Object Oriented Programming with C++

11. Inheritance

By: Prof. Pandav Patel

Second Semester, 2020-21
Computer Engineering Department
Dharmsinh Desai University

Inheritance: Introduction

- **Vehicle –** has wheels (diff num), has maximum speed, manufacturer etc..
  - **Automobile –** has engine, has gears etc...
    - **PassangeVehicle –** passanger capacity etc...
    - **GoodCarrier –** max load in weight, max load in volume etc...
  - **Cart –** Some animal to pull the cart, number of animals required etc...

```
class Vehicle {
        int wheels_count;
        float max_speed;
        string manufacturer;
public:
        Vehicle(int, float,
string);
        void print();
};
```

```
class Automobile {
        int wheels_count;
        float max_speed;
        string manufacturer;
        float engine_cc;
        int gear_count;
public:
        Automobile(int, float, string,
float, int);
        void print();
};
```

```
class Cart {
        int wheels_count;
        float max_speed;
        string manufacturer;
        string animal_type;
        int animal_count;
public:
        Cart(int, float, string,
string, int);
        void print();
};
```

- Is there duplication of code?
- Can we resuse code instead?

Inheritance: Introduction

- Inheritance (IS-A relationship)
  - Mechanism of deriving new class from existing class is called Inheritance (or derivation)
    - Existing class used in inheritance is called – base class / parent class / super class
    - Newly created class is called – derived class / child class / sub class
  - Syntax of inheritance is as follows:

```
class derived-class : visibility-mode base-class
{
        // Members of derived class
}
```

base-class

↑

derived-class

  - ***derived-class* IS-A *base-class***.
    - visibility-mode could be private / protected / public
      - Its optional, by default its private
      - Though visibility mode is often public, will look into it first

Inheritance: Introduction

Data Members

- Derived class inherits all data members of Base class
- Derived class may add data members of its own

Member Functions

- Derived class inherits all member functions of Base class
- Derived class may override a member function of Base class by redefining it with the same signature
- Derived class may overload a member function of Base class by redefining it with the same name; but different signature

Access Specification

- Derived class cannot access private members of Base class
- Derived class can access protected members of Base class

Construction-Destruction

- A constructor of the Derived class must first call a constructor of the Base class to construct the Base class instance of the Derived class
- The destructor of the Derived class must call the destructor of the Base class to destruct the Base class instance of the Derived class

```
#include<iostream>
using std::cout;
using std::endl;
class Base{

Base()
   {
      cout<<"Constructor: Base"<<endl;
   }
   ~Base()
   {
      cout<<"Destructor: Base"<<endl;
   }
};
class Other
{

Other()
   {
      cout<<"Constructor: Other"<<endl;
   }
   ~Other()
   {
      cout<<"Destructor: Other"<<endl;
   }
};
class Derived:public Base{

   Other o;
   Derived()
   {
      cout<<"Constructor: Derived"<<endl;
   }
   ~Derived()
   {
      cout<<"Destructor: Derived"<<endl;
   }
};

int main()
{
   Derived derived;
   return 0;
}
```

Output ??

```
#include<iostream>
using std::cout;
using std::endl;
class Base{

Base()
   {
      cout<<"Constructor: Base"<<endl;
   }
   ~Base()
   {
      cout<<"Destructor: Base"<<endl;
   }
};
class Other
{

Other()
   {
      cout<<"Constructor: Other"<<endl;
   }
   ~Other()
   {
      cout<<"Destructor: Other"<<endl;
   }
};
class Derived:public Base{

    Other o;
    Derived()
    {
        cout<<"Constructor: Derived"<<endl;
    }
    ~Derived()
    {
        cout<<"Destructor: Derived"<<endl;
    }
};

int main()
{
    Derived derived;
    return 0;
}
```

