
    Node* p = new Node; //Cap phat vung nho cho mot nut moi p
    //Neu cap phat thanh cong thi gan du lieu cho node
    if (p != NULL)
    {
        p->info = x;
        p->next = NULL;
    }
    return p;
}
```

## 2.3. Quy tắc chú thích

**Lưu ý:** Một số vấn đề cần xác định khi viết chú thích:

- + Mục đích của hàm là gì?
- + Biến đầu vào của hàm (tham số) là gì?
- + Các điều kiện ràng buộc của các biến đầu vào (nếu có)?
- + Kết quả trả về của hàm là gì?
- + Các ràng buộc của kết quả trả về (nếu có).
- + Ý tưởng giải thuật các thao tác trong hàm.

## 2.3. Quy tắc chú thích

### Không lạm dụng việc chú thích.

=> Chú thích tràn lan ngay cả với câu lệnh đơn giản đôi khi còn làm cho chương trình khó đọc hơn.

### Ví dụ:

```
Không nên chú thích câu lệnh đơn giản này
//Nếu nhiệt độ vượt quá mức qui định thì phải cảnh báo
if (nhietDo > nhietDoCB)
    cout<< "Nhiệt độ vượt mức qui định" ;
//i là biến chạy trong vòng lặp for để xác định các chỉ
số phần tử mảng a.
for (int i = 0 ; i<n ; i++)
    cout<< a[i] ;
```

## KISS - Nguyên tắc quan trọng trong lập trình!

- ❖ Keep it **simple**, **stupid**
- ❖ Keep it **short** and **simple**
- ❖ Keep it **simple** and **straightforward**

**Nguyên tắc KISS:** **đơn giản là mục tiêu trong thiết kế và lập trình, tránh sự phức tạp không cần thiết.**

## 1.2.2. Phong cách viết mã nguồn (tt)

Nên viết biểu thức điều kiện mang tính tự nhiên: biểu thức nên viết dưới dạng khẳng định. Vì việc viết biểu thức dạng phủ định sẽ làm khó hiểu.

Không nên	Nên
if (!(i >= a)    !(i >= b))	if ((i >= a) && (i >= b))

## 1.2.2. Phong cách viết mã nguồn (tt)

Viết các lệnh rõ ràng, tối ưu sự thực thi mã nguồn

Số	Không nên
1	int i = 0; while (i < n) { ... i++; }
2	i = i + 3 ; i += 3 ;
3	return (a + b * c) ;
4	if (a > b) return f(a); else return g(b);
5	if (a > b) return true ; else return false ;
6	return p.next == NULL ? NULL : p.next ;

## 1.2.2. Phong cách viết mã nguồn (tt)

Viết các lệnh rõ ràng, tối ưu sự thực thi mã nguồn

Số	Không nên	Nên
1	int i = 0; while (i < n) { ... i++; }	for(int i = 0; i < n; i++) { ... }
2	i = i + 3 ; i += 3 ;	i += 3 ; i += 3 ;
3	return (a + b * c) ;	return a + b * c ;
4	if (a > b) return f(a); else return g(b);	return a > b ? f (a) : g(b) ;
5	if (a > b) return true ; else return false ;	return a > b ;
6	return p.next == NULL ? NULL : p.next ;	return p.next ;

Số	Không nên
7	if (a > b) return people[current_person].rela tives.next. data[x] = f(a); else return people[current_person].rela tives.next. data[x] = f(b);
8	int countNodes (Node *root) { if (root->left == NULL) if (root->right == NULL) return 1; else return 1 + countNodes (root->right); else if (root->right == NULL) return 1 + countNodes (root->left); else return 1 + countNodes (root->left) + countNodes (root->right); }

Số	Không nên	Nên
7	if (a > b) return people[current_person].rela tives.next. data[x] = a > b ? f(a) : f(b) ;	people[current_person].rela tives.next. data[x] = a > b ? f(a) : f(b) ;
8	int countNodes (Node *root) { if (root->left == NULL) if (root->right == NULL) return 1; else return 1 + countNodes (root->right); else if (root->right == NULL) return 1 + countNodes (root->left); else return 1 + countNodes (root->left) + countNodes (root->right); }	int countNodes (Node *root) { if (root == NULL) 0 : 1 + countNodes (root->left) + countNodes (root->right); }

Số	Không nên
9	F = sqrt (dx * dx + dy * dy) + (sqrt (dx * dx + dy * dy) * sqrt (dx * dx - sqrt (dy * dy) ) ;
10	for ( int i = 0 ; i < strlen(str) ; i++) ...
11	found = FALSE; for ( int i = 0 ; i < 10000; i++) { if (list[i] == -99) found = TRUE; } if (found) cout<< "this is a - 99";

Số	Không nên	Nên
9		<pre>A = dx * dx + dy * dy; C = sqrt(dx * dx + dy * dy) * sqrt(dx * dx + dy * dy); B = C + (C + sqrt(A) - sqrt(B));</pre>
10	for (int i = 0; i < strlen(str); i++) {     ... }	<pre>int n = strlen(str); for (int i = 0; i &lt; n; i++)</pre>
11	found = FALSE; for (int i = 0; i < 10000; i++) {     if (list[i] == -99)         found = TRUE; } if (found)     cout << "this is a -99";	<pre>found = FALSE; for (int i = 0; i &lt; 10000; i++) {     if (list[i] == -99)     {         found = TRUE;         break;     } } if (found)     cout &lt;&lt; "this is a -99";</pre>
12	for (int i = 0; i < n; i++) {     a[i] = 0;     for (i = 0; i < n; i++)         b[i] = 0; }	<pre>for (int i = 0; i &lt; n; i++) {     a[i] = 0;     b[i] = 0; }</pre>

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

37

## 1.2.2. Phong cách viết mã nguồn (tt)

**Chia nhỏ chương trình:** Trong lập trình nên sử dụng chiến lược "chia để trị", nghĩa là chương trình được chia nhỏ ra thành các chương trình con. Việc chia nhỏ ra thành các chương trình con làm tăng tính modun của chương trình và mang lại cho người lập trình khả năng tái sử dụng mã code.

**Hạn chế dùng biến toàn cục:** Xu hướng chung là nên hạn chế sử dụng biến toàn cục. Khi nhiều hàm cùng sử dụng một biến toàn cục, việc thay đổi giá trị biến toàn cục của một hàm nào đó có thể dẫn đến những thay đổi không mong muốn ở các hàm khác. Biến toàn cục sẽ làm cho các hàm trong chương trình không độc lập với nhau.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

38

## Bài tập

Bài 1. Hãy chỉnh sửa và tối ưu các đoạn chương trình sau:

```
#include <stdio.h>
#include <conio.h>
int main()
{
    int ia[50];
    int i;
    int in;
    int isum = 0;
    printf("Nhập vào giá trị n: ");
    scanf("%d", &n);
    //Nhập dữ liệu vào mảng
    for(i = 0; i < in; i++)
    {
        printf("Nhập vào phần tử thứ %d: ", i + 1);
        scanf("%d", &a[i]); //Nhập giá trị cho phần tử thứ i
    }
    //Tính tổng giá trị các phần tử
    for(i = 0; i < in; i++)
        isum = isum + ia[i]; //Cộng dồn tổng phần tử vào isum
    printf("Trung bình cộng: %.2f\n", (float) isum/in);
    getch();
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

39

## Nội dung

- Tổng quan kỹ thuật lập trình
- Một số kỹ thuật tinh chỉnh mã nguồn
- Xử lý ngoại lệ
- Thao tác trên tập tin
- Quản lý bộ nhớ, con trỏ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

40

## 3. Xử lý ngoại lệ

### Cài đặt:

Bài toán: Chia hai số nguyên

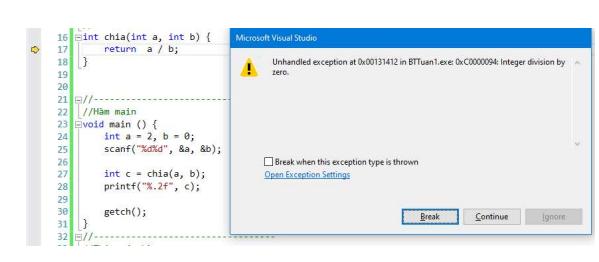
Input: a, b (nguyên)

Output: thương (thực)

Lỗi có thể xảy ra do quá trình nhập  
=> Phép chia cho 0

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

41



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

42

### 3. Xử lý ngoại lệ

Trong quá trình thực thi, chương trình có thể phát sinh lỗi:

1. Lỗi do mã nguồn (lập trình viên).
2. Lỗi do nhập sai (người sử dụng).
3. Do những điều không lường trước được.

Khi xảy ra lỗi, chương trình sẽ dừng lại và gửi một thông điệp lỗi  
=> Hay còn gọi là ném ra ngoại lệ (throw an exception)

### 3. Xử lý ngoại lệ

từ khóa: try, throw và catch

```
try {
    //các lệnh được kiểm tra lỗi trong khi thực thi
    throw exception; //từ khóa hoặc giá trị được ném ra khi phát hiện lỗi
}
catch () //các lệnh được thực hiện khi lỗi xảy ra
}
```

Vậy nên "ném ra" (throw) những ngoại lệ nào?  
và "bắt lại" (catch) ra sao?

### 3. Xử lý ngoại lệ

Trường hợp 1: Ngoại lệ là giá trị đang gây ra lỗi.

```
try {
    int age = 15;
    if (age > 18) {
        printf("Access granted - you are old enough.");
    } else {
        throw (age);
    }
}
catch (int myNum) { // bắt lấy giá trị lỗi được ném ra, đó là age
    printf("Access denied - You must be at least 18 years old.\n");
    printf("Age is: %d", myNum);
}
```

### 3. Xử lý ngoại lệ

Trường hợp 2: Ngoại lệ là tên lỗi được gây ra.

```
try {
    int age = 15;
    if (age > 18) {
        printf("Access granted - you are old enough.");
    } else {
        throw "Access denied - You must be at least 18 years old.";
    }
}
catch (const char* err) {
    puts(err);
}
```

### 3. Xử lý ngoại lệ

Trường hợp 3: Ngoại lệ là là một mã lỗi tham khảo.

```
try {
    int age = 15;
    if (age > 18) {
        printf("Access granted - you are old enough.");
    } else {
        throw 505;
    }
}
catch (int myNum) {
    printf("Access denied - You must be at least 18 years old.\n");
    printf("Error number: %d", myNum);
}
```

### 3. Xử lý ngoại lệ

Trường hợp không biết loại ngoại lệ được throw,  
=> sử dụng (...) trong catch.

```
try {
    int age = 15;
    if (age > 18) {
        printf("Access granted - you are old enough.");
    } else {
        throw "ac";
    }
}
catch (...) {
    printf("Access denied - You must be at least 18 years old.\n");
}
```

### 3. Xử lý ngoại lệ

**Bài toán:** Chia hai số nguyên

Input: a, b (nguyên)

Output: thương (thực)

Lỗi có thể xảy ra do quá trình nhập  
=> Phép chia cho 0

```
Cài đặt:
double chia(int a, int b) {
    if( b == 0 ) {
        throw "Phép chia cho 0";
    }
    return (double) a / b;
}

void main () {
    int a = 2, b = 0;
    double c = 0;
    try {
        c = chia(a, b);
        printf("%.2lf", c);
    } catch (const char* msg) {
        printf("Error: %s", msg);
    }
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

49

### 3. Xử lý ngoại lệ (tt): trường hợp có nhiều catch

```
try
{
    // phan code duoc bao ve
}
catch( ten_Exception e1 )
{
    // day la khoi catch
}
catch( ten_Exception e2 )
{
    // day la khoi catch
}
catch( ten_Exception eN )
{
    // day la khoi catch
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

50

### Ví dụ

Xét 2 mảng 1 chiều  
a và b. Tính và xuất  
kết quả các phép  
chia của phần tử  
mảng a cho phần tử  
mảng b. Hãy xét các  
trường hợp ngoại lệ  
có thể có trong bài  
toán này.

```
void main()
{
    int a[4]={4,5,6,3,9};
    int na = 5;
    int b[4]={2,1,9};
    int nb = 3;
    for(int i=0; i<na; i++)
        if(i>nb)
            throw "Ngoai le 1"; //ném ra exception dạng chuỗi
        else
            if(b[i]==0)
                throw 501; //Ném ra exception dạng mã lỗi
            else
                printf("%.5lf", 1.0*a[i]/b[i]);
    catch(const char* ex1)
    {
        printf("\nMang b không có phần tử");
    }
    catch(int num)
    {
        printf("\nPhan tu mang b là 0");
    }
    getch();
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

51

### Bài tập

- Nhập 3 số nguyên a, b, c. Xuất kết quả c/(a-b).
- Cho mảng một chiều a chứa n số nguyên. Viết hàm
  - Xóa phần tử tại vị trí k
  - Thêm phần tử x tại vị trí k.
- Viết chương trình thực hiện nhập thông tin nhân viên: họ tên, ngày sinh, giới tính. Hãy tính và xuất tháng/năm nghỉ hưu của nhân viên theo quy định:
  - Nữ: 60 tuổi
  - Nam: 62 tuổi

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

52

### Nội dung

- Tổng quan kỹ thuật lập trình
- Một số kỹ thuật tinh chỉnh mã nguồn
- Xử lý ngoại lệ
- Thao tác trên tập tin
- Quản lý bộ nhớ, con trỏ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

53

### 4.1. Nhu cầu

#### Vấn đề:

- Chương trình có quá nhiều dữ liệu input/output trong mỗi lần thực thi chương trình. Cần tiết kiệm thời gian cho việc nhập/xuất.
- C lưu dữ liệu (biến, mảng, cấu trúc, ...) trong bộ nhớ RAM.
- Kích thước dữ liệu bị hạn chế và chỉ lưu trữ tạm thời
  - Nhập: bàn phím. / Xuất: màn hình.
  - Lưu trữ: trong RAM.
- Không giải quyết được với số lượng dữ liệu lớn.

#### Nhu cầu:

Lưu trữ dữ liệu sao cho **vẫn còn khi kết thúc chương trình**,  
có thể **sử dụng nhiều lần** và **kích thước không hạn chế**.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

54

## 4.2. Tập tin

### Khái niệm

- **Tập hợp thông tin** (dữ liệu) được tổ chức theo một dạng nào đó với một tên xác định.
- Một **dãy byte liên tục** (ở góc độ lưu trữ).
- Được lưu trữ trong các thiết bị lưu trữ ngoài như đĩa mềm, đĩa cứng, USB, ...
- **Tồn tại khi chương trình kết thúc.**
- **Kích thước không hạn chế** (tùy vào thiết bị lưu trữ)
- Cho phép đọc dữ liệu (**thiết bị nhập**) và ghi dữ liệu (**thiết bị xuất**).

## 4.2. Tập tin

### Phân loại:

- Tập tin kiểu văn bản (ứng với stream văn bản).
- Tập tin kiểu nhị phân (ứng với stream nhị phân).

## 4.2. Tập tin

### Tập tin kiểu văn bản (stream văn bản)

- Dãy các dòng kế tiếp nhau.
- Mỗi dòng dài tối đa 255 ký tự và kết thúc bằng ký hiệu cuối dòng (**end\_of\_line**).
- Dòng không phải là một chuỗi vì không được kết thúc bởi ký tự '**\0**'.
- Khi ghi '**\n**' được chuyển thành cặp ký tự **CR** (về đầu dòng, mã ASCII 13) và **LF** (qua dòng, mã ASCII 10).
- Khi đọc thì cặp **CR-LF** được chuyển thành '**\n**'.

## 4.2. Tập tin

### Tập tin kiểu nhị phân (stream nhị phân)

- Dữ liệu được **đọc** và **ghi** một cách chính xác, không có sự chuyển đổi nào cả.
- Ký tự kết thúc chuỗi '**\0**' và **end\_of\_line** không có ý nghĩa là cuối chuỗi và cuối dòng mà **được xử lý** như mọi ký tự khác.

## 4.3. Thao tác trên tập tin

### Mở file



### Xử lý (Đọc / Ghi)



### Đóng file

#### 1. Mở tập tin:

Tạo một stream nối kết với tập tin cần mở, stream được quản lý bởi biến con trỏ đến cấu trúc FILE.

#### 2. Sử dụng tập tin

- 2.1. Đọc dữ liệu từ tập tin vào chương trình.
- 2.2. Ghi dữ liệu từ chương trình vào tập tin.

#### 3. Đóng tập tin

Ngắt kết nối với tập tin.

## 4.3. Thao tác trên tập tin

```
FILE *fopen(const char* fileName, const char* mode)
```

Mở tập tin có tên (đường dẫn) là chứa trong **fileName** với kiểu mở **mode**.

- ◆ Thành công: con trỏ kiểu cấu trúc **FILE**
- ◆ Thất bại: **NULL** (không mở được tập tin để đọc)

Ví dụ:

