



Môn học

Lập trình hướng đối tượng



Nội dung

❖ Chương 3: Lớp và đối tượng

- Khai báo Lớp
- Định nghĩa các phương thức
- Phương thức thiết lập
- Phép gán đối tượng
- Con trỏ đối tượng
- Hàm hủy bỏ
- Hàm thiết lập sao chép
- Đối tượng và hàm thành phần



Nội dung

❖ Chương 3: Lớp và đối tượng (t)

- Hàm bạn và lớp bạn
- Lớp bao
- Các thành phần tĩnh
- Bài tập



1. Khai báo lớp

- ❖ Lớp đóng vai trò như một kiểu dữ liệu được người dùng định nghĩa
- ❖ Việc tạo ra một đối tượng được ví như khai báo một biến có kiểu lớp.



1. Khai báo lớp (t)

❖ Cú pháp khai báo lớp

class <Tên Lớp> {

 private:

 <kiểu dữ liệu> tên thuộc tính của từng đối tượng;

 public:

 <kiểu dữ liệu> Khai báo các phương thức của đối tượng

};

<Định nghĩa phương thức trong khai báo lớp.>



1. Khai báo lớp (t)

❖ Ví dụ: Lớp Điểm trong mặt phẳng gồm:

- Thuộc tính: Tọa độ (x, y)
- Phương thức: Nhập điểm, xuất điểm, di chuyển điểm,...



1. Khai báo lớp (t)

❖ Khai báo Điểm

```
class Diem {
```

```
    private:
```

```
        double x, y; /*Khai báo thuộc tính */
```

```
    public:
```

```
        /* Khai báo các phương thức xử lý */
```

```
        void KhoiTao(double xx, double yy);
```

```
        void Nhap();
```

```
        void Xuat();
```

```
        void DiChuyen(double dx, double dy);
```

```
};
```



1. Khai báo lớp (t)

- ❖ Các thuộc tính của lớp được khai báo giống như khai báo biến, và các phương thức xử lý được khai báo giống như khai báo các hàm trong C
- ❖ Các thuộc tính được liệt kê sau từ khóa **private** chỉ được truy cập bởi các phương thức trong lớp đó.
- ❖ Các phương thức được liệt kê sau từ khóa **public** có thể được truy nhập bởi bất kỳ hàm nào trong lớp hoặc ngoài lớp.



1. Khai báo lớp (t)

```
/* Định nghĩa các phương thức trong khai  
báo lớp Điểm */  
  
void Diem::KhoiTao(double xx, double yy){  
    x = xx; y = yy;  
}  
  
void Diem::Nhap() {  
    cout<<"Nhập tọa độ: ";  
    cin>>x>>y;  
}
```

```
void Diem::Xuat(){  
    cout<<x<<", "<<y<<endl;  
}  
  
void Diem::DiChuyen(double  
    xx, double yy){  
    x += dx; y += dy;  
}
```



1. Khai báo lớp (t)

```
/* Hàm chính */  
int main(){  
    Diem a;  
    a.Nhap(); a.Xuat(); a.DiChuyen(10,20); a.Xuat();  
    Diem b;  
    b.KhoiTao(3,4); b.DiChuyen(3,5); b.Xuat();  
    return 0;  
}
```



2. Định nghĩa các phương thức

❖ Định nghĩa chồng các phương thức

```
class Diem {  
    private:  
        double x,y;  
    public:  
        void KhoiTao();  
        void KhoiTao(double xx);  
        void KhoiTao(double xx, doubl yy);  
};
```



2. Định nghĩa các phương thức (t)

❖ Định nghĩa các phương thức

```
void Diem::KhoiTao() {  
    x = 0; y = 0;  
}
```

```
void Diem::KhoiTao(double xx) {  
    x = xx; y = 0;  
}
```

```
void Diem::KhoiTao(double xx, double yy) {  
    x = xx; y = yy;  
}
```



2. Định nghĩa các phương thức (t)

❖ Hàm chính

```
int main(){  
    Diem a, b, c;  
    a.KhoiTao();  
    b.KhoiTao(2);  
    c.KhoiTao(3,4);  
    return 0;  
}
```



