CHƯƠNG 2: LỚP VÀ ĐỐI TƯỢNG

Giảng viên: Trần Thị Ngân

Email: ngantt@tlu.edu.vn

NỘI DUNG

2.1 Định nghĩa lớp

2.2 Hàm tạo và hàm hủy

2.3 Thành phần tĩnh

2.4 Hàm bạn và lớp bạn

ĐỊNH NGHĨA LỚP

- Định nghĩa lớp là tạo ra một kiểu dữ liệu trừu tượng mới để mô tả các đặc trưng của đối tượng trong thực tế.
- **Định nghĩa lớp gồm hai bước**:
  - *Khai báo lớp*: khai báo các dữ liệu và hàm thành phần (phương thức) tạo nên lớp
  - *Định nghĩa lớp*: định nghĩa cụ thể các hàm thành phần của lớp

KHAI BÁO LỚP

```
class   tên-lớp-được-định-nghĩa
{
  private:    //khai báo vùng che giấu riêng của lớp
              khai báo các dữ liệu
              khai báo các phương thức
  protected:        //khai báo vùng bảo vệ của lớp
              khai báo các dữ liệu
              khai báo các phương thức
  public:     //khai báo vùng dùng chung của lớp
              khai báo các dữ liệu
              khai báo các phương thức
};
```

KHAI BÁO LỚP

- Các dữ liệu và hàm thành phần của lớp được mô tả với 3 mức khác nhau:
  - **private**: sở hữu riêng, dùng để khai báo các thành viên là riêng của chỉ lớp đó và không thể truy cập từ các lớp khác ⇔ được dùng với mục đích che giấu thông tin của lớp
  - **protected**: chế độ bảo vệ, khai báo các thành phần của riêng lớp đó và các lớp kế thừa nó có thể truy cập được còn các lớp khác không truy cập được
  - **public**: dùng chung, dùng khai báo các thành phần có thể truy cập từ mọi nơi ⇔ truy cập được từ bên ngoài lớp
- Chú ý: Nếu không có từ khóa **private** thì mặc định hiểu là thuộc vùng private

KHAI BÁO LỚP

- **Phương thức**: xác định các phương thức hay tác vụ thực hiện xử lý trên dữ liệu của lớp, được khai báo như khai báo hàm

  <kiểu-trả-lại> Tên-phương-thức(ds đối số);

- Ví dụ:

```
class date{
....
public:
        int get_ngay( );
        int get_thang( );
        int get_nam( );
        int sosanh( date &d );
};
```

KHAI BÁO LỚP

- **Dữ liệu**: xác định các thành phần dữ liệu để mô tả lớp ⇔ gọi là trường, thuộc tính để xác định đối tượng
  - dữ liệu được mô tả thông qua tên và kiểu xác định ⇔ ***khai báo biến dữ liệu***
  - kiểu: chấp nhận được khai báo các kiểu gồm các kiểu cơ bản(int, float,…) hay con trỏ đến đối tượng lớp
  - ví dụ:

```
class date{
        private:
                int ngay, thang, nam;
….
};
```

KHAI BÁO LỚP

- ví dụ:

```
class date{
        private:
                int ngay;
                int thang;
                int nam;
         public:
                int get_ngay( );
                int get_thang( );
                int get_nam( );
                int sosanh( date &d );
};
```

Ví dụ

Định nghĩa lớp về nhân viên bao gồm dữ liệu thành viên: *mã nhân viên, họ tên, năm sinh, chuyên môn, bậc lương.*

Hàm thành viên: *nhap();*

xuat();

```
1   #include <iostream>
2   using namespace std;
3   class nhanvien
4   {
5       public:
6               void nhap();
7               void xuat();
8       private:
9           string hoten, manv, chuyenmon;
10          float bacluong;
11          int namsinh;
12  };
```

Định nghĩa hàm thành phần

- Định nghĩa hàm thành phần là thao tác định nghĩa cụ thể các hàm – thao tác của lớp được thực hiện như thế nào?
- Các hàm thành phần của lớp được định nghĩa theo hai cách:
  - *Định nghĩa từ bên ngoài khai báo lớp*
  - *Định nghĩa bên trong khai báo lớp*

