CHƯƠNG 5

LỚP VÀ ĐỐI TƯỢNG TRONG C++

Nội dung

1. Khái niệm lớp đối tượnng
2. Khai báo lớp trong C++
3. Phạm vi truy cập của thuộc tính và phương thức.
4. Phương thức khởi tạp và pgương thức hủy.
5. Các phương thức nhập/xuất
6. Mảng đối tượng
7. Hàm bạn và lớp
8. Các lớp có sẵn trong thư viện STL của C++

Khái niệm lớp đối tượng

- Đối tượng (object):
  - ✓ Là đơn vị cơ sở của lập trình hướng đối tượng, trong đó dữ liệu và các hàm xử lý trên dữ liệu được.
- Đặc trưng của đối tượng:
  - ✓ Đóng gói cả dữ liệu và xử lý.
  - ✓ Thuộc tính (attribute): dữ liệu của đối tượng.
  - ✓ Phương thức (method): xử lý của đối tượng.
- Cấu trúc đối tượng:
  - ✓ Hộp đen: thuộc tính trong, phương thức ngoài.
  - ✓ Bốn nhóm phương thức:
    - Nhóm tạo/hủy.
    - Nhóm truy xuất thông tin.
    - Nhóm xử lý nghiệp vụ.
    - Nhóm toán tử.

Khái niệm lớp đối tượng

- Lớp:
  - Là một từ khóa dùng để biểu diễn đối tượng:
    - ✓ Định nghĩa đối tượng,
    - ✓ Thiết kế đối tượng.
  - Khi biểu diễn đối tượng (định nghĩa, biểu diễn hoặc thiết kế) trong một lớp ta cần mô tả đối tượng về:
    - ✓ Đặc trưng gì (data abstraction)
    - ✓ Những hành vi nào của đối tượng áp đặt trên các đặc trưng đó (functional abstraction).
  - Từ khóa cho thành phần "lớp" là "class"

Khái niệm lớp đối tượng

Person1:
Name: Sampras.
Age: 39.
Hair Color: Black.
Eye Color: Brown.
Job: Coach.

Person2:
Name: Federer.
Age: 34.
Hair Color: Black.
Eye Color: Blue.
Job: Player.

Human:
Name.
Age.
Hair_Color.
Eye_Color.
Job.

Khai báo lớp trong C++

❖ Cấu trúc một lớp:

```
class   <class-name>
 {
    <Khai báo thuộc tính>;
    <Khai báo phương thức>;
};
```

- Khai báo lớp: file .h
- Cài đặt phương thức: file .cpp

Khai báo lớp trong C++

Ví dụ:

```
class  Shape //lớp có tên là Shape
{
    double      length;  //đặc trưng chiều dài của Shape
    double      width;   //đặc trưng chiều rộng của Shape
    double      height;  //đặc trưng chiều cao của Shape
};
```

- Khai báo đối tượng:

  class cho thấy hình hài của một đối tượng.

  Shape Box; *// Box là đối tượng kiểu Shape.*

- Truy nhập đến thành phần đối tượng: dùng toán tử ".".

  Ví dụ: Box.length = 6;

Khai báo lớp trong C++

- Ví dụ:

```
class PhanSo // file PhanSo.h
{
        private:
            int TuSo;
            int MauSo;
        public:
            PhanSo cong(PhanSo p);
};
// file PhanSo.cpp
PhanSo PhanSo::cong(PhanSo p)
{
        //Hiện thực cộng phân số
}
```

```
// file main.cpp
void main()
{
    PhanSo   p1;
    PhanSo   *p2 = new
    PhanSo;
    PhanSo   *p3 = new
    PhanSo[10];

    p3[1] = p1.cong(p3[5]);
    p3[1] = p2->cong(p3[5]);
}
```

Phạm vi truy cập của thuộc tính và phương thức

- Ẩn dấu dữ liệu, ẩn dấu chức năng là một trong những đặc trưng quan trọng của lập trình hướng đối tượng.
- Phạm vi truy cập thể hiện phần chương trình được phép truy cập đến.
- Phạm vi truy cập đến các thành phần của đối tượng dựa theo thuộc tính truy xuất.
- Các thuộc tính truy xuất:

Tầm vực	Tầm ảnh hưởng	Phạm vi hoạt động
private	Hẹp	Bên trong lớp.
public	Rộng	Bên trong lẫn bên ngoài lớp.
protected	Vừa	Bên trong lớp và lớp kế thừa.