2. Định nghĩa các phương thức (t)

❖ Phương thức với tham số mang giá trị mặc định

```
class Diem {
```

```
    private:
```

```
        double x, y;
```

```
    public:
```

```
    ....
```

```
        void KhoiTao(double xx = 0, double yy = 0);
```

```
};
```



2. Định nghĩa các phương thức (t)

❖ Sử dụng đối tượng làm tham số truyền cho phương thức

```
class Diem {  
    private:  
        double x, y;  
    public:  
        ....  
        int CheckTrungDiem(Diem u);  
        double Kcach(Diem u);  
};
```



2. Định nghĩa các phương thức (t)

❖ Định nghĩa các phương thức

```
int Diem:: CheckTrungDiem(Diem u){  
    if(x == u.x && y == u.y) return 1;  
    return 0;  
}  
  
double Diem::Kcach(Diem u){  
    return sqrt(pow(x-u.x,2) + pow(y-u.y,2));  
}
```



2. Định nghĩa các phương thức (t)

❖ Dùng đối tượng như giá trị trả về của phương thức

```
class Diem {  
    private:  
        double x, y;  
    public:  
        ....  
        Diem DoiXung();  
};
```



2. Định nghĩa các phương thức (t)

❖ /* Định nghĩa các phương thức */

```
Diem Diem::DoiXung(){
    Diem res;
    res.x = -x;
    res.y = -y;
    return res;
}
```



2. Định nghĩa các phương thức (t)

- ❖ C++ sử dụng một con trỏ đặc biệt tên là this trong định nghĩa các phương thức.
- ❖ Con trỏ này luôn trỏ tới đối tượng dùng làm tham số ngầm định trong lời gọi phương thức.
- ❖ Ví dụ 1: Do đặc tính của con trỏ this, các phương thức lớp **Diem** có thể được viết một cách tương minh như sau:



2. Định nghĩa các phương thức (t)

```
void Diem::Nhap(){
    cout<<“Nhập tọa độ:”; cin>>this->x>>this->y;
}

void Diem::Xuat(){
    cout<<this->x<<”, “<<this->y<<endl;
}

void Diem::DiChuyen(int dx, int dy){
    this->x += dx;
    this->y += dy;
}
```



2. Định nghĩa các phương thức (t)

❖ Ví dụ 2: Thêm phương thức tính chu vi và diện tích của lớp Diem

```
class Diem {  
    private:  
        ....  
    public:  
        ...  
        double ChuVi(Diem u, Diem v);  
        double DienTich(Diem u, Diem v);  
};
```



2. Định nghĩa các phương thức (t)

```
double Diem::ChuVi(Diem u, Diem v){  
    return this->Kcach(u) + u.Kcach(v) + v.Kcach(*this);  
}  
  
double Diem::DienTich(Diem u, diem v){  
    double d1 = this->Kcach(u);  
    double d2 = u.Kcach(v);  
    double d3 = v->Kcach(*this);  
    double p = (d1 + d2 + d3) / 2;  
    return sqrt(p * (p-d1) * (p-d2) * (p-d3));  
}
```



3. Phương thức thiết lập (constructor)

- ❖ Phương thức thiết lập là một hàm đặc biệt được gọi tự động mỗi khi có một đối tượng được khai báo (tạo ra)
- ❖ Phương thức thiết lập được khai báo với tên trùng tên với lớp, không có giá trị trả về và không cần khai báo void.
- ❖ Trong một lớp có thể có nhiều phương thức thiết lập (chồng phương thức)



3. Phương thức thiết lập (constructor) (t)

❖ Ví dụ:

```
class Diem {  
    private:  
        double x,y;  
    public:  
        Diem(double xx, double yy) : x(xx), y(yy) {} // Gán trực tiếp  
        Diem(); /* Phương thức thiết lập không tham số */  
        Diem (double xx); /* 1 tham số */  
        Diem (double xx, double yy); /* 2 tham số */  
};
```