```
FILE* f = fopen("file.txt", "rt");
if(f == NULL)
    printf("Khong mo duoc tap tin de doc du lieu!");
```

### 4.3. Thao tác trên tập tin

Mode	Ý nghĩa	Nếu file không tồn tại
r	Mở file chỉ cho phép đọc.	Nếu file không tồn tại, fopen() trả về NULL.
rb	Mở file chỉ cho phép đọc dưới dạng nhị phân.	Nếu file không tồn tại, fopen() trả về NULL.
w	Mở file chỉ cho phép ghi.	Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động.
wb	Mở để viết ở chế độ nhị phân.	Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

61

### 4.3. Thao tác trên tập tin

Mode	Ý nghĩa	Nếu file không tồn tại
a	Mở file ở chế độ ghi “append”. Tức là sẽ ghi vào cuối của nội dung đã có.	Nếu file không tồn tại, nó sẽ được tạo tự động.
ab	Mở file ở chế độ ghi nhị phân “append”. Tức là sẽ ghi vào cuối của nội dung đã có.	Nếu file không tồn tại, nó sẽ được tạo tự động.
r+	Mở file cho phép cả đọc và ghi.	Nếu file không tồn tại, fopen() trả về NULL.
rb+	Mở file cho phép cả đọc và ghi ở dạng nhị phân.	Nếu file không tồn tại, fopen() trả về NULL.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

62

### 4.3. Thao tác trên tập tin

Mode	Ý nghĩa	Nếu file không tồn tại
w+	Mở file cho phép cả đọc và ghi.	Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động.
wb+	Mở file cho phép cả đọc và ghi ở dạng nhị phân.	Nếu file đã tồn tại, nội dung sẽ bị ghi đè. Nếu file không tồn tại, nó sẽ được tạo tự động.
a+	Mở file cho phép đọc và ghi “append”.	Nếu file không tồn tại, nó sẽ được tạo tự động.
ab+	Mở file cho phép đọc và ghi “append” ở dạng nhị phân.	Nếu file không tồn tại, nó sẽ được tạo tự động.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

63

### 4.3. Thao tác trên tập tin

mode	Ý nghĩa
b	Mở tập tin kiểu <b>nhi phân</b> (binary)
t	Mở tập tin kiểu <b>văn bản</b> (text) ( <b>mặc định</b> )
r	Mở tập tin <b>chỉ để đọc</b> dữ liệu từ tập tin. Trả về NULL nếu không tìm thấy tập tin.
w	Mở tập tin <b>chỉ để ghi</b> dữ liệu vào tập tin. Tập tin sẽ được tạo nếu chưa có, ngược lại dữ liệu trước đó sẽ bị xóa.
a	Mở tập tin <b>chỉ để thêm</b> (append) dữ liệu vào cuối tập tin. Tập tin sẽ được tạo nếu chưa có.
r+	Giống mode r và <b>bổ sung</b> thêm tính năng ghi dữ liệu và tập tin sẽ được tạo nếu chưa có.
w+	Giống mode w và <b>bổ sung</b> thêm tính năng đọc.
a+	Giống mode a và <b>bổ sung</b> thêm tính năng đọc.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

64

### 4.3. Thao tác trên tập tin

#### Hàm nhập xuất tập tin

Thao tác	Tập tin Văn bản			Tập tin nhị phân
	(Ký tự)	(Đòng)	(Định dạng)	
Đọc	fgetc	fgets	fscanf	fread
Ghi	fputc	fputs	fprintf	fwrite

#### Hàm quản lý tập tin

Thao tác	Hàm
Xoá	remove
Đổi tên	rename

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

65

### 4.3. Thao tác trên tập tin

```
int fprintf(FILE *fp, char *fnt, ...)
```

Đọc dữ liệu có chuỗi định dạng fnt (giống hàm scanf) từ stream fp.

Nếu fp là **stdout** thì hàm giống printf.

- Thành công: trả về số byte ghi được.
- Thất bại: trả về **EOF** (có giá trị là -1, được định nghĩa trong **stdio.h**, sử dụng trong tập tin có kiểu văn bản)

Ví dụ:

```
int i = 2345; int c = 'N'; float f = 10.08;
FILE* fp = fopen("file.txt", "wt");
if (fp != NULL)
    fprintf(fp, "%d %c %.2f\n", i, c, f);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

66

### 4.3. Thao tác trên tập tin

```
int fscanf(FILE *fp, char *fnt, ...)
```

Đọc dữ liệu có chuỗi định dạng **fnt** (giống hàm scanf) từ stream **fp**. Nếu **fp** là **stdin** thì hàm giống scanf.

- Thành công: trả về số thành phần đọc và lưu trữ được.
- Thất bại: trả về **EOF**.

Ví dụ:

```
int i;
FILE* fp = fopen("file.txt", "rt");
if (fp != NULL)
    fscanf(fp, "%d", &i);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

67

### 4.3. Thao tác trên tập tin

```
int fscanf(FILE *fp, char *fnt, ...)
```

Đọc chuỗi thông tin phức hợp

%[chuỗi]: đọc cho đến khi không gặp ký tự nào trong chuỗi thì dừng.

%[^chuỗi]: đọc cho đến khi gặp một trong những ký tự trong chuỗi thì dừng.

Ví dụ: Một tập tin chứa nhiều dòng, mỗi dòng là thông tin 1 sinh viên theo định dạng sau:

< MSSV>, < Họ Tên>-< Tên>(< Khoa>) tab < ĐTB>

Ví dụ: 02345, Nguyễn Văn An(CNTT) 8.5

```
fscanf(fp, "%[^, ] , %[^-]-%[^()(%[^)])]\t%.1f",
&sv.mssv, &sv.hoTen, &sv.ten, &sv.khoa, &sv.diemTB);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

68

### 4.3. Thao tác trên tập tin

```
int fwrite(void *buf, int size, int count, FILE *fp)
```

Ghi **count** mẫu tin có kích thước mỗi mẫu tin là **size** (byte) từ vùng nhớ **buf** vào stream **fp** (theo kiểu nhị phân).

- Thành công: trả về số lượng mẫu tin (không phải số lượng byte) đã ghi.
- Thất bại: số lượng nhỏ hơn **count**.

Ví dụ:

```
int a[] = {1, 2, 3};
FILE* fp = fopen("file.dat", "wb");
if (fp != NULL)
    fwrite(a, sizeof(int), 3, fp);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

69

### 4.3. Thao tác trên tập tin

```
int fread(void *buf, int size, int count, FILE *fp)
```

Đọc **count** mẫu tin có kích thước mỗi mẫu tin là **size** (byte) vào vùng nhớ **buf** từ stream **fp** (theo kiểu nhị phân).

- Thành công: trả về số lượng mẫu tin (không phải số lượng byte) đã đọc.
- Thất bại: số lượng nhỏ hơn **count** khi kết thúc stream **fp** hoặc gặp lỗi.

Ví dụ:

```
int a[5];
FILE* fp = fopen("file.dat", "rb");
if (fp != NULL)
    fread(a, sizeof(int), 3, fp);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

70

### 4.3. Thao tác trên tập tin

```
int fclose(FILE *fp)
```

Đóng stream **fp**.

Dữ liệu trong stream **fp** sẽ được ghi hết lên đĩa trước khi đóng.

- Thành công: trả về 0.
- Thất bại: trả về **EOF**.

Ví dụ:

```
FILE* fp = fopen("file.txt", "rt");
...
fclose(fp);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

71

### Bài tập

1. Cho mảng số nguyên arr, với n là số phần tử của mảng. Hãy viết hàm:  
a. Ghi mảng arr ra thành tập tin văn bản.

b. Đọc dữ liệu từ tập tin văn bản vào mảng arr.

2. Cho mảng arr - mỗi phần tử lưu thông tin một Sinh viên, n là số phần tử của mảng. Thông tin một sinh viên gồm: mã số sinh viên, họ tên và điểm trung bình. Hãy viết hàm:

a. Ghi mảng arr ra thành tập tin nhị phân.

b. Đọc dữ liệu từ tập tin nhị phân vào mảng arr.

72

### 4.3. Thao tác trên tập tin

file.txt  
5  
2 1 3 4 5

**Ví dụ:**  
Đọc tập tin nhị phân

```
void readTextFile(int* arr[], int& n) {
    // mở file để đọc
    FILE* fi = fopen("rt");
    // đọc dữ liệu
    fscanf(fi, "%d", &n); // đọc số lượng n
    for (int i = 0; i < n; i++) {
        fscanf(fi, "%d", &arr[i]); // đọc từng phần tử
    }
    // đóng file
    fclose(fi);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

73

### 4.3. Thao tác trên tập tin

Ghi tập tin văn bản  
file.txt  
5  
2 1 3 4 5

**Ví dụ:**

```
void writeTextFile(int arr[], int& n, char* fileName) {
    // mở file để đọc
    FILE* fo = fopen(fileName, "wt");
    // đọc dữ liệu
    fprintf(fo, "%d\n", &n); // ghi số lượng phần tử
    for (int i = 0; i < n; i++) {
        // gán giá trị phần tử cho mảng arr
        fprintf(fo, "%d ", &arr[i]);
    }
    // đóng file
    fclose(fo);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

74

### 4.3. Thao tác trên tập tin

**Ví dụ:** Ghi tập tin nhị phân

```
struct SinhVien {
    char mssv[11];
    char ten[31];
    float diemTB;
};
```

```
void writeBinFile(SinhVien dssv[], int &n, char* fileName) {
    FILE* fo = fopen(fileName, "wb");
    if (fo == NULL) return;
    fwrite(&n, sizeof(int), 1, fo);
    for (int i = 0; i < n; i++) {
        fwrite(&dssv[i], sizeof(SinhVien), 1, fo);
    }
    fclose(fo);
}
```

```
struct SinhVien {
    char mssv[11];
    char ten[31];
    float diemTB;
};
```

**Ví dụ:** Đọc tập tin nhị phân

```
void readBinFile(SinhVien dssv[], int &n, char* fileName) {
    FILE* fi = fopen(fileName, "rb");
    if (fi == NULL) return;
    fread(&n, sizeof(int), 1, fi);
    for (int i = 0; i < n; i++) {
        fread(&dssv[i], sizeof(SinhVien), 1, fi);
    }
    fclose(fi);
}
```

### 4.3. Thao tác trên tập tin

#### Con trỏ chỉ vị (position indicator)

##### Khái niệm:

Được tạo tự động khi mở tập tin.  
Xác định nơi diễn ra việc đọc/ghi trong tập tin

##### Vị trí con trỏ chỉ vị:

Khi tập tin chưa mở: ở đầu tập tin (giá trị 0).  
Khi mở tập tin:  
Ở cuối tập tin khi mở để chèn (mode a hay a+)  
Ở đầu tập tin (hay giá trị 0) khi mở với các mode khác (w, w+, r, r+).

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

77

### 4.3. Thao tác trên tập tin

#### Truy xuất tuần tự & ngẫu nhiên

##### Truy xuất tuần tự (sequential access)

- Phải đọc/ghi dữ liệu từ vị trí con trỏ chỉ vị đến vị trí n-1 trước khi đọc dữ liệu tại vị trí n.
- Không cần quan tâm đến con trỏ chỉ vị do con trỏ chỉ vị tự động chuyển sang vị trí kế tiếp sau thao tác đọc/ghi dữ liệu.

##### Truy xuất ngẫu nhiên (random access)

- Có thể đọc/ghi tại vị trí bất kỳ trong tập tin mà không cần phải đọc/ghi toàn bộ dữ liệu trước đó.
- quan tâm đến con trỏ chỉ vị.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

78

### 4.3. Thao tác trên tập tin

```
void rewind(FILE *fp)
```

Đặt lại vị trí con trỏ chỉ vị về đầu (byte 0) tập tin fp.

Ví dụ:

```
FILE* fp = fopen("file.txt", "w+");
fprintf(fp, "123456789");
rewind(fp);
fprintf(fp, "aaa");
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

79

### 4.3. Thao tác trên tập tin

```
int fseek(FILE *fp, long offset, int origin)
```

Đặt vị trí con trỏ chỉ vị trong stream fp với vị trí offset so với cột mốc origin (SEEK\_SET hay 0: đầu tập tin; SEEK\_CUR hay 1: vị trí hiện tại; SEEK\_END hay 2: cuối tập tin)

- Thành công: trả về 0.
- Thất bại: trả về giá trị khác 0.

Ví dụ:

```
FILE* fp = fopen("a.txt", "w+");
fseek(fp, 0L, SEEK_SET); // ⇔ rewind(fp);
fseek(fp, 0L, SEEK_END); // cuối tập tin
fseek(fp, -2L, SEEK_CUR); // lùi lại 2 vị trí
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

80

### 4.3. Thao tác trên tập tin

```
long ftell(FILE *fp)
```

Hàm trả về vị trí hiện tại của con trỏ chỉ vị (tính từ vị trí đầu tiên của tập tin, tức là 0) của stream fp.

- Thành công: trả về vị trí hiện tại của con trỏ chỉ vị.
- Thất bại: trả về -1L.

Ví dụ:

```
FILE* fp = fopen("a.txt", "rb");
fseek(fp, 0L, SEEK_END);
long size = ftell(fp);
printf("Kích thước tập tin là %ld\n", size);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

81

### 4.3. Thao tác trên tập tin

#### Khi đã biết kích thước tập tin

- Sử dụng fwrite để lưu n mẫu tin  
→ Kích thước = n \* sizeof (1 mẫu tin);

◦ Sử dụng [hàm fseek kết hợp hàm ftell](#)

#### Khi chưa biết kích thước tập tin

- Hằng số EOF (-1) ([chỉ cho tập tin văn bản](#))
  - while ((c = fgetc(fp)) != EOF) ...
- Hàm int feof(FILE \*fp) (cho cả 2 kiểu tập tin)
  - trả về số 0 nếu chưa đến cuối tập tin
  - trả về số khác 0 nếu đã đến cuối tập tin.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

82

### 4.3. Thao tác trên tập tin

#### Tập tin nhị phân

```
void writeBinFile2(SinhVien dssv[], int &n, char* fileName) {
    FILE* fo = fopen(fileName, "wb");
    if (fo == NULL) return;
    fwrite(dssv, sizeof(SinhVien), n, fo); //ghi 1 dssv
    fclose(fo);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

83

### 4.3. Thao tác trên tập tin

#### Tập tin nhị phân

```
void readBinFile2(SinhVien dssv[], int &n, char* fileName) {
    FILE* fi = fopen(fileName, "rb");
    if(fi == NULL) return;
    int i = 0;
    while(!feof(fi)) //trong khi chua doc den cuoi file
    {
        if(fread(&dssv[i], sizeof(SinhVien), 1, fi)==1) //doc 1 sv
            i++;
    }
    n = i; //cap nhat luong phan tu cua dssv
    fclose(fi);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

84

### 4.3. Thao tác trên tập tin

- Hàm nhập xuất tập tin
- Hàm quản lý tập tin

Thao tác	Hàm
Xoá	remove
Đổi tên	rename

### 4.3. Thao tác trên tập tin

```
int remove(const char *fileName)
```

Xóa tập tin xác định bởi `fileName`.

- Thành công: trả về 0.
- Thất bại: trả về -1.

Ví dụ:

```
if(remove("a.txt") == 0)
    printf("Tap tin a.txt da bi xoai!");
else
    printf("Khong xoa duoc tap tin!");
```

### 4.3. Thao tác trên tập tin

```
int rename(const char *oldName, const char *newName)
```

Đổi tên tập tin `oldName` thành `newName`.

- Thành công: trả về 0.
- Thất bại: trả về -1

Ví dụ:

```
if(rename("a.txt", "b.txt") == 0)
    printf("Doi ten thanh cong!");
else
    printf("Doi ten that bai!");
```

## Nội dung

1. Tổng quan kỹ thuật lập trình
2. Một số kỹ thuật tinh chỉnh mã nguồn
3. Xử lý ngoại lệ
4. Thao tác trên tập tin
5. Quản lý bộ nhớ, con trỏ

## 5. Quản lý bộ nhớ, con trỏ

### 5.1. Quản lý bộ nhớ

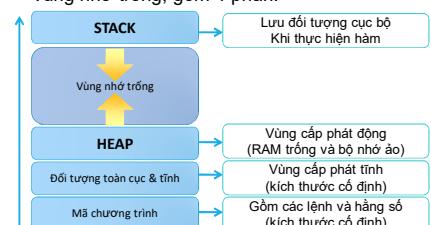
- 5.1.1. Cấu trúc chương trình trong bộ nhớ
- 5.1.2. Chuyển đổi kiểu
- 5.1.3. Cấp phát bộ nhớ động
- 5.1.2. Các thao tác trên khối nhớ

### 5.2. Con trỏ

- 5.2.1. Các thao tác trên con trỏ
- 5.2.2. Con trỏ và mảng một chiều.
- 5.2.3. Con trỏ và cấu trúc.