Phạm vi truy cập của thuộc tính và phương thức

- **public**:
  - ✓ Phương thức hoặc thuộc tính sẽ được phép truy cập ở bất kỳ vị trí nào trong chương trình.
  - ✓ Có thể thiết lập, đọc giá trị cho thuộc tính
  - ✓ Có thể gọi phương thức từ bên ngoài class.
- **private**:
  - ✓ Phương thức hoặc thuộc tính sẽ chỉ được phép truy cập trong nội bộ lớp.
  - ✓ Chỉ có lớp hoặc hàm bạn (friend class, friend function) mới được phép truy cập đến các thành viên private.
- **protected**:
  - ✓ Khả năng truy cập tương tự với private.
  - ✓ Bổ sung thêm các lớp con của lớp cơ sở có quyền truy cập.

Phạm vi truy cập của thuộc tính và phương thức

- Toán tử "::" (bốn chấm):
  - Dùng gọi tên thành phần của lớp từ bên ngoài
  - Cú pháp:

    <Tên_lớp>::<Tên_thành_phần>

```
class PhanSo // file PhanSo.h
{
        private:
            int TuSo;
            int MauSo;
        public:
            PhanSo cong(PhanSo p);
};
```

```
// file PhanSo.cpp
PhanSo PhanSo::cong(PhanSo p)
{
        //Hiện thực cộng phân số
}
```

Phạm vi truy cập của thuộc tính và phương thức

- Con trỏ this:
  - Sử dụng bên trong lớp.
  - Đại diện cho đối tượng đang gọi phương thức.
  - Hữu dụng trong một số trường hợp.

```
class PhanSo
{
private:
        int       TuSo;
        int       MauSo;
public:
        void ganTuSo(int iTuSo)
        { this->TuSo = iTuSo; }
};
```

```
void main()
{
    PhanSo   p1;
    p1.ganTuSo(3);

    PhanSo   p2;
    p2.ganTuSo(5);
}
```

Phạm vi truy cập của thuộc tính và phương thức

- Thành viên tĩnh – từ khóa static:
  - Là biến toàn cục của lớp.
  - Phương thức không tĩnh có quyền truy cập đến thành viên không tĩnh.
  - Phương thức tĩnh có thể truy cập thành viên không tĩnh.
    - ✓ Tạo thể hiện (instance) của đối tượng
    - ✓ Dùng dấu “.” (hoặc “->”) để truy cập
  - Các phương thức tĩnh và không tĩnh có thể truy cập lẫn nhau.

Phạm vi truy cập của thuộc tính và phương thức

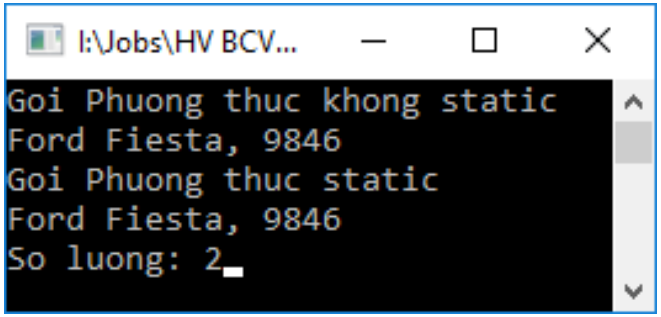
- Thành viên tĩnh – từ khóa static:
- Ví dụ:

```
class Car {
        private:
            string name;
            int serial;
        public:
            static int count;
            static void Show(){
            Car car;
            car.name="Ford Fiesta";
            car.serial=9846;
            cout<<car.name<<", "<<car.serial;}
            static void ShowStatic() {Show();}
            void ShowNonStatic() {Show();}
};
```

