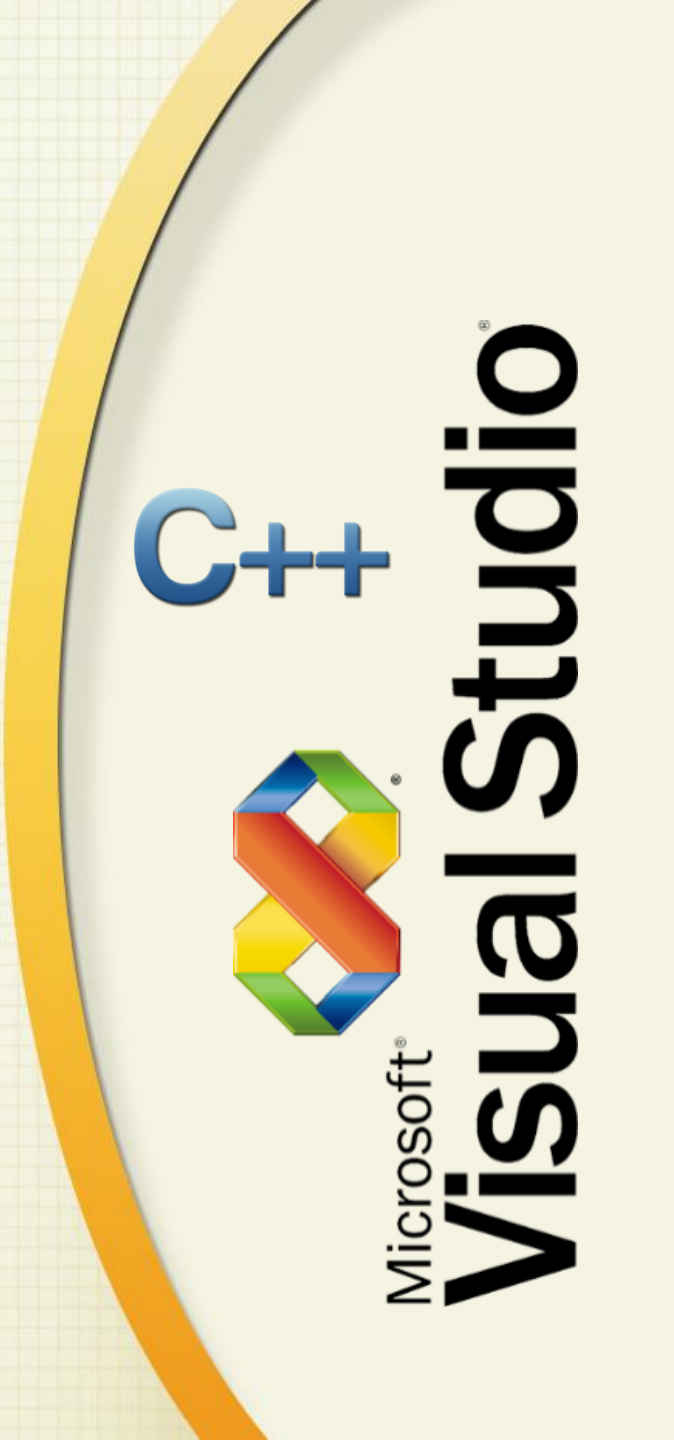
UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

ĐA HÌNH



Nội dung

1. Giới thiệu
2. Vùng chọn kiểu
3. Phương thức ảo
4. Phương thức thuần ảo
5. Bài toán Tính tiền lương

Giới thiệu

- Tính đa hình xuất hiện khi có sự kế thừa giữa các lớp.
- Có những phương thức tổng quát cho mọi lớp dẫn xuất nên có mặt ở lớp cơ sở nhưng nội dung của nó chỉ được xác định ở các lớp dẫn xuất cụ thể.
- Ví dụ, Phương thức tính diện tích của lớp hình, hình tam giác, tứ giác,…

Giới thiệu

- Đa hình: Là hiện tượng các đối tượng thuộc các lớp khác nhau có khả năng hiểu cùng một thông điệp theo các cách khác nhau.
- Ví dụ: Nhận được cùng một thông điệp “nhảy”, một con kangaroo và một con cóc nhảy theo hai kiểu khác nhau: chúng cùng có hành vi “nhảy” nhưng các hành vi này có nội dung khác nhau.