### Cấu trúc một Chương trình C trong bộ nhớ

• Toàn bộ tập tin chương trình sẽ được nạp vào bộ nhớ ở vùng nhớ trống, gồm 4 phần:



## Cấp phát bộ nhớ

Tính	Động
<ul style="list-style-type: none"> <li>Định nghĩa tại lúc biên dịch.</li> <li>Cấp phát lúc tạo liên kết.</li> <li>Giá trị ban đầu tùy theo NNLT.</li> <li>Tồn tại trong tầm vực.</li> </ul>	<ul style="list-style-type: none"> <li>Được gắn kết với một con trỏ.</li> <li>Phát sinh trong lúc thực thi.</li> <li>Không xác định giá trị ban đầu.</li> <li>Giải phóng khỏi bộ nhớ khi cần (do người lập trình).</li> </ul>

### Hạn chế:

Phải biết trước số lượng vùng nhớ cần phải dùng.  
→ tồn bộ nhớ, không thay đổi được kích thước, ...

### ↓

Cần giải phóng vùng nhớ khi không sử dụng.  
→ có thể dẫn đến rò rỉ vùng nhớ, nếu không cẩn thận.

## Chuyển đổi kiểu dữ liệu

Mọi đối tượng dữ liệu trong C đều có kiểu xác định

- Biến có kiểu: `char, int, float, double, ...`
- Con trỏ trả về kiểu: `char, int, float, double, ...`

Khi gặp một biểu thức với nhiều kiểu khác nhau?

- C tự động chuyển đổi kiểu (ép kiểu).
- Người sử dụng tự chuyển đổi kiểu.

## Chuyển đổi kiểu tự động (ngầm định)

Cú pháp: `<Biến> = <BT về phải>;`

BT ở vé phải luôn được tăng cấp (hay giảm cấp) tạm thời cho giống kiểu với Biến ở vé trái.

### Ví dụ:

```
int i;
float f = 1.23;
i = f; // → f tạm thời thành int
f = i; // → i tạm thời thành float
```

Lưu ý: Có thể làm mất tính chính xác của số nguyên khi chuyển sang số thực.



## Chuyển đổi tường minh (ép kiểu)

Cú pháp: `(<kiểu chuyển đổi>) <biểu thức>`

### Ý nghĩa:

Chủ động chuyển đổi kiểu (tạm thời) nhằm tránh những kết quả sai lầm.

### Ví dụ:

```
int x1 = 2, x2 = 4;
float f1 = x1 / x2;           // → f1 = 0.0
float f2 = (float)x1 / x2;   // → f2 = 0.5
float f3 = (float)(x1 / x2); // → f3 = 0.0
```

## 5. Quản lý bộ nhớ, con trỏ

### 5.1. Quản lý bộ nhớ

- Cấu trúc chương trình trong bộ nhớ
- Chuyển đổi kiểu
- Cấp phát bộ nhớ động
- Các thao tác trên khôi nhớ

### 5.2. Con trỏ

- Các thao tác trên con trỏ
- Con trỏ và mảng một chiều.
- Con trỏ và cấu trúc.

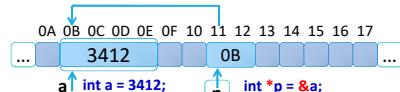
## Biến

### Khi khai báo biến

Trình biên dịch sẽ **dành riêng** một vùng nhớ với **địa chỉ duy nhất** để lưu trữ biến đó và **liên kết** địa chỉ ô nhớ với tên biến.

### Khi gọi tên biến, nó sẽ **truy xuất tự động** đến ô nhớ đã liên kết với tên biến

Ví dụ: `int a = 3412; // Giả sử địa chỉ OB (giá trị số ở hệ hexa)`



## Biến con trỏ - Pointer

- Con trỏ là biến dùng để lưu **địa chỉ** những **biến** khác hay **địa chỉ** **vùng nhớ hợp lệ** (*hay gọi là trỏ đến địa chỉ của biến khác*).
- Con trỏ **NULL** là con trỏ đặc biệt và không trỏ vào đâu cả.
- Toán tử để lấy **địa chỉ** của biến vào con trỏ: **&**  
 $\langle\text{tên biến con trỏ}\rangle = \&\langle\text{tên biến}\rangle;$
- Toán tử truy xuất đến ô nhớ mà con trỏ trỏ đến: **\***  
 $*\langle\text{tên biến con trỏ}\rangle$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

97

## Thao tác trên con trỏ

```
int a = 326;
int* p;
p = &a;
int b = *p;

printf("%d\n", a); // giá trị biến a
printf("%x\n", *(&a)); // địa chỉ ô nhớ của biến a, in thường
printf("%x\n", p); // địa chỉ ô nhớ con trỏ p trỏ đến, in thường
printf("%d\n", *p); // giá trị ô nhớ mà con trỏ p trỏ đến
printf("%d\n", b); // b chứa giá trị ô nhớ mà con trỏ p trỏ đến
```

326
BFF720
BFF720
326
326

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

98

## Biến Con trỏ

Ví dụ:

```
void main() {
    int* p;
    int a = 1, b;
    // *p = 8;           → Nếu bỏ chú thích, chương trình sẽ lỗi vì:
    //                   - biến p chưa được khởi gán.
    //                   - gán *p = 8 lỗi vì p giữ địa chỉ không hợp lệ.

    p = &a;
    *p = 11;           → a đổi thành 11

    p = &b;
    *p = 22;           → b sẽ được gán giá trị là 22
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

99

## Cấp phát vùng nhớ cho con trỏ (động)

	Ngôn ngữ C	Ngôn ngữ C++
Cấp phát	malloc calloc realloc	new
Giải phóng	free	delete

Ví dụ:

```
int *p = (int *)malloc(10*sizeof(int));      → int *a = new int;           delete a;
int *p2 = (int *)calloc(10, sizeof(int));    → int *p2 = new int[10];   delete []p2;
p = (int *)realloc(p, 20*sizeof(int));
free(p); free(p2);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

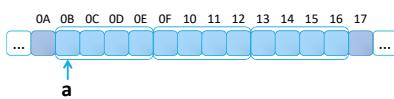
100

## Con trỏ và mảng một chiều

### Mảng một chiều

```
int a[3];
```

- Tên mảng **a** là một **con trỏ hằng**  
 $\rightarrow$  **không thể thay đổi** giá trị **địa chỉ** mà **a** **chứa**.
- Giá trị của **a** là **địa chỉ** phần tử đầu tiên của mảng  
 $\rightarrow a == \&a[0]$



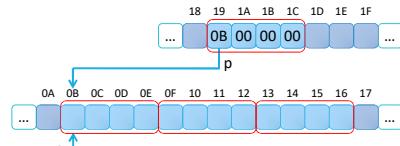
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

101

## Con trỏ và mảng một chiều

### Con trỏ đến mảng một chiều

```
int a[3], *p;
p = a; //hoặc p = &a[0];
```



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

102

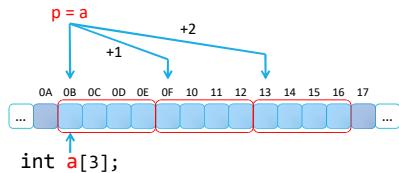
## Con trỏ và mảng một chiều

### Phép toán số học trên con trỏ

#### Phép cộng (tăng)

$+ n \Leftrightarrow + n * \text{sizeof}(<\text{kiểu dữ liệu}>)$

Có thể sử dụng toán tử gộp  $+=$  hoặc  $++$



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

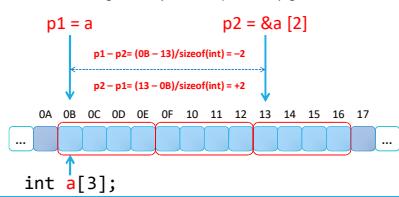
103

## Con trỏ và mảng một chiều

### Tính khoảng cách giữa 2 con trỏ

$<\text{kiểu dữ liệu}> *p1, *p2;$

$p1 - p2$  cho ta khoảng cách (theo số phần tử) giữa hai con trỏ (cùng kiểu)



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

104

## Con trỏ và mảng một chiều

Ví dụ:

```
void main() {
    int a[10], n = 4, *pa;
    pa = a; // hoặc pa = &a[0];

    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]); // scanf("%d", &p[i]);
        scanf("%d", a + i); // scanf("%d", p + i);
        scanf("%d", a++); // scanf("%d", p++);
    }
}
```

⇒  $a[i] \Leftrightarrow (a + i) \Leftrightarrow (pa + i) \Leftrightarrow &pa[i]$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

105

## Con trỏ và mảng một chiều

Có thể sử dụng con trỏ và mảng một chiều tương đương nhau trong một số trường hợp

Ví dụ:

```
void main() {
    int a[10], n = 4, *pa;
    pa = a; // hoặc pa = &a[0];

    for (int i = 0; i < n; i++) {
        scanf("%d", &a[i]); // scanf("%d", &p[i]);
        scanf("%d", a + i); // scanf("%d", p + i);
        scanf("%d", a++); // scanf("%d", p++);
    }
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

106

## Con trỏ và mảng một chiều

Có thể sử dụng con trỏ và mảng một chiều tương đương nhau trong một số trường hợp

```
void main() {
    int a[10], n = 5, *pa;
    pa = a; // hoặc pa = &a[0];

    for (int i = 0; i < n; i++) {
        printf("%d", a[i]); // printf("%d", p[i]);
        printf("%d", *(a + i)); // printf("%d", *(p + i));
        printf("%d", *(a++)); // printf("%d", *(p++));
    }
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

107

## Con trỏ và cấu trúc

### Truy xuất bằng 2 cách:

$<\text{tên biến con trỏ cấu trúc}> \rightarrow <\text{tên thành phần}>$   
 $(*<\text{tên biến con trỏ cấu trúc}>).<\text{tên thành phần}>$

#### Ví dụ:

```
struct PhanSo {
    int tu, mau;
};

PhanSo ps1, *ps2 = &ps1; // ps2 là con trỏ
ps1.tu = 1; ps1.mau = 2;
ps2->tu = 1; ps2->mau = 2;
⇒ (*ps2).tu = 1; (*ps2).mau = 2;
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

108

## Bài tập

1. Viết chương trình nhập 2 số thực bất kỳ a, b. Tạo 2 con trỏ pa và pb trỏ đến a, b. Xuất giá trị các con trỏ trên.
2. Tạo mảng 1 chiều a chứa n phần tử.  
a) Tìm max của a, xuất ra max và địa chỉ của max thông qua con trỏ mảng.
- b) Xuất địa chỉ của phần tử max và min
3. Viết chương trình nhập vào chuỗi st. Xuất giá trị từng ký tự của st thông qua con trỏ trỏ đến chuỗi.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

109



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

110



## KỸ THUẬT LẬP TRÌNH (PROGRAMMING TECHNIQUES)

### Chương 2

## KỸ THUẬT XỬ LÝ MẢNG

Khoa Công nghệ thông tin  
Bộ môn Công nghệ phần mềm  
-2022-

## NỘI DUNG

- 1. CÁC GIẢI THUẬT TÌM KIẾM VÀ SẮP XẾP**
  - 1.1. TÌM KIẾM TUYẾN TÍNH – LINEAR SEARCH**
  - 1.2. TÌM KIẾM NHỊ PHÂN – BINARY SEARCH**
  - 1.3. SẮP XẾP ĐỔI CHỖ TRỰC TIẾP – INTERCHANGE SORT**
  - 1.4. SẮP XẾP CHỌN TRỰC TIẾP – SELECTION SORT**
  - 1.5. SẮP XẾP NHANH – QUICK SORT**
- 2. XỬ LÝ MẢNG 1 CHIỀU**
- 3. XỬ LÝ MẢNG 2 CHIỀU**

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

112

### CÁC GIẢI THUẬT TÌM KIẾM

(Các thuật toán minh họa trên mảng 1 chiều chứa dữ liệu là các số nguyên)

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

113

## BÀI TOÁN

Giả sử có một mảng A gồm n phần tử. Cần **xác định** có hay không phần tử có giá trị bằng **X** trong mảng A? Nếu có phần tử X thì phần tử bằng phần tử X là phần tử thứ mấy trong mảng A?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

114

## Các giải thuật tìm kiếm nội

Các giải thuật tìm kiếm nội đưa ra 2 cách tìm kiếm:

- Tìm kiếm tuyến tính (Linear Search)
- Tìm kiếm nhị phân (Binary Search)



### 1.1. Tìm kiếm tuyến tính (Linear Search)

#### Ý tưởng:

Thuật toán tiến hành so sánh x lần lượt với các phần tử thứ 1, thứ 2,... của mảng a cho đến khi gặp phần tử có khóa cần tìm, hoặc đã tìm hết mảng mà không thấy x.

**Ví dụ:** Cho dãy số sau:

5    3    6    8    9    1    2

Tìm phần tử có giá trị x = 9, x = 10

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

116

## Minh họa ví dụ

Xét dãy số A có 7 phần tử:

5 3 6 8 9 1 2

Tìm x = 9?

Vị trí trong dãy (i)	0	1	...	4
Giá trị a[i]	5	3	...	9
Kết quả so sánh a[i] và x	5 != 9	3 != 9	...	9 == 9

Tim được x=9  
tại vị trí 4.  
Kết thúc quá  
trình

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

117

## Minh họa ví dụ

Xét dãy số A có 7 phần tử:

5 3 6 8 9 1 2

Tìm x = 10?

Vị trí trong dãy (i)	0	1	...	6	7
Giá trị a[i]	5	3	...	2	null
Kết quả so sánh a[i] và x	5 != 10	3 != 10	...	2 != 10	

i=7, a[i] không  
tồn tại.  
→ Không có 10  
trong dãy A.  
Kết thúc quá  
trình

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

118

### 1.1. Tìm kiếm tuyến tính (Linear Search) (tt)

#### Cài đặt:

```

int LinearSearch (int a[], int n, int x)
{
    for(int i = 0; i < n; i++) {
        if(a[i] == x)
            return i; // trả về vị trí của x trong a
    }
    return -1; // trả về -1 báo là không có x trong a
}
  
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

119

### 1.2. Tìm kiếm nhị phân (Binary Search)

#### Ý tưởng:

- Giả sử dãy số a đã có thứ tự tăng.
- Tại mỗi bước tiến hành so sánh x với phần tử nằm vị trí giữa của dãy tìm kiếm hiện hành, dựa vào kết quả so sánh này để quyết định giới hạn dãy tìm ở bước kế tiếp là nửa trái hay nửa phải của dãy tìm kiếm hiện hành.

#### Ví dụ:

- Cho dãy a có 7 phần tử: 3 4 6 8 9 10 13
- Tìm x = 10 và x = 2 ?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

120

## Minh họa ví dụ

Cho dãy số a: 3 4 6 8 9 10 13

Tìm x = 10?

Bước	left	right	mid = $\lfloor (left+right)/2 \rfloor$	a[mid]	Kết quả so sánh a[mid] và x?
1	0	6	$\lfloor (0+6)/2 \rfloor = 3$	8	$8 < 10$
2	left = mid+1 = 3+1 = 4	6	$\lfloor (4+6)/2 \rfloor = 5$	10	$10 == 10$

Tim được x=10 tại vị trí 5.  
Kết thúc quá trình

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

121

## Minh họa ví dụ

Cho dãy số a: -2 0 6 8 9 10 13

Tìm x = 2?

Bước	left	right	mid = $\lfloor (left+right)/2 \rfloor$	a[mid]	Kết quả so sánh a[mid] và x?
1	0	6	$\lfloor (-2+6)/2 \rfloor = 3$	8	$8 < 2$ : sai
2	0	right = mid - 1 = 3 - 1 = 2	$\lfloor (0+2)/2 \rfloor = 1$	0	$0 < 2$ : đúng
3	left = mid + 1 = 1 + 1 = 2	2	2	6	$6 < 2$ : sai
4	2	right = mid - 1 = 2 - 1 = 1	?		

