CHƯƠNG 5

Phương thức ảo và tính đa hình

NỘI DUNG

- Bài toán đa hình
- Vùng chọn kiểu
- Phương thức ảo
- Phương thức thiết lập ảo
- Phương thức ảo thuần tuý

BÀI TOÁN ĐA HÌNH

- Bài toán: Quản lý một danh sách các đối tượng có kiểu có thể khác nhau
- Hai vấn đề cần giải quyết : *Cách lưu trữ* và *thao tác xử lý*.
- *Về lưu trữ:* Ta có thể dùng union. Nhược điểm: lãng phí không gian lưu trữ.

  Giải pháp thay thế: lưu trữ đối tượng bằng đúng kích thước của nó và dùng một danh sách (mảng, dslk,...) các con trỏ để quản lý các đối tượng.

BÀI TOÁN ĐA HÌNH

- *Về thao tác*, phải thoả yêu cầu đa hình: Thao tác có hoạt động khác nhau ứng với các loại đối tượng khác nhau.
- Có hai cách giải quyết là *vùng chọn kiểu* và *phương thức ảo*.

VÙNG CHỌN KIỂU

- Xét lớp *Người* và các lớp kế thừa từ lớp Người là *sinh viên* và *công nhân*. Thao tác ta quan tâm là *Xuat.* Ta cần bảo đảm thao tác xuất áp dụng cho lớp sinh viên và lớp công nhân khác nhau.

```
class Person
{
protected:
    char *name;
    int birthYear;
    int ID;
public:
    Person(const char *n, int y, int id) : birthYear(y),
    ID(id) { name = __strdup(n); }
    ~Person() { delete[] name; }
    void print() const;
};

void Person::print() const
{
    cout << "Person, name: " << name << ", Identity: " <<
    ID << ", birth year: " << birthYear;
}
```

```
class Student : public Person
{
protected:
    char *major;
    int studentID;
    double GPA;
public:
    Student(const char *n, int y, int id, const char *m,
    int stId, double g) : Person(n, y, id),
    studentID(stId), GPA(g) { major = __strdup(m); }
    ~Student() { delete[] major; }
    void print() const; // Override Person::print
};

void Student::print() const
{
    cout << "Student, name: " << name << ", id: " <<
    studentID << ", major: " << major;
}
```

```
class Employee : public Person
{
   int empID;
   double salary;
public:
   Employee(const char *n, int y, int id, int eId,
   double s) : Person(n, y, id), empID(eId),
   salary(s) { }
   void print() const; // Override Person::print
};


void Employee::print() const
{
   cout << "Employee, name: " << name << ", id: " <<
   empID << ", salary: " << salary;
}
```

Test program 1

```
void printList(Person *ap[], int n)
{
   for (int i = 0; i < n; i++)
   {
      ap[i]->print();
      cout << "\n";
   }
}

int main()
{
   Person *a[] = {
     new Person("Le Van Nhan", 1980, 20978666),
     new Student("Vo Vien Sinh", 2000, 20978662, "Toan Ung
       Dung", 180194, 9.2),
     new Employee("Le Nhan Cong", 1994, 20971732, 2156, 5e6)
   };
   printList(a, 3);
}
```

Vùng chọn kiểu

Xuất liệu:

Person, name: Le Van Nhan, Identity: 20978666, birth year: 1980

Person, name: Vo Vien Sinh, Identity: 20978662, birth year: 2000

Person, name: Le Nhan Cong, Identity: 20971732, birth year: 1994

- Tất cả mọi đối tượng đều được quan điểm như người vì thao tác được thực hiện thông qua con trỏ đến lớp Person.

Test program 2

```
int main()
{
   Person p("Le Van Nhan", 1980, 20978666);
   Student s("Vo Vien Sinh", 2000, 20978662,
   "Toan Ung Dung", 180194, 9.2);
   Employee e("Le Nhan Cong", 1994, 20971732,
   2156, 5e6);

   p.print(); cout << "\n";
   s.print(); cout << "\n";
   e.print(); cout << "\n";

   Person *a[] = { &p, &s, &e };
   printList(a, 3);
}
```

Test program 2 output

```
Person, name: Le Van Nhan, Identity: 20978666, birth year: 1980
Student, name: Vo Vien Sinh, id: 180194, major: Toan Ung Dung
Employee, name: Le Nhan Cong, id: 2156, salary: 5e+06
Person, name: Le Van Nhan, Identity: 20978666, birth year: 1980
Person, name: Vo Vien Sinh, Identity: 20978662, birth year: 2000
Person, name: Le Nhan Cong, Identity: 20971732, birth year: 1994
```

