LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG (C++)

Giảng viên:

Đặng Hoài Phương

Bộ môn:

Công nghệ phần mềm

Khoa:

Công nghệ Thông tin

Trường Đại học Bách Khoa

Đại học Đà Nẵng

IT FACULTY

GIỚI THIỆU

- Đặng Hoài Phương.
- E-mail: danghoaiphuongdn@gmail.com
- Tel: 0935578555
- Khoa Công nghệ thông tin, trường Đại học Bách khoa, Đại học Đà Nẵng.

OBJECT RELATIONSHIP

OBJECT RELATIONSHIPS

- Relation types:
  - Inheritance
  - Composition
  - Aggregation
  - Association
  - Dependency
  - Container classes

COMPOSITION

- Composition:
  - Point2D.h

```
#include <iostream>
using namespace std;
class Point2D
{
    private:
        int xVal, yVal;
    public:
        Point2D(int = 1, int = 2);
        ~Point2D();
        void setPoint(int, int);
        friend ostream& operator<<(ostream&, const Point2D&);
};
```

COMPOSITION

- Composition:
  - Point2D.cpp

```
#include "Point2D.h"
Point2D::Point2D(int xVal, int yVal)
{
    this->xVal = xVal;
    this->yVal = yVal;
}
Point2D::~Point2D()
{
}
void Point2D::setPoint(int x, int y)
{
    this->xVal = x;
    this->yVal = y;
}
ostream & operator<<(ostream & o, const Point2D & p)
{
    o << "(" << p.xVal << ", " << p.yVal << ")" << endl;
    return o;
}
```

COMPOSITION

- Composition:
  - Creature.h

```
#include <string>
#include "Point2D.h"
using namespace std;
class Creature
{
    string name;
    Point2D location;
public:
    Creature(const string &, const Point2D &);
    ~Creature();
    friend ostream& operator<<(ostream &, const Creature &);
    void moveTo(int, int);
};
```

COMPOSITION

- Composition:
  - Creature.cpp

```
#include "Creature.h"
Creature::Creature(const string & name, const Point2D & location)
    : name(name), location(location)
{ }
Creature::~Creature()
{ }
void Creature::moveTo(int x, int y)
{
    this->location.setPoint(x, y);
}
ostream & operator<<(ostream & o, const Creature & c)
{
    o << c.name << " is at " << c.location << endl;
    return o;
}
```

COMPOSITION

- Composition:
  - Source.cpp

```
#include "Creature.h"
int main()
{
    Creature c("Creature", Point2D(4, 7));
    cout << c;
    int x = 0, y = 0;
    cout << "Enter new X location for creature: ";
    cin >> x;
    cout << "Enter new Y location for creature: ";
    cin >> y;
    c.moveTo(x, y);
    cout << c;
    system("pause");
    return 0;
}
```

AGGREGATION

- Aggregation:
  - Teacher.h

```
#include <iostream>
#include <string>
using namespace std;
class Teacher
{
    string name;
    int age;
    public:
        Teacher(string = "", int = 0);
        ~Teacher();
        friend ostream& operator<<(ostream&, const Teacher&);
};
```

AGGREGATION

- Aggregation:
  - Teacher.cpp

```
#include "Teacher.h"
Teacher::Teacher(string name, int age)
    : name(name), age(age)
{ }
Teacher::~Teacher()
{ }
ostream & operator<<(ostream &o, const Teacher &t)
{
    o << t.name << ", age: " << t.age << endl;
    return o;
}
```

AGGREGATION

- Aggregation:
  - Department.h

```
#include "Teacher.h"
class Department
{
    Teacher *teacher;
    string nameDep;
    public:
        Department(string = "", Teacher* = nullptr);
        ~Department();
        friend ostream& operator<<(ostream&, const Department&);
};
```

AGGREGATION

- Aggregation:
  - Department.cpp

```
#include "Department.h"
Department::Department(string name, Teacher * teacher)
    : nameDep(name), teacher(teacher)
{ }
Department::~Department()
{
    delete this->teacher;
}
ostream & operator<<(ostream &o, const Department &d)
{
    o << d.nameDep << ": " << *d.teacher << endl;
    return o;
}
```

AGGREGATION

- Aggregation:
  - Source.cpp

```
#include "Department.h"
int main()
{
    Teacher *t = new Teacher("NVA", 40);
    {
        Department d("CNTT", t);
        cout << d;
    }
    cout << *t;
    delete t;
    system("pause");
    return 0;
}
```

AGGREGATION – VECTOR

- Aggregation: Department can handle multiple Teachers ??? → **#include <vector>** for class Department.
- Vector là mảng động có các phần tử sắp xếp tuần tự tuyến tính, ta có thể truy cập đến từng phần tử của vector thông qua chỉ số của phần tử đó.

