```
/*khai báo lớp vector*/
class vector{
        static int n;   //số chiều của vector
        float *v;     //vùng nhớ chứa các toạ độ
  public:
        vector();
        vector(float *);
        vector(vector &);//hàm thiết lập sao chép
        ~vector();
        void display();
        static int & Size() {return n;}
        friend vector prod(matrix &, vector &);
        friend class matrix;
  };
int vector::n = 0;
/*các hàm thành phần của lớp vector*/
vector::vector()
  {
  int i;
  v= new float [n];
  for(i=0;i<n;i++) {
    cout<<"Toa do thu "<<i+1<<" : ";
    cin>>v[i];
    }
  }
vector::vector(float *a) {
  for(int i =0; i<n; i++)
    v[i]=a[i];
  }
vector::vector(vector &b)
  {
  int i;
```

```
  for(i=0;i<n;i++)
    v[i] = b.v[i];
  }
vector::~vector(){
  delete v;
  }
void vector::display()   //hiển thị kết quả
{
  for(int i=0;i<n;i++)
    cout <<v[i] <<" ";
  cout <<"\n";
}
/*khai báo lớp matrix*/
class matrix{
       static int n;   //số chiều của vector
       vector *m;     //vùng nhớ chứa các toạ độ
  public:
       matrix();
       ~matrix();
       void display();
       static int &Size() {return n;}
       friend vector prod(matrix &, vector &);
  };
int matrix::n =0;
/*hàm thành phần của lớp matrix*/
matrix::matrix(){
  int i;
  m= new vector [n];
  }
matrix::~matrix() {
  delete m;
  }
```

```
void matrix::display()   //hiển thị kết quả
{
for (int i=0; i<n; i++)
  m[i].display();
}
/*hàm prod*/
vector prod(matrix &m,vector &v) {
  float *a = new float [vector::Size()];
  int i,j;
  for (i=0; i<matrix::Size(); i++) {
    a[i]=0;
    for(j=0; j<vector::Size(); j++)
      a[i]+=m.m[i].v[j]*v.v[j];
    }
  return vector(a);
  }
void main()
  {
  clrscr();
  int size;
  cout<<"Kich thuoc cua vector "; cin>>size;
  vector::Size() = size;
  matrix::Size() = size;
  cout<<"Tao mot vector \n";
  vector v;
  cout<<" v= \n";
  v.display();
  cout<<"Tao mot ma tran \n";
  matrix m;
  cout<<" m = \n";
  m.display();
```

```
  cout<<"Tich m*v \n";
  vector u = prod(m,v);
  u.display();
  getch();
}
```

```
Kich thuoc cua vector 3
Tao mot vector
Toa do thu 1 : 1
Toa do thu 2 : 1
Toa do thu 3 : 1
 v=
1 1 1
Tao mot ma tran
Toa do thu 1 : 2
Toa do thu 2 : 3
Toa do thu 3 : 2
Toa do thu 1 : 1
Toa do thu 2 : 2
Toa do thu 3 : 3
Toa do thu 1 : 2
Toa do thu 2 : 3
Toa do thu 3 : 2
 m =
2 3 2
1 2 3
2 3 2
Tich m*v
7 6 7
```

## 9. TÓM TẮT

### 9.1 Ghi nhớ

Trong C++, tên cấu trúc là một kiểu dữ liệu không cần phải kèm theo từ khoá **struct**.

Lớp cho phép người lập trình mô tả các đối tượng thực tế với các thuộc tính và hành vi. Trong C++ thường sử dụng từ khoá **class** để khai báo một lớp. Tên lớp là một kiểu dữ liệu dùng khi khai báo các đối tượng thuộc lớp (các thể hiện cụ thể của lớp).

Thuộc tính của đối tượng trong một lớp được mô tả dưới dạng các biến thể hiện. Các hành vi là các hàm thành phần bên trong lớp.

Có hai cách định nghĩa các hàm thành phần của một lớp; khi định nghĩa hàm thành phần bên ngoài khai báo lớp phải đặt trước tên hàm thành phần tên của lớp và toán tử "::" để phân biệt với các hàm tự do cùng tên. Chỉ nên định nghĩa hàm thành phần bên trong lớp khi nó không quá phức tạp để cho chương trình dễ đọc.

Có thể khai báo và sử dụng các con trỏ đối tượng, tham chiếu đối tượng.

