

LỜI NÓI ĐẦU

Kỹ thuật lập trình là một trong những học phần cơ sở mà sinh viên các ngành máy tính và công nghệ thông tin... được tiếp cận sớm nhất. Mục tiêu của học phần không những trang bị một ngôn ngữ lập trình mà nó còn giúp hình thành, củng cố tư duy lập trình, tư duy giải quyết vấn đề và kỹ năng lập trình. Ngôn ngữ lập trình, trong trường hợp này, chỉ đóng vai trò là một công cụ giúp truyền tải các kỹ thuật. Đây là học phần nền tảng quan trọng cho nhiều học phần tiếp theo.

Con đường ngắn nhất, nếu không muốn nói là duy nhất, để rèn luyện kỹ năng và tư duy lập trình là thực hành viết mã lệnh. Do vậy, các bài tập đóng vai trò quan trọng. Với mục đích giúp sinh viên tổng hợp lại các vấn đề về lý thuyết và có thể vận dụng chúng vào việc giải các bài tập liên quan, cuốn “sách bài tập kỹ thuật lập trình” được biên soạn. Trong đó, các tác giả cố gắng trình bày tóm tắt các vấn đề về lý thuyết, các ví dụ mẫu với hướng dẫn chi tiết và các bài tập sinh viên tự giải tương ứng với từng bài học.

Nội dung sách được chia làm 8 bài:

Bài 1: Ngôn ngữ lập trình C++. Trong bài này, các tác giả giới thiệu những thành phần cơ bản của ngôn ngữ lập trình C++ như biến, biểu thức, các lệnh nhập/xuất dữ liệu.

Bài 2: Cấu trúc điều khiển trong C++. Nội dung chính của bài giới thiệu tóm tắt về cú pháp các cấu trúc điều khiển cơ bản như cấu trúc rẽ nhánh, cấu trúc chọn, cấu trúc lặp. Bài học cũng đưa ra các ví dụ minh họa về việc sử dụng kết hợp, linh hoạt các cấu trúc điều khiển khi viết chương trình.

Bài 3: Viết và sử dụng hàm. Bài học nhằm giới thiệu kiến thức cơ bản về khái niệm, phân loại hàm trong C++; cách định nghĩa, tổ chức và sử dụng hàm; các kỹ thuật liên quan tới hàm như: Hàm đệ quy, truyền tham số cho hàm.

Bài 4: Kỹ thuật xử lý mảng một chiều. Bài học tóm tắt lý thuyết cơ bản về kỹ thuật lập trình với mảng một chiều như: Các thao tác cơ bản

trên mảng, các dạng bài tập chính khi xử lý mảng kèm theo các ví dụ minh họa.

Bài 5: Kỹ thuật xử lý mảng hai chiều. Tương tự như Bài 4, trong bài này, các tác giả mở rộng các kỹ thuật xử lý mảng sang kiểu dữ liệu mảng hai chiều. Phần tóm tắt lý thuyết sẽ giới thiệu một số bài toán điển hình khi xử lý kiểu dữ liệu này.

Bài 6: Kỹ thuật xử lý chuỗi ký tự. Phần tóm lược lý thuyết sẽ giới thiệu tới độc giả một số thao tác cơ bản trên kiểu dữ liệu chuỗi ký tự và một số dạng bài tập từ cơ bản đến nâng cao.

Bài 7: Kỹ thuật lập trình với con trỏ. Bài học này trình bày tóm tắt lý thuyết về con trỏ trong C++ như: Một số thao tác cơ bản trên con trỏ, mối quan hệ giữa con trỏ và mảng, con trỏ và hàm.

Bài 8: Kỹ thuật xử lý tệp văn bản. Các tác giả giới thiệu các khái niệm ban đầu về tệp dữ liệu dạng văn bản, các thao tác cơ bản như đọc dữ liệu từ tệp, ghi dữ liệu vào tệp kèm theo các ví dụ cụ thể.

Mỗi bài học được tổ chức thành 2 phần: Phần A là tóm tắt lý thuyết kèm theo các ví dụ minh họa, phần B là các bài tập tự giải tương ứng.

Mặc dù đã có nhiều cố gắng, nhưng cuốn sách bài tập này có thể không tránh khỏi những khiếm khuyết. Các tác giả mong nhận được các ý kiến đóng góp mang tính xây dựng của độc giả để sách ngày càng được hoàn thiện hơn, phục vụ tốt hơn cho việc giảng dạy và học tập học phần Kỹ thuật lập trình.

CÁC TÁC GIẢ

MỤC LỤC

BÀI 1. NGÔN NGỮ LẬP TRÌNH C++	5
A. Tóm tắt lý thuyết.....	5
1.1. Biến.....	5
1.2. Biểu thức	7
1.3. Lệnh nhập	8
1.4. Lệnh xuất	9
1.5. Ví dụ tổng hợp.....	11
B. Các bài tập tự giải	13
BÀI 2. CẤU TRÚC ĐIỀU KHIỂN TRONG C++	15
A. Tóm tắt lý thuyết.....	15
2.1. Cấu trúc rẽ nhánh	15
2.2. Cấu trúc chọn.....	16
2.3. Cấu trúc lặp với số lần lặp xác định	17
2.4. Cấu trúc lặp với số lần lặp không xác định	18
2.5. Ví dụ tổng hợp.....	19
B. Các bài tập tự giải	25
BÀI 3. VIẾT VÀ SỬ DỤNG HÀM.....	28
A. Tóm tắt lý thuyết.....	28
3.1. Viết hàm	28
3.2. Gọi hàm	30
3.3. Tổ chức hàm	31
3.4. Kỹ thuật truyền tham số	32
3.5. Kỹ thuật đệ quy	34
3.6. Thiết kế hàm đệ quy	36
B. Các bài tập tự giải	39
BÀI 4. KỸ THUẬT XỬ LÝ MẢNG MỘT CHIỀU.....	43
A. Tóm tắt lý thuyết.....	43
4.1. Khái niệm và cách khai báo	43
4.2. Các thao tác cơ bản trên mảng một chiều	44
4.3. Một số bài toán cơ bản	45
B. Các bài tập tự giải	54
BÀI 5. KỸ THUẬT XỬ LÝ MẢNG HAI CHIỀU.....	57
A. Tóm tắt lý thuyết.....	57
5.1. Các thao tác cơ bản trên mảng hai chiều.....	57
5.2. Một số dạng bài tập trên mảng hai chiều	60
B. Các bài tập tự giải	65

BÀI 6. KỸ THUẬT XỬ LÝ CHUỖI KÝ TỰ.....	69
A. Tóm tắt lý thuyết.....	69
6.1. Một số thao tác cơ bản trên mảng ký tự	69
6.2. Một số thao tác cơ bản trên dữ liệu kiểu string	73
6.3. Một số bài toán trên chuỗi ký tự.....	73
B. Các bài tập tự giải	77
BÀI 7. KỸ THUẬT LẬP TRÌNH VỚI CON TRỎ	79
A. Tóm tắt lý thuyết.....	79
7.1. Một số thao tác cơ bản trên con trỏ	79
7.2. Con trỏ và mảng	83
7.3. Cấp phát, thu hồi bộ nhớ cho con trỏ	85
B. Các bài tập tự giải	89
BÀI 8. KỸ THUẬT XỬ LÝ TỆP VĂN BẢN	92
A. Tóm tắt lý thuyết.....	92
8.1. Mở tệp để ghi dữ liệu	92
8.2. Mở tệp để đọc dữ liệu.....	92
8.3. Ghi dữ liệu vào tệp	93
8.4. Đọc dữ liệu từ tệp	94
B. Các bài tập tự giải	96
TÀI LIỆU THAM KHẢO.....	99

BÀI 1

NGÔN NGỮ LẬP TRÌNH C++

Trong bài này, chúng ta sẽ dành thời gian tóm lược lại một số khái niệm, kỹ thuật ban đầu về lập trình: Biến, biểu thức, lệnh nhập, lệnh xuất và ứng dụng chúng trong các bài tập rèn luyện kỹ năng.

A. Tóm tắt lý thuyết

1.1. Biến

Trước khi xem xét khái niệm về biến, ta xét ví dụ sau: Viết chương trình giải và biện luận phương trình bậc hai dạng $ax^2 + bx + c = 0$.

Về mặt toán học, đại lượng x được gọi là biến của phương trình. Tuy nhiên, trong lập trình, khái niệm biến có khác đôi chút. Trước tiên, ta phân tích đầu vào, đầu ra, các đại lượng trung gian của chương trình.

- Đầu vào: a, b, c .
- Giá trị trung gian: $\Delta = b^2 - 4ac$.
- Đầu ra: x_1, x_2 (nếu có).

Khi thực thi chương trình, người dùng sẽ nhập các giá trị đầu vào (a, b, c). Khi đó, chúng cần được lưu trữ trong bộ nhớ. Tiếp theo, ta có thể phải tính đại lượng trung gian (Δ) và giá trị này cũng có thể được lưu trữ trong bộ nhớ. Cuối cùng, các giá trị x_1 và x_2 (nếu có) được tính toán và cũng có thể được lưu trữ trong bộ nhớ. Tóm lại, với chương trình trên, tối đa ta sẽ sử dụng 5 ô nhớ khác nhau để lưu trữ các đại lượng của chương trình. Năm ô nhớ này được gọi là các biến của chương trình.

Biến là khái niệm dùng để chỉ các ô nhớ để lưu trữ các giá trị đầu vào, đầu ra hoặc giá trị trung gian của chương trình trong bộ nhớ.

Mỗi biến có: địa chỉ, giá trị. Địa chỉ ô nhớ trên thiết bị lưu trữ mà biến đó đang được lưu được gọi là địa chỉ của biến. Dữ liệu do biến lưu trữ được gọi là giá trị của biến.

Để quản lý và dễ dàng truy cập vào biến, chúng được đặt tên. Tên biến được đặt theo một số quy tắc đặt tên trong C++:

Quy tắc 1: Tên phải bắt đầu bằng một chữ cái trong bảng chữ cái hoặc một dấu gạch dưới “_”.

Quy tắc 2: Sau ký tự đầu tiên, các tên cũng có thể chứa các chữ cái hoặc chữ số, không được chứa khoảng trắng và ký tự đặc biệt.

Quy tắc 3: Các tên không được đặt trùng với từ khóa của C++.

Quy tắc 4: Các tên có phân biệt chữ hoa chữ thường.

Có nhiều loại biến khác nhau gọi là “*kiểu biến*” tùy thuộc vào kích thước của chúng, loại dữ liệu được lưu trữ. Một số kiểu cơ bản như: int, long (số nguyên); float, double (số thực); char (ký tự); bool (chỉ nhận một trong hai giá trị true, false); ...

Biến cần được khai báo trước khi sử dụng. Cú pháp khai báo biến:

`<kiểu_var> <tên_var>;`

Trong đó, `<tên_var>` được người lập trình đặt tuân theo các quy tắc đặt tên ở trên và `<kiểu_var>` được lựa chọn sao cho phù hợp với kiểu dữ liệu được lưu trữ. Một số kiểu biến thông dụng như: int, long, long long (để lưu trữ các số nguyên), float, double (để lưu trữ các số thực), char (để lưu trữ các ký tự), bool (để lưu trữ giá trị logic đúng/sai)...

Các kiểu biến như: int, float, double, char, bool,... là các kiểu có sẵn trong môi trường lập trình C++ và được gọi là các kiểu “nguyên thủy”. Ngoài ra, biến cũng có thể có kiểu do ta tự định nghĩa ra (ví dụ kiểu struct, kiểu class).

1.2. Biểu thức

Để tính toán các величин, ta sử dụng các biểu thức. C++ cho phép biểu diễn các biểu thức toán học một cách dễ dàng. Một biểu thức bao gồm 2 thành phần:

Các toán tử: Được tạm phân chia làm 4 loại theo chức năng của chúng. Sau đây là một số toán tử hay dùng:

- Các toán tử số học:

Số thứ tự	Toán tử	Cách viết
1	Cộng	+
2	Trừ	-
3	Nhân	*
4	Chia	/
5	Đồng dư	%
6	Tăng 1 đơn vị	++
7	Giảm 1 đơn vị	--

- Các toán tử Logic:

Số thứ tự	Toán tử	Cách viết
1	Và	&&
2	Hoặc	
3	Phủ định	!

- Các toán tử so sánh:

Số thứ tự	Toán tử	Cách viết
1	Lớn hơn	>
2	Nhỏ hơn	<
3	Lớn hơn hoặc bằng	>=
4	Nhỏ hơn hoặc bằng	<=
5	Bằng	==
6	Không bằng	!=

- Toán tử gán:

Số thứ tự	Toán tử	Cách viết
1	Gán	=

Các toán hạng: Có thể chia làm 3 loại gồm: Hằng, biến và hàm.

Hằng: gồm hằng số, hằng chuỗi ký tự và hằng ký tự. Hằng chuỗi ký tự khi viết cần được đặt giữa hai dấu ngoặc kép “ ”; hằng ký tự được đặt giữa hai dấu nháy đơn ‘ ’ còn hằng số thì không.

Hàm: Thường được dùng trong biểu thức là những hàm tính và trả về các giá trị của một đại lượng. Một số hàm phổ biến như sau:

Số thứ tự	Đại lượng toán học	Cách viết trong C++
1	\sqrt{x}	<code>sqrt(x)</code>
2	x^n	<code>pow(x, n)</code>
3	e^x	<code>exp(x)</code>
4	$\ln(x)$	<code>log(x)</code>
5	$\log_{10}(x)$	<code>log10(x)</code>
6	$ x $	<code>fabs(x)</code>
7	$\sin x$	<code>sin(x)</code>
8	$\cos x$	<code>cos(x)</code>
9	x^n	<code>pow(x, n)</code>

☞ Để sử dụng các hàm toán học ở trên, ta cần khai báo chỉ thị tiền xử lý `#include "math.h"` hoặc `#include "CMath"`.

1.3. Lệnh nhập

Lệnh `cin` cho phép nhập dữ liệu từ bàn phím vào một biến. Thông thường, để nhập dữ liệu kiểu số hoặc kiểu ký tự, ta viết:

```
cin>>(<tên_var>);
```

Để nhập dữ liệu kiểu chuỗi ký tự, ta có thể dùng `cin` theo cú pháp:

```
cin.getline (<tên_var>, <k>);
```

Trong đó `k` là một số nguyên đủ lớn, là số ký tự tối đa muốn nhập.

☞ Lệnh `cin.getline` được dùng cho các biến được khai báo theo kiểu mảng `char`, ví dụ: `char HOTEN[30];` khác với lệnh `getline(cin, <tên_bien>);` được dùng cho các biến kiểu `string`, ví dụ: `string HOTEN;`

Đôi khi ta có thể sử dụng lệnh `gets` để nhập dữ liệu từ bàn phím vào một biến chuỗi ký tự kiểu mảng `char`:

```
fflush(stdin); gets(<tên_bien>);
```

Trong đó, lệnh `fflush` được đưa vào để làm sạch bộ đệm bàn phím trước khi nhập dữ liệu để tránh trường hợp lệnh `gets` bị vô hiệu hóa do bộ đệm bàn phím chưa được làm sạch.

1.4. Lệnh xuất

Để xuất dữ liệu ra màn hình, ta có thể sử dụng lệnh `cout` như sau:

```
cout<<<dữ_liệu>;
```

Trong đó, ký hiệu “`<<`” được gọi là toán tử xuất và `<dữ_liệu>` có thể là:

Sđt	<code><dữ_liệu></code>	Ví dụ
1	Hằng	<code>cout<<1;</code> <code>cout<<'1';</code> <code>cout<<"1";</code>
2	Biến	<code>cout<<a;</code> <code>cout<<HOTEN;</code>
3	Biểu thức	<code>cout<<a+b;</code> <code>cout<<sqrt(a) + exp(b);</code>
4	Cờ định dạng	<code>cout<<endl;</code> <code>cout<<setw(10)<<a;</code>

☞ Ta có thể sử dụng liên tiếp các toán tử xuất trên cùng một lệnh `cout`, ví dụ câu lệnh: `cout<<"a ["<<3<<"] ="<<0;` sẽ in ra màn hình:

```
a [ 3 ]=0 .
```

Các cờ định dạng sẽ ảnh hưởng tới dữ liệu ở ngay sau nó trên dòng `cout`. Một số cờ định dạng thường dùng gồm:

Số	Cờ định dạng	Ý nghĩa
1	<code>setw(n)</code>	Xác định tối thiểu n vị trí để xuất dữ liệu
2	<code>setprecision(n)</code>	Xác định số chữ số tối đa phần thập phân là n
3	<code>fixed</code>	Chỉ định xuất dữ liệu với phần thập phân kiểu fixed-point notation.
4	<code>scientific</code>	Chỉ định xuất dữ liệu với phần thập phân kiểu scientific notation.

✓ Phân biệt fixed-point và scientific notation: Với kiểu scientific notation, dữ liệu sẽ được biểu diễn dạng số mũ, ví dụ với biến a ở trên, câu lệnh:

```
cout<<setprecision(3)<<scientific<<a;
```

sẽ in ra $2.131e+000$, tức là 2.131×10^{000} .

Với phần thập phân được in ra có kiểu fixed-point notation, sẽ in ra chính xác số chữ số trong phần thập phân như được chỉ định bởi cờ setprecision, ví dụ với biến $a = 2.1314$, câu lệnh:

```
cout<<setprecision(3)<<a;
```

sẽ in ra 2.13 , nhưng lệnh

```
cout<<setprecision(3)<<fixed<<a;
```

sẽ in ra 2.131 với 3 chữ số phần thập phân.

1.5. Ví dụ tổng hợp

❶ Ví dụ 1.1: Viết chương trình tính khoảng cách Euclidean giữa hai điểm A(x_1, y_1) và B(x_2, y_2) trong không gian hai chiều theo công thức:

$$d(A, B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Trước tiên ta xác định đầu vào, đầu ra của chương trình. Đầu vào: x_1, y_1, x_2, y_2 . Đầu ra: $d(A, B)$.

Chương trình sẽ bao gồm 5 biến, ta tạm đặt tên là $x1, y1, x2, y2, d$. Chương trình sẽ thực hiện lần lượt các công việc sau:

- Khai báo các biến $x1, y1, x2, y2$ (dòng 3).
- Nhập dữ liệu từ bàn phím vào các biến trên (dòng 4 tới 7).
- Khai báo biến d và tính giá trị của d theo công thức (dòng 8).
- Xuất kết quả (d) lên màn hình (dòng 9).

```
1 int main()
2 {
3     float x1, y1, x2, y2;
4     cout << "x1 = "; cin >> x1;
5     cout << "y1 = "; cin >> y1;
6     cout << "x2 = "; cin >> x2;
7     cout << "y2 = "; cin >> y2;
8     double d = sqrt((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1));
9     cout << "d = " << fixed << d;
10 }
```

☞ *Trình tự các bước:* Một chương trình đơn giản trong C++ thông thường sẽ bao gồm các bước theo trình tự sau: Khai báo biến và nhập dữ liệu đầu vào, khai báo biến đầu ra và tính đầu ra theo công thức, xuất kết quả tính được ra màn hình.

❷ Ví dụ 1.2: Viết chương trình tính giá trị của biểu thức sau, với $x, y \in \mathbb{R}$ và $n \in \mathbb{Z}$:

$$P(x, y, n) = e^{|x|+1} + y^n \sqrt{x^2 + \frac{n^2}{3}}.$$

Ta xác định được đầu vào của chương trình là: x (số thực), y (số thực), n (số nguyên); và đầu ra: P (số thực). Chương trình sẽ thực hiện lần lượt các công việc sau:

- Khai báo các biến x, y, n (dòng 3).
- Nhập dữ liệu từ bàn phím vào các biến trên (dòng 4, 5, 6).
- Khai báo biến P và tính giá trị của P theo công thức (dòng 7, 8).
- Xuất kết quả (P) lên màn hình (dòng 9).

```

1 int main()
2 {
3     float x, y; int n;
4     cout<< "x = "; cin>>x;
5     cout<< "y = "; cin>>y;
6     cout<< "n = "; cin>>n;
7     double P = exp(fabs(x)+1) + pow(y, n)* sqrt(x*x +
8         (double) n*n/3);
9     cout<<"P = "<<setprecision(2)<<fixed<<P;
10 }

```

*☞ Thao tác ép kiểu: Trong C++, phép chia một số nguyên cho một số nguyên sẽ thu được kết quả là một số nguyên (là phần nguyên của kết quả phép chia). Ví dụ phép chia $2/3$ thu được kết quả là 0 . Để kết quả là số thực, ta cần “ép kiểu” phân số đó về kiểu thực. Khi đó, ta viết kiểu thực *đằng trước* phân số và kiểu thực này được đặt trong ngoặc đơn. Ví dụ: ta viết (double) $n*n/3$ thay vì viết $n*n/3$.*

Một số câu lệnh đơn giản dùng cho giao diện chương trình:

Số	Câu lệnh	Chức năng
1	system("cls");	Xóa màn hình
2	system("pause");	Tạm dừng chương trình cho tới khi bấm phím “Enter”
3	system("COLOR xy");	Thiết lập màu nền (x) và màu chữ (y) với x, y là các mã màu. Ví dụ: system("COLOR 14");
4	system("date");	Hiển thị ngày tháng năm của hệ thống
5	system("time");	Hiển thị giờ phút giây của hệ thống
6	system("dir");	Hiển thị thư mục hiện tại
7	sleep(n);	Tạm dừng chương trình n

		giây sau đó thực thi tiếp.
--	--	----------------------------

B. Các bài tập tự giải

Bài S1. Nhập vào họ tên, quê quán, số ngày công của một công nhân. In các thông tin vừa nhập lên màn hình kèm theo tiền công, biết rằng mỗi ngày công được trả 350 ngàn đồng.

Bài S2. Viết chương trình nhập vào họ và tên khách hàng, ngày/tháng/năm mua hàng, tên hàng, số lượng, đơn giá và in ra màn hình phiếu mua hàng tương ứng như mẫu dưới đây, biết rằng thành tiền được tính bằng đơn giá nhân với số lượng mua và được in ra với độ chính xác 3 chữ số hàng thập phân:

CONG TY GROSS			
PHIEU MUA HANG			
Khach hang: <Họ _ tên _ khách>.			Ngay mua: <ngày/tháng/năm>.
Ten hang	Don gia	So luong	Thanh tien
<Tên _ hàng>	<Đơn _ giá>	<Số _ lượng>	<Thành _ Tiền>

Hãy căn chỉnh lại code để phiếu mua hàng in ra được đẹp hơn.

Bài S3. Viết chương trình nhập vào một số thực x từ bàn phím, tính và in ra màn hình giá trị của $F(x) = \sin^2(x) + |x| + e^{ln(x)}$ với độ chính xác 2 chữ số hàng thập phân.

Bài S4. Nhập một số nguyên có ít hơn 5 chữ số, in ra màn hình cách đọc số nguyên đó (ví dụ: Số 1523 đọc là: 1 ngàn 5 trăm 2 chục 3 đơn vị). Hãy nhận xét về cách làm vừa áp dụng nếu số nguyên nhập vào không được giới hạn. Thử đưa ra phương án đọc số hoàn toàn (ví dụ: Với số 1304 đọc là: Một nghìn ba trăm linh tư).

Bài S5. Viết chương trình nhập vào các tọa độ của ba điểm $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$. Tính và in ra màn hình chu vi và diện tích của tam giác ABC, biết rằng diện tích tam giác được tính bằng công thức Heron:

$$S = \sqrt{p(p - a)(p - b)(p - c)},$$

với p là nửa chu vi và a, b, c là độ dài ba cạnh của tam giác.