AGGREGATION – VECTOR

- Aggregation: Department can handle multiple Teachers ??? → **#include <vector>** for class Department.
- **Vector** là mảng động có các phần tử sắp xếp tuần tự tuyến tính, ta có thể truy cập đến từng phần tử của vector thông qua chỉ số của phần tử đó.
- **Vector** có thể thêm 1 vùng nhớ mở rộng gọi là **capacity**, để vector có thể thêm 1 số phần tử mà không phải cấp phát lại bộ nhớ.

AGGREGATION – VECTOR

- Aggregation: Department can handle multiple Teachers ??? → **#include <vector>** for class Department.
  - **Vector** là mảng động có các phần tử sắp xếp tuần tự tuyến tính, ta có thể truy cập đến từng phần tử của vector thông qua chỉ số của phần tử đó.
  - Vector có thể thêm 1 vùng nhớ mở rộng gọi là capacity → thêm 1 số phần tử mà không phải cấp phát lại bộ nhớ.
  - Các phần tử của vector được sắp xếp tuần tự tuyến tính → truy cập đến các phần tử của vector thông qua chỉ số của phần tử đó.
  - Cho phép truy cập trực tiếp vào phần tử nào của vector, có thể thêm, sửa, xóa các phần tử.
  - Vector sử dụng một đối tượng để cấp phát bộ nhớ tự động cho quá trình lưu trữ → kích thước của vector sẽ tự động thay đổi khi ta thêm hoặc bớt phần tử.

AGGREGATION – VECTOR

- Hàm về Iterator

Tên hàm	Chức năng
begin	Trả về iterator trỏ vào phần tử đầu tiên
end	Trả về iterator trỏ vào phần tử cuối cùng
cbegin	Trả về const_iterator trỏ vào phần tử đầu tiên
cend	Trả về const_iterator trỏ vào phần tử cuối cùng
rbegin	Trả về iterator trỏ vào phần tử đầu tiên của containers bị đảo ngược
rend	Trả về iterator trỏ vào phần tử cuối cùng của containers bị đảo ngược
rcbegin	Trả về const_iterator trỏ vào phần tử đầu tiên của containers bị đảo ngược
rcend	Trả về const_iterator trỏ vào phần tử cuối cùng của containers bị đảo ngược

AGGREGATION – VECTOR

- Hàm về capacity

Tên hàm	Chức năng
size	Trả về kích thước (số phần tử) của vector
max_size	Trả về số phần tử lớn nhất mà vector có thể lưu trữ.
resize	Thay đổi kích thước (số phần tử) của vector
empty	Kiểm tra xem vector có rỗng không
capacity	Trả về kích thước của vector được phân bổ, thường bằng hoặc lớn hơn size
reserver	Thay đổi capacity của vector

AGGREGATION – VECTOR

- Hàm về truy cập các phần tử

Tên hàm	Chức năng
operator[]	Truy cập phần từ của mảng theo index arr[i].
at	Truy cập phần tử của mảng theo index arr.at(i).
front	Truy cập phần tử đầu tiên của mảng.
end	Truy cập phần tử cuối cùng của mảng.
data	Trả về con trỏ đại diện cho mảng.

AGGREGATION – VECTOR

- Hàm modify data

Tên hàm	Chức năng
assign	Gán giá trị cho vector, replace tất cả các phần tử, modify size.
push_back	Thêm phần tử vào cuối vector
pop_back	Xoá phần tử cuối vector
insert	Thêm một hoặc nhiều phần tử vào vector
erase	Xoá phần tử của vector
clear	Xoá toàn bộ vector
swap	Chuyển dữ liệu hai vector có cùng kiểu dữ liệu.

AGGREGATION – VECTOR

- Aggregation:
  - Department.h

```
#include <vector>
#include "Teacher.h"
class Department
{
    vector<Teacher *> teacher;
    string nameDept;
    public:
        Department(string);
        ~Department();
        void addTeacher(Teacher *);
        friend ostream& operator<<(ostream&, const Department&);
};
```

AGGREGATION – VECTOR

- Aggregation:
  - Department.cpp

```
#include "Department.h"
Department::Department(string name)
    : nameDept(name)
{ }
Department::~Department()
{
    this->teacher.clear();
}
void Department::addTeacher(Teacher *t)
{
    this->teacher.push_back(t);
}
ostream & operator<<(ostream &o, const Department &d)
{
    o << d.nameDept << ": " << endl;
    for (unsigned int i = 0; i < d.teacher.size(); i++)
        o << *d.teacher[i];
    o << endl;
    return o;
}
```