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 Person có cùng giá trị cho vùng chọn kiểu, các đối tượng thuộc lớp Student có giá trị của vùng chọn kiểu khác của lớp Person.

```
class Person
{
   char *name;
   int birthyear, ID;
public:
   enum PTYPE { PERSON, STUDENT, EMPLOYEE };
protected:
   PTYPE personType;
public:
   Person(const char *n, int y, int id) : birthYear(y),
   ID(id), personType(PERSON) { name = __strdup(n); }
   Person(const Person &p) : birthYear(p.birthYear),
   ID(p.ID), personType(p.personType) { name =
   __strdup(p.name); }
   ~Person() { delete[] name; }
   PTYPE getPersonType() const { return personType; }
   void print() const;
   const char *getName() const { return name; }
};
```

```
class Student : public Person
{
   char *major;
   int studentID;
   double GPA;
public:
   Student(const char *n, int y, int id, const char
   *m, int stId, double g) : Person(n, y, id),
   studentID(stId), GPA(g) { major = __strdup(m);
   personType = STUDENT; }
   Student(const Student &s) : Person(s),
   studentID(s.studentID), GPA(s.GPA) { major =
   __strdup(s.major); }
   ~Student() { delete[] major; }
   void print() const; // Override Person::print
};
```

```
class Employee : public Person
{
   int empID;
   double salary;
   public:
   Employee(const char *n, int y, int id,
   int eId, double s) : Person(n, y, id),
   empID(eId), salary(s) { personType =
   EMPLOYEE; }
   void print() const; // Override
   Person::print
};
```

```
void printList(Person *ap[], int n)
{
   for (int i = 0; i < n; i++) {
      switch (ap[i]->getPersonType()) {
      case Person::PTYPE::STUDENT:
         ((Student *)ap[i])->print();
         break;
      case Person::PTYPE::EMPLOYEE:
         ((Employee *)ap[i])->print();
         break;
      default:
         ap[i]->print();
         break;
      }
      cout << "\n";
   }
}
```

Vùng chọn kiểu

```
Xuất liệu:
Person, name: Le Van Nhan, Identity: 20978666, birth year: 1980
Student, name: Vo Vien Sinh, id: 180194, major: Toan Ung Dung
Employee, name: Le Nhan Cong, id: 2156, salary: 5e+06
```

Vùng chọn kiểu

- Cách tiếp cận trên giải quyết được vấn đề:
  - Lưu trữ được các đối tượng khác kiểu nhau và
  - Thao tác khác nhau tương ứng với đối tượng.
- Tuy nhiên nó có các nhược điểm sau:
  - Dài dòng với nhiều switch, case.
  - Dễ sai sót, khó sửa vì trình biên dịch bị cơ chế ép kiểu che mắt.
  - Khó nâng cấp, ví dụ thêm một loại đối tượng mới, đặc biệt khi chương trình lớn.
- Các nhược điểm trên có thể được khắc phục nhờ phương thức ảo.

Phương thức ảo

- Con trỏ thuộc lớp cơ sở có thể trỏ đến lớp con:

```
Person *pn = new Student("Vo Vien Sinh",
2000, 20978662, "Toan Ung Dung", 18110094,
9.2);
```

- 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->print(); // Mong muon: goi Student::print
             // Thuc te: goi Person::print
```

Phương thức ảo

- C++ có cơ chế liên kết động cho phép, thông qua con trỏ, liên kết đúng với phương thức của đối tượng mà con trỏ đang trỏ đến, khi chương trình đang thực thi, nhờ *phương thức ảo*.
- Phương thức ảo cho phép giải quyết vấn đề đa hình kể trên. Ta qui định một hàm thành phần là phương thức ảo bằng cách thêm từ khoá *virtual* vào trước khai báo hàm.
- Trong ví dụ trên, ta thêm từ khoá virtual vào trước khai báo của hàm print.