Hai từ khoá **public** và **private** dùng để chỉ định thuộc tính truy nhập cho các thành phần (dữ liệu/hàm) khai báo bên trong lớp.

Thành phần bên trong lớp được khai báo **public** có thể truy nhập từ mọi hàm khai báo một đối tượng thuộc lớp đó.

Thành phần **private** trong một đối tượng chỉ có thể truy nhập được bởi các hàm thành phần của đối tượng hoặc các hàm thành phần của lớp dùng để tạo đối tượng (ở đây tính cả trường hợp đối tượng là tham số của hàm thành phần).

Hai hàm thành phần đặc biệt của một lớp gọi là hàm thiết lập và hàm huỷ bỏ. Hàm thiết lập được gọi tự động (ngầm định) mỗi khi một đối tượng được tạo ra và hàm huỷ bỏ được gọi tự động khi đối tượng hết thời gian sử dụng.

Hàm thiết lập có thuộc tính **public**, cùng tên với tên lớp nhưng không có giá trị trả về.

Một lớp có ít nhất hai hàm thiết lập: hàm thiết lập sao chép ngầm định và hàm thiết lập do người lập trình thiết lập (nếu không được mô tả tường minh thì đó là hàm thiết lập ngầm định).

Hàm huỷ bỏ cũng có thuộc tính **public**, không tham số, không giá trị trả về và có tên bắt đầu bởi ~ theo sau là tên của lớp.

Bên trong phạm vi lớp (định nghĩa của các hàm thành phần), các thành phần của lớp được gọi theo tên. Trường hợp có một đối tượng toàn cục cùng tên, muốn xác định đối tượng ấy phải sử dụng toán tử "::"

Lớp có thể chứa các thành phần dữ liệu là các đối tượng có kiểu lớp khác. Các đối tượng này phải được khởi tạo trước đối tượng tương ứng của lớp bao.

Mỗi đối tượng có một con trỏ chỉ đến bản thân nó, ta gọi đó là con trỏ **this**. Con trỏ này có thể được sử dụng tường minh hoặc ngầm định để tham xác định các thành phần bên trong đối tượng. Thông thường người ta sử dụng this dưới dạng ngầm định.

Toán tử **new** tự động tạo ra một đối tượng với kích thước thích hợp và trả về con trỏ có kiểu lớp. Để giải phóng vùng nhớ cấp phát cho đối tượng này sử dụng toán tử **delete**.

Thành phần dữ liệu tĩnh biểu thị các thông tin dùng chung trong tất cả các đối tượng thuộc lớp. Khai báo của thành phần tĩnh bắt đầu bằng từ khoá **static.**

Có thể truy nhập tới các thành phần tĩnh thông qua các đối tượng kiểu lớp hoặc bằng tên lớp nhờ sử dụng toán tử phạm vi.

Hàm thành phần có thể được khai báo là tĩnh nếu nó chỉ truy nhập đến các thành phần dữ liệu tĩnh.

Hàm bạn của một lớp là hàm không thuộc lớp nhưng có quyền truy nhập tới các thành phần private của lớp.

Khai báo bạn bè có thể đặt bất kỳ nơi nào trong khai báo lớp.

### 9.2 Các lỗi thường gặp

Quên dấu ";" ở cuối khai báo lớp.

Khởi tạo giá trị cho các thành phần dữ liệu trong khai báo lớp.

Định nghĩa chồng một hàm thành phần bằng một hàm không thuộc lớp.

Truy nhập đến các thành phần riêng của lớp từ bên ngoài phạm vi lớp

Khai báo giá trị trả về cho hàm thiết lập và hàm huỷ bỏ.

Khai báo hàm huỷ bỏ có tham số, định nghĩa chồng hàm huỷ bỏ.

Gọi tường minh hàm thiết lập và hàm huỷ bỏ.

Gọi các hàm thành phần bên trong hàm thiết lập

Định nghĩa một hàm thành phần **const** thay đổi các thành phần dữ liệu của một đối tượng.

Định nghĩa một hàm thành phần const gọi tới một hàm thành phần không phải **const**.

Gọi các hàm thành phần không phải const từ các đối tượng **const**.

Thay đổi nội dung một đối tượng **const**.

Nhầm lẫn giữa **new** và **delete** với `malloc` và `free`.