Tại bước 4 có left > right → Vô lý.  
Kết luận không có x = 2 trong a

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

122

## 1.2. Tìm kiếm nhị phân (Binary Search) (tt)

Cài đặt:

```
int BinarySearch(int a[], int n, int x) {
    int left = 0, right = n - 1;
    while(left <= right) {
        int mid = (left + right) / 2;
        if(a[mid] == x)
            return mid; //Tìm thấy x trong a tại mid
        else if(a[mid] < x) left = mid + 1;
        else
            right = mid - 1;
    }
    return -1; //không tìm thấy x trong a
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

123

## Bài tập

Bài 1: Cho dãy số sau:

7 9 13 17 27 30 31 35 38 40

a. Tìm x = 17, x = 35, x = 40?

b. Tìm x = 23, x = 10, x = 36?

Bài 2: Cho dãy ký tự

Z R L K H F E C A

a. Tìm x = R, x = C?

b. Tìm x = D, x = Q?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

124

## Bài tập

Cho mảng 1 chiều a chứa n số nguyên. Viết chương trình thực hiện các yêu cầu sau:

- Viết hàm nhập/xuất mảng a.
- Tìm max/min của a.
- Đếm số phần tử chẵn/lẻ trong a.
- Tìm kiếm phần tử x trong a theo 2 dạng (trả về vị trí/xuất câu thông báo) với giải thuật tìm kiếm tuyến tính/tìm kiếm nhị phân.
- Đếm trên a có bao nhiêu phần tử x (x nhập từ bàn phím).

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

125

## Bài tập (tt)

6. Đếm trên a có bao nhiêu phần tử lớn hơn x.

7. Tính tổng các phần tử của a.

8. Xuất các số nguyên tố trong a.

9. Xuất các số hoàn thiện trong a.

10. Xuất các phần tử ở vị trí chẵn/ vị trí lẻ trong a.

11. Xuất giá trị max/min kèm theo vị trí của giá trị đó trong mảng a.

12. Cho 2 dãy số b có m phần tử, dãy c có n phần tử. Ghép b và c thành dãy a được xếp tăng dần.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

126

## CÁC GIẢI THUẬT SẮP XẾP

(Các thuật toán minh họa sắp xếp dãy không tăng trên mảng 1 chiều chưa dữ liệu là các số nguyên)

Sắp xếp là quá trình xử lý một danh sách các phần tử để đặt chúng theo một thứ tự thỏa mãn một tiêu chuẩn nào đó dựa trên nội dung thông tin lưu trữ tại mỗi phần tử.

Các phương pháp sắp xếp thông dụng

Sắp xếp đổi chỗ trực tiếp – Interchange Sort

Sắp xếp chọn trực tiếp – Selection Sort

Sắp xếp nhanh – Quick sort

Sắp xếp chèn trực tiếp – Insertion Sort (đọc thêm)

Sắp xếp Nối bọt – Bubble Sort (đọc thêm)

## BÀI TOÁN

Giả sử có một mảng A gồm n phần tử. Cần sắp xếp (bổ trí lại) các phần tử trong mảng A theo một thứ tự nhất định nhằm thỏa mãn một điều kiện (tiêu chuẩn) nào đó dựa trên nội dung thông tin lưu trữ tại mỗi phần tử.

### 1.3. Sắp xếp đổi chỗ trực tiếp – interchange Sort

#### Khái niệm nghịch thế:

- Xét dãy  $a$  gồm các số:  $a_1, a_2, \dots, a_n$ , với  $a$  là dãy chưa có thứ tự (chưa được sắp xếp, chẳng hạn như  $a$  là dãy không giảm). Nếu  $i < j$  và  $a_i > a_j$  thì ta gọi đó là 1 nghịch thế.

- Ví dụ: Cho dãy số  $a$  như sau:

14    5    7    8    3

Vậy dãy trên có các cặp nghịch thế sau: (14, 5); (7, 3); (8, 3); ....

### 1.3. Sắp xếp đổi chỗ trực tiếp – interchange Sort

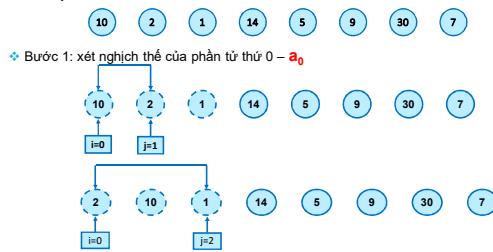
#### Ý tưởng thuật toán:

Xuất phát từ đầu dãy, lần lượt xét từng phần tử cho đến cuối dãy. Tại mỗi phần tử tìm tất cả nghịch thế chứa phần tử này, đổi chỗ phần tử này với các phần tử trong cặp nghịch thế.

Lặp lại công việc này với tất cả các phần tử còn lại cho đến khi dãy đã được sắp xếp xong.

### Minh họa ví dụ interchange sort

Cho dãy số  $a$



### Minh họa ví dụ interchange sort (tt)

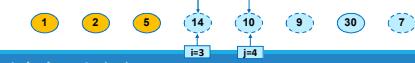
❖ Bước 2: xét nghịch thế của phần tử thứ 0 –  $a_1$



❖ Bước 3: xét nghịch thế của phần tử thứ 2 –  $a_2$

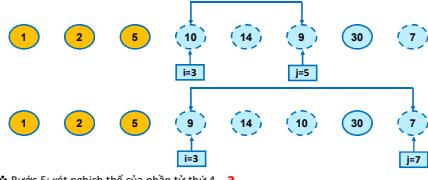


❖ Bước 4: xét nghịch thế của phần tử thứ 3 –  $a_3$



### Minh họa ví dụ interchange sort (tt)

❖ Bước 4: xét nghịch thế của phần tử thứ 3 –  $a_3$  tiếp theo

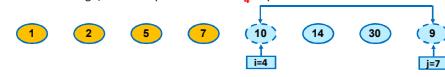


❖ Bước 5: xét nghịch thế của phần tử thứ 4 –  $a_4$

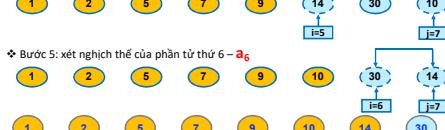


### Minh họa ví dụ interchange sort (tt)

❖ Bước 4: xét nghịch thế của phần tử thứ 4 –  $a_4$  tiếp theo



❖ Bước 5: xét nghịch thế của phần tử thứ 5 –  $a_5$



### Ví dụ

Minh họa thao tác sắp xếp dữ liệu theo phương pháp interchange Sort cho các dãy dữ liệu sau:

◦ Sắp xếp tăng:

13	8	12	6	9	10	12	7
A	H	K	R	E	C	Z	G

◦ Sắp xếp giảm:

13	8	12	6	9	10	12	7
A	H	K	R	E	C	Z	G

### Cài đặt

```
void interchangeSort(int a[], int n)
{
    for(int i = 0; i < n - 1; i++)
        for(int j = i + 1; j < n; j++)
            if(a[i] > a[j])
                swap(a[i], a[j]); //ham hoan doi gia tri
                2 so nguyen
}
void swap(int &x, int &y)
{
    int t = x;
    x = y;
    y = t;
}
```

### Bài tập cài đặt

Viết bổ sung các hàm vào chương trình xử lý mảng 1 chiều các hàm thực hiện những yêu cầu sau:

1. Viết hàm sắp xếp tăng theo PP interchange sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.
2. Viết hàm sắp xếp giảm theo PP interchange sort cho dữ liệu số nguyên/số thực/ký tự/ chuỗi ký tự.

### 1.4. Sắp xếp chọn trực tiếp – Selection sort

#### Ý tưởng giải thuật

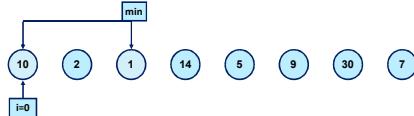
Chọn phần tử nhỏ nhất trong  $n$  phần tử ban đầu, đưa phần tử này về vị trí thứ 0 của dãy hiện hành; sau đó không quan tâm đến nó nữa. Xem dãy hiện hành chỉ còn  $(n - 1)$  phần tử, bắt đầu từ vị trí thứ 1; lặp lại quá trình đó trên dãy hiện hành... cho đến khi dãy hiện hành chỉ còn đúng 1 phần tử thì dừng.

### Minh họa ví dụ Selection sort

Cho dãy số a



- Bước 1: Tìm min của dãy số từ  $a_0 - a_{n-1}$ . Sau đó hoán đổi min với  $a_0$



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

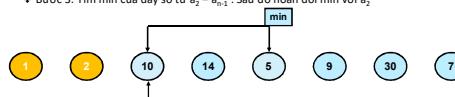
139

### Minh họa ví dụ Selection sort (tt)

- Bước 2: Tìm min của dãy số từ  $a_1 - a_{n-1}$ . Sau đó hoán đổi min với  $a_1$



- Bước 3: Tìm min của dãy số từ  $a_2 - a_{n-1}$ . Sau đó hoán đổi min với  $a_2$

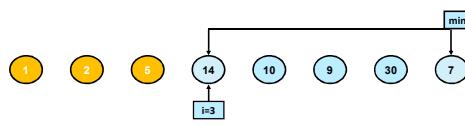


BỘ MÔN CÔNG NGHỆ PHẦN MỀM

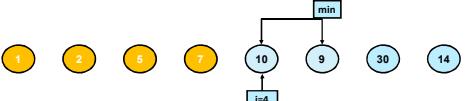
140

### Minh họa ví dụ Selection sort (tt)

- Bước 4: Tìm min của dãy số từ  $a_3 - a_{n-1}$ . Sau đó hoán đổi min với  $a_3$



- Bước 5: Tìm min của dãy số từ  $a_4 - a_{n-1}$ . Sau đó hoán đổi min với  $a_4$

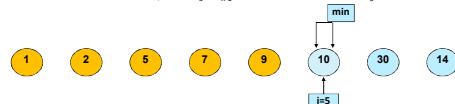


BỘ MÔN CÔNG NGHỆ PHẦN MỀM

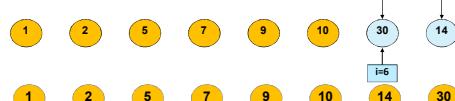
141

### Minh họa ví dụ Selection sort (tt)

- Bước 6: Tìm min của dãy số từ  $a_5 - a_{n-1}$ . Sau đó hoán đổi min với  $a_5$



- Bước 7: Tìm min của dãy số từ  $a_6 - a_{n-1}$ . Sau đó hoán đổi min với  $a_6$



Dừng. Vậy dãy đã được sắp xếp.

142

### Ví dụ

Minh họa thao tác sắp xếp dữ liệu theo phương pháp Selection Sort cho các dãy dữ liệu sau:

Sắp xếp tăng:

13	8	12	6	9	10	12	7
A	H	K	R	E	C	Z	G

Sắp xếp giảm

13	8	12	6	9	10	12	7
A	H	K	R	E	C	Z	G

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

143

### Cài đặt

```
void selectionSort(int a[], int n)
{
    for(int i = 0; i < n - 1; i++)
    {
        int imin = i;
        //tim min của dãy số từ  $a_i \rightarrow a_{n-1}$ 
        for(int j = i + 1; j < n; j++)
            if(a[imin] > a[j])
                imin = j;
        swap(a[imin], a[i]); //Đổi chỗ phần tử nhỏ nhất
                                //với phần tử đầu dãy hiện hành
    }
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

144



### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 7

Lần 1

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) swap( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 7

Lần 1

Dừng

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) swap( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 7

Lần 1

**Giải thuật:**

Bước 1: Nếu  $left \geq right$  //dãy có ít hơn 2 phần tử  
Kết thúc; //dãy đã được sắp xếp

Bước 2: Phân hoạch dãy  $a_{left} \dots a_{right}$  thành 3 đoạn:  $a_{left..}, a_j, a_{j+1..}, a_{r-1}, a_r, a_{right}$

- Đoạn 1:  $a_{left..}, a_j$  có giá trị nhỏ hơn x
- Đoạn 2:  $a_{j+1..}, a_{r-1}$  có giá trị bằng x
- Đoạn 3:  $a_r, a_{right}$  có giá trị lớn hơn x

Bước 3: Sắp xếp đoạn 1:  $a_{left..}, a_j$   
Bước 4: Sắp xếp đoạn 3:  $a_r, a_{right}$

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 2

Lần 2  
Tương tự

Dừng

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) swap( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 2

Lần 2  
Tương tự

Dừng

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) swap( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 0, right = 2

Lần 2  
Tương tự

**Giải thuật:**

Bước 1: Nếu  $left \geq right$  //dãy có ít hơn 2 phần tử  
Kết thúc; //dãy đã được sắp xếp

Bước 2: Phân hoạch dãy  $a_{left} \dots a_{right}$  thành 3 đoạn:  $a_{left..}, a_j, a_{j+1..}, a_{r-1}, a_r, a_{right}$

- Đoạn 1:  $a_{left..}, a_j$  có giá trị nhỏ hơn x
- Đoạn 2:  $a_{j+1..}, a_{r-1}$  có giá trị bằng x
- Đoạn 3:  $a_r, a_{right}$  có giá trị lớn hơn x

Bước 3: Sắp xếp đoạn 1:  $a_{left..}, a_j$   
Bước 4: Sắp xếp đoạn 3:  $a_r, a_{right}$

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 4, right = 7, x = 6

Lần 3  
Tương tự

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) **swap**( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 4, right = 7, x = 6

Lần 3  
Tương tự  
**Dừng**

Bước 2: Phát hiện và hiệu chỉnh cặp phần tử  $a[i], a[j]$  nằm sai chỗ:

- Bước 2.1: **Trong khi** ( $a[i] < x$ )  $i++$ ;
- Bước 2.2: **Trong khi** ( $a[j] > x$ )  $j--$ ;
- Bước 2.3: **Nếu** ( $i \leq j$ ) { { **Nếu** ( $i < j$ ) **swap**( $a[i], a[j]$ ); }  $i++$ ;  $j--$ ; }

Bước 3: Nếu  $i < j$ : Lặp lại Bước 2. Ngược lại: Dừng

### Quick Sort (tt)

**Minh họa:** Phân hoạch đoạn left = 6, right = 7, x = 12

Lần 3  
Tương tự

### Quick Sort (tt)

**Minh họa:**

Kết quả cuối cùng

### Quick Sort (tt)

**Cài đặt:**

```
void quickSort(int a[], int Left, int Right) {
    if (Left >= Right) return;
    int i = Left, j = Right, Mid = (Left+Right)/2;
    int x = a[Mid];
    do {
        while(a[i] < x) i++; // lặp cho đến khi a[i] >= x
        while(a[j] > x) j--; // lặp cho đến khi a[j] <= x
        if(i <= j) {
            if(i < j)
                swap(a[i], a[j]);
            i++;
            j--;
        }
    } while(i < j);
    quickSort(a, Left, j);
    quickSort(a, i, Right);
}
```

Phân hoạch bên trái  
Phân hoạch bên phải

### Quick Sort (tt)

**Bài tập:** Áp dụng giải thuật Quick Sort, hãy sắp xếp dãy số sau:

## 2. Kỹ thuật xử lý mảng 1 chiều

2.1. Duyệt mảng có điều kiện

2.2. Kiểm tra mảng thỏa điều kiện

2.3. Sắp xếp mảng (xem lại phần 1)

## Khái niệm

- Mảng 1 chiều là một dãy các phần tử có cùng tên, cùng kiểu dữ liệu được đặt liên tiếp nhau trong bộ nhớ và các phần tử được truy xuất thông qua chỉ số của mảng.
- Mỗi phần tử được xác định bởi một **chỉ số** biểu thị **vị trí** của phần tử đó trong mảng.
- Số lượng phần tử trong mảng được gọi là **kích thước** của mảng. Kích thước của mảng là cố định và phải được xác định trước.

Ví dụ:

Giá trị	34	56	47	67	12	32	34	23	123	34
Vị trí	0	1	2	3	4	5	6	7	8	9

Kích thước của mảng: 10

## Khai báo mảng một chiều

Cú pháp:

<Kiểu dữ liệu><Tên biến kiêu mảng>[<Số phần tử tối đa trong mảng>];

Ví dụ:

```
int a[100]; //Khai báo mảng số nguyên a gồm 100 phần tử
float b[50]; //Khai báo mảng số thực a gồm 50 phần tử
char str[30]; //Khai báo mảng ký tự str gồm 30 ký tự
```

Truy cập vào các phần tử của mảng

Ví dụ: mảng A có 10 phần tử: A[0] → 34; A[1] → 56; ... ; A[9] → 34

34	56	47	67	12	32	34	23	123	34
0	1	2	3	4	5	6	7	8	9

## 2.1. Duyệt mảng có điều kiện

Cú pháp: duyệt tất cả các phần tử thỏa điều kiện trong mảng có n phần tử.

```
<Kiểu hàm> <tên hàm>(int a[], int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] thỏa điều kiện)
            Xử Lý a[i]
}
```

### Ví dụ: Tính tổng các số chẵn trong mảng

```
int tinhTongSoChan(int a[], int n)
{
    int tong = 0;
    for(int i = 0; i < n; i++)
        if(a[i] % 2 == 0)
            tong += a[i];
    return tong;
}
```

### Bài tập

- Đếm số phần tử nguyên tố trong a
- Xuất các phần tử mà từng chữ số của nó là số lẻ.
- Tìm vị trí số lớn nhất (cuối cùng) của mảng, nghĩa là nếu trong mảng có nhiều phần tử là số lớn nhất thì xuất vị trí số cuối cùng.
- Kiểm tra mảng có toàn là số chẵn không?
- Kiểm tra mảng có phải là dãy số chẵn lẻ xen kẽ không?

