Object Oriented Programming (OOP)

Classes

Classes names always start with a capital letter. The syntax for classes goes like this:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

private:

string keys;

};

By default, class members are private if not explicitly specified as public or private. It's recommended to explicitly state the access modifiers for clarity.

In the `main` function, when declaring objects of a class, the class name should start with a capital letter. Additionally, you should use the correct class name (`Car`) when declaring the objects:

int main() {

Car firstCar;

Car secondCar;

firstCar.wheels = 4;

firstCar.steeringWheel = true;

firstCar.fuel = "diesel";

secondCar.wheels = 6;

secondCar.steeringWheel = false;

secondCar.fuel = "diesel";

cout << "The first car has " << firstCar.wheels << " wheels. However, the second car has " << secondCar.wheels << " Wheels." << endl;

}

Output: The first car has 4 wheels. However, the second car has 6 Wheels.

Constructors

A constructor is a special member function with the same name as the class. It is used to initialize the object's data members. Here's the corrected code for the constructor:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

// Constructor

Car(int getWheels, bool getSteeringWheel, string getFuel) {

wheels = getWheels;

steeringWheel = getSteeringWheel;

fuel = getFuel;

}

};

You can also use the same names for the constructor parameters as the member variables, but to differentiate them, you need to use the `this` keyword. Here's an example:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

// Constructor

Car(int wheels, bool steeringWheel, string fuel) {

this->wheels = wheels;

this->steeringWheel = steeringWheel;

this->fuel = fuel;

}

};

When initializing objects, you can pass the values directly to the constructor:

int main() {

Car firstCar(4, true, "diesel");

Car secondCar(6, false, "diesel");

cout << "The first car has " << firstCar.wheels << " wheels. However, the second car has " << secondCar.wheels << " Wheels." << endl;

}

Output: The first car has 4 wheels. However, the second car has 6 Wheels.

Getters and Setters (Encapsulation)

To access private member variables, you can use getter and setter methods. These methods should be public. Here's an example of using getters and setters:

class Worker {

private:

int salary;

public:

void setSalary(int getSalary) {

salary = getSalary;

}

int getSalary() {

return salary;

}

};

int main() {

Worker giwrgos;

giwrgos.setSalary(5000);

cout << giwrgos.getSalary();

}

Inheritance

Inheritance allows a class to inherit properties (variables and functions) from another class. The class being inherited from is called the base class or parent class, and the class inheriting the properties is called the derived class or child class. Here's the corrected code for inheritance:

class Phone { // Base class

public:

int buttons;

bool frontCamera;

string os;

};

Class PhoneV2 : public Phone { // Derived class inheriting from Phone

public:

string color; // Extra variable specific to PhoneV2

PhoneV2(int buttons, bool frontCamera, string os, string color) : Phone(buttons, frontCamera, os) {

this->color = color;

}

};

class PhoneV3 : public PhoneV2 { // Derived class inheriting from PhoneV2

// No additional members or constructors defined

};

int main() {

PhoneV2 newPhone(14, true, "Realme", "Cyan");

}

Marriage (Multiple Inheritance)

Multiple inheritance allows a class to inherit properties from multiple base classes. Here's the corrected code for using multiple inheritance:

class Father {

public:

string fatherName;

void fatherMessage() {

cout << "Hello from Dad!" << endl;

}

};

class Mother {

public:

string motherName;

};

class Combine : public Father, public Mother { // Derived class inheriting from both Father and Mother

public:

void motherMessage() {

cout << "Hello from " << motherName << endl;

}

};

int main() {

Combine ultimateObject;

ultimateObject.motherName = "Mama";

ultimateObject.fatherName = "Babas";

ultimateObject.motherMessage();

ultimateObject.fatherMessage();

}

Constructor Inheritance

Constructor inheritance allows a derived class to invoke the constructor of the base class. Here's the corrected code for constructor inheritance:

class Car {

public:

int wheels;

bool steeringWheel;

string fuel;

// Constructor

Car(int wheels, bool steeringWheel, string fuel) {

this->wheels = wheels;

this->steeringWheel = steeringWheel;

this->fuel = fuel;

}

};

class Car2 : public Car { // Derived class inheriting from Car

public:

string color; // Extra variable specific to Car2

// Constructor

Car2(int wheels, bool steeringWheel, string fuel, string color) : Car(wheels, steeringWheel, fuel) {

this->color = color;

}

};

int main() {

Car2 quickCar(4, true, "diesel", "black");

cout << quickCar.color;

}

Member Functions (Methods)

Methods or member functions are functions defined within a class. They can be used both inside and outside the class. Here's the corrected code for member functions:

class GoodClass {

public:

void memberFunction1() { // Internal use of member function

cout << "First method" << endl;

}

void memberFunction2(); // Declaration of an external member function

int memberFunction3(int value); // Declaration of another member function

int memberFunction4(int value) { // Definition of another member function

return value;

}

};

void GoodClass::memberFunction2() { // Definition of the external member function

cout << "First member function used outside of the class" << endl;

}

int GoodClass::memberFunction3(int value) { // Definition of another member function

return value;

}

int main() {

GoodClass class1;

class1.memberFunction3(1000);

class1.memberFunction4(120);

}

Polymorphism

Polymorphism allows different classes to be treated as instances of a common base class. Here's the corrected code for polymorphism:

// Base class

class Animal {

public:

virtual void animalSound() {

cout << "The animal makes a sound" << endl;

}

};

// Derived class

class Pig : public Animal {

public:

void animalSound() {

cout << "The pig says: wee wee" << endl;

}

};

class Dog : public Animal {

public:

void animalSound() {

cout << "The dog says: bow wow" << endl;

}

};

int main() {

Animal\* myAnimal = new Animal();

Animal\* myPig = new Pig();

Animal\* myDog = new Dog();

myAnimal->animalSound();

myPig->animalSound();

myDog->animalSound();

delete myAnimal;

delete myPig;

delete myDog;

return 0;

}