```
class Person
{
protected:
    char *name;
    int birthYear;
    int ID;
public:
    Person(const char *n, int y, int id) : birthYear(y),
    ID(id) { name = __strdup(n); }
    ~Person() { delete[] name; }
    virtual void print() const;
};

void Person::print() const
{
    cout << "Person, name: " << name << ", Identity: " <<
    ID << ", birth year: " << birthYear;
}
```

```
class Student : public Person
{
protected:
    char *major;
    int studentID;
    double GPA;
public:
    Student(const char *n, int y, int id, const char *m,
    int stId, double g) : Person(n, y, id),
    studentID(stId), GPA(g) { major = __strdup(m); }
    ~Student() { delete[] major; }
    void print() const; // Override Person::print
};

void Student::print() const
{
    cout << "Student, name: " << name << ", id: " <<
    studentID << ", major: " << major;
}
```

```
class Employee : public Person
{
   int empID;
   double salary;
public:
   Employee(const char *n, int y, int id, int eId,
   double s) : Person(n, y, id), empID(eId),
   salary(s) { }
   void print() const; // Override Person::print
};


void Employee::print() const
{
   cout << "Employee, name: " << name << ", id: " <<
   empID << ", salary: " << salary;
}
```

```
void printList(Person *ap[], int n)
{
   for (int i = 0; i < n; i++)
   {
      ap[i]->print();
      cout << "\n";
   }
}

int main()
{
   Person *a[] = {
     new Person("Le Van Nhan", 1980, 20978666),
     new Student("Vo Vien Sinh", 2000, 20978662, "Toan Ung
       Dung", 180194, 9.2),
     new Employee("Le Nhan Cong", 1994, 20971732, 2156, 5e6)
   };
   printList(a, 3);
}
```

Phương thức ảo

- Phương thức ảo print được khai báo ở lớp Person cho phép sử dụng con trỏ đến lớp cơ sở (Person) nhưng trỏ đến một đối tượng thuộc lớp con (Student, Employee) gọi đúng thao tác ở lớp con:

```
Person *pn = new Student("Vo Vien Sinh",
2000, 20978662, "Toan Ung Dung", 18110094,
9.2);
pn->print(); // Goi Student::print
```

Phương thức ảo

- Dùng phương thức ảo khắc phục được các nhược điểm của cách tiếp cận dùng vùng chọn kiểu:
  - Thao tác đơn giản không phải dùng switch/case vì vậy khó sai, dễ sửa.
  - Dễ dàng nâng cấp sửa chữa. Việc thêm một loại đối tượng mới rất đơn giản, ta không cần phải sửa đổi thao tác xử lý (như printList). Qui trình thêm chỉ là xây dựng lớp con mới kế thừa từ lớp cơ sở hoặc các lớp con đã có và định nghĩa lại phương thức (ảo) ở lớp mới tạo nếu cần.

```
class Singer : public Person
{
protected:
double cashier;
public:
Singer(const char *n, int y, int id,
double c) : Person(n, y, id), cashier(c)
{ }
void print() const; // Override
Person::print
};
```

Thêm đối tượng mới

```
void printList(Person *ap[], int n)
{
   for (int i = 0; i < n; i++)
   {
      ap[i]->print();
      cout << "\n";
   }
}

int main()
{
   Person *a[] = {
     new Person("Le Van Nhan", 1980, 20978666),
     new Student("Vo Vien Sinh", 2000, 20978662, "Toan Ung Dung",
       180194, 9.2),
     new Employee("Le Nhan Cong", 1994, 20971732, 2156, 5e6),
     new Singer("Doi Truong", 1998, 209214356, 5e7)
   };
   printList(a, 4);
}
```

Phương thức ảo

- Phương thức printLish không thay đổi, nhưng nó có thể hoạt động cho các loại đối tượng ca sĩ thuộc lớp mới ra đời.
- Có thể xem như thao tác printList được viết trước cho các lớp (con, cháu…) chưa ra đời.
- Ví dụ thêm về phương thức ảo: Mamal.cpp

Output