## 2.2 Kiểm tra mảng thỏa điều kiện cho trước

**Trường hợp 1:** Kiểm tra tồn tại một phần tử trong mảng thỏa điều kiện nào đó cho trước.

```
bool kiemTraTonTaiABC(int a[], int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] thỏa điều kiện)
            return true;
    return false;
}
```

Ví dụ: Kiểm tra mảng có tồn tại phần tử lẻ không

```
bool kiemTraTonTaiLe(int a[], int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] % 2 == 1)
            return true;
    return false;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

169

## 2.2. Kiểm tra mảng thỏa điều kiện cho trước (tt)

**Trường hợp 2:** Kiểm tra tất cả các phần tử thỏa điều kiện nào đó cho trước.

```
bool kiemTraABC(int a[], int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] không thỏa điều kiện)
            return false;
    return true;
}
```

Ví dụ: Kiểm tra mảng toàn âm

```
bool kiemTraToanAm(int a[], int n)
{
    for (int i = 0; i < n; i++)
        if (a[i] >= 0)
            return false;
    return true;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

170

## Bài tập

Cho mảng một chiều các số nguyên a. Hãy thực hiện các yêu cầu sau:

- Kiểm tra mảng có chứa toàn bộ số nguyên tố hay không?
- Kiểm tra mảng có đối xứng hay không?
- Kiểm tra mảng có tăng dần/giảm dần/không tăng không giảm?

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

171

## 3. Kỹ thuật xử lý mảng cấu trúc

Cú pháp:

```
struct <tên kiểu cấu trúc>
{
    <kiểu dữ liệu> <tên thành phần 1>;
    ...
    <kiểu dữ liệu> <tên thành phần n>;
};
```

Ví dụ

```
struct Point2D
{
    int x;
    int y;
};
```

**Bài tập:** Khai báo các kiểu dữ liệu sau:

- Phân số
- Hỗn số
- Điểm không gian 3 chiều

```
struct SinhVien
{
    char MSSV[11];
    char hoTen[31];
    float dTieuLuan, dKetThucMon, dTongKet;
};
```

172

## Cấu trúc lồng

Các cấu trúc có thể được khai báo lồng nhau

```
struct DiemThi {
    float dToan, dLy, dHoa;
}

struct ThiSinh {
    char SBD[11];
    char hoTen[31];
    struct DiemThi ketQua;
} thiSinh1, thiSinh2;
```

Khai báo trực tiếp các trường dữ liệu của một cấu trúc bên trong một cấu trúc khác:

```
struct ThiSinh {
    char SBD[11];
    char hoTen[31];
    struct [DiemThi] {
        float dToan, dLy, dHoa;
        } ketQua;
} thiSinh1, thiSinh2;
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

173

## Truy cập các trường dữ liệu

Cú pháp:

```
<tên biến cấu trúc>.<tên thành phần>
```

Lưu ý: Dấu “.” là toán tử truy cập vào trường dữ liệu trong cấu trúc.

Ví dụ:

```
struct Point2D {
    int x, y;
} p = {2912, 1706};
printf("x = %d, y = %d\n", p.x, p.y);
```

**Bài tập:** Tính khoảng cách hai điểm không gian

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

174

## Bài tập

Khai báo cấu trúc phân số cần thiết lưu trữ thông tin phân số, sau đó thực hiện các chức năng:

- Viết hàm nhập vào phân số.
- Viết hàm xuất phân số.
- Kiểm tra phân số mẫu phải khác 0.
- Viết hàm tối giản phân số.
- Nhập vào 2 phân số. Tính tổng, hiệu, tích và thương của hai phân số.

## Gán dữ liệu giữa các biến cấu trúc

Có 2 cách:

<biến cấu trúc đích> = biến cấu trúc nguồn  
<biến cấu trúc đích>.<tên thành phần> = <giá trị>

**Ví dụ:**

```
struct Point2D {
    int x, y;
} p1 = {2912, 1706}, p2;
void main() {
    p2 = p1;
    p2.x = p1.x;
    p2.y = p1.y * 2;
}
```

### Bài tập:

- Viết hàm tìm trung điểm của hai điểm trong mặt phẳng
- Viết hàm cộng, trừ, nhân, chia hai phân số

## 3.2. Mảng cấu trúc

Là tập hợp các phần tử có cùng kiểu dữ liệu là kiểu cấu trúc

**Mục đích:**

- Lưu trữ một tập hợp các phần tử có cùng kiểu.
- Mỗi phần tử là một tập hợp các thành phần có thể khác nhau: thông tin các sinh viên trong lớp, đội bóng...

**Khai báo:**

```
struct <tên cấu trúc> <tên mảng cấu trúc>[số phần tử];
struct Student {
    int id;
    char name[10];
}
Ví dụ:     };
struct Student st[5];
```

## Ví dụ

```
struct Student{
    int id;
    char name[10];
};
int i;
Student st[5];
printf("Nhập thông tin cho 5 sinh viên: \n");
for (i = 0; i < 5; i++)
{
    printf("Nhập id: ");           scanf("%d", &st[i].id);
    printf("Nhập name: ");         scanf("%s", &st[i].name);
}
printf("Danh sách sinh viên: \n");
for (i = 0; i < 5; i++)
    printf("Id: %d, Name: %s\n", st[i].id, st[i].name);
```

## Bài tập

**Bài 1:** Cho một mảng các phân số gồm n phần tử ( $n \leq 50$ ), sau đó thực hiện các chức năng:

- Viết hàm nhập danh sách các phân số.
- Viết hàm xuất danh sách các phân số.
- Tìm phân số có giá trị lớn/nhỏ nhất
- Tính tổng và tích các phân số
- Xuất ra nghịch đảo giá trị các phân số trong mảng.
- Sắp xếp mảng tăng/giảm dần theo các giải thuật đã học.

## Bài tập

**Bài 2:** Thiết kế một chương trình lưu trữ thông tin sinh viên được mô tả chi tiết như sau: Danh sách sinh viên gồm n sinh viên, với n được nhập từ bàn phím. Thông tin một sinh viên bao gồm: Số thứ tự, Mã số sinh viên, Họ tên sinh viên, Điểm tiêu luận, Điểm thi kết thúc môn. Thực hiện các yêu cầu sau:

- Hãy xây dựng hàm nhập/xuất danh sách sinh viên.
- Hãy tính điểm tổng kết = 30% Điểm tiêu luận + 70% Điểm thi kết thúc môn
- In ra sinh viên có điểm tổng kết nhất và thấp nhất
- Cho biết có bao nhiêu sinh viên đạt và không đạt
- Hãy qui đổi điểm từ hệ 10 sang hệ 4 theo cơ chế tín chỉ
- Hãy sắp xếp danh sách sinh viên tăng/giảm dần theo điểm tổng kết
- Hãy tính điểm trung bình của tất cả các sinh viên
- Hãy thực hiện việc nhập xuất dữ liệu bằng file txt

## Bài tập

- Bài 3:** Cho một danh sách sinh viên gồm n sinh viên, với n được nhập bất kỳ. Thông tin một sinh viên: Mã số sinh viên, Họ tên sinh viên, Điểm các môn: Nhập môn lập trình, Môn toán A1, Môn toán A2, Môn Vật lý kỹ thuật, Môn anh văn, Điểm trung bình tích lũy. Thông tin môn học bao gồm: Mã môn, Tên môn, Số tín chỉ, Điểm. Thực hiện các yêu cầu sau:
- Hãy thiết kế và xây dựng các cấu trúc (struct) phù hợp cho bài toán.
  - Xây dựng một chương trình hoàn chỉnh cho một lớp học đảm bảo các thao tác sau:
    - Hàm nhập/xuất danh sách sinh viên
    - Tìm một sinh viên thông qua mã số sinh viên.
    - Tìm sinh viên có điểm trung bình cao nhất
    - Thêm, xóa một sinh viên ra khỏi danh sách
    - Sắp xếp danh sách sinh viên tăng/giảm theo điểm trung bình
    - Xếp loại học tập của sinh viên theo kết quả học tập
    - Thống kê số môn đậu rớt của sinh viên.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

181

## 2. Kỹ thuật xử lý mảng hai chiều

### 1 Khái niệm

### 2 Khai báo mảng hai chiều

### 3 Các kỹ thuật xử lý mảng 2 chiều

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

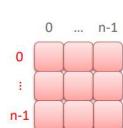
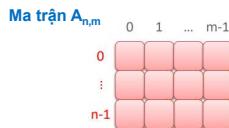
182

## 2.1. Khái niệm

**Mảng 2 chiều** thực chất là mảng 1 chiều mà mỗi phần tử của nó là một mảng khác.

**Mảng có từ 2 chiều trở lên** được gọi chung là **mảng nhiều chiều**.

Mảng nhiều chiều được dùng để lưu các dữ liệu dạng bảng, ma trận trong chương trình.



**Ma trận A<sub>n,n</sub>**  
Ma trận có số hàng và cột bằng nhau là vuông

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

183

## 2.2. Khai báo mảng hai chiều

Cú pháp:

<Kiểu dữ liệu> <Tên mảng> [<Số dòng tối đa>][<Số cột tối đa>]

Ví dụ:

Khai báo mảng 2 chiều kiểu số nguyên gồm 10 dòng, 10 cột  
int A[10][10];

Khai báo mảng 2 chiều kiểu số thực gồm 10 dòng, 10 cột  
float A[10][10];

Truy xuất các phần tử của ma trận: dựa vào chỉ số dòng, chỉ số cột của ma trận  
printf("%d", a[2][1]); //xuất ra màn hình giá trị dòng thứ 2, cột thứ 1

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

184

## 2.3. Các thuật toán xử lý mảng hai chiều

### 1 Duyệt ma trận

### 2 Tính tổng/tích

### 3 Tìm kiếm

### 4 Sắp xếp

### 5 Ma trận vuông

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

185

### 2.3.1. Duyệt ma trận

Duyệt từng dòng từ trên xuống dưới

```
void nhapMang2C_Sohnguyen(int a[][MAX], int &n, int &m)
{
    do{
        printf("Cho biết số dòng của ma trận: ");
        scanf("%d",&n);
    } while (n<0);
    do{
        printf("Cho biết số cột của ma trận: ");
        scanf("%d",&m);
    } while (m<0);
```

Ví dụ: Viết chương trình nhập ma trận các phần tử số nguyên

```
for(int i=0;i<n;i++)
    for(int j=0;j<m;j++){
        printf("Gia tri phan tu a[%d,%d]=",i,j);
        scanf("%d",&a[i][j]);
    }
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

186

## Bài tập

Cho mảng hai chiều các số nguyên a có n hàng và m cột. Hãy thực hiện các yêu cầu sau:

- Bài 1: Tính tổng các phần tử có chữ số đầu là chữ số lẻ
- Bài 2: Liệt kê các số hoàn thiện trong ma trận
- Bài 3: Tính tổng các phần tử lớn hơn trị tuyệt đối của phần tử liền sau nó
- Bài 4: Tính tổng giá trị trên dòng k ma trận
- Bài 5: Tính tổng các giá trị nằm trên biên của ma trận
- Bài 6: Đếm tần suất xuất hiện của 1 giá trị x trong ma trận
- Bài 7: Đếm số lượng các phần tử là số chẵn, số lẻ, số âm, số dương, số nguyên tố.

## 2.3.3. Tìm kiếm (tt)

Tìm phần tử lớn nhất trong mảng

Bước 1: max := a[0][0];  
Bước 2: Lặp (cho đến khi chọn hết các phần tử của mảng a)  
Chọn phần tử a[i][j], nếu max < a[i][j] thì gán max = a[i][j];

```
int timGiaTriLonNhat(int a[][MAXCOL], int n, int m)
{
    int max = a[0][0];
    for (int i=0; i<n; i++)
        for(int j=0; j<m; j++)
            if(max<=a[i][j])
                max = a[i][j];
    return max;
}
```

## 2.3.3. Tìm kiếm (tt)

Tìm phần tử nhỏ nhất trên từng dòng của ma trận

```
void printGTNN_Dong(int a[][MAX], int m, int n)
{
    for(int i=0; i<m; i++)
    {
        int min = a[i][0];
        for(int j=0; j<n; j++)
            if(min > a[i][j])
                min = a[i][j];
        printf(" Giá trị nhỏ nhất trên dòng %d là: %d", i, min);
    }
}
```

## 2.3.3. Tìm kiếm (tt)

Tìm phần tử lớn nhất trên từng cột của ma trận

```
void printGTLN_Cot(int a[][MAX], int m, int n)
{
    for(int j=0; j<n; j++)
    {
        int max = a[0][j];
        for(int i=0; i<m; i++)
            if(max < a[i][j])
                max = a[i][j];
        printf(" Giá trị lớn nhất trên cột %d là: %d", j, max);
    }
}
```

## Bài tập

Cho mảng hai chiều các số nguyên a có n hàng và m cột. Hãy thực hiện các yêu cầu sau:

- Bài 1: Liệt kê các cột có tổng nhỏ nhất trong ma trận
- Bài 2: Liệt kê các dòng có nhiều số hoàn thiện nhất trong ma trận
- Bài 3: Liệt kê chỉ số các dòng chứa toàn giá trị chẵn
- Bài 4: Tim giá trị xuất hiện nhiều nhất trong ma trận
- Bài 5: Tim số nguyên tố nhỏ nhất trong ma trận
- Bài 6: Tim phần tử lớn (nhỏ) nhất trong dòng thứ k
- Bài 7: Tim phần tử lớn (nhỏ) nhất trong cột thứ k

## 2.3.4. Sắp xếp

Sắp xếp các phần tử tăng dần trên từng dòng

Cho k chạy từ dòng 0 đến hết số dòng trong ma trận  
Cho i chạy từ cột 0 đến số cột -1 trong ma trận  
Cho j chạy từ cột i+1 đến hết số cột trong ma trận  
So sánh nếu A[k][i] > A[k][j] thì  
Đổi chỗ A[k][i] và A[k][j]

```
void sapXepMaTranTangDong(int a[][MAXSIZE], int dong, int cot)
{
    for(int k=0; k < dong; k++)
        for(int i=0; i < cot-1; i++)
            for(int j = i+1; j < cot; j++)
                if(a[k][i] > a[k][j])
                    HoanVi(a[k][i], a[k][j]);
}
```

### 2.3.4. Sắp xếp

Tương tự: Sắp xếp các phần tử tăng dần trên từng cột

Cho  $k$  chạy từ cột 0 đến hết số cột trong ma trận

Cho  $i$  chạy từ dòng 0 đến số dòng -1 trong ma trận

Cho  $j$  chạy từ dòng  $i+1$  đến hết số dòng trong ma trận

So sánh nếu  $A[i][k] < A[j][k]$  thì

Đổi chỗ  $A[i][k]$  và  $A[j][k]$

### Bài tập

Cho mảng hai chiều các số nguyên  $a$  có  $n$  hàng và  $m$  cột. Hãy thực hiện các yêu cầu sau:

**Bài 1:** Sắp xếp các phần tử trong ma trận sao cho dòng chẵn tăng dòng lẻ giảm

**Bài 2:** Sắp xếp các phần tử trong ma trận sao cho cột chẵn tăng cột lẻ giảm

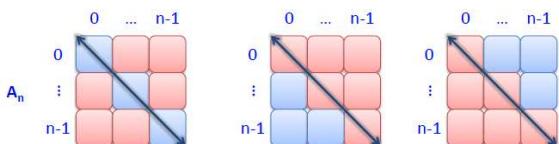
**Bài 3:** Sắp xếp ma trận tăng dần/giảm dần theo zic-zac.

### 2.3.5. Ma trận vuông

Các phần tử trên đường chéo chính:  $\text{dòng} = \text{cột}$

Các phần tử nằm bên dưới đường chéo chính:  $\text{dòng} > \text{cột}$

Các phần tử nằm bên trên đường chéo chính:  $\text{dòng} < \text{cột}$

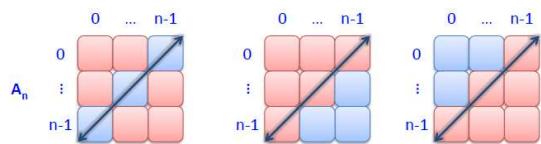


### Ma trận vuông (tt)

Các phần tử trên đường chéo phụ:  $\text{dòng} + \text{cột} = n-1$

Các phần tử nằm bên dưới đường chéo phụ:  $\text{dòng} + \text{cột} > n-1$

Các phần tử nằm bên trên đường chéo phụ:  $\text{dòng} + \text{cột} < n-1$



### Bài tập

Cho ma trận vuông  $n$ . Viết các hàm sau:

Bài 1: Tính tổng các phần tử nằm trên đường chéo chính.