Phạm vi truy cập của thuộc tính và phương thức

- Thành viên tĩnh – từ khóa static:
- Ví dụ:

```
int Car::count = 2;
int main()
{
        Car c;
        cout<<“Goi Phuong thuc khong static\n”;
        c.ShowNonStatic();
        cout<<“Goi Phuong thuc static\n”;
        Car::ShowStatic();
        cout<<“So luong: ”<<Car::count;
}
```



Phương thức khởi tạo và phương thức hủy

- Phương thức khởi tạo (constructor):
  - ✓ Là một thành viên đặc biệt của lớp.
  - ✓ Tự động thực hiện khi ta tạo một đối tượng thuộc lớp.
  - ✓ Constructor có cùng tên lớp và có một số tính chất sau:
    - Mỗi lớp có tối đa một constructor dùng để xác thực đối tượng đã được tạo ra bởi class.
    - Tên của hàm constructor trùng với tên của lớp (đây là qui định của ngôn ngữ). Constructor không có kiểu và không trả lại bất kỳ giá trị nào kể cả kiểu void.
    - Hàm constructor được ngầm định là không có biến. Tuy vậy, ta có thể sử dụng biến cho constructor để khởi tạo các biến trong lớp tại thời điểm ta tạo ra đối tượng.
    - Không có kiểu trả về

Phương thức khởi tạo và phương thức hủy

- Ví dụ:

```
class Humans
{
    private:
        string name;
        int age;
    public:
        Humans();
        Humans(string, int);
        string getName();
        string getAge();
};
```

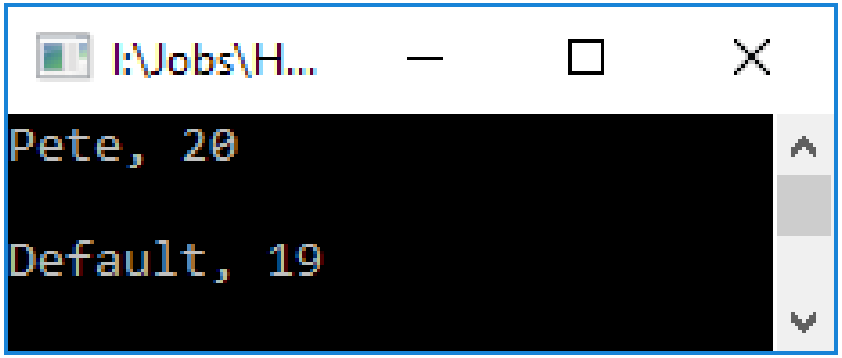
```
Humans::Humans(){
    name = "Default";
    age = 20;
}
Humans::Humans(string n, int a){
    name = n;
    age = a;
}
string Humans::getName(){
    return name;
}
int Humans::getAge(){
    return age;
}
```

Phương thức khởi tạo và phương thức hủy

❖ Ví dụ:

```
int main()
{
    Humans man("Pete",20);
    cout<<man.getName();
    cout<<", "<<man.getAge()<<"\n";

    Humans man2 = Humans();
    cout<<"\n"<<man2.getName();
    cout<<", "<<man2.getAge()<<"\n";
    return 0;
};
```



Phương thức khởi tạo và phương thức hủy

- Phương thức hủy (destructor):
  - ✓ Là một thành viên đặc biệt của lớp.
  - ✓ Tự động thực hiện khi đối tượng thuộc lớp đã ra khỏi phạm vi hoạt động của chương trình hoặc khi có toán tử delete hủy bỏ con trỏ đến đối tượng.
  - ✓ Destructor của lớp có một số tính chất sau:
    - Tên của destructor bắt đầu bằng ký tự "~" và có tên với tên của lớp.
    - Không sử dụng khai báo kiểu và tham biến cho destructor.
    - Destructor là một dạng giải phóng tài nguyên khi đối tượng không còn tồn tại.