```
(1)dog (2)cat (3)horse (4)pig: 3
(1)dog (2)cat (3)horse (4)pig: 1
(1)dog (2)cat (3)horse (4)pig: 4
(1)dog (2)cat (3)horse (4)pig: 2
(1)dog (2)cat (3)horse (4)pig: 0
Winnie!
Woof!
Oink!
Meow!
Mammal speak!
```

Cơ chế thực hiện phương thức ảo

- Khi gọi một thao tác, khả năng chọn đúng phiên bản tuỳ theo đối tượng để thực thi thông qua con trỏ đến lớp cha có được nhờ được kết nối động. Nhờ đó ta hiện thực được là tính đa hình (polymorphisms).
- Cơ chế đa hình được thực hiện nhờ ở mỗi lớp đối tượng có thêm một bảng phương thức ảo và mỗi đối tượng có một con trỏ đến bảng này. 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.
- Khi thao tác được thực hiện thông qua con trỏ, hàm có địa chỉ trong bảng phương thức ảo sẽ được gọi.

Bảng phương thức ảo

- Trong ví dụ trên, mỗi đối tượng thuộc lớp cơ sở `Person` có bảng phương thức ảo có một phần tử là địa chỉ hàm `Person::print`. Mỗi đối tượng thuộc lớp `Student` có bảng tương tự nhưng nội dung là địa chỉ của hàm `Student::print`.
- Trong ví dụ mamal, mỗi đối tượng thuộc các lớp `Mamal, Dog, Cat, Horse, Pig` đều có bảng phương thức ảo với hai phần tử, địa chỉ của hàm `Speak` và của hàm `Move`.

Phương thức huỷ bỏ ảo

- Phương thức huỷ bỏ là cần thiết ở các lớp có dữ liệu luận lý (có cấp phát tài nguyên).
- Để có thể gọi đúng phương thức huỷ bỏ ở lớp con thông qua con trỏ đến lớp cơ sở, ta khai báo phương thức huỷ bỏ ở lớp cơ sở là ảo.

```
class Person
{
protected:
    char *name;
    int birthYear;
    int ID;
public:
    Person(const char *n, int y, int id) : birthYear(y),
    ID(id) { name = __strdup(n); }
    virtual ~Person() { delete[] name; }
    virtual void print() const;
};

void Person::print() const
{
    cout << "Person, name: " << name << ", Identity: " <<
    ID << ", birth year: " << birthYear;
}
```

Phương thức huỷ bỏ ảo

```
int main()
{
   Person *a[] = {
     new Person("Le Van Nhan", 1980, 20978666),
     new Student("Vo Vien Sinh", 2000, 20978662, "Toan
       Ung Dung", 180194, 9.2),
     new Employee("Le Nhan Cong", 1994, 20971732, 2156,
     5e6),
     new Singer("Doi Truong", 1998, 209214356, 5e7)
   };
   printList(a, 4);
   const int N = sizeof(a) / sizeof(a[0]);
   printList(a, N);
   for (int i = 0; i < N; i++)
   delete a[i];
}
```

Đa hình trong Objective C

- Xem ví dụ minh hoạ các lớp Person, Student và Employee trong Xcode.

Phương thức thiết lập ảo

- C++ không cung cấp cơ chế thiết lập đối tượng có khả năng đa hình theo cơ chế hàm thành phần ảo.
- Tuy nhiên ta có thể "thu xếp" để có thể tạo đối tượng theo nghĩa "ảo". Xem ví dụ mamal02.cpp sau đây, hàm clone cho phép tạo đối tượng tùy theo lớp.

Phương thức ảo thuần tuý và lớp cơ sở trừu tượng

- Lớp cơ sở trừu tượng là lớp cơ sở không có đối tượng nào thuộc chính nó. Một đối tượng thuộc lớp cơ sở trừu tượng phải thuộc một trong các lớp con.
- Ví dụ: Lớp động vật, lớp đồ đạc, lớp xe cộ…
- Ta không muốn có một đối tượng nào thuộc lớp cơ sở trừu tượng tồn tại trong hệ thống.

Phương thức ảo thuần tuý và lớp cơ sở trừu tượng

- Lớp cơ sở trừu tượng là lớp cơ sở không có đối tượng nào thuộc chính nó. Một đối tượng thuộc lớp cơ sở trừu tượng phải thuộc một trong các lớp con.
- Ví dụ: Lớp động vật, lớp đồ đạc, lớp xe cộ…
- Ta không muốn có một đối tượng nào thuộc lớp cơ sở trừu tượng tồn tại trong hệ thống. Điều này có thể được thực hiện bằng cách dung phương thức thiết lập protected hoặc phương thức ảo thuần tuý.

Phương thức ảo thuần tuý và lớp cơ sở trừu tượng

- Phương thức ảo thuần tuý là phương thức ảo không có nội dung.
- Ta không cần định nghĩa phương thức ảo thuần tuý.
- Người sử dụng vẫn có quyền định nghĩa phương thức ảo thuần tuý nếu muốn nhưng chỉ các lớp con mới có quyền truy xuất.
- Khi một lớp có phương thức ảo thuần tuý, nó trở thành lớp cơ sở trừu tượng.

Phương thức ảo thuần tuý và lớp cơ sở trừu tượng

- Bản than lớp con của lớp cơ sở trừu tượng cũng có thể là lớp cơ sở trừu tượng.
- Xem các ví dụ: Lớp Animal, Mamal và các lớp con. Lớp Shape và các lớp con.