Output

Error: Base::Base() is private

Error Other::Other() is private

```
#include<iostream>
using std::cout;
using std::endl;
class Base{
public:
    Base()
    {
        cout<<"Constructor: Base"<<endl;
    }
    ~Base()
    {
        cout<<"Destructor: Base"<<endl;
    }
};
class Other
{
public:
    Other()
    {
        cout<<"Constructor: Other"<<endl;
    }
    ~Other()
    {
        cout<<"Destructor: Other"<<endl;
    }
};
class Derived:public Base{
public:
    Other o;
    Derived()
    {
        cout<<"Constructor: Derived"<<endl;
    }
    ~Derived()
    {
        cout<<"Destructor: Derived"<<endl;
    }
};

int main()
{
    Derived derived;
    return 0;
}
```

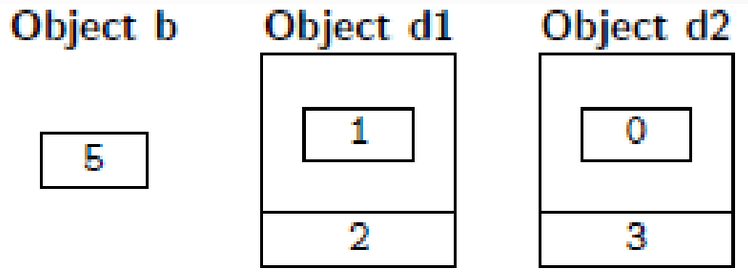
Output ??

```
#include<iostream>
using std::cout;
using std::endl;
class Base{
public:
    Base()
    {
        cout<<"Constructor: Base"<<endl;
    }
    ~Base()
    {
        cout<<"Destructor: Base"<<endl;
    }
};
class Other
{
public:
    Other()
    {
        cout<<"Constructor: Other"<<endl;
    }
    ~Other()
    {
        cout<<"Destructor: Other"<<endl;
    }
};
class Derived:public Base{
public:
    Other o;
    Derived()
    {
        cout<<"Constructor: Derived"<<endl;
    }
    ~Derived()
    {
        cout<<"Destructor: Derived"<<endl;
    }
};

int main()
{
    Derived derived;
    return 0;
}
```

Output :

```
Constructor: Base
Constructor: Other
Constructor: Derived
Destructor: Derived
Destructor: Other
Destructor: Base
```

```
class B {
public:
int data_;
B(int d = 0) : data_(d) {
    cout << "B::B(int): " << data_ << endl; }
~B() {
    cout << "B::~B(): " << data_ << endl; }
};
class D: public B {
 int info_;
public:
D(int d, int i) : B(d), info_(i) // ctor-1: Explicit construction of Base
{
   cout << "D::D(int, int): " << data_ << ", " << info_ << endl; }

D(int i) : info_(i) // ctor-2: Default construction of Base
{
    cout << "D::D(int): " << data_ << ", " << info_ << endl; }

~D() { cout << "D::~D(): " << data_ << ", " << info_ << endl; }
};
```




```
Int main()
{
   B b(5);

   // ctor-1: Explicit construction of Base
   D d1(1, 2);

  // ctor-2: Default construction of Base
  //If base class constructor do not have
    default value → Error
   D d2(3);
return 0;
}
```

```
class B {
public:
int data_;
B(int d = 0) : data_(d) {
    cout << "B::B(int): " << data_ << endl; }
~B() {
    cout << "B::~B(): " << data_ << endl; }
};
class D: public B {
 int info_;
public:
D(int d, int i)
{
B(d);
Info_= i ;
cout << "D::D(int, int): " << data_ << ", " << info_ << endl;
}
D(int i) : info_(i) // ctor-2: Default construction of Base
{ cout << "D::D(int): " << data_ << ", " << info_ << endl; }
~D() { cout << "D::~D(): " << data_ << ", " << info_ << endl; }
};
```

```
Int main()
{
   B b(5);


 D d1(1, 2);
```