Bài 2: Tính tổng các phần tử nằm trên đường chéo phụ.

Bài 3: Tính tổng các phần tử nằm phía trên đường chéo chính.

Bài 4: Tính tổng các phần tử nằm phía trên đường chéo phụ.





## KỸ THUẬT LẬP TRÌNH (PROGRAMMING TECHNIQUES)

### Chương 3

# CHUỖI KÝ TỰ

Khoa Công nghệ thông tin  
Bộ môn Công nghệ phần mềm  
-2022-

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 199

## Nội dung

1. Khái niệm
2. Khai báo và khởi tạo
3. Các thao tác trên chuỗi ký tự
4. Một số thuật toán xử lý chuỗi
  1. Một số thuật ngữ
  2. Tìm chuỗi con
  3. So khớp chuỗi
5. Bài tập

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 200

## 1. Khái niệm

**Chuỗi ký tự** là một dãy các phần tử, mỗi phần tử là một ký tự.

**Lưu ý:** Chuỗi ký tự được kết thúc bằng ký tự '\0'. Do đó khi khai báo độ dài của chuỗi luôn luôn khai báo dù 1 phần tử để chứa ký tự '\0'.

Ví dụ: `char s[5] = "CNTT";` //khai báo chuỗi s có 5 phần tử kiểu char và gán dãy ký tự CNTT vào chuỗi s.

'C'	'N'	'T'	'T'	'\0'
s[0]	s[1]	s[2]	s[3]	s[4]

Độ dài chuỗi s = 4.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 201

## 2. Khai báo và khởi tạo

Để khai báo một chuỗi, ta có 2 cách khai báo sau:

**Cách 1:**

```
char <Tên_chuỗi>[<Số ký tự tối đa của chuỗi>];
```

Ví dụ: `char hoten[50];`

**Cách 2:**

```
char *<Tên_chuỗi>;
```

Ví dụ: `char *hoten;`

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 202

## 2. Khai báo và khởi tạo

Vừa khai báo vừa khởi tạo:

```
char monhoc[50] = "Kỹ thuật lập trình";
char s[10] = {'K', 'T', 'L', 'T', '\0'};
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 203

## 3. Các thao tác trên chuỗi ký tự

### 3.1. Xuất chuỗi

- ❖ Sử dụng hàm printf với đặc tả "%s"

```
char monhoc[50] = "Tin hoc co so A";
printf("%s", monhoc); //Không xuống dòng
```

- ❖ Sử dụng hàm puts

```
char monhoc[50] = "Tin hoc co so A";
puts(monhoc); //Tự động xuống dòng
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM 204

### 3. Các thao tác trên chuỗi ký tự

#### 3.2. Nhập chuỗi

##### ❖ Sử dụng hàm `scanf` với đặc tả "%s"

- Chỉ nhận các ký tự từ bàn phím đến khi gặp ký tự khoảng trắng hoặc ký tự xuống dòng.
- Chuỗi nhận được không bao gồm ký tự khoảng trắng và xuống dòng.

Ví dụ:

```
char monhoc[50];
printf("Moi nhap chuoi: ");      Moi nhap chuoi: Tin hoc co so A
scanf("%s", monhoc);           Chuoi vua nhap: Tin_
printf("Chuoi vua nhap: %s", monhoc);
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

205

### 3. Các thao tác trên chuỗi ký tự

#### 3.2. Nhập chuỗi

##### ❖ Sử dụng hàm `gets`

- Nhận các ký tự từ bàn phím đến khi gặp ký tự xuống dòng.
- Chuỗi nhận được là những gì người dùng nhập (trừ ký tự xuống dòng).

Ví dụ:

```
char monhoc[50];
printf("Moi nhap chuoi: ");      Moi nhap chuoi: Tin hoc co so A
gets(monhoc);                  Chuoi vua nhap: Tin hoc co so A_
printf("Chuoi vua nhap: %s", monhoc);
```

Moi nhap chuoi: Tin hoc co so A

Chuoi vua nhap: Tin hoc co so A\_

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

206

### 3. Các thao tác trên chuỗi ký tự

#### Một số lưu ý khi làm việc với mảng ký tự theo quy ước chuỗi:

- Các hàm xử lý chuỗi ký tự trong C được khai báo trong `<string.h>`.
- Người lập trình phải quản lý và cấp phát đủ vùng nhớ cho các mảng ký tự dùng để lưu trữ chuỗi. Ngoài ra, phải đảm bảo chuỗi ký tự luôn chứa ký tự NULL để đánh dấu kết thúc nội dung chuỗi.
- Không sử dụng phép toán `=`, `+`, `==`, `>`, `<` và các phép toán so sánh khác đối với chuỗi được lưu ở dạng mảng ký tự mà phải sử dụng các hàm thư viện trong `<string.h>`.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

207

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

##### Thuộc thư viện `<string.h>`

- `strlen`
- `strcpy`
- `strcmp/ strncmp/ strcmpi/ stricmp`
- `strlwr/ strupr`
- `strcat`
- `strstr`
- ...

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

208

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

##### `size_t* strlen(const char *s)`



Tính độ dài chuỗi s  
`size_t` thay cho `unsigned` (trong `<stddef.h>`) dùng để đo các đại lượng không dấu.



Độ dài chuỗi s



```
char s[] = "Tin hoc co so A!!!";
int len = strlen(s); //=> 18
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

209

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

##### `char* strcpy(char dest[], const char src[])`



Sao chép chuỗi `src` sang chuỗi `dest`, dừng khi ký tự kết thúc chuỗi '`\0`' vừa được chép.  
! `dest` phải đủ lớn để chứa `src`.



Địa chỉ chuỗi dest



```
char s[100];
s = "Tin hoc co so A";//=> SAI
strcpy(s, "Tin hoc co so A");//=> ĐÚNG
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

210

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

`char* strcmp(strcmp(const char *s1, const char *s2)`



So sánh hai chuỗi s1 và s2 (không phân biệt chữ hoa chữ thường).

- ◆ <0: s1 < s2
- ◆ ==0: s1 == s2
- ◆ >0: s1 > s2



```
char s1[] = "tin hoc co so A";
char s2[] = "TIN HOC CO SO A";
int kq = strcmp(s1,s2); //=> kq == 0
```

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

`char* strcmp(const char *s1, const char *s2)`



So sánh hai chuỗi s1 và s2 (phân biệt chữ hoa chữ thường).

- ◆ <0: s1 < s2
- ◆ ==0: s1 == s2
- ◆ >0: s1 > s2



```
char s1[] = "tin hoc co so A";
char s2[] = "hoc tin co so A";
int kq = strcmp(s1,s2); //=> kq = 1 (> 0 )
```

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

STT	Tên hàm	Chức năng	Ví dụ
1	<code>strcat(char *s1, char *s2)</code>	Nối chuỗi s2 vào chuỗi s1	<code>char *s1 = "Khoa ";</code> <code>char *s2 = "CNTT";</code> <code>strcat(s1, s2);</code> <code>printf("%s\n", s1);</code> Kết quả: Khoa CNTT
2	<code>char *strupr(char *s)</code>	Chuyển chuỗi s thành chuỗi chữ thường	<code>char s[]="Tin hoc";</code> <code>strupr(s);</code> <code>puts(s);</code> Kết quả: tin hoc

### 3. Các thao tác trên chuỗi ký tự

#### 3.3. Một số hàm thao tác trên chuỗi

STT	Tên hàm	Chức năng	Ví dụ
3	<code>char* strupr(char *s)</code>	Chuyển chuỗi s thành chuỗi chữ hoa	<code>char s[]="Tin hoc";</code> <code>strupr(s);</code> <code>puts(s);</code> Kết quả: TIN HOC
4	<code>char* strstr(const char *s1, const char *s2)</code>	Tìm vị trí xuất hiện đầu tiên của s2 trong s1.	<code>char s1[]="Tin hoc";</code> <code>char s2[] = "hoc";</code> <code>if(strstr(s1,s2)!=null)</code> <code>printf("Tim thay!");</code>

### Bài tập

1. Cho biết kết quả của đoạn chương trình sau:

```
char s1[20] = "Truong DH CNTP", s2[10] = "Tp. HCM",
*input, *s3;
strcpy(input, s1);
strcpy(s3, "aeiou");
strcat(input, s2);
int n = strlen(input), k=0;
printf("Chuoi: %s", input);
for(int i=0; i<n; i++)
  if(strchr(s3, input[i])) k++;
printf("\nKet qua: %d", k);
```

### 4. Một số thuật toán xử lý chuỗi

#### 4.1. Một số thuật ngữ

❖ Một chuỗi X là chuỗi con (substring) của một chuỗi Y nếu X là một chuỗi chứa các ký tự liên tiếp của Y.

Ví dụ: **ab** và **bc** là 2 chuỗi con của **abcd**. Nhưng **ac** thì không phải là chuỗi con của **abcd**.

❖ Một chuỗi X là tiền tố (prefix) của một chuỗi Y nếu X là chuỗi con của Y và X xuất hiện ở đầu của chuỗi Y.

Ví dụ: **ab** là tiền tố của **abcd**, nhưng **bc** không phải là tiền tố của **abcd**.

❖ Một chuỗi X là hậu tố (postfix) của một chuỗi Y nếu X là chuỗi con của Y và X xuất hiện ở cuối của chuỗi Y.

Ví dụ: **cd** là hậu tố của **abcd**, nhưng **bc** không phải là hậu tố của **abcd**.

## 4.2. Thuật toán tìm xâu con của chuỗi - Brute Force

Thuật toán **Brute Force** có ý nghĩa là kiểm tra một chuỗi P có nằm trong một văn bản T hay không, nếu có thì nằm ở vị trí nào, và xuất hiện bao nhiêu lần.

### Ý tưởng:

Kiểm tra tất cả các vị trí trên văn bản từ 0 cho đến kí tự cuối cùng trong văn bản. Sau mỗi lần thử, thuật toán Brute Force dịch mẫu sang phải một kí tự cho đến khi kiểm tra hết văn bản.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

217

## 4.2. Thuật toán tìm xâu con của chuỗi - Brute Force (tt)

```
void Brute Force (chuỗi P, chuỗi T) // tìm P trong T
{
    if ((len (P) = 0) or (len (T) = 0) or (len (P) > len (T))
        P không xuất hiện trong T
    else
    {
        Tính vị trí dừng STOP = len(T) – len(P)
        Vị trí bắt đầu so sánh START = 0
        // ta cần vị trí dừng vì ra khỏi STOP
        // chiều dài P lớn hơn chiều dài đoạn T còn lại thi dĩ nhiên P không thuộc T
        While(START <= STOP)
        {
            So sánh các ký tự giữa P và T bắt đầu từ START.
            IF ( các ký tự đều giống nhau )
                P có xuất hiện trong T tại vị trí start
                Tăng START lên 1 // quay lại so sánh từ đầu của P với các ký tự của T từ vị trí START
        }Đừng khi P xuất hiện trong T hoặc START > STOP
    }
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

218

## Ví dụ

T = "I LIKE COMPUTER" n = len(T) = 14  
 P = "LIKE" m = len(P) = 4  
 start = 0  
 stop = n-m = 10  
 Bước 1: I LIKE COMPUTER i = start = 0  
 LIKE j = 0  
 P[j] = 'L' != T[i+j] = T  
 ----> tăng i lên 1 (ký tự tiếp theo trong T)  
 j = 0; (trở về đầu chuỗi P so sánh lại từ đầu P hay P dịch sang phải 1 ký tự.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

219

Bước 2: I LIKE COMPUTER	i = 1	Bước 4: I LIKE COMPUTER	i = 2
LIKE	j = 0	LIKE	j = 1
p[j] = 'L' != T[i] = ''		p[j] == T[i+j] = 'I'	
----> tăng i lên một, j = 0		----> tăng j lên một	
Bước 3: I LIKE COMPUTER	i = 2	Bước 5: I LIKE COMPUTER	i = 2
LIKE	j = 0	LIKE	j = 2
p[j] == T[i+j] = 'L'		p[j] == T[i+j] = 'K'	
----> tăng j lên một,		----> tăng j lên một	
không tăng i vì ta đang xét T[i+j].			
(chỉ khi nào có sự sai khác mới tăng i lên một)			

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

220

Bước 6: I LIKE COMPUTER i = 2

LIKE j = 3  
 p[j] == T[i+j] = 'T'  
 ----> tăng j lên một ----> j = 4 = m : hết chuỗi P -----> P có xuất hiện trong T  
 Kết quả: Xuất ra vị trí i = 2 là vị trí xuất hiện đầu tiên của P trong T

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

221

## Bài tập

2. Viết hàm đếm có bao nhiêu khoảng trắng trong chuỗi.
3. Nhập vào một chuỗi, xóa những khoảng trắng thừa trong chuỗi.
4. Viết hàm đổi những kí tự đầu tiên của mỗi từ thành chữ in hoa và những từ không phải đầu câu sang chữ thường
5. Viết chương trình tìm kiếm tên trong chuỗi họ tên. Nếu có thì xuất ra là tên này đã nhập đúng, ngược lại thông báo là đã nhập sai.
6. Viết hàm cắt chuỗi họ tên thành chuỗi họ lót và chuỗi tên.  
 Ví dụ: chuỗi họ tên là: "Nguyễn Văn A" cắt ra 2 chuỗi là chuỗi họ lót: "NguyễnVăn", chuỗi tên là: "A".

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

222

## Bài tập

7. Viết chương trình chèn 1 từ ở bất cứ vị trí nào người dùng yêu cầu.
8. Viết 1 chương trình xoá một từ nào đó trong chuỗi.  
Ví dụ: Chuỗi ban đầu: "KHOA CNTT"  
Nhập: "CNTT", và kết quả xuất ra: "KHOA".
9. Nhập 1 chuỗi bất kì, liệt kê xem mỗi ký tự xuất hiện mấy lần.
10. Viết chương trình tìm kiếm xem ký tự nào xuất hiện nhiều nhất trong chuỗi.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

223



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

224

## KỸ THUẬT LẬP TRÌNH (PROGRAMMING TECHNIQUES)



### Chương 4

## KỸ THUẬT ĐỆ QUY

Khoa Công nghệ thông tin  
Bộ môn Công nghệ phần mềm  
-2022-

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

225

## Nội dung

1. Giới thiệu
2. Khái niệm
3. Hàm đệ quy
  - a. đệ quy tuyến tính
  - b. đệ quy nhị phân
  - c. đệ quy phi tuyến
  - d. đệ quy lồng
  - e. đệ quy hỗ trợ
4. Ưu/ nhược điểm của KQĐQ.
5. Các bước tìm giải thuật đệ quy cho 1 bài toán
6. Các phương pháp khử đệ quy
7. Cách tính độ phức tạp của hàm đệ quy

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

226

## 1. Mở đầu

**Đệ quy** là kỹ thuật đưa bài toán hiện tại về một bài toán **cùng loại, cùng tính chất** (đồng dạng) nhưng ở **cấp độ thấp hơn**. Quá trình này tiếp tục cho đến khi bài toán được đưa về một cấp độ mà tại đó có thể giải được. Từ cấp độ này ta **lần ngược lại** để giải các bài toán ở cấp độ cao hơn cho đến khi giải xong bài toán ban đầu.