Định nghĩa bên ngoài khai báo lớp

- Là thực hiện đinh nghĩa bên ngoài vùng khai báo **class** ⇔ định nghĩa sau dấu " ; " của khai báo ***class***
- Định nghĩa bên ngoài phải có thêm tiếp đầu ngữ chỉ ra **tên lớp** chứa hàm cần định nghĩa.
- Dạng tổng quát:

Kiểu-trả-lại **tên-lớp** :: tên-hàm (ds đối số)

{

Định nghĩa nội dung hàm

}

Định nghĩa trong lớp

- Định nghĩa ngay tại vị trí các hàm thành phần trong phần mô tả và khai báo lớp
- Thường dùng với các hàm đơn giản và ít dòng lệnh

Ví dụ

```
#include “iostream.h”
class date
{
    int ngay,thang,nam;
public:
    void nhap()
    {
      cout<<“Nhap ngay thang nam:”;
      cin>>ngay>>thang>>nam;
    }
    void in()
    {
      cout<<“Ngay:”<<ngay<<“/”<<thang<<“/”<<nam;
    }
};
```

Ví dụ

Định nghĩa các hàm thành viên: *nhap(), xuat() cho lớp nhân viên*

Kiểu-trả-lại **tên-lớp** :: tên-hàm (ds đối số)

{

Định nghĩa nội dung hàm

}

```
1  #include <iostream>
2  using namespace std;
3  class nhanvien
4  {
5      public:
6          void nhap();
7          void xuat();
8      private:
9          string hoten, manv, chuyenmon;
10         float bacluong;
11         int namsinh;
12 };
```

Ví dụ

Định nghĩa các hàm thành viên: *nhap(), xuat() cho lớp nhân viên*

```
1   #include <iostream>
2   using namespace std;
3   class nhanvien
4   {
5       public:
6               void nhap();
7               void xuat();
8       private:
9           string hoten, manv, chuyenmon;
10          float bacluong;
11          int namsinh;
12  };
```

```
13  void nhanvien::nhap()
14  {
15      cout<<"Nhap ma nhan vien: "; getline(cin,  manv);
16      cout<<"Nhap ho ten nhan vien: "; getline(cin,  hoten);
17      cout<<"Nhap chuyen mon: "; getline(cin,  chuyenmon);
18      cout<<"Nhap bac luong: "; cin>> bacluong;
19      cout<<"Nhap nam sinh: "; cin>> namsinh;
20      cin.ignore();
21  }
22  void nhanvien:: xuat()
23  {
24      cout<<"Ma nhan vien: "<< manv;
25      cout<<"\nHo ten nhan vien: "<< hoten;
26      cout<<"\nhuyen mon: "<<  chuyenmon;
27      cout<<"\nBac luong: "<< bacluong;
28      cout<<"\nNam sinh: "<< namsinh;
29  }
```

Tạo lập đối tượng

- Để sử dụng lớp đối tượng phải tạo ra các đối tượng ⇔ khai báo biến kiểu class đã được định nghĩa
- Ví dụ:

  date ngaysinh;
- Để truy cập đến từng thành phần đối tượng thông qua tên biến kiểu class đã được khai báo và toán tử dấu chấm (.) theo dạng sau:

  Tên-biến-đối-tượng.*dữ-liệu-thành-phần***;**

  Tên-biến-đối-tượng.*hàm-thành-phần(ds đối số);*
- Ví dụ: **ngaysinh**.*ngay*

  ngaysinh.*thang*

  ngaysinh.*nhap( );*

  ts = ngaysinh.*get_thang***( );**

Truy cập các thành viên của đối tượng

- Đối với các thành viên dữ liệu của lớp đối tượng đang định nghĩa:
  - Gán dữ liệu cho đối tượng: thực hiện thay đổi từng thành phần dữ liệu của đối tượng
  - Thay đổi các dữ liệu private của lớp: chỉ các phương thức của lớp mới có quyền thay đổi giá trị các thành phần riêng của lớp
  - Thay đổi các dữ liệu trong thành phần public: có thể được thực hiện bằng bất cứ thành phần nào của lớp

Ví dụ

Cho định nghĩa lớp nhân viên.