Phương thức khởi tạo và phương thức hủy

❖ Ví dụ:

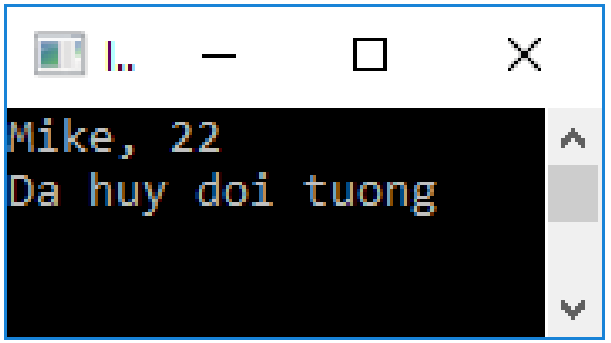
```
class Humans
{
    private:
        string name;
        int age;
    public:
        Humans();
        Humans(string, int);
        ~Humans();
        string getName();
        string getAge();
};
```

```
Humans::Humans(){
    name = "Default";
    age = 20;
}
Humans::Humans(string n, int a){
    name = n;
    age = a;
}
Humans::~Humans(){
    cout<<"\nDa huy doi tuong";
}
string Humans::getName(){
    return name;
}
int Humans::getAge(){
    return age;
}
```

Phương thức khởi tạo và phương thức hủy

- Ví dụ:

```
void TestDestructor(){
    Humans man("Mike",22);
    cout<<man.getName<<", "<<man.getAge();
}
int main()
{
     TestDestructor();
     return 0;
};
```



Các phương thức nhập/xuất

- Thuộc tính của lớp thường có thuộc tính truy xuất private.
- Cần có các phương thức để thực hiện thao tác đọc/ghi giá trị cho các thuộc tính của lớp.

```
class Humans
{
    private:
        string name;
        int age;
    public:
        void setName(string);
        void setAge(int);
        string getName();
        string getAge();
};
```

```
void setName(string s){
    name = s;
}
void setAge(int a){
    age = a;
}
string Humans::getName(){
    return name;
}
int Humans::getAge(){
    return age;
}
```

Mảng đối tượng

- Có thể dùng tên lớp để khai báo mảng đối tượng (giống như khai báo mảng int, float, char, ...) theo cú pháp:

  Tên_lớp Tên_mảng[kích_thước];

- Khai báo mảng đối tượng có khởi tạo:

  Tên_lớp Tên_mảng[kích_thước] =
  {Tên_lớp(các_tham_số), …, Tên_lớp(các_tham_số)};

- Biểu thị các thành phần của phần tử mảng:
  - Đối với thuộc tính:

    Tên_mảng[chỉ_số].Tên_thuộc_tính;

  - Đối với phương thức:

    Tên_mảng[chỉ_số].Tên_phương_thức(các_tham_số);

Hàm bạn và lớp

- Hàm bạn:
  - Nếu một thành viên của lớp là private hoặc protected thì chỉ các hàm thành viên của lớp mới có quyền truy cập.
  - Nếu một phương thức không phải là thành viên của lớp muốn truy cập chúng thì phải là hàm bạn của lớp đó.
  - Khai báo phương thức bạn bằng từ khóa friend.