Bài S6. Viết chương trình nhập vào ba tọa độ của một véc tơ $A(x, y, z)$. Tính và in ra màn hình “chuẩn 2” của A:

$$\|A\| = \sqrt{x^2 + y^2 + z^2}.$$

Bài S7. Cho hai điểm A(x₁, y₁), B(x₂, y₂) trên mặt phẳng tọa độ. Viết chương trình nhập vào x₁, x₂, y₁, y₂. Tính và in ra màn hình:

- Khoảng cách Euclidean giữa A và B theo công thức:

$$D = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

- Khoảng cách Manhattan giữa A và B:

$$M = |x_2 - x_1| + |y_2 - y_1|$$

- Khoảng cách Cosin giữa A và B:

$$C = \frac{x_1 x_2 + y_1 y_2}{\sqrt{x_1^2 + y_1^2} \sqrt{x_2^2 + y_2^2}}.$$

Bài S8. Viết chương trình nhập vào các giá trị x, y ∈ R và n ∈ Z. Tính và in ra màn hình giá trị biểu thức:

$$P(x, y, n) = (|x+y| + \sqrt[5]{x^2} + \frac{n}{3}).$$

Bài S9. Cho ba điểm A(x₁, y₁), B(x₂, y₂), C(x₃, y₃) trên mặt phẳng tọa độ XOY. Gọi K(x, y) là tâm của 3 điểm A, B, C với x (và y) là trung bình cộng các tọa độ trên trục x (và trên trục y) của 3 điểm A, B, C. Độ đo *Inter* được định nghĩa là tổng khoảng cách Euclidian giữa các điểm A, B, C đến K. Hãy:

- Nhập vào tọa độ của 3 điểm A, B, C từ bàn phím.
- Tính tọa độ của K theo định nghĩa về K ở trên.
- Tính *Inter* theo định nghĩa:

$$\begin{aligned} Inter = & \sqrt{(x_1 - x)^2 + (y_1 - y)^2} + \sqrt{(x_2 - x)^2 + (y_2 - y)^2} + \\ & \sqrt{(x_3 - x)^2 + (y_3 - y)^2}. \end{aligned}$$

BÀI 2

CẤU TRÚC ĐIỀU KHIỂN TRONG C++

Trong bài này, chúng ta sẽ ôn lại một số vấn đề liên quan tới các cấu trúc điều khiển của C++ như: Cấu trúc rẽ nhánh, cấu trúc chọn, cấu trúc lặp và sử dụng chúng trong các bài tập tương ứng.

A. Tóm tắt lý thuyết

2.1. Cấu trúc rẽ nhánh

Để biểu diễn các mệnh đề:

[1] Nếu X thì Y

[2] Nếu X thì Y, ngược lại thì Z

ta lần lượt sử dụng hai dạng của cấu trúc rẽ nhánh như sau:

Với mệnh đề [1]: **if** (X)

Y;

Với mệnh đề [2]: **if** (X)

Y;

else

Z;

Trong đó, X là một biểu thức logic (tức chỉ nhận một trong hai giá trị là đúng (`true/1`) hoặc sai (`false/0`)). Y, Z có thể là một lệnh, một khối lệnh, một cấu trúc điều khiển hoặc một khối cấu trúc điều khiển.

☞ Khối lệnh được hiểu là tập các lệnh được đặt giữa hai ký hiệu “{” và “}”. Khi đó toàn bộ khối lệnh được coi như là một lệnh gộp.

Ví dụ về khối lệnh:

Sđt	C++ code	Chú thích
1	<code>if(a > b)</code>	
2	<code>{</code>	Bắt đầu khối lệnh
3	<code> cout<<a;</code>	Lệnh thứ nhất của khối lệnh
4	<code> a++;</code>	Lệnh thứ hai của khối lệnh

☞ Theo chuẩn ANSI, nếu Y và Z có từ 2 lệnh trở lên, các lệnh đó cần được tạo thành khối lệnh. Trong trường hợp Y, Z chỉ là một lệnh, ta không tạo khối lệnh. Tương tự với Y, Z là các cấu trúc điều khiển.

2.2. Cấu trúc chọn

Với trường hợp có quá nhiều nhánh để lựa chọn, ta có thể sử dụng một cấu trúc điều khiển khác phù hợp hơn, đó là cấu trúc chọn. Cú pháp:

```
switch (X)
{
    case <giá_trị_1>:   <Nhóm lệnh 1>; [break;]
    case <giá_trị_2>:   <Nhóm lệnh 2>; [break;]
    ...
    case <giá_trị_n>:   <Nhóm lệnh n>; [break;]
    [default]:
        <Nhóm lệnh mặc định>;
}
```

Trong đó:

- X là một biểu thức bất kỳ và phải nhận giá trị nguyên.
- Không cho phép tồn tại hai **case** mà hai giá trị theo sau của nó bị trùng nhau.
- Lệnh **[break;]** để kết thúc một chuỗi câu lệnh. Khi đạt đến **break;**, **switch** kết thúc và luồng điều khiển sẽ chuyển sang dòng tiếp theo sau **switch**. Nếu thiếu **break;** luồng kiểm soát sẽ được chuyển sang các **case** tiếp theo cho đến khi gặp **break;** hoặc kết thúc **switch**.
- Nếu X không nhận bất kỳ các giá trị nào đã liệt kê trong các **case**, luồng điều khiển sẽ thực hiện <Nhóm lệnh mặc định>; và thoát khỏi cấu trúc **switch**.

*☞ Nhóm lệnh có thể là một tập hợp nhiều lệnh mà không cần tạo khối lệnh. X phải nhận các giá trị rời rạc. Trong trường hợp X nhận các giá trị liên tục, câu lệnh **switch** là không khả thi.*

Trong ví dụ sau đây, biến *a* có kiểu liên tục và câu lệnh switch là không hợp lệ:

Sđt	C++ code	Chú thích
1	float a=2;	<i>a</i> là biến liên
2	switch (<i>a</i>)	tục
3	{	Lỗi xảy ra
4	case 1: cout<<1; break;	
5	case 2: cout<<2; break;	
6	case 3: cout<<3; break;	
7	default:	
8	cout<<" ";	
9	}	

2.3. Cấu trúc lặp với số lần lặp xác định

Cú pháp:

```
for (<BT1>; <BT2>; <BT3>)
    < Khối lệnh lặp>Khối lệnh lặp>;
```

Vòng lặp sau sẽ in ra màn hình các số nguyên từ 0 tới 9, mỗi số trên một dòng.

```
for(int i=0; i<10; i++)
    cout<<i<<endl;
```

Khi đó:

<BT1>	i=0
<BT2>	i<10
<BT3>	i++
<Khối lệnh lặp>	cout<<i<<endl;

Trình tự thực hiện:

Bước 1: Thực hiện <BT1>;

Bước 2: Kiểm tra <BT2>. Nếu <BT2> sai (nhận giá trị `false`), thoát khỏi vòng lặp và chuyển tới lệnh tiếp theo sau vòng lặp (nếu có). Nếu <BT2> đúng (nhận giá trị `true`):

- Thực hiện <Khối lệnh lặp>;

- Thực hiện $\langle BT3 \rangle$;

- Quay lại Bước 2.

$\Leftrightarrow \langle BT2 \rangle$ phải là một biểu thức logic, tức chỉ nhận giá trị `true` hoặc `false`. Trong trường hợp $\langle BT2 \rangle$ luôn đúng, ta có một vòng lặp vô hạn. Khi đó, để dừng vòng lặp, ta cần đặt lệnh `break`; trong thân vòng lặp với điều kiện dừng phù hợp để dừng vòng lặp.

\Leftrightarrow *(Khởi lệnh lặp)*: Có thể là 1 lệnh, 1 khối lệnh, 1 cấu trúc điều khiển hoặc một khối cấu trúc điều khiển.

2.4. Cấu trúc lặp với số lần lặp không xác định

Vòng lặp **while**:

```
while ( $\langle$ Điều_kiện_lặp $\rangle$ )
     $\langle$ Khởi_lệnh_lặp $\rangle$ ;
```

Vòng lặp **do/while**:

```
do
     $\langle$ Khởi_lệnh_lặp $\rangle$ ;
while ( $\langle$ Điều_kiện_lặp $\rangle$ );
```

Trình tự thực hiện:

Với vòng lặp **while**: Kiểm tra \langle Điều_kiện_lặp \rangle . Nếu \langle Điều_kiện_lặp \rangle đúng, thực hiện \langle Khởi_lệnh_lặp \rangle ; Nếu \langle Điều_kiện_lặp \rangle sai, dừng vòng lặp và chuyển tới lệnh tiếp theo sau vòng lặp (nếu có).

Với vòng lặp **do/while**:

Bước 1; Thực hiện \langle Khởi_lệnh_lặp \rangle ..

Bước 2: Kiểm tra \langle Điều_kiện_lặp \rangle . Nếu \langle Điều_kiện_lặp \rangle đúng, quay lại Bước 1. Nếu \langle Điều_kiện_lặp \rangle sai, dừng vòng lặp và chuyển tới lệnh tiếp theo sau vòng lặp (nếu có).

\Leftrightarrow *(Điều_kiện_lặp)*: Phải là một biểu thức logic. Nếu \langle Điều_kiện_lặp \rangle luôn đúng, ta có vòng lặp vô hạn. Khi đó, cần sử dụng lệnh `break`; trong thân vòng lặp một cách hợp lý để dừng vòng lặp.

(Khối lệnh lặp): Có thể là 1 lệnh, 1 khối lệnh, 1 cấu trúc điều khiển hoặc một khối cấu trúc điều khiển.

2.5. Ví dụ tổng hợp

Cấu trúc rẽ nhánh:

Ví dụ 2.1: Viết chương trình nhập vào ba số thực a, b, c . Hãy kiểm tra xem dữ liệu nhập vào có thỏa mãn là ba cạnh của tam giác hay không. Nếu có, hãy kiểm tra xem đó là tam giác thường, tam giác cân hay tam giác đều.

Đầu vào của chương trình là các biến a, b, c nhập từ bàn phím. Để a, b, c là ba cạnh của một tam giác, điều kiện là “*tổng của hai cạnh bất kỳ luôn lớn hơn cạnh thứ 3*”. Nếu dữ liệu nhập vào đã thỏa mãn điều kiện này, ta cần kiểm tra:

- Tam giác đều: Các cạnh a, b, c phải thỏa mãn điều kiện $a = b = c$.
- Tam giác cân: Các cạnh a, b, c phải không là tam giác đều và thỏa mãn thêm điều kiện $(a = b)$ hoặc $(b = c)$ hoặc $(c = a)$.

C++ code của chương trình như sau:

Số	C++ code
1	int main ()
2	{
3	float a, b, c;
4	cout<<"a="; cin>>a;
5	cout<<"b="; cin>>b;
6	cout<<"c="; cin>>c;
7	if (a+b <= c b+c <= a c+a <= b)
8	cout<<"Du lieu khong hop le";
9	else
10	{
11	cout<<"Du lieu hop le"<<endl;
12	if (a==b && b==c)
13	cout<<"Tam giac deu";
14	else
15	if (a==b b==c c==a)
16	cout<<"Tam giac can";
17	else
18	cout<<"Tam giac thuong";

19	}
20	}

Cấu trúc chọn:

❶ **Ví dụ 2.2:** Viết chương trình nhập vào tháng và năm của một năm dương lịch. In ra số ngày trong tháng (để đơn giản, ta quy ước tháng 2 của các năm chia hết cho 4 thì có 29 ngày, của các năm còn lại có 28 ngày).

Ta chia các tháng ra thành ba loại với các tháng có 31, 30 ngày. Riêng tháng 2 có thể có 28 hoặc 29 ngày. Tuy nhiên, ta cần xét khả năng người dùng nhập vào một tháng không hợp lệ (không thuộc [1, 12]). Với bài toán này, có thể sử dụng cấu trúc **if**. Tuy nhiên, việc viết biểu thức điều kiện trong **if** khá dài dòng. Do vậy ta có thể sử dụng cấu trúc **switch** như sau:

Số	C++ code
1	int main ()
2	{
3	int thang, nam;
4	cout<<"thang="; cin>>thang;
5	cout<<"nam ="; cin>>nam;
6	switch (thang)
7	{
8	case 4:
9	case 6:
10	case 9:
11	case 11: cout<<"Thang co 30 ngay";
12	break;
13	case 1:
14	case 3:
15	case 5:
16	case 7:
17	case 8:
18	case 10:
19	case 12: cout<<"Thang co 31 ngay";
20	break;
21	case 2:
22	if (nam%4==0)

23	cout<<"Thang co 29 ngay";
24	else
25	cout<<"Thang co 28 ngay";
26	break;
27	default:
28	cout<<"Thang khong hop le";
}	}
}	

Cấu trúc lặp for:

❶ Ví dụ 2.3: Viết chương trình in ra màn hình một lá cờ có kích thước $n \times m$ bằng các dấu “*”.

Ta cần in ra các dấu “*” được bố trí trên n dòng, m cột. Trước tiên, ta duyệt qua các dòng bằng vòng lặp:

```
for (int i=0; i<n; i++)
```

Với mỗi dòng, ta cần làm hai việc: 1) In ra m dấu “*”, và 2) xuống dòng. Việc in ra m dấu “*” cũng được sử dụng vòng lặp:

```
for (int j=0; j<m; j++)
    cout<<"*";
```

Số	C++ code
1	int main ()
2	{
3	int n, m;
4	cout<<"n="; cin>>n;
5	cout<<"m="; cin>>m;
6	for (int i=0; i<n; i++)
7	{
8	for (int j=0; j<m; j++)
9	cout<<"*";
10	cout<<endl;
11	}
12	}

❶ **Ví dụ 2.4:** Viết chương trình nhập giá trị mua vào của một tài sản, tỷ lệ % khấu hao của tài sản qua từng năm, số năm sử dụng. Tính và in ra giá trị còn lại của tài sản.

Gọi giá trị mua vào của tài sản là G, tỷ lệ khấu hao là K (%) và số năm sử dụng là Y. Giá trị còn lại của tài sản lúc mới mua về R bằng chính giá trị mua vào G. Qua mỗi năm, giá trị còn lại của tài sản giảm đi một lượng bằng K% của giá trị còn lại tại thời điểm đó.

Sđt	C++ code
1	int main ()
2	{
3	float G, K; int Y;
4	cout<<"Gia tri mua = "; cin>>G;
5	cout<<"Ty le kh hao= "; cin>>K;
6	cout<<"So nam sd = "; cin>>Y;
7	double R = G;
8	for (int i=1; i<=Y; i++)
9	R = R - K*R/100;
10	cout<<"Gia tri con lai = "<<R;
11	}

Cấu trúc lặp while, do/while:

❷ **Ví dụ 2.5:** Viết chương trình nhập vào một số nguyên dương k. Tìm và in ra số nguyên là lũy thừa 2 nhưng bé nhất trong các số lớn hơn k.

Hiện nhiên k chỉ được nhập giá trị khi thực thi chương trình. Do vậy, khi viết code, ta thực sự chưa biết được số cần tìm là bao nhiêu. Phương pháp đơn giản là: Gọi số cần tìm là S thì S phải là lũy thừa của 2. Do vậy ta khởi gán $S = 1$ (tức 2^0), sau đó tăng dần S và vẫn đảm bảo S là lũy thừa của 2 (mỗi lần tăng ta nhân S với 2), cho tới khi S lớn hơn k thì dừng lại.

Rõ ràng là số lần tăng S lên (số lần lặp) phụ thuộc vào giá trị k. Do vậy, sử dụng **while** hay **do/while** là phù hợp trong trường hợp này. Hãy so sánh hai đoạn code dưới đây để nhận thấy sự giống và khác nhau khi sử dụng hai cấu trúc lặp này.

Sđt	C++ code sử dụng while
1	int main ()
2	{

3	int k;
4	cout<<"k="; cin>>k;
5	long S = 1;
6	while (S<=k)
7	S = S*2;
8	cout<<"So can tim: "<<S;
9	}

Stt	C++ code sử dụng do/while
1	int main ()
2	{
3	int k;
4	cout<<"k="; cin>>k;
5	long S = 1;
6	do
7	S = S*2;
8	while (S<=k);
9	cout<<"So can tim: "<<S;
10	}

Sử dụng linh hoạt các cấu trúc điều khiển:

❷ **Ví dụ 2.6:** Viết chương trình nhập vào một số thực x và một số nguyên $n \in [1, 100]$. Nếu giá trị n nhập vào không thỏa mãn $n \in [1, 100]$, cho phép nhập lại cho tới khi thỏa mãn điều kiện. Tính và in ra màn hình giá trị của:

$$P = \begin{cases} 10e^n(\sqrt[3]{x}) & \text{với } n < 10 \\ x^n + \frac{x^2}{3} + \frac{x^3}{3^2} + \dots + \frac{x^n}{3^{n-1}} & \text{với } n \geq 10 \end{cases}$$

Hiển nhiên ta không thể biết trước giá trị của n mà người dùng sẽ nhập vào. Do vậy, để kiểm tra giá trị nhập vào của n và cho phép nhập lại n nếu không thỏa mãn điều kiện, ta sẽ sử dụng vòng lặp không xác định **do/while** (hoặc có thể sử dụng **while**).

Để tính P , ta chia bài toán thành 2 trường hợp (sử dụng cấu trúc **if**). Với trường hợp $n \geq 10$, để tính tổng chuỗi số, ta sử dụng vòng lặp **for**.

Trước tiên ta xét chương trình đơn giản sau:

Sđt	C++ code sử dụng do/while
1	int main ()
2	{
3	float x; int n;
4	cout<<"x="; cin>>x;
5	do
6	{
7	cout<<"n="; cin>>n;
8	if (n<1 n>100)
9	cout<<"n khong hop le !"<<endl;
10	}
11	while (n<1 n>100);
12	double P;
13	if (n<10)
14	P = 10* exp (n)* pow (x, 1/3.0);
15	else
16	{
17	P = pow (x, n);
18	for (int i=2; i<=n; i++)
19	P = P + pow (x, i)/ pow (3, i-1);
20	}
21	cout<<"P="<<P;
22	}

☞ Hãy thử nghiệm với $n = 100$. Khi đó, giá trị của P quá lớn và được xác định là inf (*infinity, dương vô cùng*).

Chương trình sau đây tốt hơn theo nghĩa là số thao tác tính toán ít hơn nhưng lại tốn thêm bộ nhớ cho một số biến phụ. Hãy phân tích số thao tác tính toán cần thực hiện của hai chương trình với cùng một bộ giá trị (x, n) để thấy sự tốt hơn này.

Sđt	C++ code sử dụng do/while
1	int main ()
2	{
3	float x; int n;
4	cout<<"x="; cin>>x;
5	do
6	{
7	cout<<"n="; cin>>n;

```

8      if(n<1 || n>100)
9          cout<<"n khong hop le"<<endl;
10     }
11     while(n<1 || n>100);
12     double P;
13     if(n<10)
14         P = 10*exp(n)*pow(x, 1/3.0);
15     else
16     {
17         P = pow(x, n);
18         float T = x;
19         int M = 1;
20         for(int i=2; i<=n; i++)
21         {
22             T = T*x; M = M*3;
23             P = P + T/M;
24         }
25     }
26     cout<<"P="<<P;
27 }
```

B. Các bài tập tự giải

Cấu trúc rẽ nhánh và chọn:

Bài S10. Viết chương trình nhập vào một số nguyên n . Kiểm tra xem n chẵn hay lẻ.

Bài S11. Viết chương trình tính tiền điện tiêu thụ của các hộ gia đình trong một tháng. Biết rằng tiền điện được tính theo đơn giá như sau: Từ kwh thứ nhất đến kwh thứ 100 là 1500 đồng/ kwh ; từ kwh thứ 101 đến kwh thứ 200 là 2000 đồng/ kwh ; từ kwh thứ 201 đến kwh thứ 300 là 3000 đồng/ kwh ; từ kwh thứ 301 trở đi là 5000 đồng/ kwh .

Bài S12. Viết chương trình tính và in ra màn hình giá trị của biểu thức sau (tùy theo giá trị của x và n được nhập vào):

$$F(x, n) = \begin{cases} x^{n-10} & \text{nếu } n < 10 \\ \ln(x) + n\sqrt{x^2 + 3} & \text{nếu } 10 \leq n \leq 20 \\ 0 & \text{nếu } n > 20. \end{cases}$$

Bài S13. Viết chương trình nhập vào tọa độ của hai điểm A(x_1, y_1) và B(x_2, y_2) trong hệ tọa độ Descartes xOy. Hãy kiểm tra và cho biết A, B có nằm trên một đường thẳng song song với trục tung hoặc trục hoành hay không.

Bài S14. Viết chương trình nhập vào tọa độ của ba điểm A(x_1, y_1), B(x_2, y_2), C(x_3, y_3) trong hệ tọa độ Descartes xOy. Nhập vào tọa độ của điểm X(x, y) và cho biết X có nằm trong hình tam giác ABC hay không.

Bài S15. Viết chương trình nhập vào điểm tổng kết của một học sinh và in ra xếp loại cho học sinh đó với quy định:

- Xếp loại giỏi nếu tổng điểm từ 8.00 trở lên.
- Xếp loại khá nếu tổng điểm từ 7.00 tới cận 8.00.
- Xếp loại trung bình nếu tổng điểm từ 5.00 tới cận 7.00.
- Còn lại, xếp loại yếu.

Cấu trúc lặp:

Bài S16. Nhập vào một số nguyên n . Tính tổng các số nguyên tố trong đoạn $[n, 2n]$.

Bài S17. Viết chương trình nhập vào một số nguyên n , sau đó tính giá trị biểu thức:

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

Bài S18. Viết chương trình nhập vào một số nguyên n , sau đó tính giá trị biểu thức

$$F = \begin{cases} 1 + \frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^n} & \text{nếu } n \text{ chẵn} \\ \sqrt{n^2 + 1} & \text{nếu } n \text{ lẻ} \end{cases}$$

Bài S19. Viết chương trình nhập vào một số nguyên n trong khoảng [10, 20] (nếu số nhập vào không thuộc khoảng đó thì yêu cầu nhập lại tới khi thỏa mãn). Tính và in ra màn hình tổng các số liên tiếp từ 1 tới n .

Bài S20. Viết chương trình nhập vào một số nguyên n , sau đó tính tổng các số nguyên tố thuộc đoạn [1.. n]. Cho biết có bao nhiêu số nguyên tố thuộc đoạn đó.

Bài S21. Viết chương trình tìm số nguyên dương n nhỏ nhất thỏa mãn: $1 + 2 + 3 + \dots + n > 1000$.

BÀI 3

VIẾT VÀ SỬ DỤNG HÀM

Bài học này được dành để tổng hợp các kiến thức, hoàn thiện các kỹ năng về viết và sử dụng hàm trong C++. Ngoài các kiến thức cơ bản như: Viết và sử dụng hàm, tổ chức hàm, chúng ta sẽ tiếp cận tới hai kỹ thuật: Truyền tham số và đệ quy.

A. Tóm tắt lý thuyết

3.1. Viết hàm

Các đặc trưng của hàm

- **Tên hàm:** Do người lập trình tự đặt và có những đặc điểm sau:
 - + Tên hàm thường mang tính đại diện cho công việc mà hàm sẽ đảm nhiệm.
 - + Tên hàm được đặt theo quy ước đặt tên trong C++ (xem quy ước đặt tên biến).
- **Kiểu giá trị trả về của hàm:** Nếu hàm trả về một giá trị thì giá trị đó được gán vào lời gọi hàm và nó phải thuộc một kiểu dữ liệu nào đó mà ta gọi là kiểu giá trị trả về của hàm. Nó có thể là các kiểu dữ liệu nguyên thủy hoặc một kiểu do người lập trình tự định nghĩa.
 - **Kiểu và tên các tham số của hàm:** Nếu hàm sử dụng các tham số (các giá trị đầu vào) thì chúng phải thuộc một kiểu dữ liệu nào đó. Khi thiết lập một hàm, ta cần chỉ ra danh sách các tham số của hàm và kiểu dữ liệu của mỗi tham số.
 - **Thân hàm:** Là nội dung chính của hàm, chứa toàn bộ các lệnh của hàm.