Bài toán

- Giả sử, cần quản lý danh sách các đối tượng có kiểu có thể khác nhau →Cần giải quyết 2 vấn đề:
  - Cách lưu trữ
  - Thao tác xử lý
- Xét trường hợp cụ thể, các đối tượng có thể là Người, Sinh viên hoặc Công nhân.

Bài toán

❖Về mặt lưu trữ:

- Có thể dùng mảng
- Danh sách liên kết
- …

Có hai cách để giải quyết vấn đề:
- Vùng chọn kiểu
- Phương thức ảo

❖Về thao tác: Phải thõa yêu cầu đa hình, thao tác có hoạt động khác nhau ứng với các loại đối tượng khác nhau

Ví dụ

```
class Nguoi {
protected:
    char *HoTen;
    int NamSinh;
public:
    Nguoi(char *ht, int ns):NamSinh(ns){HoTen=strdup(ht);}
    ~Nguoi() {delete [ ] HoTen;}
    void An() const { cout << HoTen << " an 3 chen com";}
    void Xuat() const {
        cout << "Nguoi, ho ten: " << HoTen << " sinh “
        cout << NamSinh; }
};
```

Ví dụ

```
class SinhVien : public Nguoi{
protected:
    char *MaSo;
public:
    SinhVien(char *n, char *ms, int ns) : Nguoi(n,ns) {
        MaSo = strdup(ms);
    }
    ~SinhVien() { delete [ ] MaSo;}
    void Xuat() const {
        cout<<"Sinh vien "<<HoTen<<", ma so "<<MaSo;
    }
};
```

Ví dụ

```
class NuSinh : public SinhVien
{
  public:
    NuSinh( char *ht, char *ms, int ns) : SinhVien(ht,ms,ns) {
    }
    void An() const
    {
          cout << HoTen
          cout << " ma so " << MaSo << " an 2 to pho";
    }
};
```

Ví dụ

```
class CongNhan : public Nguoi{
protected:
    double MucLuong;
public:
    CongNhan( char *n, double ml, int ns) : Nguoi(n,ns),
    MucLuong(ml){ }
    void Xuat() const {
            cout << "Cong nhan, ten " << HoTen
            cout << " muc luong: " << MucLuong;
    }
};
```

Ví dụ

```
void XuatDs(int n, Nguoi *an[ ])
{
    for (int i = 0; i < n; i++)
    {
        an[i] →Xuat();
        cout << "\n";
    }
}
```

Ví dụ

```
const int N = 4;
void main(){
    Nguoi *a[N];
    a[0] = new SinhVien(“Vien Van Sinh”, “200001234”, 1982);
    a[1] = new NuSinh(“Le Thi Ha Dong”, “200001235”, 1984);
    a[2] = new CongNhan(“Tran Nhan Cong”, 1000000, 1984);
    a[3] = new Nguoi(“Nguyen Thanh Nhan”, 1960);
    XuatDs(4,a);
}
```

```
Nguoi, ho ten: Vien Van Sinh sinh 1982
Nguoi, ho ten: Le Thi Ha Dong sinh 1984
Nguoi, ho ten: Tran Nhan Cong sinh 1984
Nguoi, ho ten: Nguyen Thanh Nhan sinh 1960
```

Dùng vùng chọn kiểu

- Để bảo đảm xuất liệu tương ứng với đối tượng, phải có cách nhận diện đối tượng
  - Ta thêm một vùng dữ liệu vào lớp cơ sở để nhận diện
  - Vùng này có giá trị phụ thuộc vào loại của đối tượng và được gọi là vùng chọn kiểu.
- Các đối tượng thuộc lớp người có cùng giá trị cho vùng chọn kiểu, các đối tượng thuộc lớp sinh viên có giá trị của vùng chọn kiểu khác của lớp người.