Xây dựng hàm main() nhập thông tin cho 2 nhân viên (không trùng mã). Kiểm tra xem 2 người có cùng chuyên môn hay không.

```
#include <iostream>
using namespace std;
class nhanvien
{
    public:
        void nhap();
        void xuat();
        string hoten, manv, chuyenmon;
        float bacluong;
        int namsinh;
};
void nhanvien::nhap()
{
    cout<<"Nhap ma nhan vien: "; getline(cin,  manv);
    cout<<"Nhap ho ten nhan vien: "; getline(cin,  hoten);
    cout<<"Nhap chuyen mon: "; getline(cin,  chuyenmon);
    cout<<"Nhap bac luong: "; cin>> bacluong;
    cout<<"Nhap nam sinh: "; cin>> namsinh;
    cin.ignore();
}
void nhanvien:: xuat()
{
    cout<<"Ma nhan vien: "<< manv;
    cout<<"\nHo ten nhan vien: "<< hoten;
    cout<<"\nhuyen mon: "<<  chuyenmon;
    cout<<"\nBac luong: "<< bacluong;
    cout<<"\nNam sinh: "<< namsinh;
}
```

Con trỏ this

- Mỗi một lớp trong nó luôn có một con trỏ mặc định là con trỏ **this**, con trỏ để mô tả chính đối tượng lớp đang định nghĩa.
- **Các hàm thành phần của lớp luôn có tham số đầu tiên là con trỏ this.**
- Ví dụ:

```
class PS
{
 public:
      int  ts, ms;
      void  nhap();              // void nhap( PS *this );
      PS  cong( PS b );      //PS  cong( PS *this , PS b );
};
```

Con trỏ this

- Truy cập đến thành phần con trỏ this

  this->tên_thành_phần ⇔ tên_thành_phần

Ví dụ:

```
void PS::nhap()
{
  cout<<“Tu:”; cin>>ts;
  cout<<“Mau:”; cin>>ms;
}
```

```
void PS::nhap()
{
  cout<<“Tu:”;
  cin>>this->ts;
  cout<<“Mau:”;
  cin>>this->ms;
}
```

NỘI DUNG:

- 2.1 Định nghĩa lớp
- **2.2 Hàm tạo và hàm hủy**
- 2.3 Thành phần tĩnh
- 2.4 Hàm bạn và lớp bạn

Constructor và Destructor

- là phương thức được định nghĩa đặc biệt được sử dụng khi khởi tạo một đối tượng mới hay loại bỏ đối tượng.
  - **Constructor :** cho phép tự động tạo ra một đối tượng mới khi khai báo
  - **Destructor** : tự động phá bỏ đối tượng khi không cần dùng đến

Constructor

- Là hàm thành phần đặc biệt của lớp, làm nhiệm vụ tạo lập các đối tượng theo yêu cầu.
- Tên Constructor được **đặt trùng với tên lớp** đang định nghĩa
- Ví dụ:

```
class sophuc
{
private:
    double x;                   //phan thuc
    double y;                   //phan ao
public:
    //dinh nghia ham toan tu cho phep tao doi tuong so phuc
    sophuc( void );
    sophuc( double a, double b );
    void nhap();
    void in();
};
```

Tính chất của Constructor

- Phải khai báo trong vùng public
- Tên của constructor phải trùng với tên của lớp
- Tự động được thực hiện khi khai báo đối tượng
- Một lớp có thể có nhiều hàm constructor
- Constructor không có kiểu trả về
- Không thể kế thừa, nhưng lớp dẫn xuất vẫn có thể gọi constructor của lớp cơ sở

Phân loại Constructor

- Constructor *mặc định* ⇔ Constructor không có tham số=> tạo đối tượng chỉ cần đặt tên

Ví dụ: **sophuc** *a,b,c;*

- Constructor *có tham số*: là constructor có các tham số để cho phép khởi tạo một bộ giá trị nào đó cho các dữ liệu thành phần của đối tượng, kiểu tham số cho constructor có thể là kiểu bất kỳ ngoại trừ **kiểu lớp đang định nghĩa.**