Phân loại hàm

Một chương trình trong C++ được cấu tạo từ các hàm, trong đó hàm *main* là hàm bắt buộc phải có, đóng vai trò như chương trình chính.

Hàm được chia làm hai loại:

- **Hàm không có giá trị trả về:** Là hàm chỉ có chức năng thực hiện một công việc nào đó mà không có hoặc ta không quan tâm tới giá trị trả về của hàm.
- **Hàm có giá trị trả về:** Ngoài việc thực hiện một công việc nào đó, ta còn quan tâm tới giá trị thu được sau khi hàm thực thi để lưu trữ lại và dùng trong những đoạn trình tiếp theo.

Tùy theo nguồn gốc của hàm người ta phân ra:

- **Các hàm có sẵn:** Là các hàm chứa trong các thư viện của C++. Thư viện có thể là các file có phần mở rộng .h (hoặc không có phần mở rộng, ví dụ `iostream`, `iomanip...`), đã được định nghĩa từ trước. Người lập trình chỉ việc sử dụng thông qua các chỉ thị: `#include <Tên thư viện chứa hàm>`.
- **Các hàm tự định nghĩa:** Là các hàm do người lập trình tự viết ra. Các hàm này cũng có thể được tập hợp lại trong một file .h để dùng như một thư viện có sẵn.

Một hàm thường có cấu trúc như sau:

```

<Kiểu trả về> <Tên hàm> ([<kiểu tham số> <tên
tham số>])
{
    <Các lệnh trong thân hàm;>
}

```

Trong đó:

- **<Kiểu trả về>:** Là kiểu của giá trị trả về hay đầu ra của hàm. Nếu hàm không có giá trị trả về, ta dùng kiểu trả về là `void`. Ngược lại, ta thường sử dụng các kiểu nguyên thủy như `int`, `float`, `double`, `char...`
- **<Tên hàm>:** Do người dùng tự định nghĩa theo quy ước đặt tên.
- **[<kiểu tham số> <tên tham số>]:** Liệt kê danh sách các tham số của hàm (nếu có) và kiểu dữ liệu của tham số. Nếu hàm có nhiều tham số thì các tham số cách nhau bởi dấu phẩy. Một nguyên tắc trong C++ là mỗi tham số đều phải có một kiểu đi kèm trước tên tham số.

- Nếu hàm có giá trị trả về thì cần có câu lệnh **return** (Giá trị trả về); để gán giá trị này vào tên hàm. Tuyệt đối không được gán $\langle\text{Tên hàm}\rangle = \langle\text{Giá trị trả về}\rangle$; $\langle\text{Giá trị trả về}\rangle$ có thể là một biểu thức, một biến hoặc một hằng.
- Riêng hàm **void** (kiểu trả về là **void**) sẽ không có lệnh **return**.

3.2. Gọi hàm

Hàm được sử dụng thông qua lời gọi nó. Thông thường, chúng được sử dụng chúng trong hàm **main** để giải quyết bài toán đặt ra. Tuy nhiên, về nguyên tắc một hàm bất kỳ đều có thể gọi tới các hàm khác, miễn là các hàm đó đã được định nghĩa trước.

Khi gọi hàm, ta gọi tới tên hàm. Nếu hàm có tham số, ta phải truyền các đối số phù hợp về kiểu vào vị trí các tham số này. Số lượng đối số truyền vào khi gọi hàm phải bằng số lượng các tham số và theo đúng thứ tự khi ta định nghĩa hàm.

Cách viết một lời gọi hàm như sau:

$\langle\text{Tên hàm}\rangle$ ([danh sách các đối số]);

Như vậy:

- Các đối số phải có kiểu trùng với kiểu của tham số tương ứng.
- Nếu hàm không có tham số thì lời gọi hàm vẫn phải sử dụng dấu () kèm tên hàm: $\langle\text{Tên hàm}\rangle()$.

Tuy nhiên, vì hàm có 2 loại: Có và không có giá trị trả về nên cách sử dụng hai loại hàm này cũng khác nhau.

- Nếu hàm có giá trị trả về thì tên hàm được sử dụng như một *biến*, tức là ta không thể sử dụng hàm một cách độc lập mà lời gọi hàm có thể được đặt ở vé phải của phép gán, trong biểu thức hoặc kèm với một lệnh khác.
- Ngược lại, nếu hàm không có giá trị trả về, tên hàm được sử dụng như một *lệnh*, tức là lời gọi hàm được viết độc lập, không viết trong phép gán, trong biểu thức hay kèm với một câu lệnh khác.

3.3. Tổ chức hàm

Khi một chương trình có nhiều hàm, ta quan tâm tới việc tổ chức chúng như thế nào cho khoa học. Thông thường có 2 cách tổ chức các hàm:

Cách 1: Các hàm đặt trong cùng một tệp với chương trình chính

Chương trình ngoài hàm **main** còn có các hàm khác thì các hàm khác có thể đặt trước hoặc sau hàm **main** đều được:

Các hàm đặt trước hàm **main**:

```
<Hàm 1>
<Hàm 2>
...
int main()
{
    //Thân hàm main;
}
```

Các hàm đặt sau hàm **main**:

```
<Nguyên mẫu của hàm 1>;
<Nguyên mẫu của hàm 2>;
...
int main()
{
    //Thân hàm main;
}
<Hàm 1>
<Hàm 2>
...
```

Trong đó, <Nguyên mẫu của hàm> chính là dòng đầu tiên của hàm có kèm theo dấu chấm phẩy ‘;’. Nguyên mẫu của hàm có dạng:

⟨Kiểu trả về⟩ ⟨Tên hàm⟩ ([⟨kiểu tham số⟩ ⟨tên tham số⟩]);

Như vậy, nếu hàm được đặt sau hàm **main** thì cần khai báo nguyên mẫu của hàm trước hàm **main** để chương trình dịch có thể biết trước sự tồn tại của chúng khi dịch hàm **main**.

☞ Các hàm luôn đặt rời nhau. Một hàm không bao giờ được phép đặt trong một hàm khác.

Cách 2: Các hàm đặt trong tệp thư viện

Bước 1: Viết các hàm (trừ hàm **main**) trong một file sau đó lưu dưới định dạng .h. File này thường được gọi là file thư viện (hoặc header file). (để thuận tiện cho việc soát lỗi, tốt nhất trước tiên nên tổ chức các hàm như cách 1, sau khi đảm bảo các hàm chạy tốt, di chuyển toàn bộ các hàm (trừ hàm **main**) sang một file mới và lưu lại với đuôi .h)

Bước 2: Viết hàm **main** trong một tệp riêng. Để hàm **main** có thể sử dụng các hàm viết trong file thư viện đã tạo trong Bước 1, cần thêm chỉ thị:

```
#include <[đường dẫn] Tên thư viện>.
```

3.4. Kỹ thuật truyền tham số

Khái niệm và phân loại cách truyền tham số

Khi định nghĩa hàm, thông thường các giá trị đầu vào được định nghĩa một cách hình thức (giả định) và chúng được gọi là các tham số hình thức.

Khi sử dụng hàm, nếu hàm có tham số, ta phải truyền các đối số thực sự tương ứng cho hàm. Các đối số là các giá trị cụ thể và tương ứng về kiểu với các tham số của hàm. Chúng có thể là các biến hoặc các hằng.

Khi tham số là biến, có hai cách truyền vào hàm:

[1]. Truyền tham trị: Khi truyền tham số dưới dạng tham trị, tham số sẽ không được truy cập trực tiếp. Hàm sẽ cấp phát một vùng nhớ mới và sao chép giá trị của đối số vào đó. Các lệnh trong thân hàm sẽ thao tác trên vùng nhớ mới này. Như vậy, một đối số khi truyền vào một hàm sẽ không bị thay đổi giá trị của nó.

[2]. Truyền tham chiếu: Là khi ta truyền *địa chỉ* của các biến đối số vào hàm. Khi truyền tham số dưới dạng tham chiếu, đối số sẽ được truy cập trực tiếp. Như vậy, đối số có thể bị thay đổi giá trị của nó bởi các lệnh bên trong hàm.

Truyền tham số

Khi truyền một biến (không phải con trỏ) vào hàm tức là ta đã truyền dưới dạng tham trị. Hàm sẽ cấp phát vùng nhớ mới và sao chép giá trị của biến vào ô nhớ này để sử dụng. Khi hàm kết thúc, ô nhớ mới sẽ bị giải phóng và giá trị của biến truyền vào hàm không hề thay đổi. Xét chương trình sau:

```
void tang(int a)
{
    a++;
}

int main()
{
    int n=1;
    cout<<"n trước khi gọi hàm "<<n<<endl;
    tang(n);
    cout<<"n sau khi gọi hàm "<<n<<endl;
}
```

Biến *n* là một biến thông thường và đang mang một giá trị cụ thể (*n*=1), được truyền vào hàm dưới dạng tham trị nên sau khi ra khỏi hàm, giá trị của nó không hề thay đổi (vẫn là 1) mặc dù trong thân hàm `void tang(int a)` giá trị của đối số bị thay đổi (`a++;`).

Nếu ta chỉ truyền địa chỉ của biến vào hàm thì việc truyền như vậy gọi là truyền tham chiếu. Khi đó hàm sẽ tham chiếu trực tiếp tới biến và thao tác trên vùng nhớ của biến truyền vào. Kết quả là giá trị của biến có thể bị thay đổi do tác động của hàm.

```
void tang(int &a)
{
    a++;
}

int main()
{
    int n=1;
    cout<<"n trước khi gọi hàm "<<n<<endl;
    tang(n);
    cout<<"n sau khi gọi hàm "<<n<<endl;
```

}

Dễ thấy hàm: void **tang**(int &a) chỉ nhận địa chỉ của biến truyền vào. Do vậy, với đối số n truyền vào, hàm sẽ thao tác trực tiếp trên đối số này. Kết quả là khi hàm kết thúc, biến n bị thay đổi giá trị (tăng lên 1 đơn vị).

☞ Phép lấy địa chỉ của biến là: &(tên biến). Như vậy, nếu đối số không phải là con trỏ, muốn truyền tham chiếu thì đối số này cần được viết dạng: (kiểu đối) &(tên đối) (ví dụ: int &a).

☞ Khi đối số là con trỏ (ví dụ biến a dạng: int *a;, int a[100]; hoặc int a[]) thì bản thân con trỏ đang chứa địa chỉ nên không cần sử dụng toán tử & để lấy địa chỉ mà vẫn đảm bảo đối số truyền vào cho tham số này là dạng tham chiếu.

3.5. Kỹ thuật đệ quy

Trong C++, một hàm có thể gọi đến chính nó, tính chất này của hàm gọi là tính đệ quy. Khi một hàm thể hiện tính đệ quy, ta gọi hàm đó là hàm đệ quy. Ví dụ ta xét hàm tính $n!$. Ngoài cách viết sử dụng vòng lặp như trên, ta có thể có cách tiếp cận khác để giải quyết bài toán.

Ta định nghĩa: $n! = (n-1)! * n$. Với định nghĩa này, giả sử $n=5$ thì $n!$ được tính như sau:

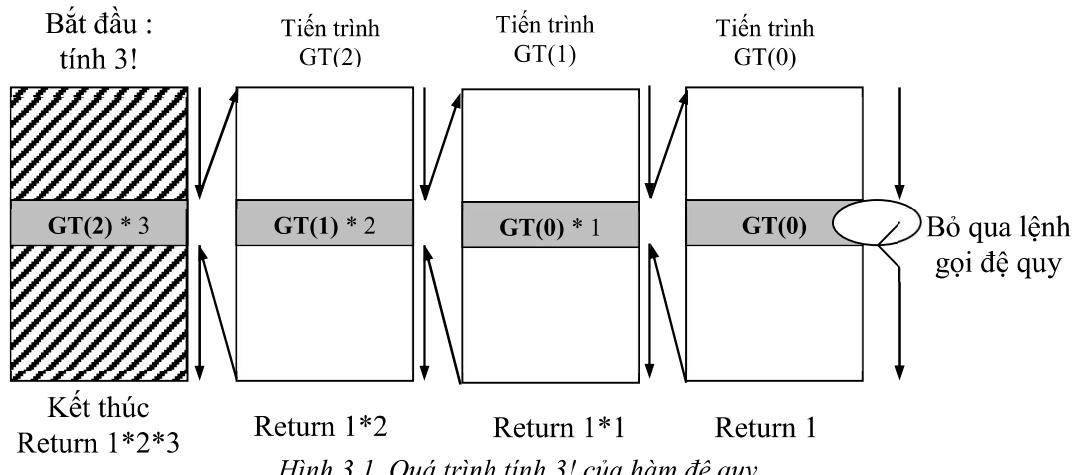
$$\begin{aligned} 5! &= 4! * 5 \\ &= 3! * 4 * 5. \\ &= 2! * 3 * 4 * 5. \\ &= 1! * 2 * 3 * 4 * 5. \\ &= 0! * 2 * 3 * 4 * 5. \end{aligned}$$

Với $0! = 1$, ta hoàn toàn có thể tính được $5!$ bằng cách tính các giá trị $2!, 3!, 4!$ để rồi thay vào công thức $5!=4! * 5$.

Một cách tổng quát, nếu ta có hàm GT(n) để tính $n!$ thì: $GT(n) = GT(n-1) * n \quad \forall n>0$. Đây chính là công thức thể hiện tính đệ quy. Từ công thức này, hàm đệ quy tính $n!$ có thể viết như sau:

```
1 long GT(int n)
2 {
3     if (n==0)
4         return 1;
5     else
6         return GT(n-1) * n;
```

Để hiểu bản chất của đệ quy, ta xét quá trình tính $3!$. Quá trình này được thể hiện qua sơ đồ sau:



Hình 3.1. Quá trình tính $3!$ của hàm đệ quy.

Khi gặp lời gọi $GT(3)$, máy tính tạo ra một tiến trình (*process*) với đầu vào là 3 . Tuy nhiên, khi thực hiện, tiến trình này gặp phải lời gọi đệ quy $3 * GT(2)$.

Khi đó, toàn bộ tiến trình này phải dừng lại và chờ để máy tính tạo ra một tiến trình mới với đối vào là 2 . Khi tiến trình mới này thực hiện xong (tức là tính xong $2!$) nó sẽ quay về tiến trình ban đầu với kết quả $2!$ tính được và tiếp tục thực thi tiến trình ban đầu với kết quả là $2!*3$.

Tuy nhiên, tiến trình tính $2!$ lại gặp lời gọi đệ quy tính $1!$ nên nó cũng phải tạm dừng và chờ 1 tiến trình thứ 3 được tạo ra để tính $1!$.

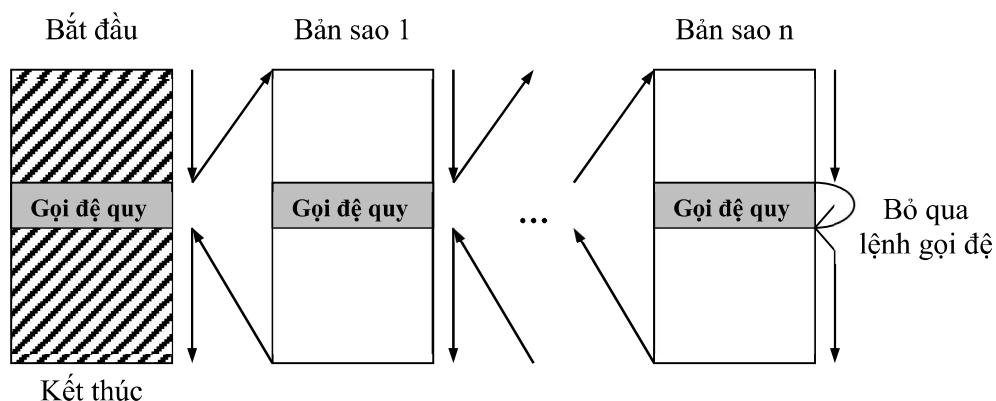
Tiến trình tính $1!$ lại gặp lời gọi đệ quy tính $0!$ nên nó cũng phải tạm dừng và chờ 1 tiến trình thứ 4 được tạo ra để tính $0!$.

Rất may là trong tiến trình tính $0!$ không có lời gọi đệ quy (vì lệnh `if (n==0) return 1;`) nên quá trình dừng-chờ-tạo tiến trình mới không xảy ra. Do vậy các tiến trình đang chờ trước đó lần lượt được khôi phục và trả về kết quả mong muốn.

Như vậy, khi gặp một lời gọi đệ quy, máy tính sẽ:

- Tạm dừng tiến trình hiện tại, lưu địa chỉ của dòng lệnh gọi đệ quy vào ngăn xếp.
- Tạo một tiến trình hoàn toàn mới, cấp phát các vùng nhớ mới cho các biến cục bộ, thực hiện tiến trình mới này.
- Khi việc thực thi tiến trình mới hoàn thành, chương trình quay về ngăn xếp, lấy địa chỉ của dòng lệnh gọi đệ quy và khôi phục tiến trình ban đầu.

Một cách tổng quát ta có sơ đồ quá trình thực thi hàm đệ quy như sau:



Hình 3.2. Quá trình thực thi của hàm đệ quy.

Nó chung, một hàm được viết theo kiểu đệ quy sẽ tốn bộ nhớ hơn, thực thi phức tạp hơn và do vậy người ta thường tìm cách khử đệ quy (tức viết chương trình không theo kiểu đệ quy). Tuy nhiên, cách tiếp cận đệ quy lại tỏ ra phù hợp với các bài toán liên quan tới duyệt cây, đồ thị, danh sách tuyến tính .v.v...

3.6. Thiết kế hàm đệ quy

Các bài toán áp dụng giải thuật đệ quy thường có đặc điểm sau:

- Bài toán dễ dàng giải quyết trong một số trường hợp riêng ứng với các giá trị đặc biệt của tham số. Trong trường hợp này, ta có thể giải quyết bài toán mà không cần gọi đệ quy. Quan trọng hơn, tất cả các trường hợp còn lại đều có thể quy về trường hợp này sau một số hữu hạn

lần thay đổi tham số (lần gọi đệ quy). Ta gọi trường hợp này là trường hợp **suy biến**.

- Trong trường hợp **tổng quát**, bài toán có thể quy về bài toán cùng dạng nhưng giá trị của tham số thay đổi. Sau một số hữu hạn bước biến đổi đệ quy, sẽ dẫn tới trường hợp suy biến.

Trường hợp suy biến rất quan trọng trong bài toán đệ quy. Nếu không có trường hợp này, quá trình tạo tiến trình mới sẽ không thể dừng lại và ta gặp phải trường hợp đệ quy vô hạn. Nếu trường hợp tổng quát mà sau một số hữu hạn lần gọi đệ quy không thể quy về trường hợp suy biến thì cũng không thể thoát khỏi quá trình gọi đệ quy vô hạn này.

Giả sử bài toán tính $n!$. Với n nguyên dương, để dàng thấy:

- Với $n = 1$ thì $n! = 1$. Khi đó ta không cần gọi đệ quy vẫn có thể tính được $n!$. Quan trọng hơn, trong tất cả các trường hợp còn lại ($n > 1$), ta có thể quy về việc giải bài toán trong trường hợp $n=1$ bằng công thức $n! = n*(n-1)!$. Do vậy, đây là trường hợp suy biến.

- Trường hợp tổng quát là $n > 1$. Khi đó, $n! = n*(n-1)!$. Tức là để tính $n!$, ta có thể quy về bài toán tính $(n-1)!$. Sau một số hữu hạn bước biến đổi, ta có thể quy về bài toán tính $1!$.

☞ Tại sao ta không chọn một giá trị khác 1 cho n , chẳng hạn $n=3$ (khi đó $n!=6$) làm trường hợp suy biến? Để dàng nhận thấy, nếu như vậy, các trường hợp $n < 3$ sẽ không thể quy về suy biến theo công thức $n!=n(n-1)!$.*

☞ Tại sao công thức tổng quát lại biểu diễn $n!$ qua $(n-1)!$ mà không phải biểu diễn qua $(n-2)!$ hoặc $(n-3)!$...? Điều này cũng là để đảm bảo các trường hợp tổng quát luôn quy về được suy biến.

Phương pháp thiết kế hàm đệ quy có thể theo 3 bước như sau:

Bước 1: Xác định các trường hợp

- Trường hợp suy biến, giá trị của tham số, công thức để tính toán trong trường hợp này.
- Trường hợp tổng quát, giá trị của tham số, công thức để tính toán trong trường hợp này.

Bước 2: Viết nội dung hàm đệ quy

if (suy biến)

```

        return <công thức suy biến>;
else
        return <công thức tổng quát>;

```

Bước 3: Hoàn thiện hàm đệ quy.

② **Ví dụ 3.1:** Thiết kế hàm đệ quy tính $n!$.

Ta thực hiện thiết kế hàm đệ quy theo ba bước ở trên.

Bước 1: Xác định các trường hợp

- Suy biến: $n = 1$. Công thức suy biến: $n! = 1$.
- Tổng quát: $n > 1$. Công thức tổng quát: $n! = n * (n-1)!$.

Bước 2: Viết nội dung hàm đệ quy

```

if (n== 1)
    return 1;
else
    return n * GT(n-1);

```

Bước 3: Hoàn thiện để thu được hàm đệ quy tính $n!$.

1	long GT(int n)
2	{
3	if (n== 1)
4	return 1;
5	else
6	return n * GT(n-1);
7	}

② **Ví dụ 3.2:** Dãy số Catalan được phát biểu đệ quy như sau:

$$C_1 = 1;$$

$$C_n = \sum_{i=1}^{n-1} C_i C_{n-i} \quad \forall n > 1.$$

Hãy xây dựng hàm đệ quy tìm số Catalan thứ n .

Hàm đệ quy được thiết kế như sau:

Bước 1: Xác định các trường hợp

- Suy biến: $n = 1$. Công thức suy biến: $C_n = 1$;
- Tổng quát: $n > 1$. Công thức tổng quát:

$$C_n = \sum_{i=1}^{n-1} C_i C_{n-i}$$

Bước 2: Viết nội dung hàm đệ quy

```

if (n == 1)
    return 1;
else
{
    int C = 0;
    for (int i=1; i<n; i++)
        C+= Catalan(i) * Catalan(n-i);
    return C;
}

```

Bước 3: Hoàn thiện hàm đệ quy.

1	int Catalan (int n)
2	{
3	if (n==1)
4	return 1;
5	else
6	{
7	int C=0;
8	for (int i=1; i<n; i++)
9	C+= Catalan (i)* Catalan (n-i);
10	return C;
11	}
12	}

☞ Hãy thực thi chương trình với $n = 20$ và $n = 25$ để thấy thời gian thực thi của chương trình. Hãy tìm cách khử đệ quy để chương trình có thể thực thi nhanh hơn và có thể tính được với $n=100$.

B. Các bài tập tự giải

Bài H1. Giả sử ta cần tính giá trị của biểu thức sau ($n, m \in \mathbb{Z}^+$):

$$S = \frac{n!+m!}{(n+m)!}.$$

Hãy tự tổ chức chương trình thành các hàm cần thiết để viết chương trình nhập vào n, m là hai số nguyên dương từ bàn phím. Tính và in ra màn hình giá trị của S .