Sử dụng this bên trong các hàm thành phần tĩnh.

**9.3 Một số thói quen lập trình tốt**

Nhóm tất cả các thành phần có cùng thuộc tính truy nhập ở một nơi trong khai báo lớp, nhờ vậy mỗi từ khoá mô tả truy nhập chỉ được xuất hiện một lần. Khai báo lớp vì vậy dễ đọc hơn. Theo kinh nghiệm, để các thành phần **private** trước tiên rồi đến các thành phần **protected**, cuối cùng là các thành phần **public**.

Định nghĩa tất cả các hàm thành phần bên ngoài khai báo lớp. Điều này nhằm phân biệt giữa hai phần giao diện và phần cài đặt của lớp.

Sử dụng các chỉ thị tiền xử lý `#ifndef`, `#define`, `#endif` để cho các tập tin tiêu đề chỉ xuất hiện một lần bên trong các chương trình nguồn.

Phải định nghĩa các hàm thiết lập để đảm bảo rằng các đối tượng đều được khởi tạo nội dung một cách đúng đắn.

Khai báo là **const** tất cả các hàm thành phần chỉ để sử dụng với các đối tượng **const**.

Nên sử dụng **new** và **delete** thay vì `malloc` và `free`.

## 10. BÀI TẬP

**Bài tập 3.1**

So sánh ý nghĩa của struct và class trong C++

**Bài tập 3.2**

Tạo một lớp gọi là Complex để thực hiện các thao tác số học với các số phức. Viết một chương trình để kiểm tra lớp này.

Số phức có dạng

<Phần thực> + <Phần ảo>*j

Sử dụng các biến thực để biểu diễn các thành phần dữ liệu riêng của lớp. Cung cấp một hàm thiết lập để tạo đối tượng. Hàm thiết lập sử dụng các tham số có giá trị ngầm định. Ngoài ra còn có các hàm thành phần **public** để thực hiện các công việc sau:

+ Cộng hai số phức: các phần thực được cộng với nhau và các phần ảo được cộng với nhau.
+ Trừ hai số phức: Phần thực của số phức thứ hai được trừ cho phần thực của số phức thứ nhất. Tương tự cho phần ảo.
+ In số phức ra màn hình dưới dạng (a, b) trong đó a là phần thực và b là phần ảo.

**Bài tập 3.3**

Tạo một lớp gọi là `PS` để thực hiện các thao tác số học với phân số. Viết chương trình để kiểm tra lớp vừa tạo ra.

Sử dụng các biến nguyên để biểu diễn các thành phần dữ liệu của lớp-tử số và mẫu số. Viết định nghĩa hàm thiết lập để tạo đối tượng sao cho phần ảo phải là số nguyên dương. Ngoài ra còn có các hàm thành phần khác thực hiện các công việc cụ thể:

+ Cộng hai phân số. Kết quả phải được tối giản.
+ Trừ hai phân số. Kết quả phải được tối giản.
+ Nhân hai phân số. Kết quả dưới dạng tối giản.
+ Chia hai phân số. Kết quả dưới dạng tối giản.
+ In ra màn hình phân số dưới dạng a/b trong đó a là tử số, còn b là mẫu số.
+ In phân số dưới dạng số thập phân.

**Bài tập 3.4**

Khai báo, định nghĩa và sử dụng lớp `time` mô tả các thông tin về giờ, phút và giây với các yêu cầu như sau:

Tạo tập tin tiêu đề TIME.H chứa khai báo của lớp time với các thành phần dữ liệu mô tả giờ, phút và giây: `hour, minute, second`.

Trong lớp `time` khai báo :

+ một hàm thiết lập ngầm định, dùng để gán cho các thành phần dữ liệu giá trị 0.
+ hàm thành phần `set(int, int, int)` với ba tham số tương ứng mang giá trị của ba thành phần dữ liệu.
+ hàm hiển thị trong đó giờ được hiển thị với giá trị 0 tới 24.
+ hàm hiển thị chuẩn có phân biệt giờ trước và sau buổi trưa.

Tạo tập tin chương trình `TIME.CPP` chứa định nghĩa của các hàm thành phần trong lớp `time`, và chương trình minh hoạ cách sử dụng lớp `time`.

**Bài tập 3.5**

Tương tự như bài 3.4 nhưng ở đây hàm thiết lập có ba tham số có giá trị ngầm định bằng 0.