AGGREGATION – VECTOR

- Aggregation:
  - Source.cpp

```
#include "Department.h"
int main()
{
    Teacher *t1 = new Teacher("Bob");
    Teacher *t2 = new Teacher("Frank");
    Teacher *t3 = new Teacher("Beth");
    {
        Department dept("CNTT");
        dept.addTeacher(t1);
        dept.addTeacher(t2);
        dept.addTeacher(t3);
        cout << dept;
    }
    cout << *t1 << *t2 << *t3;
    delete t1;
    delete t2;
    delete t3;
    system("pause");
    return 0;
}
```

ASSOCIATION

- Association:
  - Patient.h

```
#include <iostream>
#include <string>
#include <vector>
#include "Doctor.h"
using namespace std;
class Doctor;
class Patient
{
    string nameP;
    vector<Doctor *> doctor;
    void addDoctor(Doctor *);
    public:
        Patient(string = NULL);
        ~Patient();
        string getName() const;
        friend ostream& operator<<(ostream &, const Patient &);
        friend class Doctor;
};
```

ASSOCIATION

- Association:
  - Patient.cpp

```
#include "Patient.h"
void Patient::addDoctor(Doctor *d)
{
    this->doctor.push_back(d);
}
Patient::Patient(string name)
    : nameP(name)
{ }
Patient::~Patient()
{ }
string Patient::getName() const
{
    return this->nameP;
}
```

ASSOCIATION

- Association:
  - Doctor.h

```
#include "Patient.h"
using namespace std;
class Patient;
class Doctor
{
    string nameD;
    vector<Patient *> patient;
    public:
        Doctor(string = NULL);
        ~Doctor();
        void addPatient(Patient *);
        string getName() const;
        friend ostream& operator<<(ostream &, const Doctor &);
};
```

ASSOCIATION

- Association:
  - Doctor.cpp

```
#include "Doctor.h"
Doctor::Doctor(string name)
    : nameD(name)
{ }
Doctor::~Doctor()
{ }
void Doctor::addPatient(Patient *p)
{
    this->patient.push_back(p);
    p->addDoctor(this);
}
string Doctor::getName() const
{
    return this->nameD;
}
```

ASSOCIATION

- Association:
  - Doctor.cpp

```
ostream & operator<<(ostream &o, const Doctor &d)
{
    int number = d.patient.size();
    if (number == 0)
    {
        o << d.nameD << " has no patients right now" << endl;
    }
    else
    {
        o << d.nameD << " is seeing patients: ";
        for (int i = 0; i < number; i++)
            o << d.patient[i]->getName() << " ";
        o << endl;
    }
    return o;
}
```

ASSOCIATION

- Association:
  - Source.cpp

```
#include "Patient.h"
int main()
{
    Patient *p1 = new Patient("Dave");
    Patient *p2 = new Patient("Frank");
    Patient *p3 = new Patient("Betsy");
    Doctor *d1 = new Doctor("James");
    Doctor *d2 = new Doctor("Scott");
    d1->addPatient(p1);
    d2->addPatient(p1);
    d2->addPatient(p3);
    cout << *d1; cout << *d2;
    cout << *p1; cout << *p2; cout << *p3;
    delete p1; delete p2; delete p3;
    delete d1; delete d2;
    system("pause");
    return 0;
}
```

INDIRECT ASSOCIATION

- Association:
  - Car.h

```
#include <iostream>
#include <string>
using namespace std;
class Car
{
    string nameCar;
    int ID;
    public:
        Car(string = NULL, int = 0);
        ~Car();
        string getName() const;
        int getID() const;
};
```

INDIRECT ASSOCIATION

- Association:
  - Car.cpp

```
#include "Car.h"
Car::Car(string name, int ID)
    : nameCar(name), ID(ID)
{ }
Car::~Car()
{ }
string Car::getName() const
{
    return this->nameCar;
}
int Car::getID() const
{
    return this->ID;
}
```

INDIRECT ASSOCIATION

- Association:
  - ListCar.h

```
#include "Car.h"
class ListCar
{
    static Car car[4];
    public:
        ListCar() = delete;
        ~ListCar();
        static Car* getCar(int);
};
```

INDIRECT ASSOCIATION

- Association:
  - ListCar.cpp

```
#include "ListCar.h"
Car ListCar::car[4] = { Car("Prius", 4),
                        Car("Corolla", 17),
                        Car("Accord", 84),
                        Car("Matrix", 62) };
ListCar::~ListCar()
{ }
Car * ListCar::getCar(int ID)
{
    for (int i = 0; i < 4; i++)
        if (car[i].getID() == ID)
            return &(car[i]);
    return nullptr;
}
```

INDIRECT ASSOCIATION

- Association:
  - Driver.h

```
#include <iostream>
using namespace std;
class Driver
{
    string nameDriver;
    int ID; //ID of object Car
    public:
        Driver(string = NULL, int = 0);
        ~Driver();
        string getName() const;
        int getCarID() const;
};
```