Bài H2. Người ta định nghĩa một biểu thức như sau $x \in \mathbb{R}, n \in \mathbb{Z}$:

$$F(x, n) = 2021n + 2x^2 + 3x + 5.$$

Hãy tự tổ chức chương trình thành các hàm một cách khoa học để viết chương trình nhập vào giá trị của $x, y \in \mathbb{R}$ và $n \in \mathbb{Z}$ từ bàn phím. Tính và in ra màn hình kết quả của biểu thức: $P = F(x, n) + F(y, n) - F(x+y, n)$.

Bài H3. Để linh hoạt trong việc giải nhiều phương trình bậc hai ($ax^2 + bx + c = 0$) cùng một lúc mà không phải viết lặp lại các đoạn code nhiều lần, hãy tự tổ chức chương trình thành các hàm phù hợp để viết chương trình nhập vào từ bàn phím các hệ số của phương trình bậc hai bất kỳ và giải phương trình.

Bài H4. Viết hàm giải hệ phương trình bậc nhất hai ẩn dạng:

$$\begin{cases} ax + by + c = 0 \\ dx + ey + f = 0 \end{cases}.$$

Trong đó a, b, c, d, e, f là các hệ số thực. Người ta cần lưu trữ các nghiệm của hệ phương trình (nếu có) để sử dụng về sau. Do vậy, yêu cầu hàm phải phân biệt các trường hợp hệ phương trình vô nghiệm, vô số nghiệm, có nghiệm và trả về nghiệm x, y (nếu có). Viết chương trình chính minh họa việc sử dụng hàm trên.

Bài H5. Cho ba số thực a, b, c bất kỳ. Hãy tự xác định các hàm, đầu vào, đầu ra của các hàm một cách hợp lý để xây dựng chương trình đáp ứng yêu cầu sau:

- 1). Nhập vào ba số thực a, b, c .
- 2). Tìm và in ra giá trị nhỏ nhất, giá trị lớn nhất trong ba số.
- 3). Nhập thêm hai số thực d, e in ra số nhỏ nhất (tương tự là số lớn nhất) trong 5 số a, b, c, d, e .

Bài H6. Hàm *swap* hoán vị giá trị hai biến thực a, b đã có sẵn trong C++. Tuy nhiên, hãy tự viết hàm này để sử dụng. Viết chương trình chính nhập hai số thực a, b và sử dụng hàm vừa viết để hoán vị giá trị của a và b .

Bài H7. Viết hàm tính:

$$F1 = 1 + 2 + 3 + 4 + \dots + n;$$

$$F2 = 1 + 2^2 + 3^3 + \dots + n^n;$$

$$F3 = 1/3 + 1/5 + \dots + 1/(2n+1).$$

Viết chương trình chính nhập vào một biến nguyên n . Tính và in ra: $T = F1 + F2 + F3$.

Bài H8. Cho 3 điểm $A(x_1, y_1)$, $B(x_2, y_2)$, $C(x_3, y_3)$ trong mặt phẳng tọa độ XOY. Người ta muốn xác định xem điểm nào là điểm gần tâm O nhất, điểm nào là điểm xa tâm O nhất, tính theo khoảng cách Euclidiens. Hãy tự tổ chức thành các hàm để viết chương trình thực hiện yêu cầu trên.

Bài H9. Viết chương trình gồm các hàm thực hiện các yêu cầu sau:

- Hàm nhập vào một số nguyên dương từ bàn phím, nếu số nhập vào không thỏa mãn là số dương, yêu cầu nhập lại cho tới khi thỏa mãn điều kiện.
- Hàm tính $n!$ với n nguyên dương bất kỳ.
- Hàm tính tổ hợp chập k của n theo công thức:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

- Hàm main sử dụng các hàm trên để nhập vào 2 số nguyên dương n và k , tính và in ra màn hình tổ hợp chập k của n .

Bài H10. Tự xác định các hàm để: viết chương trình nhập vào hai số thực x, y và một số nguyên dương n . Tính và in ra:

- Diện tích hình chữ nhật có cạnh là x, y .
- Tổng các số chẵn và chia hết 3 trong đoạn $[1, n]$.
- Hàm $F1(x, n) = 2016x^n + \frac{x^2}{3} + \frac{x^3}{3^2} + \dots + \frac{x^n}{3^n}$.
- Hàm $F2(x, n) = \begin{cases} 2016x & \text{neu } n > 10 \\ e^x + 1 + 2 + 3 + \dots + n & \text{neu } n \leq 10 \end{cases}$.
- Viết hàm main nhập vào x, y, n . Sử dụng 4 hàm ở trên để tính và in ra các kết quả tương ứng.

Bài H11. Viết hàm để quy tính giá trị của biểu thức: $F(x, n) = x^n / n!$, với $x \in \mathbb{R}$ và $n \in \mathbb{Z}^+$. Sử dụng hàm này trong chương trình chính để tính và in ra màn hình giá trị của biểu thức (với x, n nhập từ bàn phím):

$$P = \frac{F(x,n)+F(2x,n)}{F(x,n)}.$$

Hãy sửa hàm để quy để tính được biểu thức: $F(x, n) = 2021x^n / n!$.

Bài H12. Viết hàm để quy tính số chữ số trong 1 số nguyên dương bất kỳ (ví dụ số 1423 có 4 chữ số). Viết chương trình chính nhập vào một số nguyên dương n bất kỳ, sử dụng hàm trên để cho biết n có bao nhiêu chữ số. Hãy thử nghiệm với trường hợp n không dương.

Bài H13. Viết hàm đệ quy tính giá trị của tổng chuỗi sau:

$$Q(x, n) = 2021x + \frac{x}{3} + \frac{x^2}{5} + \cdots + \frac{x^n}{2n+1},$$

với x thực và n nguyên dương. Viết hàm *main* để minh họa việc sử dụng hàm đệ quy ở trên.

Bài H14. Viết hàm đệ quy trả về tổng các chữ số trong một số nguyên dương n . Ví dụ: $n = 25$, tổng của các chữ số của n bằng $2 + 5 = 7$.

BÀI 4

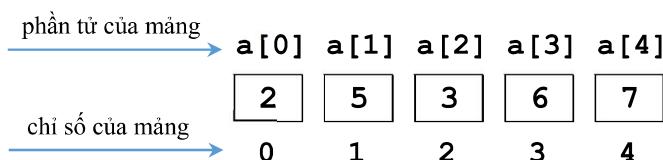
KỸ THUẬT XỬ LÝ MẢNG MỘT CHIỀU

Bài này được dành để trình bày các kiến thức cơ bản và bài tập về kỹ thuật xử lý mảng một chiều: Khái niệm, các thao tác cơ bản trên mảng một chiều, các dạng bài tập cơ bản như tìm kiếm, sắp xếp, thống kê, tách, chèn, xóa... trên mảng một chiều.

A. Tóm tắt lý thuyết

4.1. Khái niệm và cách khai báo

Mảng là một cấu trúc bộ nhớ bao gồm một dãy liên tiếp các ô nhớ cùng tên, cùng kiểu nhưng khác nhau về chỉ số, dùng để lưu trữ một dãy các phần tử cùng kiểu.



Hình 4.1. Ví dụ về một mảng `a` có 5 phần tử.

Cú pháp khai báo:

`<Kiểu mảng> <Tên mảng> [<p>];`

Trong đó:

`<Kiểu mảng>`: Là kiểu dữ liệu của mỗi phần tử trong mảng, có thể là một kiểu dữ liệu nguyên thủy hoặc kiểu dữ liệu tự định nghĩa.

`<Tên mảng>`: Được đặt tuỳ ý tuân theo quy ước đặt tên trong C++.

`<p>`: Là một số nguyên chỉ ra số ô nhớ tối đa được dành cho mảng cũng như số phần tử tối đa mà mảng có thể chứa được.

Ví dụ, với khai báo `int a[3];` sẽ cấp phát 3 ô nhớ liên tiếp cùng kiểu nguyên dành cho mảng `a`. Mảng này có thể chứa được tối đa 3 số nguyên.

Để khởi tạo mảng, ta viết các phần tử của mảng cách nhau bởi dấu phẩy vào giữa cặp dấu ngoặc nhọn {} với cú pháp khởi tạo mảng như sau:

$\langle \text{Kiểu mảng} \rangle \langle \text{Tên mảng} \rangle [\langle p \rangle] = \{\text{giá trị } 1, \text{ giá trị } 2, \dots\};$

Ví dụ: `int a[3]={11,12,13};` sẽ cấp phát 3 ô nhớ liên tiếp cùng kiểu nguyên dành cho mảng *a* và khởi tạo giá trị cho các ô nhớ tương ứng là 11,12,13.

4.2. Các thao tác cơ bản trên mảng một chiều

[1]. **Nhập mảng:** Giả sử ta cần nhập mảng *a* gồm *n* phần tử từ bàn phím. Cách duy nhất là nhập từng phần tử cho mảng. Do vậy, ta có thể sử dụng một vòng lặp **for** duyệt qua từng phần tử và nhập dữ liệu cho chúng. Nhưng trước tiên, cần nhập số phần tử thực tế của mảng (*n*).

```
1 cout<<"n="; cin>>n;
2 for(int i=0; i<n; i++)
3 {
4     cout<< "a["<<i<< "]=";
5     cin>>a[i];
6 }
```

[2]. **Xuất mảng:** tương tự như nhập mảng, ta cũng cần sử dụng vòng lặp **for** để xuất từng phần tử của mảng lên màn hình. Việc xuất dữ liệu có thể thực hiện như sau (có thể sử dụng 1 trong ba dòng 2, 3, 4):

```
1 for(i = 0; i<n; i++)
2     cout<<a[i];
3 //cout<<a[i]<< " ";
4 //cout<<a[i]<<endl;
```

[3]. **Duyệt mảng:** Là thao tác “thăm” lần lượt từng phần tử của mảng. Thao tác duyệt mảng có trong hầu hết các bài toán có sử dụng mảng.

```
1 for(i = 0; i<n; i++)
2 {
3     //thăm phần tử a[i]
4 }
```

❷ **Ví dụ 4.1:** Nhập vào một mảng *a* gồm *n* phần tử thực. Xuất mảng *a* vừa nhập ra màn hình. Tính và in ra màn hình giá trị của phần tử lớn nhất trong mảng *a*.

- Hàm nhập mảng từ bàn phím như sau (chú ý đối số n được viết dưới dạng tham chiếu):

```
1 void nhapmang(float a[], int & n)
2 {
3     cout<<"n="; cin>>n;
4     for(int i=0; i<n; i++)
5     {
6         cout<<"a["<<i<<"]=";
7         cin>>a[i];
8     }
9 }
```

- Hàm xuất mảng ra màn hình như sau:

```
1 void xuatmang(float a[], int n)
2 {
3     for(int i=0; i<n; i++)
4         cout<<setw(5)<<a[i];
5 }
```

- Hàm trả về giá trị lớn nhất của mảng sử dụng kỹ thuật duyệt mảng như sau:

```
1 float timmax(float a[], int n)
2 {
3     float Max = a[0];
4     for(int i=1; i<n; i++)
5         if(a[i]>Max) Max = a[i];
6     return Max;
7 }
```

- Hàm main để vận hành chương trình:

```
1 int main()
2 {
3     float a[100]; int n;
4     nhapmang(a, n);
5     xuatmang(a, n);
6     cout<<endl<<"MAX = "<<timmax(a, n);
7     return 0;
8 }
```

4.3. Một số bài toán cơ bản

- Bài toán sắp xếp mảng

Bài toán: Cho một dãy gồm n phần tử. Hãy sắp xếp dãy theo chiều tăng dần (hoặc giảm dần).

Có nhiều phương pháp để sắp một mảng theo chiều tăng dần (hoặc giảm dần). Sau đây ta xem xét một số cách phổ biến:

Sắp xếp nổi bọt (Bubble sort)

- Sắp lần lượt từng phần tử của dãy, bắt đầu từ phần tử đầu tiên.

- Giả sử cần sắp phần tử thứ i , ta tiến hành duyệt lần lượt qua tất cả các phần tử đứng sau nó trong dãy, nếu gặp phần tử nào nhỏ hơn phần tử đang sắp thì đổi chỗ.

```
1 | for(int i = 0; i < n; i++)
2 |   for(int j = i+1; j<n; j++)
3 |     if(a[j] < a[i])
4 |     {
5 |       tg = a[i];
6 |       a[i] = a[j];
7 |       a[j] = tg;
8 |     }
```

Sắp xếp bằng phương pháp này trung bình cần $n^2/2$ lần so sánh và $n^2/2$ lần hoán vị. Trong trường hợp tồi nhất độ phức tạp tính toán là $O(n^2)$.

Sắp xếp chọn (Selection sort)

Trong phương pháp sắp xếp nổi bọt ở trên, để sắp một phần tử nhiều khi ta phải đổi chỗ nhiều lần phần tử đang sắp với các phần tử đứng sau nó. Vì vậy, người ta đề xuất một ý tưởng: Làm thế nào để chỉ cần đổi chỗ 1 lần duy nhất khi sắp một phần tử trong dãy. Đây chính là ý tưởng của phương pháp sắp xếp chọn.

Để làm được điều này, khi sắp phần tử thứ i , người ta tiến hành tìm phần tử nhỏ nhất trong số các phần tử đứng sau nó, tính cả phần tử đang sắp rồi tiến hành đổi chỗ phần tử nhỏ nhất tìm được với phần tử đang sắp.

Ví dụ với dãy $a = \{1, 6, 4, 2, 5, 7\}$, để sắp phần tử thứ 2 (có giá trị là 6) ta tiến hành tìm phần tử nhỏ nhất trong số các phần tử $\{6, 4, 2, 5, 7\}$. Khi đó Min $(6, 4, 2, 5, 7) = 2$ và phần tử có giá trị 2 được đảo chỗ cho phần tử đang sắp. Kết quả thu được:

1	2	4	6	5	7
---	---	---	---	---	---

Tuy nhiên, khi tìm giá trị nhỏ nhất trong một tập các phần tử ta cần phải lưu ý một số điểm. Chẳng hạn ta cần biết chính xác vị trí của phần tử nhỏ nhất nằm ở đâu để tiến hành đổi chỗ chứ không quan tâm tới giá trị của nó là bao nhiêu. Phương pháp được mô tả như sau:

```

1 | for(int i = 0; i < n; i++)
2 | {
3 |     Min = i;
4 |     for(int j = i+1; j<n; j++)
5 |         if(a[j] < a[Min]) Min = j;
6 |         tg = a[i];
7 |         a[i] = a[Min];
8 |         a[Min] = tg;
9 |

```

Sắp xếp bằng phương pháp chọn cần $n^2/2$ lần so sánh và n lần hoán vị. Sau đây là hàm sắp xếp chọn với đối vào là mảng a gồm n phần tử:

```

1 | void SapChon(int a[], int n)
2 | {
3 |     for (int i=0; i<n; i++)
4 |     {
5 |         int min = i;
6 |         for (int j = i+1; j<n; j++)
7 |             if (a[j] < a[min]) min = j;
8 |             int tg = a[i];
9 |             a[i] = a[min];
10 |            a[min]=tg;
11 |

```

Sắp xếp chèn (Insertion sort)

Một thuật toán gần như đơn giản ngang với thuật toán sắp xếp chọn nhưng có lẽ mềm dẻo hơn, đó là sắp xếp chèn. Đây là phương pháp người ta dùng để sắp xếp các thanh ngang cho một chiếc thang.

Đầu tiên, người ta rút ngẫu nhiên 1 thanh ngang và đặt vào 2 thanh dọc để làm thang. Tiếp đó, người ta lần lượt chèn từng thanh ngang vào sao cho không phá vỡ tính được sắp của các thanh đã được đặt trên 2 thanh dọc.

Giả sử với mảng $a = \{3, 5, 1, 7, 4, 8\}$ và các phần tử có giá trị là 3 và 5 đã được chèn vào đúng vị trí (đã được sắp):

3	5	1	7	4	8
---	---	---	---	---	---

Ta xem xét quá trình chen phần tử tiếp theo (tức phần tử $a[2]$, có giá trị 1) vào phần đã được sắp của mảng. Khi đó, quá trình diễn ra như sau:

- Đặt phần tử $a[2]$ vào biến $tg = a[2];$

• Duyệt qua các phần tử đứng trước phần tử đang sắp (tức duyệt qua các phần tử đã được sắp). Nếu gặp phần tử nào lớn hơn tg thì đẩy phần tử đó sang phải 1 vị trí. Ngược lại, nếu gặp phần tử nhỏ hơn phần tử tg thì chèn tg vào ngay sau phần tử nhỏ hơn này. Nếu đã duyệt hết các phần tử đứng trước mà vẫn chưa tìm thấy phần tử nhỏ hơn tg thì chèn tg vào đầu mảng. Kết thúc quá trình này, phần tử $a[2]$ đã được chèn đúng vị trí và 3 phần tử đã được sắp là:

1	3	5
---	---	---

Như vậy trong quá trình thực hiện, để chèn một phần tử vào đúng vị trí của nó, ta luôn thực hiện 2 công việc: *Đẩy một phần tử sang phải 1 vị trí* hoặc *chèn phần tử cần chèn vào vị trí của nó*. Nếu gọi phần tử cần chèn là $a[i]$ đang chứa trong biến tg và j là biến duyệt qua các phần tử đứng trước $a[i]$ thì:

- Khi chưa hết mảng và gặp một phần tử lớn hơn phần tử cần chèn thì đẩy nó sang phải 1 vị trí:

```

while (j >= 0 && a[j] > tg)
{
    a[j+1] = a[j];
    j--;
}

```

- Ngược lại thì chèn tg vào sau j : $a[j+1] = tg;$

```

1 void SapChen(int a[], int n)
2 {
3     for (int i=1; i< n; i++)
4     {
5         int Tg = a[i];
6         int j = i-1;
7         while (j >= 0 && a[j] > Tg)
8         {
9             a[j+1] = a[j];
}

```

```

10           j--;
11       }
12   a [ j+1 ] = Tg;
13 }
14 }

```

Sắp xếp bằng phương pháp chèn trong trường hợp trung bình cần $n^2/4$ lần so sánh và $n^2/8$ lần hoán vị. Trường hợp tồi nhất gấp đôi số này.

- **Bài toán tìm kiếm**

Bài toán: Cho một mảng a gồm n phần tử và một phần tử c cùng kiểu. Hỏi c có xuất hiện trong a không?

Tìm kiếm tuần tự (Sequential search)

Một phương pháp đơn giản để giải quyết bài toán trên là duyệt mảng. Giải thuật được trình bày như sau:

- Sử dụng một biến đếm $d = 0$;
- Duyệt qua các phần tử của mảng, nếu gặp c ta tăng biến đếm: $d++$;

Kết thúc quá trình duyệt mảng, nếu d bằng 0 chứng tỏ c không xuất hiện trong mảng và ngược lại.

Trường hợp tồi nhất, phương pháp trên cần phải duyệt qua tất cả các phần tử của mảng một cách tuần tự, do vậy, độ phức tạp của giải thuật là $O(n)$ với n là kích thước của mảng.

Tìm kiếm nhị phân (Binary search):

Nếu mảng a đã được sắp (tăng hoặc giảm) thì do tính chất đặc biệt này, ta có thể giải quyết bài toán mà không cần duyệt qua tất cả các phần tử của mảng bằng tìm kiếm nhị phân.

Giả sử ta cần tìm kiếm c trong một đoạn từ vị trí L tới vị trí R trong mảng a (trường hợp tìm kiếm trên toàn bộ mảng thì $L=0$ và $R=n-1$). Ta làm như sau:

- Gọi M là vị trí giữa của đoạn mảng ta đang tìm kiếm ($M = (L+R)/2$). Trước tiên ta tiến hành kiểm tra $a[M]$. Khi đó, chỉ có thể xảy ra một trong 3 trường hợp sau:

$a[M] = c$: Kết luận c có trong mảng a và ta có thể dùng quá trình tìm kiếm.

$a[M] > c$: Vì mảng được sắp (giả sử sắp tăng) nên rõ ràng c (nếu có) chỉ nằm trong đoạn bên phải tức $[M+1, R]$. Ta tiến hành lặp lại quá trình tìm kiếm trên đoạn $[M+1, R]$, tức điều chỉnh cận trái $L=M+1$.

$a[M] < c$: Khi đó c (nếu có) chỉ nằm trong đoạn bên trái của mảng tức $[L, M-1]$. Ta tiến hành lặp lại quá trình tìm kiếm trên đoạn $[L, M-1]$, tức điều chỉnh cận phải $R=M-1$.

Kết thúc quá trình tìm kiếm là khi xảy ra một trong hai trường hợp:

[1]. Nếu $L > R$: c không xuất hiện trong a .

[2]. Nếu $a[M] = c$: c xuất hiện trong a tại vị trí M .

Vậy quá trình “chia đôi-tìm kiếm” sẽ được lặp lại nếu: $(a[M] != c \&\& L <= R)$. Hàm sau thực hiện việc tìm kiếm nhị phân trên một mảng a có kích thước n với một khoá c cần tìm, sử dụng vòng lặp:

```

1 void TKNP_Lap(int a[], int n, int c)
2 {
3     int L=0, R=n-1, M;
4     do
5     {
6         M = (L+R)/2;
7         if (a[M]>c) R=M-1;
8         if (a[M]<c) L=M+1;
9     }
10    while (a[M] != c && L < R);
11    if (a[M]==c)
12        cout<<c<<" xuat hien tai "<<M;
13    else
14        cout<<c<<" khong xuat hien";
15 }
```

Việc chia đôi và tìm kiếm trên một nửa của mảng được lặp đi lặp lại cho tới khi xảy ra 1 trong 2 trường hợp mà ta có thể kết luận: *Tìm thấy* và *không tìm thấy* gợi ý cho ta một cách đặt đệ quy cho hàm này:

- **Trường hợp suy biến:** Là trường hợp $a[M] = c$ hoặc $L > R$. Khi đó, nếu $a[M]=c$ hàm trả về giá trị M là vị trí xuất hiện của c trong mảng; nếu $L > R$ thì c không xuất hiện trong mảng và hàm trả về giá trị -1.

- **Trường hợp tổng quát:** Là trường hợp $a[M] > c$ hoặc $a[M] < c$. Khi đó việc gọi đệ quy là cần thiết với các cận L hoặc R được thay đổi cho phù hợp.

```

1 int TKNP_DQ(int a[], int c, int L, int R)
2 {
3     int M=(L+R)/2;
4     if(a[M]==c) //trường hợp suy biến thứ nhất
5         return M;
6     else if(L>R) //trường hợp suy biến thứ 2
7         return -1;
8     else //trường hợp tổng quát
9         if(a[M]>c)
10            return TKNP_DQ(a,c,L,M-1);
11        else
12            return TKNP_DQ(a,c,M+1,R);
13 }

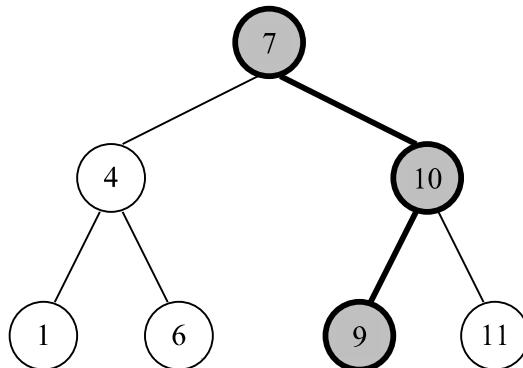
```

Giải thuật trên giống như việc ta thăm phần tử chính giữa của đoạn mảng đang tìm kiếm, nếu không gặp c ta có thể “rẽ” sang bên trái hoặc bên phải để tìm tiếp tuỳ thuộc vào giá trị của phần tử chính giữa này. Việc này giống như việc tìm kiếm trên cây nhị phân (cây mà mỗi nút có 2 con sao cho các giá trị thuộc nhánh trái luôn nhỏ hơn hoặc bằng giá trị của nút và các giá trị của nhánh bên phải luôn lớn hơn hoặc bằng giá trị của nút).