Hàm bạn và lớp

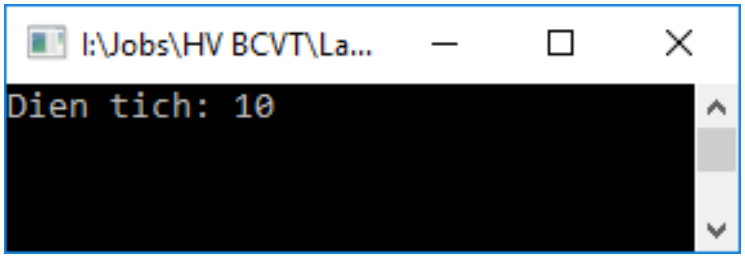
- Hàm bạn:
  - Ví dụ:

```
class Rectangle {
        private:
            int w;
            int h;
        public:
            Rectangle(int, int);
            friend int Area(Rectangle);
};
Rectangle::Rectangle(int w, int h) {
        this->w = w;
        this->h = h;
}
int Area(Rectangle r) {
        return (r.w * r.h) ;
}
```

Hàm bạn và lớp

- Hàm bạn:
  - Ví dụ:

```
int main()
{
    Rectangle rec(2, 5);
    cout<<“Dien tich: ”<<Area(rec);
    return 0;
};
```



Hàm bạn và lớp

- Lớp bạn:
  - Nếu lớp B là bạn của lớp A thì các phương thức của lớp A có thể truy cập đến các thuộc tính là private và protected của lớp B.

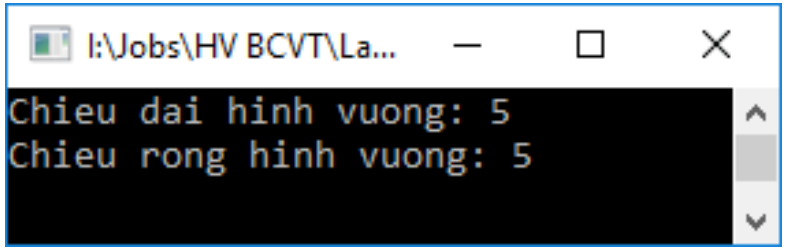
```
class Rectangle {
        private:
            int w;
            int h;
        public:
            Rectangle(int, int);
            friend class Square;
};
Rectangle::Rectangle(int w, int h) {
        this->w = w;
        this->h = h;
}
```

```
class Square {
        private:
            int w;
            int h;
        public:
            Square(Rectangle);
            void Show();
};
Square::Square(Rectangle r){
    this->w=r.h;
    this->h=r.h;
}
```

Hàm bạn và lớp

- Lớp bạn:

```
void Square::Show(){
    cout<<"Chieu dai hinh vuong: "<<w;
    cout<<"\nChieu rong hinh vuong: "<<h;
}
int main()
{
    Rectangle rec(2, 5);
    Square s(rec);
    s.Show();
    return 0;
}
```



Các lớp có sẵn trong thư viện STL của C++

- ✓ STL (Standard Template Library) là thư viện chuẩn của C++, được xây dựng sẵn.
- ✓ Cài đặt các cấu trúc dữ liệu và thuật toán thông dụng.
- ✓ Bao gồm các lớp và hàm khuôn mẫu, cho phép làm việc với dữ liệu tổng quát.
- ✓ Nằm trong một namespace có tên std.

Các lớp có sẵn trong thư viện STL của C++

- ✓ Các phần chính của STL:
  - Các lớp dữ liệu cơ bản: string, complex
  - Xuất nhập (IO)
  - Các lớp chứa (containers): list, vector, deque, stack, map, set, …
  - Duyệt phần tử của các lớp chứa (iterators)
  - Một số thuật toán thông dụng: tìm kiếm, so sánh, sắp xếp,…
  - Quản lý bộ nhớ, con trỏ
  - Xử lý ngoại lệ (exception handling)

Các lớp có sẵn trong thư viện STL của C++

- ✓ Xử lý chuỗi: #include<string>
  - Lớp string cho chuỗi ASCII và wstring cho Unicode
  - Các thao tác cơ bản: +, += (nối chuỗi); ==, !=, >, <, >=, <= (so sánh); << (xuất), >> (nhập)
  - Độ dài chuỗi: int string::length() const
  - Chuỗi con: string string::substr(int off, int count) const
  - Tìm chuỗi con: int string::find(const char* str, int pos) const
  - Đổi sang chuỗi của C: const char* string::c_str() const
  - Đổi sang số và ngược lại (C++11):