3. Phương thức thiết lập (constructor) (t)

❖ Định nghĩa các phương thức

```
Diem::Diem(){
```

```
    x = 0; y = 0;
```

```
}
```

```
Diem::Diem(double xx){
```

```
    x = xx; y = 0;
```

```
}
```

```
Diem::Diem(double xx, double yy){
```

```
    x = xx; y = yy;
```

```
}
```

```
int main(){
```

```
    Diem a; // Gọi Diem::Diem();
```

```
    Diem b(2); // Gọi Diem::Diem(x)
```

```
    Diem c(3,4); //Diem(x,y)
```

```
}
```



4. Phép gán đối tượng

- ❖ Phép gán đối tượng thực chất là việc sao chép các thuộc tính từ đối tượng nguồn (đứng bên phải phép gán) sang đối tượng đích (đứng bên trái phép gán) tương ứng từng đôi một.
- ❖ Ví dụ:
 - Diem a (4,5);
 - a.Xuat();
 - Diem b; b = a;
 - b.Xuat();
- ❖ Lưu ý: Nhưng đối với những đối tượng là con trỏ thì phép sao chép cần được định nghĩa lại, nếu không sẽ bị LỖI



5. Con trỏ đối tượng

```
int main(){
    Diem a(10,20);
    x.Xuat();
    Diem *p = &a; p->Xuat();
    p->DiChuyen(3,5); p->Xuat();
    p = new Diem;
    p->Xuat(); delete p;
    p = new Diem(5,3);
    p->Xuat(); delete p;
}
```



6. Hàm hủy bỏ

- ❖ Là phương thức đặc biệt được gọi tự động khi có một đối tượng bị xóa khỏi vùng nhớ.
- ❖ Nó được khai báo tương tự các phương thức khác với tên bắt đầu dấu ~ theo sau là tên lớp tương ứng, không có tham số, không có giá trị trả về và không cần khai báo void.



6. Hàm hủy bỏ (t)

- ❖ Một lớp chỉ có duy nhất một hàm hủy bỏ
- ❖ Dùng để giải phóng bộ nhớ động mà đối tượng đang quản lý.
- ❖ Khi người sử dụng không khai báo tường minh một hàm hủy bỏ cho lớp thì trình biên dịch sẽ tự động cung cấp cho lớp một **hàm hủy bỏ ngầm định**.



6. Hàm hủy bỏ (t)

❖ Ví dụ:

```
Class Mang1Chieu {  
    private:  
        int n; int *a;  
    public:  
        Mang1Chieu();  
        Mang1Chieu(int nn);  
        ~Mang1Chieu();  
        void Nhap(); void Xuat();  
};
```



6. Hàm hủy bỏ (t)

```
Mang1Chieu::Mang1Chieu(){  
    n = 0; a = NULL;  
}  
  
Mang1Chieu::Mang1Chieu(int nn){  
    this->n = nn; a = new int[n];  
}  
  
Mang1Chieu::~Mang1Chieu(){  
    delete [] a; a = NULL;  
}
```



6. Hàm hủy bỏ (t)

```
void Mang1Chieu::Nhap(){
    if(a == NULL) {
        cin>>n; a = new int [n];
    }
    for(int i=0;i<n;i++) cin>>a[i];
}

void Mang1Chieu::Xuat(){
    for(int i=0;i<n;i++) cout<<a[i]<<" ";
}
```



6. Hàm hủy bỏ (t)

```
int main(){
    Mang1Chieu m;
    m.Nhap(); m.Xuat();
    Mang1Chieu m2(3);
    m2.Nhap(); m2.Xuat();
    return 0;
}
```



6. Hàm hủy bỏ (t)

❖ Ví dụ 2: Ma trận

```
class MaTran {  
    private:  
        int m, n; int **a;  
    public:  
        MaTran();  
        MaTran(int m, int n);  
        ~MaTran();  
        void Nhap(); void Xuat();  
};
```