Ví dụ: **sophuc** tg(10,-2);;

sophuc *tong(2,-3), hieu(0,0);*

Destructor

- Là phương thức(method) thực hiện giải phóng bộ nhớ đã cấp cho đối tượng khi không cần sử dụng đối tượng.
- Mỗi lớp chỉ có duy nhất một Destructor
- Destructor là method rỗng không có tham số và là hàm thành phần của lớp có cùng tên lớp và thêm tiếp đầu ngữ "~"
- Ví dụ: **~date( );**

Ví dụ

```
#include <iostream>
class date
{
        int ngay,thang,nam;
public:
        date( ){ ngay=0; thang=0; nam=0; }
        date( int d, int m, int y ){ ngay = m; thang = n; nam = y }
        ~date( ) { }
        void  nhap( )
        {
                cout<<“Nhap ngay thang nam:”;
               cin>>ngay>>thang>>nam;
          }
        void  in( )
        {
                cout<<“Ngay:”<<ngay<<“/”<<thang<<“/”<<nam;
        }
 };
```

Ví dụ

```
void main()
{
   date  ngaysinh(01,04,2020), ngay_vcq;
   cout<<“ngay sinh cua ban la \n”;
   ngaysinh.in( );
   cout<<“Nhap ngay vao co quan \n”;
   ngay_vcq.nhap( );
   cout<<“Ban da nhap ngay vao co quan la \n”;
   ngay_vcq.in( );
  getch( );
}
```

Constructor sao chép

- Là Constructor cho phép tạo ra một đối tượng mới từ một đối tượng đã có nhưng hoàn toàn độc lập với đối tượng đã có đó.

VD: **PS d(2,3);**

PS u(d);

- Khi lớp không có thuộc tính kiểu con trỏ hoặc tham chiếu thì ta chỉ dùng hàm tạo sao chép mặc định.
- Khi lớp có các thuộc tính con trỏ hoặc tham chiếu thì phải định nghĩa hàm tạo sao chép mới.

Constructor sao chép

Tên_lớp (const **Tên_lớp** &đối_tượng)

{

// Các lệnh dùng các thuộc tính của đối tượng

//khởi gán cho các thuộc tính của đối tượng mới

……

}

* ví dụ:

```
class  PS
{
public:
        int ts, ms;
        PS( PS &x);  //Hàm tạo sao chép
        void nhap();
        void in();
};
//Định nghĩa hàm tạo sao chép
PS :: PS(const PS &x)
{
        ts = x.ts;
        ms = x.ms;
 }
```

NỘI DUNG:

2.1 Định nghĩa lớp

2.2 Hàm tạo và hàm hủy

2.3 Thành phần tĩnh

2.4 Hàm bạn và lớp bạn

Hàm thành phần tĩnh

- Được khai báo với từ khóa static ở đầu.
- Đặc tính của hàm thành phần tĩnh:
  - *Hàm khai báo static chỉ có thể truy nhập tới những thành phần tĩnh trong lớp*
  - *Hàm thành phần tĩnh có thể được gọi với tên của lớp thay cho tên của đối tượng*

Từ khóa const

- Được dùng khi không muốn thay đổi dữ liệu của đối tượng lớp ở trong một số phương thức (hàm) nào đó.
- **Sử dụng**: thêm từ khóa **const** vào trước khai báo của tham số hàm
- **ví dụ:**

friend sophuc cong(**const** sophuc &c1, **const** sophuc &c2);

friend sophuc tru(**const** sophuc &c1, **const** sophuc &c2);

NỘI DUNG:

- 2.1 Định nghĩa lớp
- 2.2 Hàm tạo và hàm hủy
- 2.3 Thành phần tĩnh
- **2.4 Hàm bạn và lớp bạn**

Phân loại hàm thành phần

- Hàm thành phần của lớp gồm các loại sau:
  - Hàm thành phần riêng của lớp ⇔ được khai báo trong vùng **private**
  - Hàm thành phần thân thiện ⇔ được khai báo với từ khóa **friend**
  - Hàm thành phần tĩnh ⇔ được khai báo với từ khóa **static**

Hàm thành phần private

- Hàm che giấu không cho phép các đối tượng khác truy cập đến
- Hàm được khai báo trong vùng private
- Hàm private chỉ gọi được thông qua các hàm khác trong cùng lớp
- Đối tượng trong cùng lớp cũng không truy cập được đến hàm private khi khai báo ngoài vùng định nghĩa lớp