Giả sử ta có mảng a như sau:

a	1	4	6	7	9	10	11
----------	---	---	---	---	---	----	----

Các giá trị của a có thể biểu diễn trên cây nhị phân tìm kiếm như sau:



Hình 4.2. Ví dụ về cây nhị phân tìm kiếm xây dựng từ mảng.

Xuất phát từ gốc cây, nếu ta cần tìm một giá trị c bất kỳ, trường hợp tồi nhất ta chỉ cần đi hết chiều cao của cây. Giả sử c = 9, đường đi tương ứng trong trường hợp này được tô đậm trong Hình 4.2. Nếu mảng có n phần tử, ta luôn có thể sử dụng một cây có chiều cao không quá $\log_2(n)$ để biểu diễn các giá trị của mảng. Do vậy, độ phức tạp trong trường hợp tồi nhất của giải thuật này là $O(\log_2(n))$.

- **Kiểm tra tính chất của mảng:**

Các bài toán kiểm tra xem mảng có một tính chất nào đó hay không có thể quy về bài toán tìm kiếm, tức chúng ta có thể sử dụng phương pháp “duyệt và đếm” một vài đại lượng nào đó. Số liệu thống kê sẽ được dùng làm căn cứ để xác định: Có hay không một tính chất nào đó của mảng. Tuy nhiên, với một bài toán cụ thể, ta cần duyệt và đếm đại lượng nào lại thường không được biết trước và ta cần tự xác định.

Ví dụ 4.2: Trong một hàng cây, người ta đo chiều cao của từng cây và lưu trữ trong một mảng a . Một hàng cây được gọi là “khá đồng đều” nếu có ít nhất 3 cây liên tiếp có độ cao bằng nhau. Hãy xác định xem một hàng cây có thể gọi là khá đồng đều hay không.

Dễ thấy bài toán thuộc dạng kiểm tra tính chất của mảng. Câu trả lời sẽ có khi ta duyệt mảng và đếm “những chỗ có ba phần tử liên tiếp bằng nhau”. Hàm sau trả về giá trị `true` nếu mảng a có tồn tại ít ba phần tử liên tiếp bằng nhau, ngược lại hàm trả về `false`.

```

1 | bool Check(int a[], int n)
2 | {
3 |     for (int i=0; i<n-2; i++)
4 |         if(a[i]==a[i+1] && a[i+1]==a[i+2])
5 |             return true;
6 |         return false;
7 | }

```

Trong hàm trên, cần chú ý là vòng lặp `for` không duyệt tới hết mảng mà chỉ thực hiện $i < n-2$. Bởi vì trong thân vòng lặp, ta truy cập tới $a[i+1]$ và $a[i+2]$ nên nếu duyệt tới hết mảng (tức $i=n-1$) thì $i+1$ và $i+2$ sẽ chạy ra khỏi giới hạn của mảng.

Ví dụ 4.3: Một dãy số a gọi là được sắp “tăng” nếu $a[i] \leq a[i+1]$; dãy gọi là được sắp “giảm” nếu $a[i] \geq a[i+1]$; dãy gọi là được sắp “tăng ngắt” nếu $a[i] < a[i+1]$ và dãy gọi là được sắp “giảm ngắt” nếu $a[i] > a[i+1]$,

$\forall i \in [0, n-1]$. Viết chương trình nhập một dãy n số thực, kiểm tra xem dãy đã được sắp hay chưa. Nếu đã được sắp thì sắp theo trật tự nào (tăng, tăng ngắt, giảm, giảm ngắt).

Xét hai phần tử liên tiếp bất kỳ của mảng: $a[i]$ và $a[i+1]$ $\forall i \in [0, n-1]$. Khi đó: Chỉ có thể xảy ra một trong ba trường hợp sau:

[1]. Phần tử đứng trước nhỏ hơn phần tử đứng sau ($a[i] < a[i+1]$): Trường hợp này ta gấp một bước “tăng”.

[2]. Phần tử đứng trước lớn hơn phần tử đứng sau ($a[i] > a[i+1]$): Trường hợp này ta gấp một bước “giảm”.

[3]. Hai phần tử bằng nhau ($a[i] = a[i+1]$): Trường hợp này ta gấp một bước “bằng”.

Gọi T là số bước tăng, G là số bước giảm, B là số bước bằng. Ta dễ dàng đếm được T , G , B trong mảng chỉ bằng một lần duyệt mảng. Căn cứ vào T , G , B tính được, ta có thể kết luận về tính được sắp của mảng:

- Nếu $T = n-1$: Toàn bộ các bước trong mảng đều là bước tăng. Mảng được sắp “tăng ngắt”.

- Nếu $G = n-1$: Mảng được sắp “giảm ngắt”.

- Nếu $G=0$ và $B>0$: Mảng được sắp tăng.

- Nếu $T=0$ và $B>0$: Mảng được sắp giảm.

- Các trường hợp còn lại: Mảng chưa được sắp.

Tuy nhiên, do định nghĩa về các trường hợp của mảng ở trên là giao nhau, nên trong trường hợp tất cả các phần tử của mảng bằng nhau ($B=n-1$) ta có thể kết luận mảng được sắp “tăng” hay “giảm” đều được. Do vậy ta sẽ xử lý riêng trường hợp này.

```

1 void CheckSort(float a[], int n)
2 {
3     int T, G, B ;
4     T = G = B = 0;
5     for(int i=0 ; i < n-1 ; i++)
6     {
7         if (a[i] < a[i+1])          T++;
8         if (a[i] > a[i+1])          G++;
9         if (a[i] == a[i+1])          B++;

```

```

10 }
11 if(B==n-1)
12     cout<< "Mang gom cac phan tu bang nhau";
13 else if(T == n-1)
14     cout<< "Mang duong sap tang ngat";
15 else if(G == n-1)
16     cout<< "Mang duoc sap giam ngat";
17 else if(T == 0 && B != 0)
18     cout<< "Mang duoc sap giam";
19 else if(G == 0 && B != 0)
20     cout<< "Mang duoc sap tang";
21 else
22     cout<< "Mang chua duoc sap";
23 }

```

B. Các bài tập tự giải

Bài V1. Viết chương trình nhập vào từ bàn phím một mảng a gồm n số thực. Tìm giá trị phần tử lớn nhất, nhỏ nhất của a và in ra màn hình.

Bài V2. Viết chương trình nhập vào từ bàn phím một mảng a gồm n số nguyên. Tính và in ra màn hình trung bình cộng của các phần tử trong a , tổng các số chẵn, tổng các số lẻ trong a .

Bài V3. Viết chương trình nhập vào từ bàn phím một mảng a gồm n phần tử nguyên. Sắp xếp mảng a tăng dần và in mảng đã sắp ra màn hình.

- Cho biết mảng a có bao nhiêu phần tử thuộc $[10, 20]$.
- Cho biết mảng a có tồn tại ba số chẵn liên tiếp hay không.

Bài V4. Cho hai véc tơ $x(x_1, x_2, \dots, x_n)$ và $y(y_1, y_2, \dots, y_n)$.
Viết chương trình tính và in ra:

- Tích vô hướng của hai vector trên:

$$P = \langle x, y \rangle = \sum_{i=1}^n x_i y_i.$$

- Chuẩn 1 của x (và của y):

$$\|x\|_1 = \sum_{i=1}^n |x_i|.$$

- Chuẩn 2 của x (và của y):

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \dots + x_n^2}.$$

- Chuẩn 0 của x (và của y):

$$\|x\|_0 = Card(x).$$

$\text{Card}(x)$ được tính bằng số phần tử khác 0 của x .

- Chuẩn vô cùng của x (và của y):

$$\|x\|_{\infty} = \text{Max}_i(x_i).$$

Bài V5. Viết chương trình nhập vào một mảng a gồm n số nguyên. Hãy cho biết trong mảng a có chứa những số nguyên nào và số lần xuất hiện của nó trong a .

Bài V6. Nhập một mảng a gồm n phần tử nguyên. Mảng a được gọi là hợp lệ nếu nó chứa đúng 3 phần tử dương và 3 phần tử âm. Hãy cho biết a có hợp lệ không. Nếu a không hợp lệ hãy sắp a theo chiều tăng dần bằng phương pháp chèn.

Bài V7. Một mảng a gồm n phần tử nguyên được gọi là hợp lệ nếu tất cả các phần tử có chỉ số lẻ đều nguyên tố. Hãy kiểm tra tính hợp lệ của mảng.

Bài V8. Nhập vào một mảng a chỉ gồm các phần tử 0 hoặc 1. Một đường đi trên a là một dãy liên tiếp các phần tử 1. Độ dài của đường đi là số phần tử trên đường đi đó. Ví dụ: Với mảng $a = \{3, 1, 1, 1, 1, 5, 1, 1\}$ có hai đường đi $\{1, 1, 1, 1\}$ và $\{1, 1\}$. Hãy cho biết:

- Mảng vừa nhập có bao nhiêu đường đi.
- Mảng vừa nhập có đường đi dài nhất xuất phát từ vị trí nào.
- Độ dài của đường đi dài nhất trong mảng.

Bài V9. Nhập một mảng a gồm n phần tử thực. Hãy sắp xếp mảng a sao cho: Các phần tử lớn nhất ở đầu mảng, các phần tử bé nhất ở cuối mảng, các phần tử còn lại sắp tăng dần. In mảng đã sắp ra màn hình.

Bài V10. Nhập một mảng a gồm n phần tử nguyên. Hãy sắp xếp mảng a sao cho các phần tử lớn nhất về cuối mảng, các phần tử còn lại được sắp giảm dần. In kết quả ra màn hình. Cho biết mảng vừa sắp có bao nhiêu phần tử nhỏ nhất.

Bài V11. Cho hai mảng a gồm n phần tử và b gồm m phần tử. Các phần tử của cả a và b đều đã được sắp tăng dần. Hãy trộn hai mảng trên để thu được một mảng c cũng sắp theo thứ tự tăng dần. Hãy đưa ra các phương án khác nhau để giải quyết bài toán này và đánh giá độ phức tạp thuật toán của từng phương án.

Bài V12. Công nhân của một công ty cây xanh thực hiện trồng một hàng cây gồm n cây liên tiếp nhau. Sau khi trồng xong, ban quản lý công ty tiến

hành kiểm tra xem hàng cây có đạt yêu cầu hay không. Nếu hàng cây có ít nhất 50% số cây cao trên 1.2 mét và không có chỗ nào có hai cây liên tiếp đều thấp hơn 0.3 mét là đạt yêu cầu.

Gọi a là mảng chứa chiều cao của các cây trong hàng cây trên theo đúng thứ tự, đơn vị tính là mét. Hãy cho biết hàng cây tương ứng với mảng a có đạt yêu cầu hay không.

Bài V13. Cho n bình đựng nước với dung tích của bình thứ i là $a[i]$ lít. Người ta cần lấy ra các bình để chứa một lượng nước là k lít. Hãy cho biết cần lấy ra ít nhất bao nhiêu bình để chứa hết k lít nước.

Ví dụ với 5 bình có dung tích lần lượt là 3, 6, 2.1, 8, 5 lít. Giả sử $k=16$. Khi đó cần lấy ra ít nhất là 3 bình để chứa hết 16 lít nước.

BÀI 5

KỸ THUẬT XỬ LÝ MẢNG HAI CHIỀU

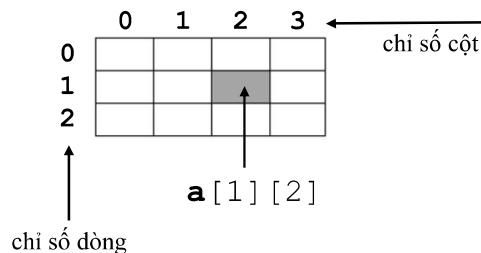
Bài này được dành cho việc tổng hợp lại các kiến thức cơ bản về mảng hai chiều như: Khái niệm, các thao tác cơ bản, một số dạng bài tập cơ bản trên mảng hai chiều. Cuối cùng là các bài tập tự giải liên quan.

A. Tóm tắt lý thuyết

Mảng hai chiều là một cấu trúc dữ liệu quan trọng trong lập trình, đặc biệt trong các chương trình liên quan tới xử lý ảnh, thị giác máy tính, các mạng học sâu, xử lý dữ liệu đồ thị...

Mảng 2 chiều là một cấu trúc bộ nhớ gồm nhiều ô nhớ cùng tên, cùng kiểu nhưng khác nhau về chỉ số dùng để lưu trữ một bảng các phần tử cùng kiểu gồm nhiều dòng và nhiều cột.

Giải sử bảng dữ liệu có n dòng và m cột. Khi đó, ta có chỉ số dòng và chỉ số cột và chúng luôn bắt đầu từ 0. Phần tử tại dòng i , cột j có chỉ số là $[i][j]$. Ví dụ: Một mảng a gồm 3 dòng, 4 cột có dạng như sau:



Hình 5.1. Ví dụ về hình dạng của một mảng hai chiều.

5.1. Các thao tác cơ bản trên mảng hai chiều

- **Khai báo mảng:** Để khai báo một mảng hai chiều, ta cần chỉ ra tên mảng, số dòng và số cột tối đa của mảng. Ví dụ: Để khai báo một mảng số thực a gồm 3 dòng và 4 cột như trên, ta viết:

```
float a [ 3 ] [ 4 ] ;
```

Một cách tổng quát: Cú pháp để khai báo một mảng có p dòng và q cột (p, q là các số nguyên):

`<Kiểu_mảng> <Tên_mảng> [p] [q];`

Trong đó: `<Kiểu_mảng>` là kiểu của tất cả các phần tử trong mảng, có thể là kiểu nguyên thủy (`int, float, double, char, ...`) hoặc một kiểu tự định nghĩa (`struct, class, ...`). `<Tên_mảng>`: Do ta đặt tuân theo quy tắc đặt tên trong C++.

- **Khởi tạo mảng:** có 2 cách như sau:

```
int a[3][4] =  
{  
    {0, 1, 2, 3}, /* khai tao gia tri cho hang ma co chi muc la 0 */  
    {4, 5, 6, 7}, /* khai tao gia tri cho hang ma co chi muc la 1 */  
    {8, 9, 10, 11} /* khai tao gia tri cho hang ma co chi muc la 2 */  
};  
Hoặc:  
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

- **Nhập mảng:** Thao tác này bao gồm việc nhập số dòng và số cột thực tế của mảng (giả sử số dòng là n , số cột là m). Tiếp theo ta sử dụng hai vòng lặp `for` để nhập lần lượt từng phần tử của mảng. Hàm sau đây nhập vào từ bàn phím n, m và các giá trị của mảng a gồm n dòng, m cột.

```
1 void nhapmang(float a[100][100], int &n, int &m)  
2 {  
3     cout<<"n="; cin>>n;  
4     cout<<"m="; cin>>m;  
5     for(int i=0; i<n; i++)  
6         for(int j=0; j<m; j++)  
7         {  
8             cout<<"a [ "<<i<<" ] [ "<<j<<" ] =";  
9             cin>>a[i][j];  
10        }  
11 }
```

- **Xuất mảng:** Thao tác này đơn giản chỉ là sử dụng hai vòng lặp `for` để xuất lần lượt các phần tử của mảng ra màn hình. Hàm sau xuất các giá trị của mảng a gồm n dòng, m cột lên màn hình, có tạo dáng ma trận.

```
1 void xuatmang(float a[100][100], int n, int m)  
2 {  
3     for(int i=0; i<n; i++)  
4     {  
5         for(int j=0; j<m; j++)  
6             cout<<setw(5)<<a[i][j];  
7         cout<<endl;
```

8	}
9	}

- **Duyệt mảng:** Thao tác này xuất hiện phổ biến trong các bài xử lý mảng hai chiều. Để duyệt mảng, ta sử dụng hai vòng lặp `for` lồng nhau để “thăm” lần lượt từng phần tử của mảng. Đoạn giả code sau giúp ta duyệt một mảng a gồm n dòng, m cột:

1	<code>for(int i=0; i<n; i++)</code>
2	<code> for(int j=0; j<m; j++)</code>
3	<code> //Thăm a[i][j]</code>

- **Ví dụ 5.1:** Nhập vào một ma trận gồm n dòng, m cột các số thực. In ma trận vừa nhập ra màn hình. Cho biết giá trị lớn nhất trong ma trận.

Để giải quyết bài tập này, chúng ta đơn giản chỉ kết hợp 4 thao tác cơ bản trên mảng hai chiều: Khai báo, nhập, xuất, duyệt mảng.

1	<code>void nhapmang(float a[100][100], int &n, int &m)</code>
2	<code>{</code>
3	<code> cout<<"n="; cin>>n;</code>
4	<code> cout<<"m="; cin>>m;</code>
5	<code> for(int i=0; i<n; i++)</code>
6	<code> for(int j=0; j<m; j++)</code>
7	<code> {</code>
8	<code> cout<<"a["<<i<<"] ["<<j<<"]=";</code>
9	<code> cin>>a[i][j];</code>
10	<code> }</code>
11	<code> }</code>
12	<code>void xuatmang(float a[100][100], int n, int m)</code>
13	<code>{</code>
14	<code> for(int i=0; i<n; i++)</code>
15	<code> {</code>
16	<code> for(int j=0; j<m; j++)</code>
17	<code> cout<<setw(5)<<a[i][j];</code>
18	<code> cout<<endl;</code>
19	<code> }</code>
20	<code> }</code>
21	<code>float MAX(float a[100][100], int n, int m)</code>
22	<code>{</code>
23	<code> float Max = a[0][0];</code>
24	<code> for(int i=0; i<n; i++)</code>
25	<code> for(int j=0; j<m; j++)</code>
26	<code> if(a[i][j] > Max)</code>

```

27         Max = a[i][j];
28     return Max;
29 }
30 int main()
31 {
32     float a[100][100]; int n,m;
33     nhapmang(a, n, m);
34     xuatmang(a, n, m);
35     cout<<endl<<"MAX = "<<MAX(a, n, m);
36     return 0;
37 }
```

5.2. Một số dạng bài tập trên mảng hai chiều

- **Tìm kiếm:** Bài toán này về cơ bản tương tự như bài toán tìm kiếm trên mảng một chiều. Tuy nhiên, do đặc thù của mảng hai chiều, ta chia bài toán thành 3 trường hợp: Tìm kiếm trên toàn mảng, tìm kiếm trên từng dòng, tìm kiếm trên từng cột. Phương pháp chung là “duyệt và đếm”.

Ví dụ 5.2: Nhập vào một mảng a gồm $n \times m$ số thực và một số thực c . Hãy cho biết:

- + c có xuất hiện trong mảng hay không.
- + Dòng nào của mảng có xuất hiện c .
- + Cột nào của mảng có xuất hiện c .

Ba hàm tìm kiếm sau đây áp dụng phương pháp duyệt trên toàn mảng (**Search**), duyệt trên từng dòng (**SByRow**), duyệt trên từng cột (**SByCol**) để giải quyết bài toán trên.

```

1 void nhapmang(float a[100][100], int &n, int &m)
2 {
3     cout<<"n="; cin>>n;
4     cout<<"m="; cin>>m;
5     for(int i=0; i<n; i++)
6         for(int j=0; j<m; j++)
7         {
8             cout<<"a["<<i<<" ] ["<<j<<"] =" ;
9             cin>>a[i][j];
10        }
11    }
12
13 void xuatmang(float a[100][100], int n, int m)
14 {
15     for(int i=0; i<n; i++)
```

```

16     {
17         for(int j=0; j<m; j++)
18             cout<<setw(5)<<a[i][j];
19         cout<<endl;
20     }
21 }
22
23 void Search(float a[100][100], int n, int m, int
24 c)
25 {
26     bool found = false;
27     for(int i=0; i<n; i++)
28         for(int j=0; j<m; j++)
29             if(a[i][j]==c)
30                 found = true;
31     if(found)
32         cout<<c<<" found"<<endl;
33     else
34         cout<<c<<" not found"<<endl;
35 }
36 void SByRow(float a[100][100], int n, int m, int
37 c)
38 {
39     //Duyệt qua các dòng của ma trận
40     for(int i=0; i<n; i++)
41     {
42         bool found = false;
43         for(int j=0; j<m; j++)
44             if(a[i][j]==c) found = true;
45         if(found)
46             cout<<"Row "<<i+1<<" includes
47 "<<c<<endl;
48     }
49 }
50 void SByCol(float a[100][100], int n, int m, int
51 c)
52 {
53     //Duyệt qua các cột của ma trận
54     for(int j=0; j<m; j++)
55     {
56         bool found = false;
57         for(int i=0; i<n; i++)
58             if(a[i][j]==c) found = true;
59         if(found)
60             cout<<"Col "<<j+1<<" includes
61 "<<c<<endl;
62     }

```

```

63 }
64 int main()
{
65     float a[100][100]; int n,m;
66     nhapmang(a, n, m);
67     xuatmang(a, n, m);
68     float c;
69     cout<<"c="; cin>>c;
70     Search(a, n, m, c);
71     SByRow(a, n, m, c);
72     SByCol(a, n, m, c);
73     return 0;
74 }
75 }
```

Hàm **Search** ở trên có thể viết dưới dạng hàm có giá trị trả về. Hàm sau trả về true nếu tìm thấy c trong mảng a , ngược lại hàm trả về false.

```

1 bool Search(float a[100][100], int n, int m, int c)
2 {
3     for(int i=0; i<n; i++)
4         for(int j=0; j<m; j++)
5             if(a[i][j]==c)      return true;
6     return false;
7 }
```

- **Kiểm tra tính chất của mảng**

Một dạng bài toán khá phổ biến trên mảng hai chiều là thống kê trên ma trận. Việc thống kê một đại lượng nào đó nhằm mục đích kiểm tra một tính chất của mảng. Phương pháp chung để giải quyết bài toán thuộc dạng này là duyệt mảng. Một vấn đề khó khăn đặt ra là xác định chính xác xem cần thống kê đại lượng nào nếu để bài chưa nói rõ.

❶ Ví dụ 5.3: Nhập vào một ma trận vuông a ($n \times n$) các phần tử thực. Ma trận a được gọi là “hợp lệ” nếu tất cả các phần tử trên đường chéo chính đều bằng 0; các phần tử phía trên đường chéo chính đều dương và các phần tử còn lại đều âm. Hãy kiểm tra xem ma trận vừa nhập có hợp lệ không.

Ta biểu diễn điều kiện để ma trận vuông a là “hợp lệ” như sau ($i, j \in [0, n-1]$):

$$\begin{cases} \forall i = j, a[i][j] = 0 \\ \forall i < j, a[i][j] > 0 \\ \forall i > j, a[i][j] < 0 \end{cases}$$

Lấy phủ định của điều kiện trên, ta thu được điều kiện để mảng không hợp lệ là: “*Tồn tại ít nhất 1 phần tử trên đường chéo chính mà khác 0 hoặc tồn tại ít nhất 1 phần tử nằm phía trên đường chéo chính mà không dương, hoặc tồn tại ít nhất 1 phần tử nằm phía dưới đường chéo chính mà không âm*”. Áp dụng phương pháp duyệt mảng ta thu được hàm **Check** trả về giá trị `true` nếu mảng hợp lệ, ngược lại hàm trả về giá trị `false` như sau:

```

1 | bool Check(float a[100][100], int n)
2 | {
3 |     for(int i=0; i<n; i++)
4 |         for(int j=0; j<n; j++)
5 |         {
6 |             if(i==j && a[i][j] != 0) return false;
7 |             if(i<j && a[i][j] <= 0) return false;
8 |             if(i>j && a[i][j] >= 0) return false;
9 |         }
10 |     return true;
11 | }
```

❷ Ví dụ 5.4: Có n cửa hàng kinh doanh trong m tháng. Doanh thu của mỗi cửa hàng trong mỗi tháng đều được lưu trữ trong một ma trận có n dòng, m cột. Một cửa hàng sẽ bị đóng cửa nếu doanh thu của nó giảm liên tiếp trong $m-1$ tháng (trừ tháng đầu tiên). Hãy cho biết cửa hàng nào trong số n cửa hàng trên sẽ bị đóng cửa.

Bài toán được giải quyết bằng cách duyệt qua từng cửa hàng. Với mỗi cửa hàng i , ta đếm số lần giảm doanh thu của nó. Nếu cửa hàng nào có đúng $m-1$ lần giảm doanh thu, cửa hàng đó sẽ bị đóng cửa.