INDIRECT ASSOCIATION

- Association:
  - Driver.cpp

```
#include "Driver.h"
Driver::Driver(string name, int ID)
    : nameDriver(name), ID(ID)
{ }
Driver::~Driver()
{ }
string Driver::getName() const
{
    return this->nameDriver;
}
int Driver::getCarID() const
{
    return this->ID;
}
```

INDIRECT ASSOCIATION

- Association:
  - Source.cpp

```
#include "Driver.h"
#include "Car.h"
#include "ListCar.h"
int main()
{
    //Franz is driving the car with ID 17
    Driver d("Franz", 17);
    // Get that car from the car lot
    Car *car = ListCar::getCar(d.getCarID());
    if (car)
        cout << d.getName() << " is driving a "
        << car->getName() << endl;
    else
        cout << d.getName()
        << " couldn't find his car" << endl;
    system("pause");
    return 0;
}
```

CONTAINER CLASSES

- 2 type:
  - Value containers are compositions;
  - Reference containers are aggregations.

CONTAINER CLASSES

- List.h

```
#include <iostream>
using namespace std;
class List
{
    int length;
    int *data;
    public:
        List();
        List(int);
        ~List();
        void Erase();
        int& operator[](int);
        int GetLength() const;
        void Reallocate(int);   //fast
        void Resize(int);       //slow
        void Insert(int, int);
        void Remove(int);
        void InsertFirst(int);
        void InsertLast(int);
        friend ostream& operator<<(ostream&, const List&);
};
```

CONTAINER CLASSES

- List.cpp

```
#include "List.h"
List::List() : length(0), data(nullptr)
{ }
List::List(int length) : length(length)
{
    assert(this->length >= 0);
    if(length > 0)
        this->data = new int[this->length];
    else
        this->data = nullptr;
}
List::~List()
{
    delete[] this->data;
}
void List::Erase()
{
    delete[] this->data;
    this->data = nullptr;
    this->length = 0;
}
```

CONTAINER CLASSES

- List.cpp

```
int & List::operator[](int index)
{
    assert(index >= 0 && index < this->length);
    return this->data[index];
}
int List::GetLength() const
{
    return this->length;
}
ostream & operator<<(ostream &o, const List &l)
{
    o << "List = ";
    for (int i = 0; i < l.length; i++)
    {
        o << l.data[i];
    }
    o << endl;
    return o;
}
```

CONTAINER CLASSES

- List.cpp

```
void List::Reallocate(int newLength)
{
    Erase();
    if (newLength <= 0)
        return;
    this->data = new int[newLength];
    this->length = newLength;
}
void List::Resize(int newLength)
{
    if (newLength == this->length)
        return;
    if (newLength <= 0)
    {
        Erase(); return;
    }
    int *data = new int[newLength];
    if (this->length > 0)
    {
        int maxlength = (newLength > this->length) ? this->length : newLength;
        for (int i = 0; i < maxlength; ++i)
            data[i] = (*this)[i];
    }
    delete[] this->data;
    this->data = data;
    this->length = newLength;
}
```

CONTAINER CLASSES

- List.cpp

```
void List::Insert(int value, int index)
{
    assert(index >= 0 && index <= this->length);
    int *data = new int[this->length + 1];
    for (int i = 0; i < index; ++i)
        data[i] = (*this)[i];
    data[index] = value;
    for (int j = index; j < this->length; ++j)
        data[j + 1] = (*this)[j];
    delete[] this->data;
    this->data = data;
    ++this->length;
}
void List::InsertFirst(int value)
{
    Insert(value, 0);
}
void List::InsertLast(int value)
{
    Insert(value, this->length);
}
```

CONTAINER CLASSES

- List.cpp

```
void List::Remove(int index)
{
    assert(index >= 0 && index < this->length);
    if (this->length == 1)
    {
        Erase();
        return;
    }
    int *data = new int[this->length - 1];
    for (int i = 0; i < index; ++i)
        data[i] = (*this)[i];
    for (int j = index + 1; j < this->length; ++j)
        data[j - 1] = (*this)[j];
    delete[] this->data;
    this->data = data;
    --this->length;
}
```

CONTAINER CLASSES

- Source.cpp

```
#include "List.h"
int main()
{
    List arr(10);                cout << arr;
    for (int i = 0; i < 8; i++)
        arr[i] = i + 1;
    cout << arr;
    arr.Resize(8);               cout << arr;
    arr.Insert(0, 5);            cout << arr;
    arr.Remove(8);               cout << arr;
    arr.InsertFirst(0);          cout << arr;
    arr.InsertLast(8);           cout << arr;
    system("pause");
    return 0;
}
```

Thank You !

IT FACULTY