    [int|long|float|double] sto[i|l|f|d](const string& s);

    string to_string([int|long|float|double] n);

    wstring to_wstring([int|long|float|double] n);

Các lớp có sẵn trong thư viện STL của C++

- ✓ Các lớp chứa (containers): vector
  - Là mảng động.
  - Có thể chứa dữ liệu bất kỳ (template): vector<type>

```
#include<vector>
int p[] = {4, 2, 6};
vector<int> a(p, p+3);              // khởi tạo từ mảng C
a.push_back(1);                     // thêm vào cuối
a.insert(a.begin() + 2, 3);         // thêm ở vị trí 2
a.insert(a.end() - 1, 5);           // thêm ở vị trí 1 từ cuối
a[3] = 10;                          // phần tử thứ 4
vector<int>::iterator i;            // duyệt xuôi
for (i = a.begin(); i != a.end(); i++) *i += 5;
vector<int>::reverse_iterator j;    // duyệt ngược
for (j = a.rbegin(); j != a.rend(); j++)
   cout << *j << " ";
```

Các lớp có sẵn trong thư viện STL của C++

- ✓ Các lớp chứa (containers): iterator
  - Các lớp chứa của STL (vector, list,…) có định nghĩa kiểu iterator tương ứng để duyệt các phần tử (theo thứ tự xuôi)
    - Mỗi iterator chứa vị trí của một phần tử
    - Các hàm begin() và end() trả về một iterator tương ứng với các vị trí đầu và cuối.
    - Các toán tử với iterator:

      i++ phần tử kế tiếp

      i-- phần tử liền trước

      *i giá trị của phần tử

      i-> lấy thành phần của phần tử
  - Tương tự, có reverse_iterator để duyệt theo thứ tự ngược
    - Các hàm rbegin() và rend()

Các lớp có sẵn trong thư viện STL của C++

- ✓ Các lớp chứa (containers): danh sách liên kết list
  - Có thể chứa dữ liệu kiểu bất kỳ (template): list<type>
  - #include<list>
  - Duyệt danh sách dùng iterator tương tự như với vector.

```
double p[] = {1.2, 0.7, 2.2, 3.21, 6.4};
list<double> l(p, p+5);                    // khởi tạo từ mảng C
l.push_back(3.4);                          // thêm vào cuối
l.pop_front();                             // xoá phần tử đầu
list<double>::iterator i = l.begin();      // phần tử đầu
*i = 4.122;                                // gán giá trị
i++;                                       // phần tử kế tiếp
l.insert(i, 5.0);                          // chèn phần tử
l.erase(i);                                // xoá phần tử
l.sort();                                  // sắp xếp (tăng dần)
for (i = l.begin(); i != l.end(); i++)   // duyệt xuôi
cout << *i <<" ";
```

Các lớp có sẵn trong thư viện STL của C++

- ✓ Thuật toán tìm kiếm
  - Phần tử lớn nhất, bé nhất:

    ```
    vector<float>::iterator p =
                                max_element(a.begin()+2, a.end()-3);
    list<string>::iterator p =min_element(l.begin(), l.end());
    ```
  - Dựa trên các toán tử so sánh → định nghĩa nếu chưa có
  - Tìm đúng giá trị:

    ```
    list<float>::iterator p = find(p1, p2, 2.5f);
    ```
  - Tìm theo tiêu chuẩn: cần định nghĩa một hàm đánh giá

    ```
    bool isOdd(int i) { return i%2 == 1; }
    list<int>::iterator p = find_if(p1, p2, isOdd);
    ```
  - Tìm kiếm và thay thế, xoá:

    ```
    replace_if(p1, p2, isOdd, 10);
    remove_if(p1, p2, isOdd);
    ```

Các lớp có sẵn trong thư viện STL của C++

- ✓ Thuật toán sắp xếp
  - Sắp xếp mảng:
    - + Dùng toán tử so sánh:

```
sort(a.begin(), a.end());
```