Ví dụ: Định nghĩa n giai thừa:

$$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdots n. \rightarrow \text{định nghĩa tự nhiên}$$

$$n \cdot (n-1)! \text{ với } 0!=1 \rightarrow \text{định nghĩa bằng đệ quy}$$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

227



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

228



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

229

## Hàm đệ quy (tt)

Mọi hàm đệ quy đều có 2 phần

### Phản cơ sở (phản neo – trường hợp dừng)

- Là trường hợp khi thực hiện không cần kỹ thuật gọi đệ quy.
- Nếu hàm đệ quy không có phản này thì hàm sẽ bị lặp vô hạn và sinh lỗi khi thực hiện.
- Phản đệ quy
  - Là phản trong khi thực hiện có dùng kỹ thuật gọi đệ quy nhưng dành cho cấp độ thấp hơn.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

231

## 2. Khái niệm

**Kỹ thuật đệ quy:** là kỹ thuật định nghĩa một khái niệm có sử dụng chính khái niệm đang cần định nghĩa.

**Hàm đệ quy:** là hàm mà trong thân của nó có lệnh gọi lại chính nó dành cho đối tượng ở cấp thấp hơn.

**Ví dụ: Hàm tính n!**

```
int tinhGiaiThua (int n)
{
    int gt = 1;
    if(n<=1)
        gt = 1;
    else
        gt = n * tinhGiaiThua(n-1);
    return gt;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

230

## Cách viết hàm đệ quy

Định nghĩa tác vụ đệ quy thế nào thì viết hàm đệ quy như vậy. Xét 2 trường hợp neo (thực hiện không đệ quy) và trường hợp đệ quy.

Ví dụ: Chuyển các định nghĩa sau về dạng đệ quy:

- $S1(n) = 1+2+3+\dots+n$
- $S2(n) = 1+1/2 + 1/3 + \dots + 1/n$
- $S3(n) = 1^2 + 2^2 + 3^2 + 4^2 + \dots + n^2$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

233

## Bài toán dãy số Fibonacci

Fibonacci (1175 - 1250) được biết đến nhiều nhất với dãy số mang tên ông - dãy số Fibonacci.

Dãy số này xuất hiện trong bài toán dưới đây viết trong cuốn Liber Abaci: "Trong một năm, bắt đầu chỉ từ một đôi thỏ, bao nhiêu đôi thỏ sẽ được sinh ra nếu mỗi tháng một đôi thỏ sinh được một đôi thỏ con và cặp thỏ này lại để được từ tháng thứ hai trở đi?"

Dãy số Fibonacci có nguồn gốc từ bài toán trên là một dãy sao cho mỗi số hạng, kể từ sau số hạng thứ nhất, bằng tổng của hai số đứng ngay trước nó. Dãy số đó là:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144....

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

234

## Bài toán dãy số Fibonacci (tt)

Dãy số Fibonacci:

- $F(n) = F(n-1) + F(n-2)$  (phản đệ quy)
- $F(n) = 1$  với  $n \leq 2$  (phản neo).

Hàm đệ quy tính số thứ n trong dãy số Fibonacci

```
int tinhSoFibonacci (int n)
{
    if(n == 1) return 1;
    if(n == 2) return 1;
    return tinhSoFibonacci(n-1) + tinhSoFibonacci(n-2);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

235

## Bài toán dãy số Fibonacci (tt)

Dãy số Fibonacci:

- $F(n) = F(n-1) + F(n-2)$  (phản đệ quy)
- $F(n) = 1$  với  $n \leq 2$  (phản neo).

Hàm đệ quy tính số thứ n trong dãy số Fibonacci

```
int tinhSoFibonacci (int n)
{
    int kq;
    if(n<=2)
        kq = 1;
    else
        kq = tinhSoFibonacci(n-1) + tinhSoFibonacci(n-2);
    return kq;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

236

## Bài tập Viết hàm đệ quy cho các bài toán sau:

Bài 1: Tính  $S(n)$

- $S(n) = 1+2+3+\dots+n$
- $S(n) = 1+1/2+1/3+\dots+1/n$
- $S(n) = 1*2+2*3+3*4+4*5+\dots+n(n+1)$

Bài 2. Tìm giá trị phần tử thứ n của cấp số cộng có số hạng đầu là a, công sai là r:

$$U_n = \begin{cases} a, & \text{nếu } n = 1 \\ U_{n-1} + r, & \text{nếu } n > 1 \end{cases}$$

Bài 3. Tìm trị phần tử thứ n của 1 cấp số nhân có số hạng đầu là a, công bội là q:

$$U_n = \begin{cases} a, & \text{nếu } n = 1 \\ qU_{n-1}, & \text{nếu } n > 1 \end{cases}$$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

237

## 3. Các loại đệ quy

- a. Đệ quy tuyến tính
- b. Đệ quy nhị phân
- c. Đệ quy phi tuyến
- d. Đệ quy lồng
- e. Đệ quy hỗ trợ

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

238

## 3.1. Đệ quy tuyến tính

Đệ quy tuyến tính: bên trong thân hàm chỉ gọi hàm đệ quy 1 lần  
Cấu trúc hàm dạng đệ quy tuyến tính

```
KieuDuLieu TenHam (Thamso)
{
    if(Dieu Kieu Dung)
    {
        ...
        return Gia tri tra ve;
    }
    TenHam (Thamso)
    ...
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

239

## Ví dụ: tính giai thừa

```
int tinhGiaiThua (int n)
{
    int gt = 1;
    if(n <= 1)
        gt = 1;
    else
        gt = n * tinhGiaiThua(n-1);
    return gt;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

240

### 3.2. đệ quy nhị phân

Đệ quy nhị phân: bên trong thân hàm gọi hàm đệ quy 2 lần  
Cấu trúc hàm dạng đệ quy nhị phân

```
KieuDuLieu TenHam(Thamso)
{
    if(Dieu Kieu Dung)
    {
        ...
        return Gia tri tra ve;
    }
    ...
    TenHam(Thamso) ;
    TenHam(Thamso) ;
    ...
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

241

### Ví dụ: tính số trong dãy Fibonacci

```
int tinhSoFibonacci (int n)
{
    int kq;
    if(n <= 2)
        kq = 1;
    else
        kq = tinhSoFibonacci(n-1) + tinhSoFibonacci(n-2);
    return kq;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

242

### 3.3. Đệ quy phi tuyến

Đệ quy phi tuyến: nếu bên trong thân hàm có lời gọi lại chính nó được đặt bên trong thân của vòng lặp.

Cấu trúc hàm dạng đệ quy phi tuyến

```
KieuDuLieu TenHam(Thamso) {
    if(Dieu Kieu Dung) {
        ...
        return Gia tri tra ve;
    }
    ...
    vòng lặp (diều kiện lặp) {
        ...
        TenHam(Thamso) ;
        ...
    }
    return giá trị trả về;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

243

### Ví dụ:bài tập 3 ở slide 13

```
long Un ( int n)
{
    if (n < 6) return n;
    long S= 0;
    for (int i = 5; i > 0; i--)
        S = S + Un(n-i);
    return S;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

244

### 3.4. Đệ quy lồng

Hàm được gọi là đệ quy lồng nếu tham số trong lời gọi hàm là một lời gọi đệ quy.

Cấu trúc hàm dạng đệ qui lồng

```
<kiểu dữ liệu><tên hàm> (<danh sách tham số>)
{
    if(<điều kiện dừng cho phần cơ sở>)
    {
        ...
        return <giá trị trả về>;
    }
    ...
    //lời gọi hàm đệ quy có tham số là hàm gọi đệ quy
    <tên hàm>(<tên hàm>(<danh sách tham số>), tham số) ;
    ...
    return <giá trị trả về>;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

245

### Ví dụ

Viết hàm tính giá trị Ackermann's theo công thức sau:

$$Acker(m, n) = \begin{cases} n + 1, & \text{nếu } m = 0 \\ Acker(m - 1, 1), & \text{nếu } n = 0 \text{ và } m > 0 \\ Acker(m - 1, Acker(m, n - 1)), & \text{nếu } m > 0 \text{ và } n > 0 \end{cases}$$

```
int ackerman (int m, int n)
{
    if (m == 0)
        return (n+1);
    if (n == 0)
        return ackerman(m-1, 1);
    return ackerman(m-1, ackerman(m, n-1));
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

246

### 3.5. Đệ quy tương hỗ

Trong đệ quy tương hỗ thì thường có 2 hàm, và trong thân của hàm này có lời gọi của hàm kia , điều kiện dừng và giá trị trả về của cả hai hàm có thể giống nhau hoặc khác nhau.

Cấu trúc hàm dạng đệ quy

```
KieuDuLieu TenHamX(Thamso)
{
    if(Dieu Kieu Dung)
        ( ...; return Gia tri tra ve; )
    ...
    return TenHamX(Thamso)
    <Lien ket hai han>
    TenHamY(Thamso);
}
```

```
KieuDuLieu TenHamY(Thamso)
{
    if(Dieu Kieu Dung)
        ( ...; return Gia tri tra ve; )
    ...
    return TenHamY(Thamso)
    <Lien ket hai han>
    TenHamX(Thamso);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

247

### Ví dụ

- 2 hàm đệ quy gọi nhau
- $U_n = n$ ,  $n < 5$   
 $U_n = U_{n-1} + G_{n-2}$ ,  $n \geq 5$
- $G_n = n-3$ ,  $n < 8$   
 $G_n = U_{n-1} + G_{n-2}$ ,  $n \geq 8$

```
long Un(int n) {
    if (n < 5) return n;
    return Un(n-1) + Gn(n-2);
}

long Gn(int n) {
    if (n < 8) return n-3;
    return Un(n-1) + Gn(n-2);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

248

### 4.1 Ưu điểm của kỹ thuật đệ quy

Viết chương trình đơn giản dễ hiểu.

Có thể thực hiện một số lượng lớn các thao tác tính toán thông qua một đoạn chương trình ngắn gọn.

Định nghĩa một tập vô hạn các đối tượng thông qua một số hữu hạn lời phát biểu.

Hầu hết các ngôn ngữ lập trình đều hỗ trợ kỹ thuật đệ quy.

Đưa ra phương án tối ưu, giải quyết được một số vấn đề mà vòng lặp thông thường không giải quyết được.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

249

### 4.2 Nhược điểm của kỹ thuật đệ quy

Do mỗi lần gọi hàm đệ quy, chương trình phát sinh vùng nhớ mới trong bộ nhớ nên dễ tốn bộ nhớ.

Chương trình có tốc độ chạy chậm.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

250

### BÀI TẬP

**Bài 1.** Viết hàm tính các biểu thức  $S(n)$  theo 2 cách đệ quy và khử đệ quy (nếu có thể), với  $n$  là số nguyên dương nhập từ bàn phím :

1.  $S(n) = 1 + 2 + 3 + \dots + n$ .

2.  $S(n) = \sqrt{2 + \sqrt{2 + \dots + \sqrt{2 + \sqrt{2}}}}$  có  $n$  dấu căn.

3.  $S(n) = \frac{1}{2} + \frac{2}{3} + \dots + \frac{n}{n+1}$

4.  $S(n) = 1 + \frac{1}{3} + \frac{1}{5} + \dots + \frac{1}{2n+1}$

5.  $S(n) = 1.2 + 2.3 + 3.4 + 4.5 + \dots + n.(n+1)$

6.  $S(n) = \frac{1}{1.2.3} + \frac{1}{2.3.4} + \frac{1}{3.4.5} + \dots + \frac{1}{n.(n+1).(n+2)}$

7.  $S(n) = 1^2 + 2^2 + \dots + n^2$

8.  $S(n) = 1 + (1+2) + (1+2+3) + \dots + (1+2+3+\dots+n)$

9.  $S(n) = -\frac{1+2}{2!} + \frac{3+4}{4!} - \frac{5+6}{6!} \dots + (-1)^n \frac{(2n-1)+(2n)}{(2n)!}$

10.  $S(n) = \frac{1.2!}{2+\sqrt{3}} + \frac{2.3!}{3+\sqrt{4}} + \frac{3.4!}{4+\sqrt{5}} + \dots + \frac{n.(n+1)!}{(n+1)+\sqrt{(n+2)}}$

11.  $S(n) = \frac{1+\sqrt{2}}{2+\sqrt{3}} + \frac{2+\sqrt{3}}{3+\sqrt{4}} + \frac{3+\sqrt{4}}{4+\sqrt{5}} + \dots + \frac{n+\sqrt{(n+1)}}{(n+1)+\sqrt{(n+2)}}$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

251

### BÀI TẬP

**Bài 2.** Viết hàm tính  $m^n$  với  $m$  là số nguyên và  $n$  là số nguyên dương nhập từ bàn phím.

**Bài 3.** Viết hàm tìm ước chung lớp nhất của 2 số nguyên dương  $a$ ,  $b$ .

Gợi ý : Nếu  $a > b$  thi  $\text{UCLN}(a,b) = \text{UCLN}(b,a-b)$ , ngược lại  $\text{UCLN}(a,b) = \text{UCLN}(a-b, a)$ .

**Bài 4.** Viết hàm tìm giá trị phần tử thứ  $n$  của cấp số cộng có hạng đầu là  $a$ , công sai là  $r$ :

$$U_n = \begin{cases} a, & \text{nếu } n = 1 \\ U_{n-1} + r, & \text{nếu } n > 1 \end{cases}$$

**Bài 5.** Viết hàm tìm giá trị phần tử thứ  $n$  của cấp số nhân có hạng đầu là  $a$ , công bội là  $q$ :

$$U_n = \begin{cases} a, & \text{nếu } n = 1 \\ qU_{n-1}, & \text{nếu } n > 1 \end{cases}$$

**Bài 6.** Viết hàm tính biểu thức  $U(n)$  sau đây, với  $n$  là số nguyên dương nhập từ bàn phím :

$$U_n = \begin{cases} n, & \text{với } n < 6 \\ U_{n-5} + U_{n-4} + U_{n-3} + U_{n-2} + U_{n-1} & \text{với } n \geq 6 \end{cases}$$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

252

**Bài 7.** Cho dãy số  $A_n$  theo các công thức quy nạp như sau, hãy viết chương trình tính số hạng thứ  $n$ , với  $n$  là số nguyên dương :

1.  $A_0 = 1$  ;  $A_1 = 0$  ;  $A_2 = -1$  ;  $A_3 = 2A_{n-1} - 3A_{n-2} - A_{n-3}$ .
2.  $A_1 = 1$  ;  $A_2 = 2$  ;  $A_3 = 3$  ;  $A_{n+3} = 2A_{n+2} + A_{n+1} - 3A_n$

Viết chương trình tính số hạng thứ  $n$ .

**Bài 8.** Với  $n \geq 1$ ,  $n$  là số nguyên dương, biết rằng  $f(n)$  được tính theo công thức đệ quy như sau :  $f(1) = 1$  ;  $f(2n) = 2f(n)$  ;  $f(2n+1) = 2f(n) + 3f(n+1)$ .

1. Tính  $f(5)$ .
2. Viết chương trình tính  $f(n)$ .

**Bài 10.** Cho máng 1 chiều a đã có thứ tự chứa  $n$  số nguyên, viết hàm tìm kiếm số  $x$  trên a theo thuật toán tìm kiếm nhị phân bằng kỹ thuật đệ quy.

**Bài 11.** Viết hàm xuất dãy có  $n$  số Fibonacci, biết rằng số Fibonacci là số có dạng :

$$F(n) = \begin{cases} 1, & \text{với } n \leq 2 \\ F(n-1) + F(n-2), & \text{với } n > 2 \end{cases}$$

Ví dụ : nhập  $n=10 \rightarrow$  dãy số Fibonacci có 10 số là : 1 1 2 3 5 8 13 21 34 55

**Bài 12.** Viết hàm tìm số Fibonacci lớn nhất nhưng nhỏ hơn số nguyên  $n$  cho trước theo 2 cách đệ quy và khử đệ quy.

**Ví dụ :** nhập  $n=15 \rightarrow$  số Fibonacci lớn nhất nhỏ hơn 15 là 13.

**Bài 13.** Viết hàm tính số hạng thứ  $n$  của 2 dãy sau :

$$\begin{aligned} x_0 &= 1, y_0 = 0, \\ x_n &= x_{n-1} + y_{n-1} \text{ với mọi } n > 0 \\ y_n &= 3x_{n-1} + 2y_{n-1} \text{ với mọi } n > 0 \end{aligned}$$

**Bài 14.** Dãy An được cho như sau :  $A_1=1$  ;  $A_n = n(A_1+A_2+ \dots + A_{n-1})$ .

Viết hàm tính  $A_n$  sử dụng kỹ thuật đệ quy.

**Bài 15.** Với mỗi  $n \geq 1$ , số  $Y_n$  được tính như sau :

$$Y_1=1 ; Y_2=2 ; Y_3=3 ; Y_n = Y_{n-1} + 2Y_{n-2} + 3Y_{n-3} \text{ nếu } n \geq 4$$

Viết hàm tính  $Y_n$  bằng 2 cách đệ quy và khử đệ quy.

**Bài 16.** Với mỗi  $n \geq 1$ , số  $X_n$  được tính như sau :

$$X_1=1 ; X_2=1 ; X_n = X_{n-1} + (n-1)X_{n-2} \text{ với } n \geq 3$$

Viết hàm tính  $X_n$  bằng 2 cách đệ quy và khử đệ quy.

## 5. Các bước tìm giải thuật đệ quy cho bài toán

**Bước 1:** Thông số hóa bài toán.

**Bước 2:** Tìm các trường hợp cơ bản (phần neo) cùng giải thuật tương ứng cho các trường hợp này.

**Bước 3:** Tìm giải thuật giải trong trường hợp tổng quát (phản đệ quy) bằng cách phân rã bài toán theo kiểu đệ quy.

## 5.1. Thông số hóa bài toán

Chuyển bài toán cần giải thành bài toán tổng quát.

Tìm ra các thông số cho bài toán tổng quát, đặc biệt là nhóm các thông số biểu thị kích thước của bài toán.

$$\text{Acker}(m, n) = \begin{cases} n + 1, & \text{nếu } m = 0 \\ \text{Acker}(m - 1, 1), & \text{nếu } n = 0 \text{ và } m > 0 \\ \text{Acker}(m - 1, \text{Acker}(m, n - 1)), & \text{nếu } m > 0 \text{ và } n > 0 \end{cases}$$

Bài toán tính  $n!$ :  $n$  là tham số tổng quát, biểu thị kích thước bài toán.

Bài toán tính giá trị Ackermann's là  $n$  và  $m$  là tham số tổng quát, biểu thị kích thước bài toán.

## 5.2. Xác định thành phần cơ bản của bài toán

Xác định trường hợp neo của bài toán tổng quát.

Đây là các trường hợp tương ứng với các giá trị biên của các biến điều khiển mà giải thuật giải không đệ quy. Thông thường trong trường hợp này giải thuật rất đơn giản.

**Ví dụ:**

- Giá trị giải thừa của  $n=0$  hoặc  $n=1$  là 1 (ví dụ 3.1)
- Giá trị hàm  $U(n)=n$ , với  $n<6$  (ví dụ 3.5)
- Giá trị hàm Ackerman =  $n+1$ , với  $m=0$  và  $n>0$  (ví dụ 3.8)

## 5.3. Phân rã bài toán theo phương thức đệ quy

Tìm giải thuật giải bài toán trong trường hợp tổng quát bằng cách phân chia nó thành các thành phần:

- hoặc có giải thuật không đệ quy
- hoặc là bài toán trên nhưng có kích thước nhỏ hơn (có đệ quy).

Mục đích của bước thực hiện này là tìm giải thuật để giải bài toán theo hướng đệ quy với giá trị tham số tổng quát.

**Ví dụ** Tính giải thừa số nguyên  $n! = n * (n-1)!$ , với  $n>=2$

Tính tổ hợp chập  $k$  của  $n$ :  $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$  nếu  $0 < k < n$

## 5.4. Xây dựng giải thuật đệ quy cho một số bài toán thông dụng

**Bài toán 1.** Tim phần tử lớn nhất trong mảng 1 chiều a có n phần tử là số nguyên.  
a: a[0], a[1], a[2]... a[n-2], a[n-1]

Phân tích bài toán:

- **B1: Thông số hóa bài toán**
  - n chính là kích thước dữ liệu và là thông số tổng quát cho bài toán.
- **B2: Xác định phân cơ bản (neo)**
  - Nếu n=1 thì mảng a có 1 phần tử a[0], và nó cũng chính là phần tử lớn nhất của mảng.
- **B3: Xác định phân đệ quy ( $n \geq 2$ )**
  - n=2: max của mảng a chính là  $\max(\max(a[0]), a[1])$
  - n=3: max của mảng a chính là  $\max(\max(\max(a[0]), a[1]), a[2])$
  - ...
  - n=n: max của mảng a chính là  $\max(\max(a[0]...a[n-2]), a[n-1])$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

259

## Cài đặt