Hàm thành phần friend

- Hàm friend là hàm được định nghĩa cho phép nhiều lớp cùng sử dụng chung
- Có quyền truy cập đến các thành viên **private** và **protected**
- Khai báo: thêm từ khóa **friend** vào trước kiểu của hàm thành phần
- Cú pháp khai báo:

    friend type tên-hàm(*parameter*);

Hàm thành phần friend

- Hàm friend có thể định nghĩa ở mọi nơi và không cần dùng từ khóa friend hay toán tử :: ⇔ **Hàm** friends **là một hàm được định nghĩa thông thường**

Đặc điểm hàm friend

- Không nằm trong miền xác định của lớp nơi được khai báo
- Khi truy cập đến hàm không cần gắn với đối tượng của lớp ⇔ truy nhập đến các hàm friend thực hiện như các hàm khai báo thông thường
- Thông thường đối số của các hàm friend là các đối tượng đang định nghĩa
- Hàm friend không sử dụng con trỏ this

Ví dụ

- Xây dựng lớp phân số với các thao tác cần định nghĩa như sau:
  - Cộng, trừ hai phân số
  - Nhân, chia hai phân số (*hàm friend)*

Ví dụ

```
#include…
//Mo ta lop

class ps
{
   int ts, ms;      //tp du lieu
public:
   //dinh nghia ham nhap xuat
   void nhap( );
   void in( );
   //dinh nghia ham cong, tru, nhan, chia
   ps cong( ps b);
   ps tru( ps b);

   friend ps nhan(ps a, ps b);
   friend ps chia(ps a, ps b);
};
```

Ví dụ

```
//Định nghĩa hàm thành phần
void ps::in()
{
  if(ms==1) cout<<ts;
  else
  {
     if(ts==0) cout<<"0";
     else cout<<ts<<"/"<<ms;
  }
}
```

Ví dụ

```
void ps::nhap()
{
  cout<<"\n";
  cout<<"Nhap tu so:"; cin>>ts;
  cout<<"Nhap mau so:"; cin>>ms;
  if (ms<0)
  {
     ts=-ts;
     ms=-ms;
  }
}
```

Ví dụ

```
ps ps::cong( ps b)
{
  ps kq;
  kq.ts = ts*b.ms+b.ts*ms;
  kq.ms = ms*b.ms;
  return kq;
}
ps ps::tru( ps b)
{
  ps kq;
  kq.ts = ts*b.ms - b.ts*ms;
  kq.ms = ms*b.ms;
  return kq;
}
```

Ví dụ

```
ps nhan( ps a, ps b)
{
  ps kq;
  kq.ts = a.ts*b.ts;
  kq.ms = a.ms*b.ms;
  return kq;
}
ps chia( ps a, ps b)
{
  ps kq;
  kq.ts = a.ts*b.ms;
  kq.ms = a.ms*b.ts;
  return kq;
}
```

```
void main()
{
  ps x,y,kq;
  cout<<"Nhap phan so thu nhat: \n";
  x.nhap();
  cout<<"\n\n Nhap phan so thu hai:\n";
  y.nhap();
  kq = x.cong(y);
  cout<<"\n Tong:";
  kq.in();
  kq = nhan(x,y);
  cout<<"\n Tich:";
  kq.in();
  getch();
}
```

Bài tập:

Cho định nghĩa lớp ma trận như sau.

- Hãy viết các hàm thành viên của lớp.
- Viết hàm main() nhập vào 2 ma trận.
- Thực hiện các phép toán trên ma trận đã định nghĩa và hiển thị kết quả (nếu thực hiện được phép toán đó).

viết các hàm thành viên của lớp. Viết hàm main()

```
class Matrix
{
private:
    int m; // dòng
    int n; // cột
    double elements[100][100];
public:
    Matrix();
    ~Matrix();
    Matrix(const Matrix & a);
    void nhap();
    void xuat();
    int Cong(const Matrix & a); // return 1 nếu cộng dc
    void Nhan(const double & k);// Nhân với 1 số K
    int Nhan(const Matrix & a);// return 1 nếu nhân đc
    friend Vector multiply(const Matrix &a, const Vector &b);
};
```