Error: declaration of ‘B d’ shadows a parameter

```
/*
as we must have base object prior to call to constructor of D. We haven’t supplied it so compiler call default constructor of B and set data_ =0 (default).

Now, code B(d) inside constructor try to create new object of B but as we already have an object of B created by compiler using default constructor of base class
*/

return 0;
}
```

```
class B {
public:
int data_;
B(int d = 0) : data_(d) {
    cout << "B::B(int): " << data_ << endl; }
~B() {
    cout << "B::~B(): " << data_ << endl; }
};
class D: public B {
 int info_;
public:
D(int d, int i)
{
//B(d);
Info_= i ;
cout << "D::D(int, int): " << data_ << ", " << info_ << endl;
}
D(int i) : info_(i) // ctor-2: Default construction of Base
{ cout << "D::D(int): " << data_ << ", " << info_ << endl; }
~D() { cout << "D::~D(): " << data_ << ", " << info_ << endl; }
};
```

```
Int main()
{
   B b(5);


 D d1(1, 2);

/*
Now the code will work fine, the
only issue is compiler will set data_
variable to 0 (default) rather setting
it to 1 (user's input)
*/

return 0;
}
```

```
#include<iostream>
#include<string>

using std::cout;
using std::endl;
using std::string;

class Vehicle {
    int wheels_count;
    float max_speed;
    string manufacturer;
public:
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed, string manufacturer) {
    this->wheels_count = wheels_count;
    this->max_speed = max_speed;
    this->manufacturer = manufacturer;
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
int main() {

    Vehicle vehicle(2, 7.1, "Ambica carts");
    vehicle.vehicle_print();

    return 0;
}
```

```
Wheel Count: 2
Max Speed: 7.1
Manufacturer: Ambica carts
```

```
class Vehicle {

    int wheels_count;
    float max_speed;
    string manufacturer;
public:
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                        string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                        string manufacturer,
                        float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    // error: no matching function for call to 'Vehicle::Vehicle()'
    {}

    void print() {
        cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    return 0;
}
```

```
class Vehicle {

    int wheels_count;
    float max_speed;
    string manufacturer;
public:
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                        string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                            string manufacturer,
                    float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        // error: 'int Vehicle::wheels_count' is private
        cout << "Wheel Count: " << wheels_count << endl;
        // similar error
        cout << "Max Speed: " << max_speed << endl;
        // similar error
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
Wheel Count: 2
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {
public:
    int wheels_count;
    float max_speed;
    string manufacturer;

    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                        string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {

        cout << "Wheel Count: " << wheels_count << endl;

        cout << "Max Speed: " << max_speed << endl;

        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
Wheel Count: 2
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {

    int wheels_count;
    float max_speed;
    string manufacturer;
public:
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                        string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                        string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
            vehicle_print();
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
Wheel Count: 2
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {
protected:
    int wheels_count;
    float max_speed;
    string manufacturer;

    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                        string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
class Vehicle {
protected:
    int wheels_count;
    float max_speed;
    string manufacturer;

    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
                                        string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer) {
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                            string manufacturer,
                    float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: 'void Vehicle::vehicle_print()' is protected
    automobile.vehicle_print();
    return 0;
}
```

Inheritance: access specifiers

Access Specifier	Access from methods of class itself	Access from methods of derived class	Acess from methods of other classes and independent functions
public	Allowed	Allowed	Allowed
protected	Allowed	Allowed	Not Allowed
private	Allowed	Not Allowed	Not Allowed

