Chương 2

Lớp và đối tượng (phần 1)

Nội dung

- 2.1 Giới thiệu lập trình hướng đối tượng (OOP)
- 2.2 Kiểu dữ liệu class (Class)
- 2.3 Hàm thành phần (Member function)
- 2.4 Hàm tạo (Constructor)
- 2.5 Hàm hủy (Destructor)

2.1 Giới thiệu lập trình hướng đối tượng

- Lập trình hướng đối tượng là một mô hình và phương pháp lập trình rất phổ biến hiện nay
- Rất nhiều ngôn ngữ lập trình hiện nay sử dụng mô hình này như: C++, Java, C#, PHP, Python, Javascript, TypeScript, Dart, Kotlin, Scala...
- Lập trình hướng đối tượng là phần quan trọng nhất của ngôn ngữ lập trình C++

So sánh lập trình hướng đối tượng và lập trình hướng thủ tục

Lập trình hướng thủ tục (ngôn ngữ C)

- Chương trình sử dụng **biến** để lưu dữ liệu, sử dụng **hàm** (function) để thực hiện các chức năng thông qua xử lý dữ liệu đó.
- **Dữ liệu và hàm xử lý dữ liệu là các thành phần riêng biệt và tách rời nhau**

Lập trình hướng đối tượng (ngôn ngữ C++)

Ví dụ về xử lý ma trận trong lập trình thủ tục

- Dữ liệu ma trận sẽ được tạo riêng

```
int  m,n ; // số hàng và số cột
int  A[100][100];  // phần tử ma trận
```

- Hàm xử lý dữ liệu ma trận được đặt riêng
- Khi gọi hàm, dữ liệu ma trận sẽ truyền vào dưới dạng tham số của hàm

```
Ví dụ hàm cộng ma trận với 1 số nguyên k
void  addMatrix(int a[][100],int m, int n, k);
```


So sánh lập trình hướng đối tượng và lập trình hướng thủ tục

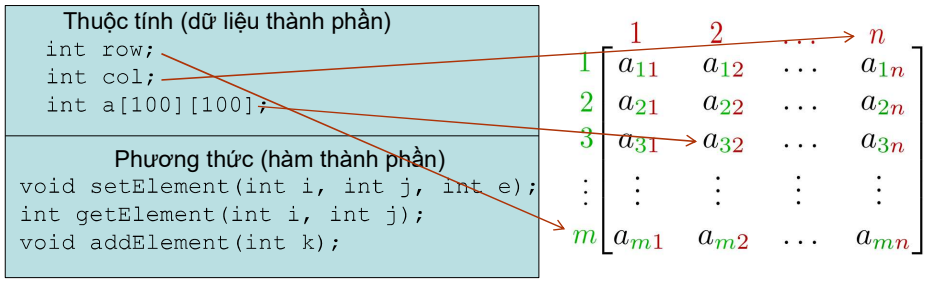
Lập trình hướng thủ tục (ngôn ngữ C)

- Chương trình sử dụng **biến** để lưu dữ liệu, sử dụng **hàm** (function) để thực hiện các chức năng thông qua xử lý dữ liệu đó.
- **Dữ liệu và hàm xử lý dữ liệu là các thành phần riêng biệt và tách rời nhau**

Lập trình hướng đối tượng (ngôn ngữ C++)

Ví dụ về kết hợp dữ liệu và hàm trong hướng đối tượng

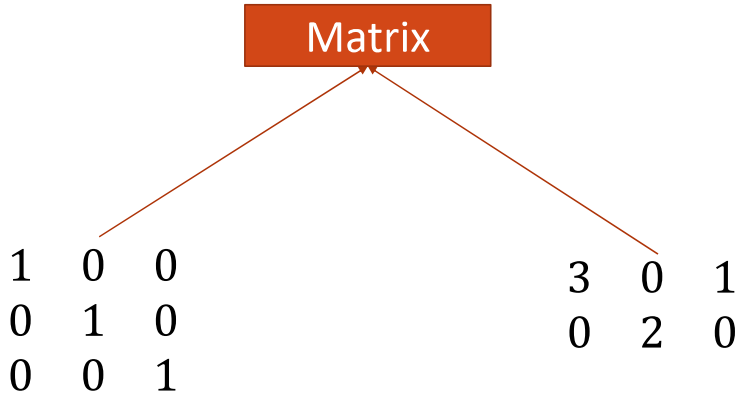
Đối tượng Matrix (ma trận)