```

1 | void CheckClosed(float a[100][100], int n, int m)
2 | {
3 |     for(int i=0; i<n; i++)
4 |     {
5 |         int d=0;
6 |         for(int j=0; j<m-1; j++)
7 |             if(a[i][j]>a[i][j+1]) d++;
8 |         if (d==m-1)
9 |             cout<<"Close the store: "<<i+1<<endl;
```

10	}
11	}

- **Tính toán trên ma trận:**

Về mặt hình thức, mảng có dạng ma trận. Do vậy, một dạng bài tập phổ biến về mảng là thực hiện các bài toán trên ma trận. Thuộc dạng này có các bài như: Cộng, trừ, nhân hai ma trận; chuyển vị ma trận, đổi dấu ma trận, nghịch đảo ma trận...

❷ Ví dụ 5.5: Nhập vào một ma trận a ($n \times k$) và b ($k \times m$) các phần tử thực. Tính ma trận $c = a * b$.

Với hai ma trận a , b đã cho, tích của chúng là ma trận c có kích thước $n \times m$ với:

$$c[i][j] = \sum_{t=0}^{k-1} a[i][t]^* b[t][j].$$

Hàm sau thực hiện phép nhân hai ma trận a ($n \times k$) và b ($k \times m$) và thu được ma trận c ($n \times m$):

1	void Mul (float a[100][100], float b[100][100], float
2	c[100][100], int n, int m, int k)
3	{
4	for (i=0; i<n; i++)
5	for (j=0; j<m; j++)
6	{
7	c[i][j] = 0;
8	for (int t=0; t<k; t++)
9	c[i][j] += a[i][t]^*b[t][j];
10	}
11	}

❸ Ví dụ 5.6: Nhập vào một ma trận $a(n \times m)$ phần tử nguyên. Hãy chuyển vị ma trận a và in ma trận chuyển vị ra màn hình.

Chuyển vị của ma trận $a(n \times m)$ cho ta ma trận $b(m \times n)$ với:

$$b[j][i] = a[i][j], \forall 0 \leq i < n, 0 \leq j < m.$$

Trong ví dụ này, một lần nữa ta ôn lại các thao tác nhập, xuất mảng. Hàm **Trans** thực hiện chuyển vị một ma trận $a(n \times m)$ để thu được một ma trận $b(m \times n)$.

```

1 void input(int a[100][100], int n, int m)
2 {
3     for(int i=0; i<n; i++)
4         for(int j=0; j<m; j++)
5         {
6             cout<<"a["<<i<<"] ["<<j<<"]=";
7             cin>>a[i][j];
8         }
9 }
10 void output(int a[100][100], int n, int m)
11 {
12     for(int i=0; i<n; i++)
13     {
14         for(int j=0; j<m; j++)
15             cout<<setw(5)<<a[i][j];
16         cout<<endl;
17     }
18 }
19 void Trans(int a[100][100], int b[100][100], int n,
20 int m)
21 {
22     for(int i=0; i<n; i++)
23         for(int j=0; j<m; j++)
24             b[j][i] = a[i][j];
25 }
26 int main()
27 {
28     int a[100][100], b[100][100];
29     int n, m;
30     cout<<"n="; cin>>n;
31     cout<<"m="; cin>>m;
32     input(a, n, m);
33     Trans(a, b, n, m);
34     cout<<"Ma tran a: "<<endl;
35     output(a, n, m);
36     cout<<"Ma tran chuyen vi cua a:"<<endl;
37     output(b, m, n);
38 }

```

B. Các bài tập tự giải

Bài M1. Nhập một ma trận $a(n \times m)$ với các phần tử là số thực. Hãy tính tổng các phần tử dương trên ma trận và in kết quả ra màn hình. Hãy tính tổng của các phần tử xung quanh ma trận đồng thời in ma trận vừa nhập ra màn hình.

Bài M2. Nhập một ma trận vuông $a(n \times n)$ với các phần tử là số nguyên. Ma trận được gọi là hợp lệ nếu tổng các phần tử trên mỗi dòng của tất cả các dòng đều bằng nhau. Hãy kiểm tra xem ma trận vừa nhập có hợp lệ không. In mảng vừa nhập ra màn hình.

Bài M3. Nhập một ma trận vuông $a(n \times n)$ với các phần tử là số thực. Ma trận được gọi là hợp lệ nếu tất cả các dòng của nó, mỗi dòng chỉ chứa đúng 1 phân tử âm. Hãy kiểm tra xem ma trận vừa nhập có hợp lệ không. In mảng vừa nhập ra màn hình.

Bài M4. Viết chương trình nhập vào một ma trận $a(n \times m)$ với các phần tử là số nguyên. Tìm các phần tử lớn nhất và bé nhất trên các dòng (tương tự các cột) của ma trận.

Bài M5. Viết chương trình tìm phần tử âm đầu tiên (theo chiều từ trái qua phải, từ trên xuống dưới) trong ma trận $a(n \times m)$ với các phần tử là số thực.

Bài M6. Viết chương trình nhập vào hai ma trận A, B đều có n hàng, m cột với các phần tử là số thực. Tính ma trận $C = A + B$ và in kết quả ra màn hình.

Bài M7. Ma trận vuông $a(n \times n)$ được gọi là đối xứng qua đường chéo chính nếu $a[i, j] = a[j, i] \forall 0 \leq i, j \leq n-1$. Viết chương trình nhập vào một ma trận vuông a . Kiểm tra xem a có đối xứng qua đường chéo chính không và in kết luận lên màn hình.

Bài M8. Một ma trận vuông $a(n \times n)$ với các phần tử là số thực được gọi là hợp lệ nếu tất cả các phần tử nằm trên đường chéo chính bằng 1, tất cả các phần tử nằm phía trên đường chéo chính đều dương, tất cả các phần tử nằm phía dưới đường chéo chính đều âm. Hãy kiểm tra xem a có hợp lệ không?

Bài M9. Một đồ thị vô hướng không có trọng số $G(V, E)$ với V là tập đỉnh (n đỉnh) và E là tập cạnh (m cạnh) được biểu diễn bằng ma trận kề. Trong đó, người ta sử dụng một ma trận $a(n \times n)$ với các phần tử của a được xác định như sau: Nếu đỉnh i có cạnh nối tới đỉnh j thì $a[i][j] = a[j][i] = 1$, ngược lại $a[i][j] = a[j][i] = 0$. Một đỉnh được xem là không có đường nối tới chính nó. Độ tuổi của một đỉnh là số cạnh nối tới đỉnh đó. Viết chương trình để:

- Nhập vào 1 ma trận kề từ bàn phím.
- Tính độ tuổi của mỗi đỉnh trong đồ thị tương ứng với ma trận kề đó.

Bài M10. Một ảnh số đa cấp xám được biểu diễn (lưu trữ) bởi một ma trận $a(n \times m)$ với các phần tử nguyên và $a[i][j] \in [0, 255] \forall 0 \leq i < n, 0 \leq j < m$. Phép biến đổi âm bản ảnh a cho ta một ảnh a' theo công thức:

$$a'[i][j] = L - a[i][j], \forall 0 \leq i < n, 0 \leq j < m.$$

Ở đây, L là giá trị lớn nhất trong mảng a . Hãy nhập vào một ảnh số đa cấp xám, biến đổi âm bản và in kết quả ra màn hình.

Bài M11. Một ảnh số được lưu trữ dưới dạng một ma trận $a(n \times m)$ với $a[i][j] \in [0, 255]$. Người ta dùng 1 bộ lọc (filter) là một ma trận $b(3 \times 3)$ để thực hiện phép tích chập \otimes trên ảnh a thu được ảnh $a' = a \otimes b$, với:

$$a'_{i,j} = \sum_{k=0}^2 \sum_{t=0}^2 a_{i-1+k, j-1+t} * b_{kt}$$

$$\forall i \in [1, n-2], \forall j \in [1, m-2].$$

Nói cách khác, ta tiến hành “rê” ma trận b trên ảnh a sao cho tâm của b trùng với 1 điểm $a[i][j]$ (trừ các điểm ở viền của a) và tiến hành nhân các điểm của ma trận a với các điểm của ma trận b trùng với nó rồi cộng lại (thực hiện 9 phép nhân). Ví dụ:

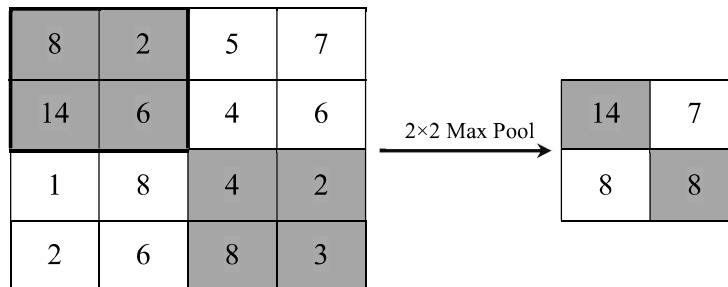
a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

b[0][0]	b[0][1]	b[0][2]
b[1][0]	b[1][1]	b[1][2]
b[2][0]	b[2][1]	b[2][2]

$$\begin{aligned} a'[1][2] = & a[0][1]*b[0][0] + a[0][2]*b[0][1] + a[0][3]*b[0][2] + \\ & a[1][1]*b[1][0] + a[1][2]*b[1][1] + a[1][3]*b[1][2] + \\ & a[2][1]*b[2][0] + a[2][2]*b[2][1] + a[2][3]*b[2][2] + \end{aligned}$$

Hãy nhập vào từ bàn phím ma trận ảnh a và ma trận bộ lọc b như trên. Tính ma trận a' và in kết quả ra màn hình.

Bài M12. Phép biến đổi 2×2 Max-Pool thực hiện trên một ma trận $a(n \times m)$ bằng cách rẽ một khung hình vuông kích thước 2×2 trên a sao cho các hình vuông không giao nhau. Bốn điểm trong hình vuông sẽ được biến đổi thành 1 điểm với giá trị mới là giá trị lớn nhất trong số 4 điểm trong hình vuông của a . Sau khi biến đổi, ma trận a về cơ bản sẽ giảm số dòng, số cột đi 2 lần.



Nhập vào từ bàn phím một ma trận $a(n \times m)$ với n, m là các số chẵn. Thực hiện biến đổi 2×2 Max-Pool trên a và in kết quả ra màn hình.

BÀI 6

KỸ THUẬT XỬ LÝ CHUỖI KÝ TỰ

Trong bài này, chúng ta sẽ dành thời gian để tìm hiểu các kỹ thuật cơ bản xử lý các bài toán trên chuỗi ký tự như: khai báo, nhập, xuất, duyệt, các thao tác đặc thù trên chuỗi ký tự. Sau đó là một số dạng bài tập chính trên dữ liệu kiểu này.

A. Tóm tắt lý thuyết

Chuỗi ký tự (hay xâu ký tự) là một dãy liên tiếp các ký tự. Như vậy, về bản chất chuỗi ký tự giống với một mảng một chiều mà mỗi phần tử của mảng là một ký tự. Trong thực tế, kiểu dữ liệu này rất phổ biến. Chúng ta có thể gặp dữ liệu kiểu chuỗi ký tự khi xử lý các trường dữ liệu như họ và tên, quê quán, địa chỉ... hoặc xử lý các tin nhắn, các văn bản ngắn.

C++ kế thừa kiểu dữ liệu `char` trong C để có thể sử dụng cho các biến kiểu chuỗi ký tự như là mảng các ký tự. Ngoài ra, C++ mới bổ sung vào kiểu dữ liệu `string` để giúp cho việc xử lý dữ liệu kiểu chuỗi ký tự được thuận tiện hơn (để sử dụng kiểu dữ liệu này, cần khai báo `#include <string>;`). Trong phần đầu, chúng ta xem xét các thao tác cơ bản sử dụng kiểu `char`. Phần thao tác xử lý xâu kiểu `string`, xin xem mục 6.2.

6.1. Một số thao tác cơ bản trên mảng ký tự

- **Khai báo biến kiểu chuỗi ký tự:**

```
char <Tên_biến> [<p>];
```

Trong đó p là một số nguyên, biểu diễn kích thước tối đa của mảng, cũng là độ dài tối đa của biến xâu ký tự.

- Khai báo như con trỏ kiểu xâu ký tự:

```
char * <Tên_biến>;
```

Với cách khai báo này, ta thu được `<Tên_biến>` là một con trỏ kiểu `char`. Để có thể sử dụng nó như một biến kiểu xâu ký tự, ta cần cấp phát bộ nhớ cho con trỏ (xem Bài 7):

```
char * <Tên_biến> = new char [<p>];
```

Hai lệnh khai báo sau đây là tương đương:

```
char S[30];  
char * S = new char[30];
```

Kết quả ta đều thu được biến `S` kiểu chuỗi ký tự. Biến này chỉ chứa được các chuỗi có độ dài không quá 30 ký tự.

- **Nhập/ xuất biến kiểu chuỗi ký tự:**

Thông thường ta sử dụng lệnh `gets` cho việc nhập dữ liệu từ bàn phím vào một biến kiểu chuỗi ký tự:

```
gets (<Biến>);
```

Trong đó: `<Biến>` là một biến kiểu mảng `char`. Nếu nhập liên tiếp các biến chuỗi ký tự kiểu mảng `char`, ta có thể làm sạch bộ đệm bàn phím bằng lệnh `fflush(stdin)`:

```
fflush(stdin); gets (<Biến>);
```

Việc xuất dữ liệu trong biến kiểu chuỗi ký tự cũng trở nên rất đơn giản bằng cách sử dụng một trong các lệnh xuất `puts`, `printf`, `cout`. Ví dụ, để xuất chuỗi ký tự trong biến `S` lên màn hình, ta viết:

```
cout<<S; hoặc puts(S);
```

☞ Khi chuỗi ký tự được lưu trong mảng, ký tự kết thúc chuỗi có ký hiệu ‘\0’ được thêm vào cuối chuỗi. Ví dụ khi bạn nhập chuỗi “HA NOI” vào biến `S`, cấu trúc của `S` như sau:

H	A		N	O	I	\0
---	---	--	---	---	---	----

- **Duyệt chuỗi ký tự:**

Để truy xuất tới từng ký tự trong một chuỗi ký tự, ta có thể truy xuất trực tiếp hoặc có thể duyệt chuỗi. Việc này cũng tương tự như ta truy xuất hoặc duyệt các phần tử của mảng. Tuy nhiên, ta cần sử dụng hàm `strlen(<Biến_xâu>)` trả về độ dài của chuỗi ký tự cần duyệt (hàm này có sẵn trong thư viện `string.h`). Giả sử duyệt xâu S, ta viết:

```
for (int i=0; i < strlen(S); i++)
{
    //Thăm ký tự S[i];
}
```

- **Phép gán chuỗi:**

Nếu *a* và *b* là hai biến kiểu nguyên (`int`), ta dễ dàng gán *a* sang *b* và ngược lại bằng cách viết: `a=b`; hoặc `b = a`;

Tuy nhiên, nếu *a* và *b* là hai biến chuỗi ký tự kiểu mảng `char`, việc sử dụng phép gán trên là không hợp lệ. Để gán chuỗi *b* sang biến kiểu mảng `char a`, ta phải sử dụng hàm `copy` chuỗi.

```
strcpy(a, b);
```

Một cách tổng quát, hàm `copy` chuỗi được viết như sau:

```
strcpy(S1, S2);
```

Trong đó, *S1* là một biến chuỗi ký tự kiểu mảng `char` còn *S2* có thể là một biến có kiểu tương tự hoặc một hằng chuỗi ký tự. Khi đó, chuỗi ký tự *S2* sẽ được gán sang biến *S1*. Ví dụ:

```
char S1[30], S2[30];
strcpy(S1, "Ha Noi"); //Gán "Ha Noi" vào S1
strcpy(S2, S1); //Gán S1 vào S2
```

- **Phép so sánh chuỗi ký tự:**

Để so sánh 2 biến chuỗi ký tự kiểu mảng `char`, ta cũng không được phép sử dụng toán tử so sánh (`==`) mà sử dụng hàm so sánh chuỗi:

```
strcmp(S1, S2);
```

Hàm này sẽ trả về giá trị bằng 0 nếu hai chuỗi bằng nhau; giá trị dương nếu *S1* > *S2* và giá trị âm nếu *S1* < *S2*. Đoạn chương trình sau đây

khai báo hai biến chuỗi ký tự kiểu mảng char có độ dài không quá 30 ký tự, nhập dữ liệu từ bàn phím vào S1 và S2, sau đó so sánh xem hai chuỗi vừa nhập có giống nhau hay không:

```
1  char S1[30], S2[30];
2  fflush(stdin); gets(S1);
3  fflush(stdin); gets(S2);
4  if (strcmp(S1, S2)==0)
5      cout<< "Xau S1 bang xau S2";
6  else
7      cout<< "Xau S1 khac xau S2";
```

☞ Hai hàm strcpy() và strcmp() có sẵn trong thư viện string.h.

- **Chuyển đổi qua lại giữa mã ASCII và ký tự:**

Một điểm đặc biệt của chuỗi ký tự kiểu mảng char là nó có thể hoạt động như một mảng với mỗi phần tử là một ký tự hoặc có thể hoạt động như một mảng số nguyên. Điều này có được là do dữ liệu được lưu trữ trong mảng là các mã trong bảng mã ASCII của chúng. Các mã này là các số nguyên từ 0 tới 255.

Trong một số bài toán ta cần lấy mã ASCII của các ký tự. Công việc này trong C++ được thực hiện một cách dễ dàng mà không cần sử dụng tới hàm chuyển đổi. Để lấy mã ASCII của một ký tự S[i] trong chuỗi S, ta chỉ cần đặt toán tử ép kiểu (int) ngay trước ký tự đó. Hai câu lệnh sau sẽ cho kết quả khác nhau:

```
cout<<S[i];
```

(1)

```
cout<<(int) S[i];
```

(2)

Câu lệnh (1) in ra màn hình ký tự S[i], còn câu lệnh (2) chỉ in ra mã ASCII của ký tự đó.

Tương tự như trên, việc chuyển mã ASCII thành ký tự cũng được thực hiện dễ dàng bằng toán tử ép kiểu (char). Giả sử muốn in ra màn hình ký tự có mã 65, ta chỉ cần viết:

```
cout<<(char) 65;
```

Khi đó ký tự ‘A’ sẽ được in ra màn hình do nó có mã ASCII là 65.

6.2. Một số thao tác cơ bản trên dữ liệu kiểu string

Việc sử dụng kiểu dữ liệu string trong C++ cho phép ta thực hiện các thao tác xử lý chuỗi ký tự một cách đơn giản hơn nhiều. Để sử dụng kiểu dữ liệu này, ta cần thêm chỉ thị tiền xử lý `#include <string>`.

Biến kiểu string có một ưu điểm là độ dài của mảng ký tự được thay đổi linh hoạt tùy theo dữ liệu lưu trữ trong đó. Trong trường hợp này, bộ nhớ được cấp phát động và vì ta không chỉ định kích thước của chuỗi ngay khi khai báo nên không có ô nhớ nào bị lãng phí. Tuy nhiên, việc thao tác trên chuỗi ký tự kiểu mảng `char` tỏ ra nhanh hơn là thao tác trên chuỗi ký tự kiểu string.

Hơn nữa, dữ liệu kiểu mảng `char` không được cung cấp nhiều hàm có sẵn để thao tác, trong khi lớp `string` định nghĩa một tập phong phú các hàm chức năng (trong lập trình hướng đối tượng gọi là các phương thức) cho phép các thao tác trên dữ liệu một cách tiện lợi. Bảng 6.1 dưới đây trình bày một số thao tác cơ bản trên một biến `S` có kiểu `string`:

Bảng 6.1. Một số thao tác trên dữ liệu kiểu string

Số	Thao tác	Thực hiện
1	Khai báo biến	<code>string S;</code>
2	Nhập dữ liệu	<code>getline(cin, S);</code>
3	Xuất dữ liệu	<code>cout<<S;</code>
4	Lấy độ dài của chuỗi	<code>int k = S.length();</code>
5	Duyệt chuỗi	<code>for(int i=0; i<S.length(); i++) cout<<S[i];</code>
5	So sánh chuỗi	<code>if(S1 == S2)</code>
6	Gán chuỗi	<code>S = "Ha noi"; S = S1;</code>

Do phạm vi của học phần, các phương thức xử lý dữ liệu kiểu string không được giới thiệu ở đây. Các bạn có thể tham khảo trong các tài liệu khác sau khi đã có kiến thức căn bản về lập trình hướng đối tượng.

6.3. Một số bài toán trên chuỗi ký tự

- Tìm kiếm, thống kê

Bài toán phổ biến trên dữ liệu kiểu chuỗi ký tự là tìm kiếm, thống kê một đại lượng nào đó, ví dụ như đếm số lần xuất hiện của một ký tự, đếm số từ, số câu... Phương pháp chung để xử lý các bài tập dạng này là “duyệt” và “đếm”.

Ví dụ 6.1: Cho một chuỗi ký tự bất kỳ. Một từ trong chuỗi là một dãy liên tiếp, dài nhất các ký tự khác ký tự trống. Ví dụ chuỗi

“ _Ha__Noi_Ngay__Thang____Nam__ ”

có 5 từ. Hãy cho biết chuỗi đã cho có bao nhiêu từ.

Dễ thấy với một chuỗi chưa chuẩn hóa thì số từ không tỷ lệ thuận với số dấu cách (ký tự trống). Do vậy việc đếm số dấu cách là không phù hợp. Thay vào đó, ta đi đếm số lần bắt đầu của một từ, đó là số lần $S[i]$ bằng một dấu cách và $S[i+1]$ khác dấu cách. Tuy nhiên, trong trường hợp $S[0]$ khác dấu cách thì ta đã đếm thiếu 1 từ đầu tiên nên phải tăng biến đếm lên 1. Hàm sau đây trả về số từ trong một chuỗi ký tự S:

```

1 int countword(char S[])
2 {
3     int d=0;
4     for(int i=0; i<strlen(S)-1; i++)
5         if(S[i] == ' ' && S[i+1] != ' ')
6             d++;
7     if(S[0] != ' ')
8         d++;
9     return d;
10 }
```

Ví dụ 6.2: Nhập một xâu ký tự S bất kỳ. Hãy đếm số lần xuất hiện của tất cả các chữ cái có trong S, không phân biệt chữ hoa và chữ thường.

Nếu không phân biệt chữ hoa và chữ thường thì bảng mã ASCII có tổng cộng 26 chữ cái. Trường hợp tồi nhất là cả 26 chữ cái này đều xuất hiện trong S. Do vậy ta sử dụng 26 biến đếm (mảng d gồm 26 phần tử nguyên). Nếu ta gặp một ký tự nào đó trong S thì biến đếm tương ứng của nó sẽ được tăng lên 1.

Để thực hiện đếm, ta duyệt lần lượt từng ký tự của chuỗi. Trước tiên ta kiểm tra xem ký tự được duyệt có nằm trong dải mã của chữ cái in hoa (mã từ 65 đến 90) hay không. Nếu có, ta chỉ việc tăng biến đếm của $d[k-65]$ lên 1. Sau đó, ta lại kiểm tra xem ký tự được duyệt có nằm trong dải

mã của chữ cái thường (mã từ 97 đến 122) hay không. Nếu có, ta chỉ việc tăng biến đếm của $d[k-97]$ lên 1.