Dùng vùng chọn kiểu – Ví dụ

```
class Nguoi{
public:          enum LOAI {NGUOI, SV, CN};
protected:
    char *HoTen;       int NamSinh;
public:
    LOAI pl;
    Nguoi(char *ht, int ns):NamSinh(ns), pl(NGUOI) {HoTen
    = strdup(ht);}
    ~Nguoi() {delete [] HoTen;}
    void An() const { cout << HoTen << " an 3 chen com";}
    void Xuat() const { cout << "Nguoi, ho ten: " << HoTen <<
    " sinh " << NamSinh; }
};
```

Dùng vùng chọn kiểu – Ví dụ

```
class SinhVien : public Nguoi{
protected:
    char *MaSo;
public:
    SinhVien(char *n, char *ms, int ns) :    Nguoi(n,ns) {
        MaSo = strdup(ms); pl = SV;
    }
    ~SinhVien() {delete [ ] MaSo;}
    void Xuat() const {
        cout<<"Sinh vien "<<HoTen<<", ma so " << MaSo;
    }
};
```

Dùng vùng chọn kiểu – Ví dụ

```
class CongNhan : public Nguoi{
protected:
    double MucLuong;
public:
    CongNhan( char *n, double ml, int ns) : Nguoi(n,ns),
    MucLuong(ml){
        pl = CN;
    }
    void Xuat() const{
        cout << "Cong nhan, ten " << HoTen
        cout << " muc luong: " << MucLuong;
    }
};
```

Dùng vùng chọn kiểu – Ví dụ

```
void XuatDs(int n, Nguoi *an[]) {
    for (int i = 0; i < n; i++){
        switch(an[i]->pl){
            case Nguoi::SV:
                ((SinhVien*)an[i])→Xuat();  break;
            case Nguoi::CN:
                ((CongNhan*)an[i])→Xuat(); break;
            default:
                an[i]->Xuat(); break;
        }
        cout << "\n";
    }
}
```

Dùng vùng chọn kiểu – Ví dụ

```
const int N = 4;
void main(){
    Nguoi *a[N];
    a[0] = new SinhVien(“Vien Van Sinh”, “200001234”, 1982);
    a[1] = new NuSinh(“Le Thi Ha Dong”, “200001235”, 1984);
    a[2] = new CongNhan(“Tran Nhan Cong”, 1000000, 1984);
    a[3] = new Nguoi(“Nguyen Thanh Nhan”, 1960);
    XuatDs(4,a);
}
```

```
Sinh vien Vien Van Sinh, ma so 200001234
Sinh vien Le Thi Ha Dong, ma so 200001235
Cong nhan, ten Tran Nhan Cong muc luong:1000000
Nguoi, ho ten: Nguyen Thanh Nhan sinh 1960
```

Dùng vùng chọn kiểu

- Cách tiếp cận trên giải quyết được vấn đề: Lưu trữ các đối tượng khác kiểu nhau và thao tác khác nhau tương ứng từng đối tượng. Tuy nhiên, tồn tại một số khuyết điểm:
  - Mã lệnh dài dòng (nhiều switch case)
  - Dễ sai sót, khó sửa
  - Khó nâng cấp, bảo trì
- Các nhược điểm trên có thể khắc phục được nhờ phương thức ảo.

Phương thức ảo

- Phương thức ảo:
  - Là cách thể hiện tính đa hình trong ngôn ngữ C++.
  - Các phương thức ở lớp cơ sở có tính đa hình phải được định nghĩa là một phương thức ảo
- Con trỏ thuộc lớp cơ sở có thể trỏ đến lớp con:

```
Nguoi* pn=new SinhVien("Le Vien Sinh",TH11001,1982);
```

Phương thức ảo

- Ta mong muốn thông qua con trỏ thuộc lớp cơ sở có thể truy xuất hàm thành phần được định nghĩa lại ở lớp con

```
pn->Xuat();
//Mong muon: goi Xuat cua lop sinh vien,
//Thuc te: goi Xuat cua lop Nguoi
```

Phương thức ảo

- Phương thức ảo cho phép giải quyết vấn đề trên.
- Ta qui định một hàm thành phần là phương thức ảo bằng cách thêm từ khóa virtual vào trước khai báo hàm.
- Trong ví dụ trên, ta thêm từ khóa virtual vào trước khai báo của hàm Xuat.