6. Hàm hủy bỏ (t)

```
MaTran::MaTran(){  
    m = n = 0;  
    a = NULL;  
}  
  
MaTran::~MaTran(){  
    for(int i=0;i<m;i++) delete [] a[i];  
    delete []a;  
    a = NULL;  
}
```

```
MaTran::MaTran(int m, int n){  
    this->m = m;  
    this->n = n;  
    a = new int *[m];  
    for(int i=0;i<m;i++)  
        a[i] = new int[n];  
}
```



6. Hàm hủy bỏ (t)

```
void MaTran::Nhap(){
    if(a == NULL){
        cin>>m>>n;
        a = new int *[m];
        for(int i=0;i<m;i++) a[i] = new int [n];
    }
    for(int i=0;i<m;i++){
        for(int j=0;j<n;j++) cin>>a[i][j];
    }
}
```



6. Hàm hủy bỏ (t)

```
void MaTran::Xuat(){  
    for(int i=0;i<m;i++) {  
        for(int j=0;j<n;j++) cout<<a[i][j]<<" ";  
        cout<<endl;  
    }  
}  
  
int main(){  
    MaTran a; a.Nhap(); a.Xuat();  
    MaTran b (2,3); b.Nhap(); b.Xuat();  
}
```



7. Hàm thiết lập sao chép

❖ Là một hàm đặc biệt được gọi tự động khi xảy ra trong các tình huống sau:

- Khởi tạo nội dung cho một đối tượng thông qua một đối tượng khác (TH1)
- Sử dụng đối tượng làm tham số truyền cho hàm theo giá trị (TH2)



7. Hàm thiết lập sao chép (t)

- ❖ Khi người sử dụng không khai báo tường minh một hàm thiết lập sao chép cho lớp thì trình biên dịch tự động cung cấp cho lớp một **hàm thiết lập sao chép ngầm định**.
- ❖ Hàm này chỉ có nhiệm vụ: Sao chép giá trị các thành phần dữ liệu có trong đối tượng nguồn cho các thành phần dữ liệu có trong đối tượng đích



7. Hàm thiết lập sao chép (t)

```
class Diem {  
    private:  
        ...  
    public:  
        ...  
        Diem(Diem &u); // Hàm sao chép  
};  
Diem::Diem(Diem &u){  
    x = u.x; y = u.y;  
}
```



7. Hàm thiết lập sao chép (t)

```
int main(){
    Diem a(2,6);
    Diem b = a; // Gọi Diem::Diem(Diem &); TH1
    Diem c(4,5);
    double kq = a.Kcach(c); // Gọi Diem::Diem(Diem &); TH2
}
```



7. Hàm thiết lập sao chép (t)

❖ Chú ý:

- Với các lớp không có thành phần dữ liệu kiểu con trỏ thì chỉ cần sử dụng hàm thiết lập sao chép ngầm định là đủ.
- Các lớp có thành phần dữ liệu kiểu con trỏ thì không được sử dụng hàm thiết lập sao chép ngầm định có sẵn mà phải dùng đến hàm thiết lập sao chép tường minh do người dung sử dụng khai báo.



7. Hàm thiết lập sao chép (t)

```
class mang1Chieu {  
    private:  
        int n;  
        int *a;  
    public:  
        ....  
        mang1Chieu(mang1Chieu &m);  
        .....  
}
```



7. Hàm thiết lập sao chép (t)

```
mang1Chieu::mang1Chieu(mang1Chieu &m){  
    n = m.n;  
    a = new int[n];  
    for(int i=0;i<n;i++) a[i] = m.a[i];  
}
```

Khi đó để sao chép đối tượng ta chỉ cần gọi:

```
mang1Chieu m1; m1.NhapMang();  
mang1Chieu m2 = m1; // Gọi hàm thiết lập sao chép
```



8. Đối tượng & hàm thành phần hằng

❖ Thành viên dữ liệu hằng

- Khi một thành viên được khai báo là **const**, thành viên đó sẽ được giữ nguyên giá trị trong suốt thời gian sống của đối tượng.

❖ Hàm thành viên hằng

- Được phép gọi trên hằng đối tượng.
- Không được thay đổi giá trị dữ liệu thành viên.