```

1 void countallchar(char S[])
2 {
3     int d[26];
4     for(int i=0; i<26; i++)
5         d[i]=0;
6     for(int i=0; i<strlen(S); i++)
7     {
8         if(S[i]>=65 && S[i]<=90)
9             d[S[i]-65]++;
10        if(S[i]>=97 && S[i]<=122)
11            d[S[i]-97]++;
12    }
13    for(int i=0; i<26; i++)
14    if(d[i]>0)
15    {
16        cout<<(char)(i+65)<<" hoặc "<<(char)(i+97);
17        cout<<" xuất hiện "<<d[i]<<" lần !";
18    }
19 }
```

④ Ví dụ 6.3: Cho một biểu thức toán học với các dấu mở/đóng ngoặc ‘(‘ và ‘)’ dưới dạng một chuỗi ký tự. Các dấu mở/đóng ngoặc được gọi là hợp lệ nếu nó được đặt đúng chỗ cho phép trong biểu thức toán học đó. Ví dụ biểu thức: $(a+b) * (c+d)$ có các dấu mở/đóng ngoặc hợp lệ, nhưng biểu thức: $(a+b) * (c+d)$ hoặc $(a+b)) * ((c+d)$ lại không hợp lệ. Hãy cho biết một biểu thức có các dấu mở/đóng ngoặc hợp lệ hay không.

Trước tiên ta duyệt chuỗi ký tự theo chiều từ trái sang và sử dụng biến đếm $d1$ để đếm các ký tự là dấu mở ngoặc ‘(‘ và biến đếm $d2$ để đếm các ký tự là dấu đóng ngoặc ‘)’. Một biểu thức toán học được gọi là “có các dấu mở/đóng ngoặc hợp lệ” nếu thỏa mãn:

- + Mở trước đóng: Trong quá trình đếm từ trái qua phải, $d1$ luôn không nhỏ hơn $d2$.
- + Đóng mở bằng nhau: Kết thúc quá trình đếm, $d1$ phải bằng $d2$.

Hàm **isValid()** sau trả về giá trị `true` nếu biểu thức toán học đang chứa trong chuỗi ký tự S là hợp lệ, `false` nếu ngược lại:

```

1 bool isValid(char S[])
2 {
```

```

3   int d1=0, d2=0;
4   for(int i=0; i<strlen(S); i++)
5   {
6       if( S[i] == '(' ) d1++;
7       if( S[i] == ')' ) d2++;
8       if(d1<d2) return false;
9   }
10  if(d1 != d2) return false;
11  return true;
12 }
13 int main()
14 {
15     char s[100];
16     gets(s);
17     if(isValid(s))
18         cout<<"Bieu thuc hop le !";
19     else
20         cout<<"Bieu thuc khong hop le !";
21     return 0;
22 }
```

• Chuẩn hóa chuỗi

Cắt các ký tự trống ở hai đầu chuỗi, xoá bớt dấu cách nếu có 2 dấu cách liền nhau trong thân chuỗi, trước dấu chấm câu không có dấu cách, sau dấu chấm câu có 1 dấu cách, ký tự đầu câu cần viết hoa... là các thao tác chuẩn hóa chuỗi. Nói chung, ta cần đưa một chuỗi ký tự (giả sử chưa chuẩn) về một chuỗi mà dữ liệu của nó đã được chuẩn hóa.

Kỹ thuật chuẩn hóa còn được mở rộng thành các bài toán phức tạp hơn như: Điều dấu văn bản tiếng Việt, sửa lỗi chính tả... Trong trường hợp này, các kỹ thuật lập trình là chưa đủ. Người ta thường sử dụng các kỹ thuật xử lý ngôn ngữ tự nhiên, trí tuệ nhân tạo mới có thể giải quyết được một phần bài toán.

❶ Ví dụ 6.4: Nhập vào từ bàn phím một chuỗi ký tự S bất kỳ. Xóa mọi dấu cách (nếu có) ở đầu và cuối chuỗi. Xóa bớt các dấu cách trong chuỗi sao cho không tồn tại những chỗ có nhiều hơn 1 dấu cách liền nhau.

Hàm **strdelete** () sau sẽ thực hiện xóa một ký tự trong chuỗi S tại vị trí *k* (với giả thiết là *k* hợp lệ). Khi xóa, chúng ta “đẩy” toàn bộ các ký tự từ vị trí *k*+1 lên phía trước 1 vị trí, kể cả ký tự kết thúc chuỗi.

1	void strdelete (char s[], int k)
---	---

```

2  {
3      for (int j = k; j<strlen(s); j++)
4          s[j] = s[j+1];
5  }
6  int main()
7  {
8      char s[100]; gets(s);
9      //Xóa các ký tự trắng ở đầu chuỗi
10     while(s[0] == ' ')
11         strdelete(s, 0);
12     //Xóa các ký tự trắng ở cuối chuỗi
13     while(s[strlen(s)-1] == ' ')
14         strdelete(s, strlen(s)-1);
15     cout<<endl<<s;
16     //Xóa bớt các ký tự trắng liên tiếp
17     for (int i=0; i<strlen(s)-1; i++)
18         while(s[i]==' ' && s[i+1]==' ')
19             strdelete(s, i);
20     cout<<endl<<s;
}

```

B. Các bài tập tự giải

Bài C1. Nhập vào từ bàn phím một chuỗi ký tự gồm toàn các chữ cái. Chuỗi có chứa nhiều câu với các câu được phân tách bởi dấu chấm ‘.’ Hãy cho biết chuỗi ký tự vừa nhập có bao nhiêu câu.

Bài C2. Nhập vào từ bàn phím một chuỗi ký tự bất kỳ. Hãy cho biết chuỗi vừa nhập có bao nhiêu chữ cái hoa và chữ cái thường.

Bài C3. Nhập vào một chuỗi ký tự S từ bàn phím và một ký tự C. Xóa mọi ký tự C trong S và in chuỗi S sau khi xóa lên màn hình.

Bài C4. Viết chương trình cho phép chèn một ký tự C vào vị trí k trong xâu S. In kết quả sau khi chèn ra màn hình.

Bài C5. Nhập vào từ bàn phím một chuỗi ký tự S. Một đường đi trong S là dãy liên tiếp các ký tự giống nhau (không phân biệt chữ hoa và chữ thường), độ dài của đường đi là số ký tự có trong đường đi. Hãy cho biết độ dài của đường đi dài nhất trong S.

Bài C6. Nhập vào từ bàn phím một chuỗi ký tự gồm toàn chữ cái. Hãy chuyển các ký tự bắt đầu của tất cả các từ trong chuỗi thành chữ viết hoa nếu nó là chữ thường (một từ được định nghĩa là một cụm chữ cái liên tiếp, dài nhất không chứa dấu cách).

Bài C7. Nhập vào từ bàn phím một chuỗi ký tự gồm toàn chữ cái. Hãy in ra các từ trong chuỗi, mỗi từ trên một dòng. Cho biết từ dài nhất trong chuỗi có bao nhiêu chữ cái.

Bài C8. Viết chương trình C ++ để kiểm tra xem một chuỗi đã cho có phải là một Palindrome hay không. Palindrome là một chuỗi ký tự mà khi đọc lần lượt các ký tự theo chiều từ trái sang phải hoặc từ phải sang trái là như nhau. Ví dụ chuỗi “madam” hay “racecar” là các Palindrome.

Bài C9. Viết chương trình C ++ để kiểm tra xem một chuỗi S có phải là một chuỗi con của chuỗi Q hay không. Ví dụ nếu chuỗi S = “Ha Noi” và chuỗi Q = “Ha Noi Ngay Thang Nam” thì S là chuỗi con của Q.

BÀI 7

KỸ THUẬT LẬP TRÌNH VỚI CON TRỎ

Trong bài này, chúng ta sẽ dành thời gian tóm lược lại một số khái niệm, kỹ thuật cơ bản về con trỏ: Khi báo biến con trỏ, cấp phát/thu hồi bộ nhớ cho con trỏ cũng như các ứng dụng của con trỏ trong mảng và hàm.

A. Tóm tắt lý thuyết

Trong C++, mỗi byte trong bộ nhớ đều được đánh địa chỉ là một con số hệ thập lục phân. Địa chỉ của biến là địa chỉ của byte đầu tiên trong ô nhớ dành cho biến.

Con trỏ (hay biến con trỏ) là một biến đặc biệt dùng để chứa địa chỉ của các biến khác. Con trỏ cũng có thể có kiểu nguyên thủy (int, float, double, char...) hoặc một kiểu tự định nghĩa (struct, class...). Con trỏ thuộc kiểu nào chỉ chứa địa chỉ của biến thuộc kiểu đó. Khi con trỏ đang chứa địa chỉ của biến nào, ta nói nó đang “trỏ” tới biến đó. Việc sử dụng một ô nhớ (con trỏ) để tham chiếu tới các ô nhớ khác nhau như vậy cho phép ta linh hoạt truy xuất, quản lý các biến thông qua con trỏ.

7.1. Một số thao tác cơ bản trên con trỏ

- **Khai báo biến con trỏ**

$\langle\text{Kiểu}\rangle \ * \ \langle\text{Tên_con_trỏ}\rangle;$

Trong đó: $\langle\text{Kiểu}\rangle$ có thể là một trong các kiểu nguyên thủy của C++ hoặc kiểu tự định nghĩa. $\langle\text{Tên_con_trỏ}\rangle$ được đặt theo quy ước đặt tên biến. Ví dụ:

```
float a, *p;
```

Dòng trên khai báo a là biến thông thường (*variable*) và có kiểu float, còn p là con trỏ (*pointer*) cùng kiểu với a .

- **Phép lấy địa chỉ của biến**

Giả sử a là một biến kiểu float và p là một con trỏ cùng kiểu với a . Để p trỏ tới a , ta viết:

```
p = &a;
```

Toán tử `&` cho phép lấy địa chỉ của một biến bất kỳ. Một cách tổng quát, để lấy địa chỉ của một biến ta viết:

```
&<Tên biến>;
```

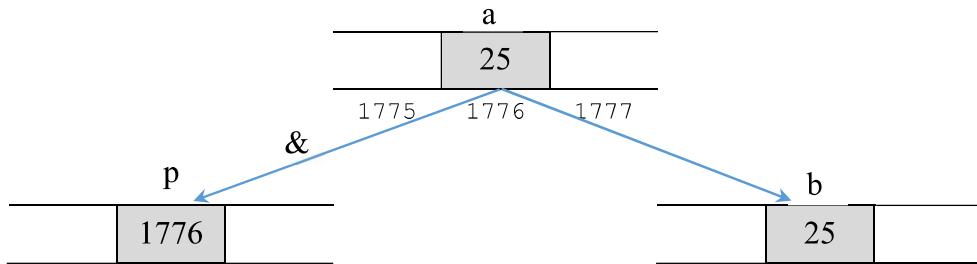
Xét các câu lệnh sau:

```
int a = 25;
```

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

```
int b = a;
```

Các giá trị chứa trong mỗi biến sau khi thực hiện các câu lệnh này được hiển thị trong sơ đồ sau:



• Phép truy xuất tới biến

Khi một con trỏ đang trỏ tới một biến, ta có thể sử dụng con trỏ để truy xuất tới biến đó. C++ sử dụng dấu `*` ở trước tên con trỏ để biểu thị biến mà nó đang trỏ tới. Cụ thể hơn: `*p` chính là biến mà `p` đang trỏ tới. Ví dụ với biến `a` kiểu `int`, `p` là con trỏ cùng kiểu và `p` đang trỏ tới `a`:

```
int a = 25;
```

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

Khi đó, `*p` chính là `a` và hai câu lệnh sau cùng in ra màn hình số 25:

```
cout<<a;  
cout<<*p;
```

Thậm chí, chúng ta có thể thay đổi giá trị của `a` khi sử dụng con trỏ `p`, ví dụ:

```
*p = 50;  
cout<<a;
```

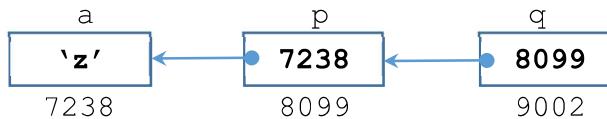
Lúc này giá trị của *a* đã bị thay đổi thành 50 và kết quả sẽ in ra màn hình số 50.

- **Con trỏ trỏ tới con trỏ**

C ++ cho phép sử dụng các con trỏ trỏ tới các con trỏ khác cùng kiểu, đến lượt các con trỏ này lại trỏ tới dữ liệu (hoặc thậm chí tới các con trỏ khác). Cú pháp khai báo chỉ cần thêm một dấu '*' cho mỗi mức chuyển hướng. Xét các câu lệnh sau:

```
char a;  
char * p;  
char ** q;  
a = 'z'; p = &a; q = &p;
```

Trong trường hợp này, *q* trỏ tới *p* và *p* trỏ tới *a*. Ta nói *q* là con trỏ trỏ tới con trỏ. Nếu muốn sử dụng *q* để truy xuất tới dữ liệu trong *a*, ta viết ***q*. Giả sử các vị trí bộ nhớ được chọn ngẫu nhiên cho mỗi biến *a*, *p*, *q* lần lượt là: 7238, 8099 và 9002, giá trị của chúng được biểu diễn như sau:



- **Con trỏ void và con trỏ null**

Con trỏ `void` là một loại con trỏ đặc biệt. Trong C ++, `void` thể hiện sự vắng mặt của kiểu dữ liệu. Do đó, con trỏ `void` là con trỏ trỏ đến một biến thuộc bất kỳ kiểu dữ liệu nào, từ giá trị số nguyên hoặc số thực đến một chuỗi ký tự.

Đổi lại, chúng có một hạn chế lớn: Dữ liệu được trỏ đến bởi chúng không thể được truy xuất trực tiếp thông qua toán tử `*`. Vì lý do đó, ta chỉ có thể truy cập tới dữ liệu một cách gián tiếp thông qua một con trỏ có định kiểu. Trong ví dụ dưới đây, ta sử dụng một biến nguyên *a*, một biến thực *b* và một con trỏ không định kiểu *p*.

```
int a = 25;  
float b = 30.5;  
void *p;
```

Khi đó, *p* có thể trỏ tới *a* hoặc tới *b* bất kể *a*, *b* có hai kiểu khác nhau. Tuy nhiên, để truy xuất giá trị của biến mà *p* trỏ tới, ta không thể viết **p*, mà phải sử dụng các con trỏ có kiểu phù hợp để trỏ tới *p* và truy xuất gián tiếp thông qua con trỏ này.

```

p = &a;
int*q = (int*)p;
cout<<*q;

p=&b;
float*t = (float*)p;
cout<<*t;

```

Đôi khi, một con trỏ thực sự cần phải trỏ đến hư không, hay nói khác đi, con trỏ không trỏ tới đâu cả. Đôi với những trường hợp như vậy, tồn tại một giá trị đặc biệt mà bất kỳ kiểu con trỏ nào cũng có thể nhận: Giá trị con trỏ *null*. Giá trị này có thể được thể hiện trong C++ theo hai cách: Hoặc với giá trị nguyên bằng 0 hoặc với giá trị *NULL*. Ví dụ, với con trỏ *p* có kiểu bất kỳ, ta có thể gán *p* = 0; hoặc *p* = *NULL*; . Trong C++ 11, ta có thể sử dụng cách thứ 3 là gán con trỏ bằng giá trị *nullptr* thay cho giá trị 0 hoặc *NULL*, cái mà có thể gây hiểu lầm trong một số trường hợp.

• Con trỏ trỏ tới hàm

C++ cho phép truyền trực tiếp một hàm làm đối số cho một hàm khác. Con trỏ đến hàm được khai báo với cú pháp giống như khai báo hàm thông thường, ngoại trừ tên của hàm được đặt giữa dấu ngoặc đơn (.) và dấu '*' được chèn trước tên.

Xét ví dụ sau đây. Trước tiên, ta định nghĩa hai hàm *int cong(int, int)* và *int tru(int, int)* để thực hiện phép cộng và phép trừ hai số nguyên:

1	<i>int cong (int a, int b)</i>
2	{ return (a+b); }
3	
4	<i>int tru(int a, int b)</i>
5	{ return (a-b); }

Sau đó, ta định nghĩa một hàm *thuchien()* với 3 đối vào. Đôi thứ nhất và đối thứ 2 là hai số nguyên cần thực hiện phép toán. Đối thứ 3

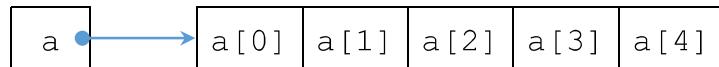
là một hàm có hai đối số nguyên, trả về giá trị kiểu nguyên được viết theo cú pháp `int (*tenham)(int, int)`. Khi sử dụng hàm `thuchien()`, tùy theo đối số thứ 3, hàm sẽ thực hiện phép cộng hoặc phép trừ tương ứng:

```

1 int thuchien(int x, int y, int (*ham)(int,int))
2 {
3     return (*ham)(x,y);
4 }
5 int main()
6 {
7     int m = thuchien(7, 5, cong);
8     int n = thuchien(2, 9, tru);
9     cout <<n<<" , "<<m;
10    return 0;
11 }
```

7.2. Con trỏ và mảng

Tên mảng chính là con trỏ: Khi ta khai báo mảng, tên mảng sẽ là một con trỏ trỏ tới phần tử đầu tiên của mảng. Ví dụ, với khai báo `int a[5]`; ta thu được cấu trúc bộ nhớ như sau:



Phép cộng/trừ địa chỉ:

Với các phép toán số học, chỉ các phép toán cộng và trừ mới được phép thực hiện trên các con trỏ. Những phép toán khác không có ý nghĩa gì trong thế giới của các con trỏ. Nhưng cả phép cộng và phép trừ đều có hành vi hơi khác biệt so với cộng trừ các số nguyên thông thường. Chúng là các phép cộng, trừ địa chỉ. Tùy theo kích thước k byte của kiểu dữ liệu chúng trỏ tới mà mỗi khi ta cộng thêm 1 đơn vị vào con trỏ, địa chỉ đang chứa trong con trỏ bị dịch chuyển đi k byte. Do vậy, chúng sẽ có giá trị là địa chỉ của ô nhớ tiếp theo. Ví dụ: Với a đang chứa địa chỉ của $a[0]$ thì $(a+1)$ sẽ là địa chỉ của $a[1]$. Vậy để truy xuất tới phần tử thứ i trong mảng a , thay vì viết $a[i]$, ta có thể viết: $* (a+i)$.

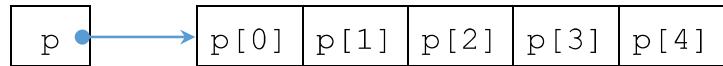
Tuy nhiên, một con trỏ thì không hẳn sẽ là một mảng. Ta có quy tắc sau:

Con trỏ + Cáp phát bộ nhớ = Mảng

Ví dụ, ta có một con trỏ nguyên p và cáp phát bộ nhớ cho p bằng toán tử **new**:

```
int *p = new int[5];
```

Lúc này, ta sẽ thu được một mảng p có 5 phần tử:



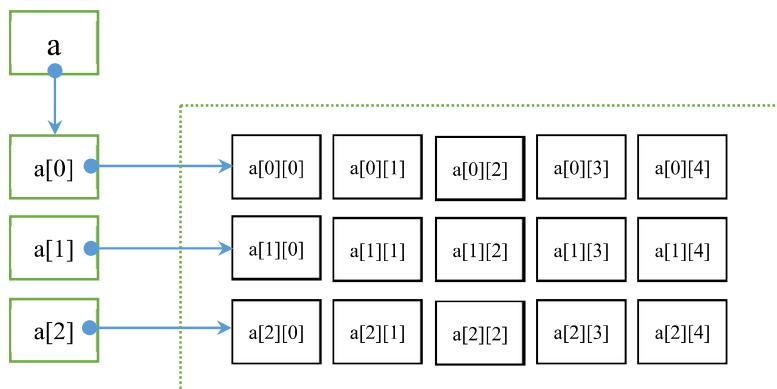
Hiển nhiên, vì p là mảng, ta có thể truy xuất các phần tử của p theo hai cách. Phần tử thứ i của p được viết là $*(p+i)$ hoặc viết tương tự như một mảng thông thường là $p[i]$.

Vậy hai câu lệnh khai báo sau là tương đương:

```
int a[5];  
int* a = new int[5];
```

Con trỏ và mảng hai chiều:

Mảng hai chiều về bản chất có thể xem như là mảng của các mảng một chiều (tức là mảng một chiều mà mỗi phần tử của nó lại là một mảng một chiều). Giả sử ta có mảng a (3×5) gồm 3 dòng và 5 cột, cấu trúc bộ nhớ của mảng sẽ như sau:



Hình 7.1. Cấu trúc bộ nhớ của mảng $a(3 \times 5)$.

Tên mảng (a) là một con trỏ, nó trỏ tới một mảng bao gồm 3 con trỏ ($a[0]$, $a[1]$, $a[2]$). Đến lượt nó, mỗi con trỏ $a[i]$ lại quản lý một

mảng một chiều gồm 5 phần tử. Như vậy ta có một mảng 3×5 phần tử. Do vậy, khi khai báo mảng hai chiều, ta có thể viết (ví dụ mảng a kiểu thực):

```
float ** a;
```

Khi cấp phát bộ nhớ cho mảng $a(n \times m)$, ta cần có hai bước:

Bước 1: Cấp phát n phần tử cho mảng a , mỗi phần tử là một con trỏ:

```
a = new float* [n];
```

Bước 2: Duyệt qua các con trỏ $a[i]$. Với mỗi con trỏ ta cấp cho chúng m ô nhớ:

```
for (int i=0; i<n; i++)
    a[i] = new float[m];
```

7.3. Cấp phát, thu hồi bộ nhớ cho con trỏ

Việc sử dụng con trỏ cho các bài toán về mảng sẽ giúp tiết kiệm bộ nhớ nếu như ta cấp phát bộ nhớ động cho con trỏ (tức sử dụng tới đâu, cấp phát tới đó). Ta sử dụng các hàm định vị bộ nhớ (allocation memory) của C như sau (giả sử ta sử dụng con trỏ p):

- **Hàm malloc**

Cú pháp:

```
p = (<Kiểu_của_p>*) malloc(<n>, <size>);
```

Trong đó, $\langle n \rangle$ là số ô nhớ cần cấp phát hay số phần tử của mảng; $\langle size \rangle$ là kích thước của một ô nhớ. Hàm malloc nếu thực hiện thành công sẽ cấp phát một vùng nhớ có kích thước $\langle n \rangle * \langle size \rangle$ byte và con trỏ sẽ trỏ tới ô nhớ đầu tiên của vùng nhớ này. Ngược lại, nếu thực hiện không thành công (do không đủ bộ nhớ hoặc $\langle n \rangle$ hoặc $\langle size \rangle$ không hợp lệ) hàm sẽ trả về giá trị NULL.

Tùy theo kiểu của p mà kích thước của một ô nhớ cấp phát cho p là khác nhau. Để lấy được kích thước này một cách chính xác, ta hay sử dụng toán tử sizeof với cú pháp: `sizeof(<kiểu>)`. Câu lệnh sau sẽ khai báo một con trỏ p kiểu thực và cấp phát cho nó 5 ô nhớ để biến nó thành một mảng thực có 5 phần tử:

```
float *p = (float*) calloc(5, sizeof(float));
```

- **Hàm malloc**

Tương tự như hàm `calloc`, hàm `malloc` được dùng để cấp phát bộ nhớ cho con trỏ. Cú pháp:

```
p = (<Kiểu_của_p>*) malloc(<size>);
```

Trong đó `<size>` là kích thước của vùng nhớ cần cấp phát tính bằng byte. Chẳng hạn ta cần cấp phát bộ nhớ cho một mảng `p` gồm 5 phần tử kiểu thực. Khi đó kích thước vùng nhớ cần cấp phát là `5*sizeof(float)` ta viết:

```
p = (float*) malloc(5 * sizeof(float));
```

- **Cấp phát lại bộ nhớ bằng realloc**

Đôi khi, trong quá trình hoạt động, kích thước của mảng lại thay đổi. Khi đó, ta nên cấp phát lại bộ nhớ cho mảng để tránh trường hợp thừa hoặc thiếu bộ nhớ. Cấp phát lại sẽ đảm bảo thay đổi kích thước vùng nhớ đang được quản lý bởi con trỏ (mảng) nhưng không làm thay đổi các giá trị mà con trỏ đang quản lý. Để làm điều đó, ta sử dụng hàm `realloc` với cú pháp:

```
p = (<Kiểu_của_p>*) realloc(p, <new_size>);
```

Trong đó, `<new_size>` là kích thước mới của vùng nhớ và được tính bằng byte. Câu lệnh sau cấp phát lại bộ nhớ cho mảng `p` gồm `n` phần tử kiểu thực với số ô nhớ được tăng lên 1:

```
p = (float*) realloc(p, (n+1)*sizeof(float));
```

☞ Hàm `realloc` chỉ thực sự có hiệu quả khi cấp phát lại các vùng nhớ đã được cấp phát bởi hàm `calloc` hoặc `malloc`. Trong trường hợp sử dụng toán tử `new` để cấp phát ban đầu, ta không sử dụng `realloc` để cấp phát lại bộ nhớ.

❷ Ví dụ 7.1: Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào một mảng `a` gồm `n` phần tử nguyên. Xóa mọi phần tử chẵn ra khỏi mảng. In mảng kết quả ra màn hình.

Hàm `input`: Cho phép nhập `n`, cấp phát bộ nhớ cho con trỏ `a` và nhập các phần tử của `a`. Do vậy, cần chú ý truyền tham chiếu cho đối `n`.

Trong hàm **input**, ta cũng cấp phát bộ nhớ cho *a* nên đối thứ nhất phải là `int*& a` thay vì `int* a`.

Hàm **print**: Thực hiện việc in mảng ra màn hình.

Hàm **evenDelete**: Thực hiện việc xóa mọi phần tử chẵn trong *a* theo các bước sau:

- Duyệt lần lượt các phần tử của *a*, từ *a[0]* tới *a[n-1]* (dòng 19).
- Chừng nào mà còn gặp một phần tử chẵn và chưa duyệt hết mảng thì còn (dòng 20): 1) xóa *a[i]*: Đẩy toàn bộ các phần tử từ *a[i+1]* trở đi lên phía trước 1 vị trí (dòng 22, 23) và 2) giảm *n* đi 1 đơn vị (dòng 24).
- Cuối cùng, ta cấp phát lại bộ nhớ để giải phóng những ô nhớ thừa (dòng 26).