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed
                                                    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                                string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        // error: 'int Vehicle::wheels_count' is private
        cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: 'int Vehicle::wheels_count' is private
    cout << "Wheel Count: " << automobile.wheels_count << endl;
    // error: 'float Vehicle::max_speed' is protected
    cout << "Max Speed: " << automobile.max_speed << endl;
    cout << "Manufacturer: " << automobile.manufacturer << endl;
    automobile.print();
    return 0;
}
```

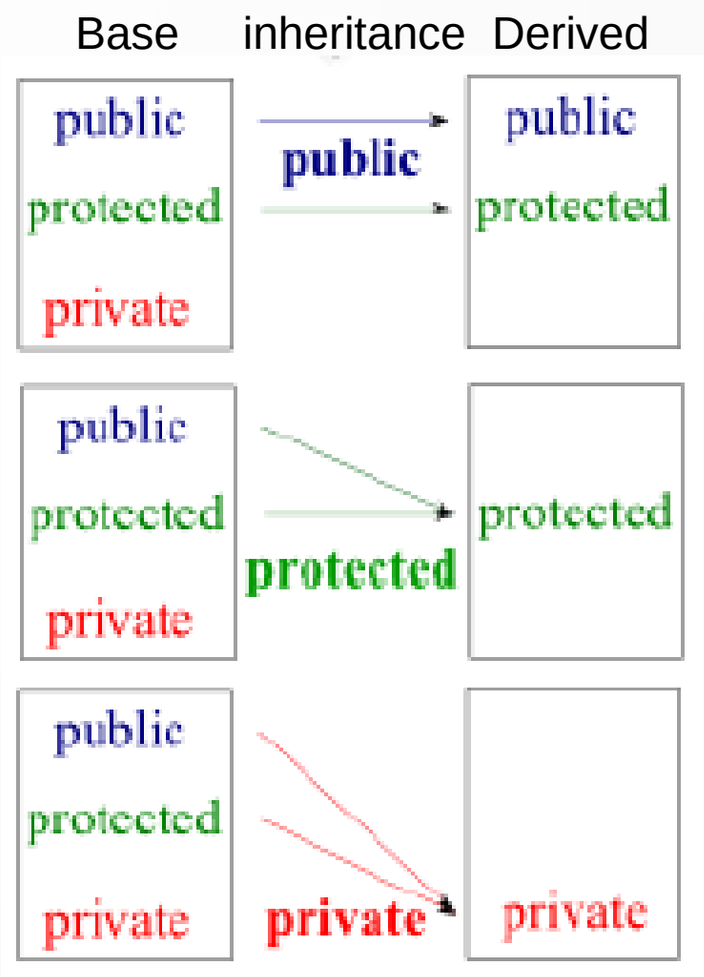
```
Vehicle constructor called
Automobile constructor called
Manufacturer: Honda //(from main)
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed
                                        string manufacturer):

wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : public Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                                string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        // error: 'int Vehicle::wheels_count' is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: 'int Vehicle::wheels_count' is private
    // cout << "Wheel Count: " << automobile.wheels_count << endl;
    // error: 'float Vehicle::max_speed' is protected
    // cout << "Max Speed: " << automobile.max_speed << endl;
    cout << "Manufacturer: " << automobile.manufacturer << endl;
    automobile.print();
    return 0;
}
```

Inheritance: visibility mode

class derived-class : **visibility-mode** base-class

Visibility mode	Base class	Derived class
public	public	public
	protected	protected
	private	**No access**
protected	public	protected
	protected	protected
	private	**No access**
private	public	private
	protected	private
	private	**No access**



```
Vehicle constructor called
Automobile constructor called
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed
                                        string manufacturer):

wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : protected Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                                string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: ‘int Vehicle::wheels_count’ is private
    // cout << "Wheel Count: " << automobile.wheels_count << endl;
    // error: ‘float Vehicle::max_speed’ is protected
    // cout << "Max Speed: " << automobile.max_speed << endl;
    // error: string Vehicle::manufacturer is inaccessible
    // cout << "Manufacturer: " << automobile.manufacturer << endl;
    automobile.print();
    return 0;
}
```

Protected Understanding:
For Derive class it is like public
For Other classes and functions it is like private

```
Vehicle constructor called
Automobile constructor called
Max Speed: 100
Manufacturer: Honda
Engine Capacity: 125
Gear Count: 4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed
                                        string manufacturer):

wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}

class Automobile : private Vehicle{
    float engine_cc;
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
                                string manufacturer,
                float engine_cc, int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: ‘int Vehicle::wheels_count’ is private
    // cout << "Wheel Count: " << automobile.wheels_count << endl;
    // error: ‘float Vehicle::max_speed’ is protected
    // cout << "Max Speed: " << automobile.max_speed << endl;
    // error: string Vehicle::manufacturer is inaccessible
    // cout << "Manufacturer: " << automobile.manufacturer << endl;
    automobile.print();
    return 0;
}
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void automobile_print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
```