❖ Hằng đối tượng

- Không được thay đổi giá trị.



8. Đối tượng & hàm thành phần hằng (t)

```
class SV {  
    private:  
        string masv;  
        string hoten;  
        double diem;  
    public:  
        SV(){ masv = ""; hoten = ""; diem = 0; }  
        SV(string t1, string t2, double t3) : masv(t1), hoten(t2), diem(t3) {};  
        void nhap();  
        void xuat();  
};
```



8. Đối tượng & hàm thành phần hằng (t)

```
class Lop {  
    private:  
        const int spt; // Thành viên dữ liệu hằng  
        SV *sv;  
    public:  
        Lop(int t) : spt(t) {  
            sv = new SV[spt];  
        }  
        void TimKiemSV() const; // Hàm thành viên hằng  
};
```



8. Đối tượng & hàm thành phần hằng (t)

```
void Lop::TimKiemSV() const{  
    ....  
}  
  
int main(){  
    const Lop s(2); // Hằng đối tượng.  
    s.TimKiemSV();  
}
```



9. Hàm bạn và lớp bạn

- ❖ Khi một hàm được khai báo là ***friend*** của lớp, hàm này được xem là bạn của lớp và được phép truy xuất đến các thành phần ***private*** của lớp giống như hàm thành của lớp đó.



9. Hàm bạn và lớp bạn (t)

❖ Ví dụ 1

```
class Diem {  
    private:  
        double x,y;  
    public:  
        Diem(int xx, int yy) : x(xx), y(yy) {}  
        bool Trung(Diem u);  
        friend bool Trung(Diem u, Diem v);  
};
```



9. Hàm bạn và lớp bạn (t)

```
bool Diem::Trung(Diem u){  
    if(u.x == this->x && u.y == this->y) return true;  
    return false;  
}  
  
bool Trung(Diem u, Diem v) {  
    if(u.x == v.x && u.y == v.y) return true;  
    return false;  
}
```



9. Hàm bạn và lớp bạn (t)

```
int main(){
    Diem a1(2,3);
    Diem b1(3,4);
    cout<<Trung(a1, b1)<<"→"<<a1.Trung(b1)<<endl;
    return 0;
}
```



9. Hàm bạn và lớp bạn (t)

- ❖ Lớp bạn trong C++ cũng tương tự như hàm bạn, việc khai báo lớp bạn sẽ cho phép lớp bạn của lớp kia được truy cập tất cả các thành viên của nó.
- ❖ Tính chất:
 - Khi khai báo A là bạn của lớp B không có nghĩa là B xem A là bạn, điều đó có nghĩa chỉ có lớp A truy cập được thành viên của lớp B. Nhưng lớp B không thể truy cập ngược lại của lớp A
 - Không đối xứng
 - Không bắt đầu



9. Hàm bạn và lớp bạn (t)

❖ Ví dụ 2

```
class TOM {  
    private:  
        int secretTom;  
    public:  
        TOM();  
        void xuat();  
        friend class JERRY;  
};
```



9. Hàm bạn và lớp bạn (t)

```
TOM::TOM(){  
    this->secretTom = 0;  
}  
  
void TOM::xuat(){  
    cout << "secretTom: " << secretTom << endl;  
}
```



9. Hàm bạn và lớp bạn (t)

```
class JERRY {  
    private:  
        int secretJERRY;  
    public:  
        void change(TOM &T);  
};
```

```
void JERRY::change(TOM &T){  
    T.secretTom++;  
}
```



9. Hàm bạn và lớp bạn (t)

```
int main(){
    TOM t;
    t.xuat();
    JERRY j;
    j.change(t);
    t.xuat();

    return 0;
}
```



10. Lớp bao

- ❖ Một lớp có thành phần dữ liệu là đối tượng thuộc một lớp khác thì lớp đó được gọi là lớp bao.
- ❖ Khi xây dựng hàm thiết lập của lớp bao, có thể sử dụng các hàm thiết lập của lớp thành phần để khởi gán cho các đối tượng thành phần của lớp bao.