**Bài tập 3.6**

Vẫn dựa trên lớp `time`, nhưng ở đây ta bổ sung thêm các hàm thành phần để thiết lập riêng rẽ giờ, phút, giây:

+ `void setHour(int)`
+ `void setMiniute(int)`
+ `void setSecond(int)`

Đồng thời có các hàm để lấy từng giá trị đó của từng đối tượng:

+ `int getHour()`
+ `int getMiniute();`
+ `int getSecond()`

**Bài tập 3.7**

Thêm một hàm thành phần `tick()` vào lớp `time` để tăng thời gian trong một đối tượng `time` mỗi lần một giây. Lưu ý các trường hợp, tăng sang phút tiếp theo, tăng sang giờ tiếp theo, tăng sang ngày tiếp theo.

**Bài tập 3.8**

Khai báo lớp `date` mô tả thông tin về ngày tháng năm: `month, day, year`.

Lớp date có hai hàm thành phần:

+ hàm thiết lập với ba tham số có giá trị ngầm định.
+ hàm `print()` in thông tin về ngày tháng dưới dạng quen thuộc `mm-dd-yy`.

Viết chương trình kiểm ta việc sử dụng phép gán cho các đối tượng thuộc lớp date.

**Bài tập 3.9**

Dựa trên lớp `date` của bài 3.8 người ta thực hiện một số thay đổi để kiểm soát lỗi trên giá trị các tham số của hàm thiết lập. Đồng thời bổ sung thêm hàm thành phần `nextDay()` để tăng `date` từng ngày một.

**Bài tập 3.10**

Kết hợp lớp `time` trong bài 3.7 và lớp `date` trong bài 3.9 để tạo nên một lớp `date_time` mô tả đồng thời thông tin về ngày, giờ. Thay đổi hàm thành phần `tick()` để gọi tới hàm tăng ngày mỗi khi cần thiết. Thêm các hàm hiển thị thông tin về ngày giờ. Hoàn thiện chương trình để kiểm tra hoạt động của lớp.

**Bài tập 3.11**

Viết chương trình khai báo lớp mô tả hoạt động của một ngăn xếp hoặc hàng đợi chứa các số nguyên.

*Chương 4*

# ĐỊNH NGHĨA TOÁN TỬ TRÊN LỚP

## (class operators)

Mục đích chương này :

1. Cách định nghĩa các phép toán cho kiểu dữ liệu lớp và cấu trúc
2. Các toán tử chuyển kiểu áp dụng cho kiểu dữ liệu lớp

## 1. GIỚI THIỆU CHUNG

Thực ra, vấn đề định nghĩa chồng toán tử đã từng có trong C, ví dụ trong biểu thức:

```
a + b
```

ký hiệu + tuỳ theo kiểu của a và b có thể biểu thị:

1. phép cộng hai số nguyên,
2. phép cộng hai số thực độ chính xác đơn (**float**)
3. phép cộng hai số thực chính xác đôi (**double**)
4. phép cộng một số nguyên vào một con trỏ.

Trong C++, có thể định nghĩa chồng đối với hầu hết các phép toán (một ngôi hoặc hai ngôi) trên các lớp, nghĩa là một trong số các toán hạng tham gia phép toán là các đối tượng. Đây là một khả năng mạnh vì nó cho phép xây dựng trên các lớp các toán tử cần thiết, làm cho chương trình được viết ngắn gọn dễ đọc hơn và có ý nghĩa hơn. Chẳng hạn, khi định nghĩa một lớp `complex` để biểu diễn các số phức, có thể viết trong C++: `a+b, a-b, a*b, a/b` với `a,b` là các đối tượng `complex`.

Để có được điều này, ta định nghĩa chồng các phép toán `+, -, *` và `/` bằng cách định nghĩa hoạt động của từng phép toán giống như định nghĩa một hàm, chỉ khác là đây là hàm toán tử (operator function). Hàm toán tử có tên được ghép bởi từ khoá **operator** và ký hiệu của phép toán tương ứng. Bảng 4.1 đưa ra một số ví dụ về tên hàm toán tử.

Hàm toán tử có thể dùng như là một hàm thành phần của một lớp hoặc là hàm tự do; khi đó hàm toán tử phải được khai báo là bạn của các lớp có các đối tượng mà hàm thao tác.