Các khái niệm trong lập trình hướng đối tượng

- **Đối tượng (object)**: là một thành phần trong chương trình
- **Thuộc tính (attribute)**: là những đơn vị dữ liệu của đối tượng,
- **Phương thức (method)**: là các thao tác xử lý
- **Lớp (class)**: là phân lớp chung

Mối quan hệ giữa lớp và đối tượng



Các khái niệm trong Lập trình hướng đối tượng

- **Ẩn dữ liệu (data hiding)**: giới hạn truy nhập
- **Bao đóng (encapsulation)**: nhóm dữ liệu và hàm thành phần

Tại sao cần ẩn dữ liệu bên trong đối tượng ?

- **Bảo vệ dữ liệu**: Hàm thành phần tạo ra một class bảo vệ để
- **Ẩn đi các thao tác xử lý dữ liệu đối với bên sử dụng đối tượng:** chỉ cần sử dụng đối tượng và hàm thành phần để thực hiện công việc cần thiết

2.2) Khai báo kiểu dữ liệu class

- Để có đối tượng trong chương trình,
- Class là một kiểu dữ liệu do
- Khai báo kiểu dữ liệu class bao gồm
  - Đặt tên kiểu **class**
  - Khai báo các
  - Khai báo và định nghĩa

Cú pháp khai báo kiểu dữ liệu class

```
class <tên class>
{
  <phạm vi truy nhập>:
      dữ_liệu_thành_phần;
      hàm_thành_phần;
};
```


Phạm vi truy nhập

- Mỗi dữ liệu và hàm thành phần của class đều phải có phạm vi truy nhập nhất định.
- Nó được dùng để

Phạm vi truy nhập

- Mỗi thành phần sẽ được khai báo với một trong các kiểu phạm vi truy nhập sau
  - **public** (công khai) :
  - **private** (riêng):

Ví dụ khai báo class Matrix (ma trận)

```
class Matrix {
private:
    int row,col;
    int a[100][100];
public:
    void setElement(int i, int j, int e);
    int getElement(int i, int j);
};
```

Một số dạng hàm thành phần phổ biến

- **Hàm truy nhập lấy dữ liệu (get function)**:

```
getElement :  lấy về giá trị phần tử của ma trận
```

- **Hàm thay đổi, đặt giá trị cho dữ liệu (set function)**

```
setElement :  đặt giá trị mới cho phần tử ma trận
```


Một số quy tắc khi khai báo class

- Dữ liệu thành phần thường có phạm vi **`private`**
- Các hàm thành phần thường là **`public`**
- Sử dụng 'get' để bắt đầu tên hàm lấy dữ liệu thành phần
- Sử dụng 'set' để bắt đầu tên hàm gán giá trị cho dữ liệu thành phần

Chương trình minh họa matrix1.cpp

- Khai báo class Matrix với các thành phần
  - Số hàng **row**
  - Số cột **col**
  - Phần tử **a[100][100]**
  - Hàm thành phần **getElement** trả về 1 phần tử nhất định của ma trận
  - Hàm thành phần **setElement** để gán giá trị cho 1 phần tử nhất định

2.3) Định nghĩa hàm thành phần

- Là
- Định nghĩa hàm thành phần như sau, trước tên hàm cần có

```
int Matrix::getElement(int i, int j)
{
    return a[i][j];
}
```

Chú ý

- Phần định nghĩa hàm thành phần cần

```
class Matrix {
    private:
          int row,col;
          int a[100][100];
    public:
          void setElement(int i, int j, int e);
}; // kết thúc khai báo class

// Định nghĩa hàm setRow
void Matrix::setElement(int i, int j, int e){
     a[i][j] = e;
}
```


Chương trình minh họa matrix2.cpp

- Bổ sung khai báo hàm thành phần
  - getRow()
  - getCol()
- Viết phần định nghĩa cho hàm thành phần
  - getElement()
  - setElement()
  - getRow()
  - getCol()

