Chương 4: Mảng, Chuỗi và con trỏ

Phần 1: Lý thuyết:
1. Lớp (Class) và Đối Tượng (Object)
Class: Bản thiết kế của đối tượng, chứa thuộc tính (data) và phương thức (functions).
Object: Thể hiện cụ thể của lớp.
Ví dụ:

```
class Student {private:
  string name;      Thuộc tính private
  int age;  public:
   Constructor: Khởi tạo đối tượng
  Student(string n, int a) {
    name = n;
    age = a;
  }

   Phương thức public
  void display() {
    cout << "Ten: " << name << ", Tuoi: " << age << endl;
  }};
 Tạo đối tượng
Student s1("Alice", 20);
s1.display();   Output: Ten: Alice, Tuoi: 20
```

Constructor/Destructor:
Constructor: Khởi tạo giá trị mặc định khi đối tượng được tạo.
Destructor: Hủy đối tượng, giải phóng bộ nhớ (kí hiệu ~).

```
~Student() {
  cout << "Doi tuong da bi huy";}
```

---

2. Tính Kế Thừa (Inheritance)
Kế thừa: Lớp con (derived class) kế thừa thuộc tính/phương thức từ lớp cha (base class).
Ví dụ:

```
class Person {protected:
  string name;public:
  void setName(string n) { name = n; }};
 Lớp Student kế thừa từ Personclass Student : public Person {private:
  int studentId;public:
  void setStudentId(int id) { studentId = id; }
  void display() {
    cout << "Ten: " << name << ", ID: " << studentId << endl;
  }};
```

---

3. Tính Đa Hình (Polymorphism)
Virtual Function: Cho phép ghi đè (override) phương thức ở lớp con.

```
class Animal {public:
  virtual void sound() {   Hàm ảo
    cout << "Tieng keu" << endl;
  }};
class Dog : public Animal {public:
  void sound() override {   Ghi đè
    cout << "Gau gau!" << endl;
  }};
int main() {
  Animal *animal = new Dog();
  animal->sound();   Output: Gau gau! (Đa hình runtime)
  delete animal;
  return 0;}
```

---

4. Thư Viện STL (Standard Template Library)

vector: Mảng động, tự quản lý bộ nhớ.

```
#include <vector>
vector<int> numbers = {3, 1, 4};
numbers.push_back(5);   Thêm phần tử
cout << numbers[2];     Output: 4
```

algorithm: Cung cấp hàm tiện ích như sort(), find().

```
#include <algorithm>sort(numbers.begin(), numbers.end());   Sắp xếp tăng dần
```

Phần 2: Thực hành

Bài 1: Xây Dựng Lớp Student Quản Lý Điểm

Yêu cầu:

Thuộc tính: studentId, name, grades (vector<float>).

Phương thức: Thêm điểm, tính điểm trung bình.

Code mẫu:

```
class Student {private:
  string name;
  string id;
  vector<float> grades;public:
  Student(string n, string i) : name(n), id(i) {}

  void addGrade(float grade) {
    grades.push_back(grade);
  }

  float calculateAverage() {
    float sum = 0;
    for (float g : grades) sum += g;
    return sum / grades.size();
  }
```

```
  void displayInfo() {
    cout << "ID: " << id << ", Ten: " << name
         << ", Diem TB: " << calculateAverage() << endl;
  }};
int main() {
  Student s("Alice", "SV001");
  s.addGrade(8.5);
  s.addGrade(7.0);
  s.displayInfo();   Output: ID: SV001, Ten: Alice, Diem TB: 7.75
  return 0;}
```

Bài 2: Sắp Xếp Danh Sách Sinh Viên Bằng STL

Yêu cầu: Sắp xếp danh sách sinh viên theo điểm trung bình giảm dần.

Code mẫu:

```
bool compareStudents(const Student &a, const Student &b) {
  return a.calculateAverage() > b.calculateAverage();}
int main() {
  vector<Student> students;
  students.push_back(Student("Bob", "SV002"));
  students.push_back(Student("Alice", "SV001"));

   Thêm điểm cho từng sinh viên
  students[0].addGrade(9.0);
  students[1].addGrade(8.5);

   Sắp xếp bằng hàm compareStudents
  sort(students.begin(), students.end(), compareStudents);

   In danh sách đã sắp xếp
  for (Student &s : students) {
    s.displayInfo();
  }
  return 0;}
```

Bài 3: Ứng Dụng STL Khác

Tìm kiếm phần tử trong vector:

```
vector<int> nums = {5, 3, 7, 1};auto it = find(nums.begin(), nums.end(), 7);if
(it != nums.end()) {
  cout << "Tim thay tai vi tri: " << it - nums.begin();   Output: 2}
```

Lỗi Thường Gặp & Cách Khắc Phục

Quên public khi kế thừa:

```
class Student : Person { ... };   Mặc định là private → Lỗi!class Student : public
Person { ... };   Đúng
```

Không khởi tạo vector: Truy cập phần tử khi vector rỗng gây lỗi.

Sai phạm vi truy cập: Truy cập thuộc tính `private` từ bên ngoài lớp.

Tips Tối Ưu Hoá
Dùng STL thay tự implement: Tiết kiệm thời gian (ví dụ: `vector` thay mảng động).
Virtual destructor: Luôn khai báo destructor ảo trong lớp cơ sở nếu có đa hình.
Range-based for loop: Duyệt STL container dễ dàng.