```
1 void input(int*& a, int& n)
2 {
3     cout<<"n="; cin>>n;
4     a = (int*) malloc(n, sizeof(int));
5     for(int i=0; i<n; i++)
6     {
7         cout<<"a["<<i<<"]=";
8         cin>>a[i];
9     }
10 }
11 void print(int*a, int n)
12 {
13     for(int i=0; i<n; i++)
14         cout<<setw(5)<<a[i];
15     cout<<endl;
16 }
17 void evenDelete(int*a, int &n)
18 {
19     for(int i=0; i<n; i++)
20     while(a[i]%2==0 && i<n)
21     {
22         for(int j=i+1; j<n; j++)
23             a[j-1]=a[j];
24         n--;
25     }
26     a = (int*) realloc(a, n*sizeof(int));
27 }
28 int main()
```

```

29  {
30      int *a; int n;
31      input(a, n);
32      print(a, n);
33      evenDelete(a, n);
34      print(a, n);
35      return 0;
36  }

```

- **Cấp phát bộ nhớ bằng toán tử new**

Trong C++, ta thường sử dụng toán tử `new` để cấp phát bộ nhớ, thay vì sử dụng các hàm `malloc`, `calloc`, `realloc`. Cú pháp sau cấp phát cho `p` một ô nhớ và `p` trỏ tới ô nhớ này:

`p = new <Kiểu_của_p>;`

Nếu muốn cấp phát nhiều ô nhớ cho `p`, ta sử dụng cú pháp sau:

`p = new <Kiểu_của_p> [n];`

Trong đó, `n` là số ô nhớ (hay số phần tử của mảng) cần cấp phát.

- **Thu hồi bộ nhớ bằng toán tử delete**

Để thu hồi bộ nhớ đã được cấp phát bởi toán tử `new` cho con trỏ `p`, ta có thể sử dụng toán tử `delete` với cú pháp:

`delete [] p;`

☞ Toán tử `delete` sẽ chỉ hiệu quả khi thu hồi bộ nhớ đã được cấp phát bởi toán tử `new`. Trường hợp bộ nhớ được cấp phát bởi hàm `malloc` hay `realloc`, ta không sử dụng `delete`. Thay vào đó, ta sử dụng hàm `free`, ví dụ: `free(p);`

❷ **Ví dụ 7.2:** Nhập một mảng `a` gồm `n` phần tử thực. Tách các phần tử âm sang mảng `b` và các phần tử dương sang mảng `c`. In ba mảng `a`, `b`, `c` ra màn hình.

Giả sử sau khi tách, mảng `b` có `m` phần tử và mảng `c` có `k` phần tử. Hàm `Separate` sau đây nhận đối vào là `double*a`, `int n` và tính toán đổi ra là các mảng `b`, `c`: `double*& b`, `int&m`, `double*& c`, `int& k`.

Trước tiên, ta thực hiện đếm số phần tử âm và số phần tử dương trong mảng a (tức xác định kích thước m, k của b và c) để có thể cấp phép bộ nhớ cho b và c (dòng 4 tới dòng 9). Sau đó, ta thực hiện việc cấp phát bộ nhớ cho b và c bằng toán tử `new` (dòng 10, 11).

Để thực hiện tách mảng, ta sử dụng hai chỉ số j, t trong đó j chạy trên mảng b và t chạy trên mảng c . Cả hai chỉ số j, t đều xuất phát từ 0 (dòng 12). Ta duyệt qua lần lượt các phần tử của mảng a (dòng 13). Nếu gặp phần tử $a[i]$ âm, ta gán $a[i]$ sang $b[j]$ và tăng j lên 1 đơn vị (dòng 15 tới 18). Nếu gặp phần tử $a[i]$ dương, ta gán $a[i]$ sang $c[t]$ và tăng t lên 1 đơn vị (dòng 19 tới 22).

```

1 void Separate(double*a, int n, double*& b, int&
2   m, double*& c, int& k)
3 {
4     m = k = 0;
5     for(int i=0; i<n; i++)
6     {
7         if(a[i]<0) m++;
8         if(a[i]>0) k++;
9     }
10    if(m>0) b = new double[m];
11    if(k>0) c = new double[k];
12    int j=0, t=0;
13    for(int i=0; i<n; i++)
14    {
15        if(a[i]<0)
16        {
17            b[j]=a[i]; j++;
18        }
19        if(a[i]>0)
20        {
21            c[t]=a[i]; t++;
22        }
23    }
24 }
```

B. Các bài tập tự giải

Bài P.1. Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào một mảng a gồm n phần tử nguyên, sao chép các phần tử lẻ của mảng đặt vào cuối mảng. In mảng kết quả ra màn hình.

Bài P.2. Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào từ bàn phím một mảng a gồm n phần tử nguyên. Nhập vào một số nguyên k từ bàn phím. Xóa một phần tử tại vị trí k (nếu có) của mảng và in mảng kết quả ra màn hình.

Bài P.3. Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào từ bàn phím một mảng a gồm n phần tử kiểu `double`. Xóa mọi phần tử tại vị trí lẻ trong a và in kết quả ra màn hình.

Bài P.4. Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào từ bàn phím một mảng a gồm n phần tử kiểu `float`. Chèn một số thực C vào vị trí k trong mảng a và in mảng kết quả ra màn hình.

Bài P.5. Sử dụng con trỏ, cấp phát bộ nhớ động để nhập vào từ bàn phím một mảng a gồm n số nguyên. Hãy tách mảng a thành hai mảng b và c sao cho b chứa toàn bộ các số chẵn, còn c chứa toàn bộ các số lẻ của a . In ba mảng a , b , c ra màn hình.

Bài P.6. Viết các hàm thực hiện việc cộng, trừ nhân, chia hai số thực bất kỳ. Sử dụng con trỏ hàm để viết một hàm `Calculate()` thực hiện các phép toán cộng, trừ nhân chia hai số thực bất kỳ, trong đó, đối vào của nó tham chiếu tới các hàm đã viết ở trên. Viết hàm main minh họa cách sử dụng hàm `Calculate()`.

Bài P.7. Một ảnh số đa cấp xám được cho dưới dạng một ma trận $a(n \times m)$ các phần tử nguyên có giá trị thuộc $[0, 255]$. Phép khử nhiễu trên ảnh được thực hiện đơn giản bằng cách tính lại các giá trị của mảng, theo đó, giá trị $a[i][j]$ được tính lại bằng trung bình cộng của 8 giá trị xung quanh nó (trừ các điểm ảnh trên viền ảnh được giữ nguyên):

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

$$\begin{aligned}
 a[1][1] &= \\
 a[0][0] + a[0][1] + a[0][2] + a[1][2] + \\
 a[2][2] + a[2][1] + a[2][0] + a[1][0]
 \end{aligned}$$

Sử dụng con trỏ, cáp phát bộ nhớ động để nhập vào một ảnh số như trên, thực hiện khử nhiễu và in ma trận kết quả lên màn hình.

BÀI 8

KỸ THUẬT XỬ LÝ TỆP VĂN BẢN

Bài số 8 sẽ giới thiệu với chúng ta một số thao tác cơ bản khi xử lý tệp văn bản như: Khai báo dòng nhập/xuất tệp, ghi dữ liệu vào tệp, đọc dữ liệu từ tệp. Sau đó, một số dạng bài tập sẽ được trình bày.

A. Tóm tắt lý thuyết

8.1. Mở tệp để ghi dữ liệu

```
ofstream <Tên_bien>;  
<Tên_bien>.open(<Tên_tệp>, <Kiểu_mở>);
```

Trong đó, `<Tên_bien>` do ta đặt và được gọi là một dòng xuất tệp. `<Tên_tệp>` là một chuỗi ký tự biểu thị tên của tệp, có thể bao gồm cả đường dẫn. `<Kiểu_mở>` thường sử dụng chỉ thị `ios::out` nếu muốn dữ liệu ghi vào tệp sẽ đè lên dữ liệu cũ của tệp, ngược lại, chỉ thị `ios::app` sẽ cho phép ghi bổ sung dữ liệu mới vào cuối tệp.

Ví dụ ta khai báo một biến tệp `f` để ghi dữ liệu và sử dụng nó để mở tệp VANBAN.TXT trên ổ đĩa D, sử dụng chế độ ghi đè:

```
ofstream f;  
f.open("D:/VANBAN.TXT", ios::out);
```

Nếu tệp VANBAN.TXT chưa tồn tại trên ổ đĩa D, chương trình sẽ tạo ra một tệp như vậy để sẵn sàng cho việc ghi dữ liệu. Ngược lại, nếu tệp đã tồn tại, dữ liệu cũ sẽ bị ghi đè. Ta cũng có thể thực hiện gộp hai dòng lệnh trên thành một như sau:

```
ofstream f("D:/VANBAN.TXT", ios::out);
```

8.2. Mở tệp để đọc dữ liệu

```
ifstream <Tên_bien>;  
<Tên_bien>.open(<Tên_tệp>, ios::in);
```

Trong đó, `<Tên_bien>` do ta đặt và được gọi là một dòng nhập từ tệp. `<Tên_tệp>` là một chuỗi ký tự biểu thị tên của tệp, có thể bao gồm cả đường dẫn. Ví dụ để khai báo một biến tệp `f` và mở tệp “VANBAN.TXT” trên ổ đĩa D để đọc dữ liệu:

```
ifstream f;  
f.open("D:/VANBAN.TXT", ios::in);
```

Ta cũng có thể thực hiện gộp hai dòng lệnh trên thành một như sau:

```
ifstream f("D:/VANBAN.TXT", ios::in);
```

✓ Để sử dụng một biến tệp mà vừa có thể ghi dữ liệu vào tệp, vừa có thể đọc dữ liệu từ tệp, ta có thể sử dụng kiểu `fstream` thay cho `ofstream` hoặc `ifstream`. Ví dụ :

```
fstream f("D:/VANBAN.TXT", ios::in); hoặc  
fstream f("D:/VANBAN.TXT", ios::out);
```

8.3. Ghi dữ liệu vào tệp

C++ sử dụng `cout` như một dòng xuất dữ liệu gắn với màn hình. Nói cách khác, `cout` là dòng xuất gắn với thiết bị xuất là màn hình. Để xuất dữ liệu vào tệp, ta chỉ đơn giản là khai báo một dòng xuất tệp, mở một tệp và dùng nó thay cho `cout`. Ví dụ ta khai báo dòng xuất tệp có tên `fout` và mở tệp “D:/vidu.txt” như sau:

```
ofstream fout("D:/vidu.txt", ios::out);
```

Sau đó ta có thể sử dụng `fout` như đối tượng `cout`:

```
fout<< "Hello"<< endl;
```

✓ Tương tự `cout`, với dòng xuất tệp `fout` ta cũng có thể sử dụng `endl` để xuống dòng, sử dụng `setw(n)`, `setprecision(n)`... để định dạng dữ liệu xuất... Kết thúc việc xuất dữ liệu, ta cần đóng tệp lại bằng phương thức `close()`, ví dụ: `fout.close();`.

❶ **Ví dụ 8.1:** Nhập vào một mảng `a` gồm `n` phần tử nguyên, xuất dữ liệu trong mảng `a` vào tệp “mang.txt” với định dạng:

- Dòng đầu tiên: Ghi số `n`.

- Dòng thứ 2: Ghi các phần tử của mảng, mỗi phần tử cách nhau bởi dấu cách.

Hàm `Ghidulieu()` sau sẽ thực hiện khai báo một dòng xuất tệp `fout` với tên tệp đang chứa trong đối số `tentep` để ghi dữ liệu theo chế độ ghi đè (dòng 3), sau đó xuất dữ liệu vào tệp (dòng 4, 5, 6) và không quên đóng tệp (dòng 7).

```

1 void Ghidulieu(int*a, int n, char* tentep)
2 {
3     ofstream fout(tentep, ios::out);
4     fout<<n<<endl;
5     for(int i=0; i<n; i++)
6         fout<<a[i]<<" ";
7     fout.close();
8 }
```

8.4. Đọc dữ liệu từ tệp

Việc đọc dữ liệu từ tệp sẽ có chút phức tạp hơn là việc ghi dữ liệu vào tệp. Ta có hai kiểu đọc dữ liệu: Đọc từng dòng và đọc từng cụm. Hơn nữa, ta cần sử dụng một số thao tác hỗ trợ cho việc đọc này.

- **Đọc từng dòng trong tệp**

Lệnh `getline` sau đây cho phép đọc từng dòng dữ liệu trong tệp đang được mở bởi dòng đọc tệp `f` ra một biến kiểu chuỗi ký tự `str`:

```
f.getline(str, <p>);
```

Trong đó, `<p>` là một số nguyên đủ lớn, là số ký tự tối đa sẽ đọc. Để đọc toàn bộ nội dung tệp, ta sử dụng phương thức `eof()` để kiểm tra xem đã đọc hết tệp hay chưa `f.eof()` sẽ trả về giá trị `true` nếu ta đã đọc hết dữ liệu trong tệp `f` (con trỏ tệp trỏ tới cuối tệp), ngược lại nó trả về giá trị `false`. Hàm `ReadLineFile()` sau đây sẽ đọc toàn bộ nội dung tệp với tên tệp chứa trong đối số `tentep` và in chúng ra màn hình:

```

1 void ReadLineFile(char* tentep)
2 {
3     ifstream f(tentep, ios::in);
4     char str[255];
5     while (!f.eof())
6     {
7         f.getline(str, 255);
8         cout<<str<<endl;
```

8	}
9	f.close();
10	}
11	

- **Đọc từng cụm trong tệp**

Với dữ liệu kiểu số (ví dụ một mảng số), việc đọc từng dòng dữ liệu trong tệp sẽ có thể gây khó khăn khi phải xử lý tách các phần tử trong dòng đó. Trong trường hợp này, ta sử dụng kỹ thuật đọc từng cụm.

Toán tử nhập (`>>`) sẽ thực hiện việc đọc từng cụm dữ liệu trong tệp (mỗi lần đọc một cụm), với “cụm” được định nghĩa là một dãy ký tự liên tiếp, dài nhất, không chứa dấu cách. Câu lệnh sau sẽ đọc 1 cụm ký tự trong tệp đang được mở bởi dòng đọc tệp `f` và chứa vào một biến `a`:

`f>>a;`

Ta cũng có thể đọc nhiều cụm liên tiếp bằng toán tử nhập. Câu lệnh sau đọc ba cụm liên tiếp của tệp `f` và chứa vào ba biến nguyên `a, b, c`:

`f>>a>>b>>c;`

☞ Thông thường, biến a, b, c có kiểu chuỗi ký tự. Tuy nhiên, tùy theo dữ liệu có kiểu gì, ta có thể sử dụng biến a, b, c có kiểu đó mà không cần thực hiện thao tác chuyển đổi kiểu.

❷ Ví dụ 8.2. Cho một tệp “`mang.txt`” trong ổ đĩa D. Dòng đầu tiên chứa số phần tử của một mảng số nguyên. Dòng tiếp theo chứa các phần tử của mảng với mỗi phần tử cách nhau bởi dấu cách, ví dụ như sau:

5
12 24 28 11 17

Hãy đọc dữ liệu của tệp lên một biến mảng `a`, in mảng đọc được ra màn hình.

Hàm `ReadFile` sau trả về giá trị `true` nếu đọc tệp thành công, ngược lại trả về `false`. Hàm sử dụng phương thức `good()` để kiểm tra việc tệp có tồn tại hay không. Nếu tệp tồn tại, `good()` trả về giá trị `true`, ngược lại, nó trả về giá trị `false`.

```

1  bool ReadFile(char* tentep, int*&a, int&n)
2  {
3      ifstream f(tentep, ios::in);
4      if(f.good())
5      {
6          f>>n;
7          a = new int[n];
8          for(int i=0; i<n; i++)
9              f>>a[i];
10         f.close();
11         return true;
12     }
13     else
14     {
15         cout<<"File not found !";
16         return false;
17     }
18 }
19 int main()
20 {
21     int*a; int n;
22     bool fExists = ReadFile("mang.txt",a, n);
23     if(fExists)
24     {
25         cout<<n<<endl;
26         for(int i=0; i<n; i++)
27             cout<<a[i]<<" ";
28     }
29 }
```

B. Các bài tập tự giải

Bài F1. Viết chương trình cho phép tạo một tệp tin firstfile.txt với nội dung như dưới đây. Đọc và hiển thị nội dung của tệp tin firstfile.txt lên màn hình.

Problem name: exp1 Maximize obj: x1 + 2 x2 + 3 x3 + x4 Subject To c1: x2 - 3.5 x4 = 0 Bounds 0 <= x1 <= 40 General x4 End
--

Bài F2. Viết chương trình nhập vào từ bàn phím một mảng a gồm n phần tử nguyên. Xuất dữ liệu của mảng a vào tệp theo định dạng:

- Dòng đầu tiên là kích thước của a (giá trị biến n).
- Dòng thứ 2 là các phần tử của a , mỗi phần tử cách nhau bởi dấu cách.
- Dòng thứ 3 là giá trị lớn nhất của mảng a .

Đọc toàn bộ nội dung của tệp dữ liệu và in lên màn hình.

Bài F3. Cho một tệp văn bản chứa dữ liệu là một ma trận số thực với định dạng:

- Dòng đầu tiên là số dòng và số cột của ma trận, cách nhau bởi dấu cách.
- n dòng tiếp theo, mỗi dòng là m phần tử là dữ liệu từng dòng của ma trận, các phần tử cách nhau bởi dấu cách.

Hãy đọc dữ liệu của tệp chứa vào một mảng hai chiều $a(n \times m)$. In mảng đọc được ra màn hình.

Bài F4. Viết chương trình cho phép tạo ra hai tệp văn bản `file1.txt` và `file2.txt` với nội dung như hình dưới đây. Ghép hai tệp để thu được tệp `file3.txt`. Đọc và hiển thị nội dung tệp tin `file3.txt` lên màn hình.

file1.txt file2.txt

$\begin{matrix} 7 & 2 \\ 1 & 2 & 3 & 1 \\ 2 & 2 & 4 & 1 \\ 3 & 3 & 5 & 1 \end{matrix}$	$\begin{matrix} 4 & 1 & 1 & 2 \\ 5 & 2 & 2 & 2 \\ 6 & 3 & 3 & 2 \\ 7 & 4 & 4 & 2 \end{matrix}$
--	--

Bài F5. Nhập một mảng a gồm n phần tử thực từ bàn phím. Tạo một tệp tin `dathuc.txt` với nội dung như sau:

- Dòng thứ nhất là số nguyên n kích thước của mảng a .
- Dòng thứ 2 là n phần tử của mảng a , mỗi phần tử cách nhau bởi dấu cách.
- Dòng thứ 3 là đa thức dạng:

$$a[0] \ x_0 + a[1] \ x_1 + \dots + a[n-1] \ x_{n-1}.$$

Đọc và in nội dung của tệp tin dathuc.txt ra màn hình.

Bài F6. Tạo một tệp tin với tên là file.txt với nội dung:

- Dòng đầu tiên là một số nguyên n là kích thước của một mảng a .
- Dòng thứ 2 là n phần tử nguyên là các phần tử của mảng a .

Đọc dữ liệu từ tệp file.txt lên các biến n và a . Sắp xếp mảng a tăng dần và ghi dữ liệu vào tệp sorted_file.txt.

Bài F7. Hãy tạo một tệp có nội dung như hình minh họa dưới đây:

```
1 Problem name: exp2
2 Minimize
3 obj:      3 x1 + 2 x2 + 3 x3 + x4
4 Subject To
5 c1: x2 - 3.5 x4 = 0
6 Bounds
7 0 <= x1 <= 40
8 General
9      x4
End
```

Hãy đọc các hệ số của đa thức trong dòng 3 và chứa trong mảng một mảng a có n phần tử. In mảng đọc được từ tệp ra màn hình.

TÀI LIỆU THAM KHẢO

- [1]. Programming Notes for Professionals books. An unofficial free book of GoalKicker.com; 2021.
- [2]. Richard Grimes: Beginning C++ Programming. Packt Publishing (April 24, 2017).
- [3]. Bjarne Stroustrup: Programming: Principles and Practice Using C++. Addison-Wesley Professional; 2nd edition (May 15, 2014).
- [4]. Marc Gregoire: Professional C++, 3rd Edition. Addison-Wesley Professional; September 2014.
- [5]. Stephen Prata: C++ Primer Plus (6th Edition). Addison-Wesley Professional; October 28, 2011.
- [6]. Bruce Eckel: Thinking in C++. MindView, Inc; Volume 1, 2nd Edition; January 13, 2000.
- [7]. Mike Hendrickson, Andrew Koenig, Barbara Moo: Accelerated C++: Practical Programming by Example. Addison-Wesley Professional; 1st edition (August 14, 2000).