10. Lớp bao (t)

```
class Diem {  
    private:  
        double x,y;  
    public:  
        Diem(double xx=0, double yy=0);  
        void Nhap(); void Xuat();  
};
```



10. Lớp bao (t)

```
class TamGiac {  
    private:  
        Diem A,B,C;  
    public:  
        TamGiac(Diem AA, Diem BB, Diem CC);  
        void Nhap(); void Xuat();  
};
```



10. Lớp bao (t)

```
TamGiac::TamGiac(Diem AA, Diem BB, Diem CC){  
    A = AA; B = BB; C = CC;  
}  
  
void TamGiac::Nhap(){  
    A.NhapDiem(); B.NhapDiem(); C.NhapDiem();  
}  
  
void TamGiac::Xuat(){  
    A.XuatDiem(); B. XuatDiem(); C. XuatDiem();  
}
```



10. Lớp bao (t)

```
int main(){
    TamGiac b(Diem(2,3), Diem(3,4), Diem (4,5) );
    b.Xuat();
    TamGiac c;
    c.Nhap(); c.Xuat();
    return 0;
}
```



11. Các thành phần tĩnh (static)

❖ Thành phần dữ liệu tĩnh

- Là thành phần dữ liệu chung cho cả lớp chứ không phải riêng của từng đối tượng. Nó tồn tại ngay cả khi lớp chưa có đối tượng nào.
- Thành phần dữ liệu tĩnh được cấp phát bộ nhớ và khởi gán giá trị ban đầu bằng một câu lệnh khai báo bên ngoài.



11. Các thành phần tĩnh (static) (t)

```
class Counter {  
    private:  
        static int n; // Thành phần dữ liệu tĩnh  
    public:  
        Counter();  
        ~Counter();  
};  
int Counter::n = 0; // Khởi tạo bên ngoài lớp
```



11. Các thành phần tĩnh (static) (t)

```
Counter::Counter(){  
    cout<<“Khoi tao: Bay gio co: “<< ++n<<“ doi tuong” <<endl;  
}  
  
Counter::Counter(){  
    cout<<“Huy: Bay gio con: “<< --n << “ doi tuong” <<endl;  
}
```



11. Các thành phần tĩnh (static) (t)

```
int main(){  
    Counter a;  
    Counter b;  
    return 0;  
}
```



11. Các thành phần tĩnh (static) (t)

❖ Hàm thành phần tĩnh

- Là hàm thành phần không cần phụ thuộc vào một đối tượng cụ thể và tồn tại khi chưa có đối tượng nào.
- Lời gọi hàm thành phần tĩnh có thể thực hiện thông qua tên lớp hoặc đối tượng
- Trong thân hàm thành phần tĩnh không được phép truy cập đến các thành phần dữ liệu, ngoại trừ thành phần dữ liệu tĩnh.



11. Các thành phần tĩnh (static) (t)

```
class Counter {  
    private:  
        static int n; // Thành phần dữ liệu tĩnh  
    public:  
        Counter(){n++;}  
        ~Counter(){n--;}  
        static void Display();  
};  
int Counter::n = 0; // Khởi tạo bên ngoài lớp
```



11. Các thành phần tĩnh (static) (t)

```
void Counter::Display(){  
    cout<<“Hien dang co: “<<n << “ doi tuong”<<endl;  
}  
  
int main(){  
    Counter a, b;  
    Counter::Display();  
    return 0;  
}
```



12. Bài tập

1. Xây dựng đối tượng điểm gồm 2 thuộc tính (x, y) và thực hiện yêu cầu sau:
 - a) Tính khoảng cách 2 điểm
 - b) Kiểm tra 3 điểm thẳng hàng
 - c) Tính chu vi và diện tích tạo thành bởi 3 điểm
 - d) Kiểm tra điểm có thuộc trong tam giác hay không?
2. Bài tập thao tác với mảng một chiều và ma trận (xây dựng đối tượng sử dụng con trỏ)
3. Bài toán quản lý lớp sinh viên



Thank you!

Any questions?