Phương thức ảo – Ví dụ

```
class Nguoi {
protected:
    char *HoTen;
    int NamSinh;
public:
    Nguoi( char *ht,int ns):NamSinh(ns){HoTen = strdup(ht);}
    ~Nguoi() {delete [ ] HoTen;}
    void An() const { cout << HoTen << " an 3 chen com";}
    virtual void Xuat() const {
            cout << "Nguoi, ho ten: " << HoTen
            cout << " sinh " << NamSinh;
    }
};
```

Thêm lớp con mới

- Dùng phương thức ảo, ta dễ dàng nâng cấp sửa chữa.
- Thêm một loại đối tượng mới rất đơn giản, không cần sửa đổi thao tác xử lý (XuatDs).
- Qui trình thêm chỉ là xây dựng lớp con kế thừa lớp cơ sở và định nghĩa lại phương thức (ảo) ở lớp mới tạo nếu cần.

Thêm lớp con mới – Ví dụ

```
class CaSi : public Nguoi{
protected:
    double CatXe;
public:
    CaSi( char *ht, double cx, int ns): Nguoi(ht,ns),CatXe(cx)
    { }
    void Xuat() const {
            cout<<"Ca si, "<<HoTen<<" co cat xe "<< CatXe;
    }
};
```

Thêm lớp con mới

```
void XuatDs( int n, Nguoi *an[]){
    for ( int i = 0; i < n; i++){
        an[i]->Xuat();
        cout << "\n";
    }
}
```

- Hàm XuatDs không thay đổi, nhưng nó có thể hoạt động cho các loại đối tượng ca sĩ thuộc lớp mới ra đời.

Lưu ý khi sử dụng phương thức ảo

- Phương thức ảo chỉ hoạt động thông qua con trỏ.
- Muốn một hàm trở thành phương thức ảo có hai cách:
  - Khai báo với từ khoá virtual
  - Hoặc phương thức tương ứng ở lớp cơ sở đã là phương thức ảo.

Lưu ý khi sử dụng phương thức ảo

- Phương thức ảo chỉ hoạt động nếu các phương thức ở lớp cơ sở và lớp con có nghi thức giao tiếp giống hệt nhau.
- Nếu ở lớp con định nghĩa lại phương thức ảo thì sẽ gọi phương thức ở lớp cơ sở (gần nhất có định nghĩa).

Cơ chế thực hiện phương thức ảo

- Khi gọi một thao tác, khả năng chọn đúng phiên bản tùy theo đối tượng để thực hiện thông qua con trỏ đến lớp cơ sở được gọi là tính đa hình (polymorphisms).
- Cơ chế đa hình được thực hiện nhờ ở mỗi đối tượng có thêm một bảng phương thức ảo. Bảng này chứa địa chỉ của các phương thức ảo và nó được trình biên dịch khởi tạo một cách ngầm định khi thiết lập đối tượng.

Cơ chế thực hiện phương thức ảo

- Khi thao tác được thực hiện thông qua con trỏ, hàm có địa chỉ trong bảng phương thức ảo sẽ được gọi.
- Trong ví dụ trên, mỗi đối tượng thuộc lớp cơ sở Nguoi có bảng phương thức ảo có một phần tử là địa chỉ hàm Nguoi::Xuat. Mỗi đối tượng thuộc lớp SinhVien có bảng tương tự nhưng nội dung là địa chỉ của hàm SinhVien::Xuat.

Các đặc trưng của phương thức ảo

- Phương thức ảo không thể là các hàm thành viên tĩnh.
- Một phương thức ảo có thể được khai báo là friend trong một lớp khác nhưng các hàm friend của lớp thì không thể là phương thức ảo.
- Không cần thiết phải ghi rõ từ khóa virtual khi định nghĩa một phương thức ảo trong lớp dẫn xuất (để cũng chẳng ảnh hưởng gì).

Các đặc trưng của phương thức ảo

- Để sự kết nối động được thực hiện thích hợp cho từng lớp dọc theo cây phả hệ, một khi phương thức nào đó được xác định là ảo, từ lớp cơ sở đến các lớp dẫn xuất đều phải đ/n thống nhất.
- Nếu đối với phương thức ảo ở lớp dẫn xuất, chúng ta lại sơ suất định nghĩa các tham số khác đi một chút thì trình biên dịch sẽ xem đó là phương thức khác. Đây chính là điều kiện để kết nối động.

Q & A