```
int timMax_Dequy (int a[], int n)
{
    if(n==1)
        return a[0];
    else
        return max2so(timMax_Dequy(a, n-1), a[n-1]);
}
int max2so (int x, int y)
{
    return x>y?x:y;
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

260

## 5.4. Xây dựng giải thuật đệ quy cho một số bài toán thông dụng (tt)

**Bài toán 2.** Đếm số chữ số của số nguyên dương n.

Ví dụ:  $n=29084952106 \rightarrow kq = 11$

Phân tích bài toán

Cài đặt

```
int DemSLChuSoCuaN (int n)
{
    n = abs(n);
    if(n < 10)
        return 1;
    return 1 + DemSLChuSoCuaN(n/10);
}
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

261

## 5.4. Xây dựng giải thuật đệ quy cho một số bài toán thông dụng (tt)

**Bài toán 3.** Tìm tất cả các hoán vị của 1 dãy phần tử

Phân tích bài toán:

- Thông số hóa: số phần tử n của dãy là giá trị xác định kích thước dữ liệu bài toán.
- Phân tích tìm phần neo và đệ quy:
  - Nếu dãy A có n=1 phần tử A[1] = a thì số hoán vị chỉ có 1 là: a.
  - Nếu dãy A có n=2 phần tử: A[1]=a, A[2]=b thì số hoán vị là 2 dãy sau :
  - a b ; b a.
- Nếu dãy A có n=3 phần tử: A[1]=a, A[2]=b, A[3]=c thì các hoán vị của dãy A là 6 dãy sau:

```
a b c;
b a c
a c b
c a b
b c a
c b a
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

262

## Các bước giải bài toán bằng đệ quy

Giới thiệu HoanVi(A, m) là hàm xuất tất cả các dạng hoán vị khác nhau của A có được bằng cách hoán vị m thành phần đầu của dãy A.

Phản ứng bài toán:

    m=1, hàm HoanVi(A,1) chỉ có 1 cách xuất A.

Phân rã tổng quát

```
HoanVi(A,m) ≡ { Swap (A[m], A[m]) ; HoanVi (A, m-1) ;
                    Swap (A[m], A[m-1] ; HoanVi (A, m-1) ;
                    Swap (A[m], A[m-2] ; HoanVi (A,m-1) ;
                    ...
                    Swap (A[m], A[2] ; HoanVi (A,m-1) ;
                    Swap (A[m], A[1] ; HoanVi (A,m-1) ;
                }
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

263

## Code mã hóa

```
1 //#include<conio.h>
2 //#include<stdio.h>
3 //#include<stdlib.h>
4 #define N 10
5 typedef int Data;
6
7 Data A[N]; //mảng A chứa dãy dữ liệu với kiểu dữ liệu Data
8 //...
9 void Swap(Data &x, Data &y)
10 {
11     Data t = x;
12     x = y;
13     y = t;
14 }
15 //.....
16 void print(Data a[], int n)// mảng A có N phần tử
17 {
18     for (int i = 0; i < n; i++) // phần tử mảng bắt đầu từ vị trí 0
19         printf("%d", a[i]);
20     printf("\n");
21 }
```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

264

**Code mã hóa**

```

23 //Copy n phần tử của mảng source cho mảng dest
24 //Copy n phần tử của mảng source cho mảng dest
25 for (int i = 0; i < n; i++)
26     dest[i] = source[i];
27 }
28 .....
29 void HoanVi(Data A[], int n, int m, int k=1)// hoán vị n thành phần đầu của dãy A
30 {
31     if (n == 1)
32     {
33         printf("%d.", k+1);
34         printf(A, n);
35     }
36     else
37     {
38         Data *B = (Data *)malloc(n*sizeof(Data));
39         Copy(B, A, n); //giữ bản sao của A khi chưa hoán đổi để lặp chuẩn
40         for (int i = m - 1; i >= 0; i--)// Phản từ mang bắt đầu từ vị trí 0
41         {
42             Copy(A, B, n); //chuyển A về thứ tự chuẩn ban đầu
43             Swap(A[m - 1], A[i]);
44             HoanVi(A, n, m - k, k);
45         }
46     }
47 }

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

265

**Code mã hóa**

```

48 void main()
49 {
50     Data A[N] = { 5, 2, 4, 3 };
51     int n = 4, solan = 0;
52     HoanVi(A, n, n, solan);
53     getch();
54 }

```

1.	5	2	4	3
2.	2	5	4	3
3.	5	4	2	3
4.	4	5	2	3
5.	3	4	5	2
6.	2	4	3	5
7.	5	2	3	4
8.	2	5	3	4
9.	5	3	2	4
10.	3	5	2	4
11.	3	2	5	4
12.	2	3	5	4
13.	5	4	2	
14.	3	5	4	2
15.	5	4	3	2
16.	4	5	3	2
17.	4	3	5	2
18.	3	4	5	2
19.	3	2	4	5
20.	2	5	4	3
21.	5	4	3	
22.	4	3	2	5
23.	4	2	3	5
24.	2	4	3	5

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

266

## 6.2. Bài toán chia thường

Tìm số cách chia  $m$  vật (phần thường) cho  $n$  đối tượng (học sinh) có thứ tự.

Gọi Distribute là hàm tính số cách chia, khi đó Distribute là hàm có 2 tham số nguyên  $m$  và  $n$  (Distribute( $m, n$ )).

Gọi  $n$  đối tượng theo thứ tự xếp hạng 1, 2, 3, ...,  $n$ ;  $S_i$  là số vật mà đối tượng thứ  $i$  nhận được.

Khi đó các điều kiện ràng buộc cho cách chia là :

- $S_i \geq 0$
- $S_1 \geq S_2 \geq \dots \geq S_n$
- $S_1 + S_2 + \dots + S_n = m$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

267

### Ví dụ

Chia 5 phần thường cho 3 em học sinh theo yêu cầu của bài toán chia thường, ta có các cách chia như sau:

5	0	0
4	1	0
3	2	0
3	1	1
2	2	1

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

268

## Bước 1: Tìm cơ sở và cách giải

Nếu  $m=0$  (số vật bằng 0) thì sẽ có 1 cách chia : mọi đối tượng đều nhận 0 vật. Vậy Distribute( $0, n$ ) = 1 với mọi giá trị  $n$ .

Nếu  $n=0$ ,  $m \neq 0$  thì không có cách nào để thực hiện việc phân chia. Vậy Distribute( $m, 0$ )=0 với mọi giá trị  $m \neq 0$ .

Hoặc nếu  $n=1$ ,  $m \neq 0$  thì sẽ có 1 cách chia : đối tượng nhận  $m$  vật. Khi đó Distribute( $m, 1$ )=1.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

269

## Bước 2: Phân rã bài toán trong trường hợp tổng quát

Nếu số vật nhỏ hơn số đối tượng ( $m < n$ ):  $m-n$  đối tượng xếp sau cuối sẽ không nhận được gì trong mọi cách chia. Vậy khi  $m < n$  thì Distribute( $m, n$ )=Distribute( $m, n-1$ ).

Nếu số vật lớn hơn hoặc bằng số đối tượng ( $m \geq n$ ) ta phân các cách chia thành 2 nhóm :

- + **Nhóm 1** : không dành cho đối tượng thứ  $n$  (đối tượng xếp cuối) số vật nào cả ( $S_n = 0$ ). Vậy số cách = Distribute( $m, n-1$ ).
- + **Nhóm 2** : có phân chia vật cho người cuối cùng ( $S_n > 0$ ). Ta thấy rằng số cách chia của nhóm này bằng số cách chia ( $m-n$ ) vật cho  $n$  đối tượng. Vì cách chia mà tất cả đối tượng đều nhận được phân thường có thể thực hiện bằng cách cho mỗi đối tượng nhận trước 1 vật, sau đó chia phần vật còn lại.

Distribute( $m, n$ ) = Distribute( $m, n-1$ ) + Distribute( $m-n, n$ )

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

270

## Mã hóa bài toán

```
int distribute(int m, int n)
{
    if(n == 0) return 0;
    if(n == 1) return 1;
    if(m == 0) return 1;
    if(m < n) return distribute(m, m);
    return distribute(m, n-1) + distribute(m-n, n);
}
```

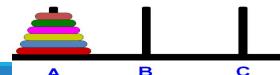
BỘ MÔN CÔNG NGHỆ PHẦN MỀM

271

## 6.3. Bài toán tháp Hà Nội

Mô tả bài toán: chuyển  $n$  đĩa từ cột A sang cột C có cột B làm trung gian, với yêu cầu sau:

- Mỗi lần di chuyển 1 đĩa.
- Khi chuyển có thể dùng cột trung gian B.
- Trong quá trình chuyển các chồng đĩa ở các cột luôn được xếp đúng (đĩa có kích thước bé đặt trên đĩa có kích thước lớn).



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

272

## Bước 1. Tìm phần cơ sở và cách giải

$n=1: \text{HaNoiTower}(1, A, B, C)$ : tìm dãy thao tác để chuyển chồng 1 đĩa từ cột A sang cột C, lấy cột B làm cột trung gian.  
 $\text{HaNoiTower}(1, A, B, C) \equiv (\text{Move}(A, C))$

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

273

## Bước 2. Phân rã bài toán

$\text{HaNoiTower}(n, A, B, C)$ : chuyển  $n$  đĩa từ cột A sang cột C, lấy cột B làm trung gian thành dây tuần tự 3 công việc sau:

- Chuyển  $(n-1)$  đĩa từ cột A sang cột B, lấy cột C làm trung gian : **HaNoiTower ( $n-1$ , A, C, B)**.
- Chuyển 1 đĩa từ cột A sang cột C : **Move(1, A, C)**.
- Chuyển  $(n-1)$  đĩa từ cột B sang cột C, lấy A làm trung gian : **HaNoiTower ( $n-1$ , B, A, C)**

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

274

## Tổng quát giải thuật

```
 $\text{HaNoiTower}(A, B, C) \equiv$ 
{
     $\text{Thap\_HN}(n - 1, A, C, B);$ 
     $\text{Move}(1, A, C);$ 
     $\text{Thap\_HN}(n - 1, B, A, C);$ 
}
```

**Chạy Demo**

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

275

## 7. Khử đệ quy

Là quá trình chuyển đổi 1 giải thuật đệ quy thành giải thuật không đệ quy.

Chưa có giải pháp cho việc chuyển đổi này một cách tổng quát. Ta thường khử đệ quy cho đệ quy tuyến tính.

Cách tiếp cận:

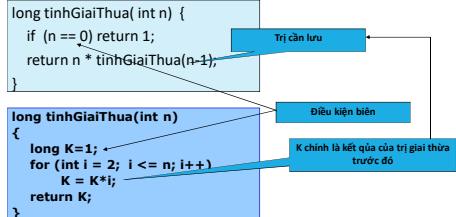
- Dùng quan điểm đệ quy để tìm giải thuật cho bài toán.
- Mã hóa giải thuật đệ quy.
- Khử đệ quy để có giải thuật không-đệ-quy.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

276

## 7.1. Khử đệ quy bằng vòng lặp

Ý tưởng: Lưu lại các trị của các lần tính toán trước làm dữ liệu cho việc tính toán của lần sau.



BỘ MÔN CÔNG NGHỆ PHẦN MỀM

277

## 7.2. Khử đệ quy dùng Stack

Để thực hiện 1 hàm đệ quy, hệ thống phải tổ chức vùng nhớ lưu trữ theo quy tắc LIFO (last in first out), còn gọi là Stack (ngăn xếp). Như vậy thường rất tốn bộ nhớ, vì cách tổ chức này mặc định cho mọi trường hợp.

Dựa vào ý tưởng trên, chúng ta chủ động tạo CTDL stack riêng cho từng hàm cụ thể.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

278

## Cấu trúc chuyển đệ quy thành khử đệ quy

Giả sử thủ tục đệ quy tuyến tính  $F(X)$  có cấu trúc như sau :

```

F(X)
{
    if(dk(X))//phần neo
        A(X);
    B(X);
    F(f(X));
    C(X);
}

```

$F(X)$ : hàm đệ quy có danh sách tham số là  $X$ .

$dk(X)$ : biểu thức điều kiện theo  $X$ .

$A(X)$ ,  $B(X)$ ,  $C(X)$ : nhóm lệnh không đệ quy theo  $X$ .

$f(X)$ : hàm thay đổi  $X$  để gọi đệ quy cấp thấp hơn.

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

279

## Giải thuật khử đệ quy bằng Stack

```

F(X) {
    taoStack(S);
    //đưa các giá trị đã tính vào Stack
    while(dk(X) == false) {
        B(X);
        //Xgt là giá trị cần lưu trữ sau mỗi
        bước tính với X.
        Push(S, Xgt);
        X = f(X);
    }
    A(X);
    //Lần lượt lấy các giá trị trong Stack ra.
    while(Empty(S) == false) {
        Pop(S, Xgt);
        C(X);
    }
}

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

280

## VD: Xét hàm đệ quy chuyển số từ hệ 10 sang hệ 2

```

void convert10To2(int n)
{
    if(n > 0)
    {
        convert10To2(n/2);
        printf("%d", n%2);
    }
}

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

281

## Hàm khử đệ quy của Convert10To2

```

void convert10To2(int n)
{
    if(n > 0)
    {
        convert10To2(n/2);
        printf("%d", n%2);
    }
}

void convert10To2_NonRecursive (int n)
{
    int temp;
    taoStack (S);
    while(n>0)
    {
        temp = n%2;
        push(S, temp);
        n/=2;
    }
    while(Empty(S)==false)
    {
        pop(S, temp);
        printf("%d", temp);
    }
}

```

BỘ MÔN CÔNG NGHỆ PHẦN MỀM

282