2.4 Hàm tạo (constructor)

- Đối tượng sẽ được tạo ra khi
- Hàm tạo là hàm

Tác dụng của hàm tạo constructor

- Khởi tạo giá trị ban đầu cho
- Hàm tạo phải có phạm vi truy nhập **public**
- Tên hàm tạo phải trùng với

Ví dụ về hàm tạo

Hàm tạo 2 tham số:

```
class Matrix
{
  //. . .
  public:
    Matrix(int r, int c);
  //. . .
};

Matrix::Matrix(int r, int c)
{
   row = r; // khởi tạo
   col = c;
}
```


Nhiều hàm tạo khác nhau về tham số

- Một class có thể có nhiều

```
Matrix();  // không tham số
Matrix(int size); // 1 tham số
Matrix(int r, int c); // 2 tham số
```


Hàm tạo không tham số được gọi là

```
class Matrix
{
  private:
    int row, col;
    int a[100][100];

  public:
    Matrix(); // hàm tạo mặc định

};


Matrix::Matrix() // hàm tạo mặc định
{
   row = col = 0;
}
```


Hàm tạo được gọi như thế nào

- Hàm tạo của 1 đối tượng **sẽ tự động được gọi** khi
- Để tạo 1 đối tượng và gọi hàm tạo mặc định,
- Để tạo đối tượng và gọi hàm tạo có tham số, danh sách tham số

Sử dụng đối tượng sau khi khởi tạo

- Gọi hàm thành phần của đối tượng theo cú pháp sau

```
<đối tượng>.<tên hàm>(<tham số thực sự>);
```


Chú ý khi sử dụng đối tượng

- Chỉ gọi hàm thành phần có
- **Không** truy nhập trực tiếp vào
- Các dữ liệu và hàm thành phần có phạm vi

- Ví dụ

```
…
//  hàm thành phần là nội bộ của class Matrix
void Matrix::setElement(int i, int j, int e){
     a[i][j] = e; // ok
}

// hàm main không phải là nội bộ của class Matrix
int main() {
   Matrix m1, m2;
   m1.setElement(0,0,5);  // ok vì hàm là public
   m2.a[0][0] = 4 ;       // sai vì a là private
 }
```


2.5) Hàm hủy (destructor)

- Là hàm thành phần được
- Cú pháp *~<tên class>();*
- Hàm hủy không có kiểu trả về
- Không có tham số
- Mỗi class

Khi nào 1 đối tượng kết thúc tồn tại

- Khi chương trình thực thi ra khỏi
- Nếu đối tượng là biến toàn cục (tổng thể) thì đối tượng hủy khi
- Nếu đối tượng là biến cục bộ của hàm thì sẽ hủy khi

Ví dụ hàm hủy

```
class Matrix
{
  private:
    int row, col;
    int a[100][100];
  public:
    Matrix(){ // hàm tạo mặc định
       row = col = 1;
    }
    ~Matrix(); // hàm hủy

};
Matrix::~Matrix(){
  cout << "Matrix destructor" << endl;
}
```


Chương trình minh họa matrix3.cpp

- Khai báo các hàm tạo và hàm hủy
  - Hàm tạo mặc định Matrix()
  - Hàm tạo 1 tham số Matrix(int)
  - Hàm tạo 2 tham số Matrix(int, int)
  - Hàm hủy ~Matrix()
- Viết phần định nghĩa cho hàm tạo và hàm hủy
- Viết hàm main() để tạo các đối tượng lớp Matrix và gọi hàm thành phần trên đối tượng đã tạo.

Bài tập minh họa

- Hãy khai báo 1 class biểu diễn thông tin Đối tác bao gồm các thuộc tính sau (class DoiTac)
  - Họ tên
  - Tên công ty
  - Chức vụ
  - Số di động
  - Địa chỉ email
- Khai báo hàm tạo mặc định và hàm tạo có 2 tham số là họ tên, số di động
- Khai báo các hàm **get/set** cho tất cả các thuộc tính

- Viết định nghĩa cho tất cả các hàm tạo và hàm thành phần của class

- Viết hàm main()
  - Tạo 2 đối tượng DoiTac sử dụng 2 hàm tạo
  - Nhập dữ liệu cho 2 đối tượng
  - In ra thông tin (giá trị thuộc tính) của 2 đối tượng, mỗi thuộc tính trên 1 dòng