Phải định nghĩa toán tử "<" cho kiểu dữ liệu được chứa

- + Dùng hàm so sánh tự định nghĩa:

```
bool compare(const table& a, const table& b) {
    return a.c1 < b.c1 ||(a.c1 == b.c1 && a.c2 < b.c2);
}
sort(a.begin(), a.end(), compare);
```

  - Sắp xếp danh sách:

```
l.sort();
l.sort(compare);
```

Các lớp có sẵn trong thư viện STL của C++

- Xuất/nhập (input/output):
  - Trong STL, việc xuất nhập được thông qua các "luồng thông tin" (data streams)
    - ✓ #include <iostream>
    - ✓ Luồng xuất (output streams): để xuất dữ liệu
      - Toán tử <<
      - Lớp cơ sở: *basic_istream<type>*
    - ✓ Luồng nhập (input streams): để nhập dữ liệu
      - Toán tử >>
      - Lớp cơ sở: *basic_ostream<type>*
    - ✓ Luồng xuất nhập (input/output streams): cả xuất và nhập
      - Toán tử >> và <<
      - Lớp cơ sở: *basic_iostream<type>*
  - Mỗi kiểu đối tượng muốn làm việc được với các lớp trên cần phải được định nghĩa toán tử >> và <<

Các lớp có sẵn trong thư viện STL của C++

- Xuất/nhập (input/output): I/O chuẩn
  - Các đối tượng
    - ✓ cin thuộc kiểu istream, tương đương với stdin
    - ✓ cout thuộc kiểu ostream, tương đương với stdout
    - ✓ cerr thuộc kiểu ostream, tương đương với stderr
    - ✓ Các đối tượng wcin, wcout, wcerr để làm việc với Unicode
  - Ví dụ:

```
int i; float a[10];
cout << "Nhap i va a[i]: ";
if (cin >> i >> a[i]) {
      cout << "a[" << i << "] = " << a[i] << endl;  cout.flush();
} else cerr << "Nhap du lieu loi" << endl;
```

  - Đọc một dòng:
    - ✓ string s;
    - ✓ getline(cin, s);

Các lớp có sẵn trong thư viện STL của C++

- Xuất/nhập (input/output): Đọc ghi file
  - Tương tự chương 4

Các lớp có sẵn trong thư viện STL của C++

- Xuất/nhập (input/output): Định dạng xuất dữ liệu
  - Tương tự phần Xâu ký tự của chương 2

Các lớp có sẵn trong thư viện STL của C++

- ✓ Tuần tự hóa (serialize):
  - Là việc chuyển đổi một đối tượng có cấu trúc dữ liệu bất kỳ thành một luồng thông tin có thứ tự để có thể ghi ra rồi đọc lại
  - Ứng dụng trong việc truyền tin và lưu trữ dữ liệu
  - Với STL, ta có thể định nghĩa các toán tử >> và << để thực hiện serialize.
  - Ví dụ:

```
istream& operator >>(istream& is, SinhVien& sv) {
        return is >> sv.ten >> sv.khoa >> sv.nam_sinh;
}
ostream& operator <<(ostream& os, SinhVien& sv) {
        return os << sv.ten << sv.khoa << sv.nam_sinh;
}
```

Case study

- Thông tin một học sinh bao gồm:
  - Họ tên.
  - Điểm văn, toán.

  Xây dựng lớp **học sinh** cho phép thực hiện các thao tác:
  - ✓ Nhập, xuất.
  - ✓ Lấy họ tên, điểm văn, toán.
  - ✓ Gán giá trị cho họ tên, điểm văn, điểm toán.
  - ✓ Tính điểm trung bình.
  - ✓ Xếp loại theo tiêu chí:
    - Giỏi (>= 8.0), Khá (>= 7.0).
    - Trung bình (>= 5.0), Yếu (< 5).
- Lập trình bài tập nhóm theo hướng đối tượng:
  - Nhận diện một số đối tượng
  - Hiện thức các đối tượng này