```
class PassangerVehicle : public Automobile {
    int max_passangers;
public:
    PassangerVehicle(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count, int max_passangers):
    max_passangers(max_passangers),
    Automobile(wheels_count, max_speed, manufacturer,
        engine_cc, gear_count) {
        cout << "PassangerVehicle constructor called\n";
    }
    void print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        // error: 'float Automobile::engine_cc' is private
        // cout << "Eng Cap: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
        cout << "Max Passan: " << max_passangers << endl;
    }
};

int main() {
    PassangerVehicle pv(2, 100, "Honda", 125, 4, 60);
    // error: ‘int Vehicle::wheels_count’ is private
    // cout << "Wheel Count: " << pv.wheels_count << endl;
    // error: ‘float Vehicle::max_speed’ is protected
    // cout << "Max Speed: " << pv.max_speed << endl;

    cout << "Manufacturer: " << pv.manufacturer << endl;
    // error: 'float Automobile::engine_cc' is private
    // cout << "Engine Capacity: " << pv.engine_cc << endl;
    // error: 'int Automobile::gear_count' is protected
    // cout << "Gear Count: " << pv.gear_count << endl;
    pv.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
PassangerVehicle constructor called
Manufacturer: Honda
Max Speed: 100
Manufacturer: Honda
Gear Count: 4
Max Passan: 60
```

Vehicle	Automobile	PassangerVehicle	pv (object in main)
wheels_count	wheels_count	wheels_count	wheels_count
max_speed	max_speed	max_speed	max_speed
manufacturer	manufacturer	manufacturer	manufacturer
vehicle_print	vehicle_print	vehicle_print	vehicle_print
	engine_cc	engine_cc	engine_cc
	gear_count	gear_count	gear_count
	automobile_print	automobile_print	automobile_print
		max_passangers	max_passangers
		print	print

Vehicle

↑

Automobile

↑

PassangerVehicle

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : protected Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void automobile_print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
```

```
class PassangerVehicle : public Automobile {
    int max_passangers;
public:
    PassangerVehicle(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count, int max_passangers):
    max_passangers(max_passangers),
    Automobile(wheels_count, max_speed, manufacturer,
        engine_cc, gear_count) {
        cout << "PassangerVehicle constructor called\n";
    }
    void print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        // error: 'float Automobile::engine_cc' is private
        // cout << "Eng Cap: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
        cout << "Max Passan: " << max_passangers << endl;
    }
};

int main() {
    PassangerVehicle pv(2, 100, "Honda", 125, 4, 60);
    // error: ‘int Vehicle::wheels_count’ is private
    // cout << "Wheel Count: " << pv.wheels_count << endl;
    // error: ‘float Vehicle::max_speed’ is protected
    // cout << "Max Speed: " << pv.max_speed << endl;
    // error: string Vehicle::manufacturer is inaccessible
    // cout << "Manufacturer: " << pv.manufacturer << endl;
    // error: 'float Automobile::engine_cc' is private
    // cout << "Engine Capacity: " << pv.engine_cc << endl;
    // error: 'int Automobile::gear_count' is protected
    // cout << "Gear Count: " << pv.gear_count << endl;
    pv.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
PassangerVehicle constructor called
Max Speed: 100
Manufacturer: Honda
Gear Count: 4
Max Passan: 60
```

Vehicle	Automobile	PassangerVehicle	pv (object in main)
wheels_count	wheels_count	wheels_count	wheels_count
max_speed	max_speed	max_speed	max_speed
manufacturer	manufacturer	manufacturer	manufacturer
vehicle_print	vehicle_print	vehicle_print	vehicle_print
	engine_cc	engine_cc	engine_cc
	gear_count	gear_count	gear_count
	automobile_print	automobile_print	automobile_print
		max_passangers	max_passangers
		print	print

Vehicle
↑
Automobile
↑
PassangerVehicle

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : private Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void automobile_print() {
        // error: 'int Vehicle::wheels_count' is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
```

```
class PassangerVehicle : public Automobile {
    int max_passangers;
public:
    PassangerVehicle(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count, int max_passangers):
    max_passangers(max_passangers),
    Automobile(wheels_count, max_speed, manufacturer,
        engine_cc, gear_count) {
        cout << "PassangerVehicle constructor called\n";
    }
    void print() {
        // error: 'int Vehicle::wheels_count' is private
        // cout << "Wheel Count: " << wheels_count << endl;
        // error: 'float Vahicle::max_speed' is protected
        // cout << "Max Speed: " << max_speed << endl;
        // error: 'string Vehicle::manufacturer' is inaccessible
        // cout << "Manufacturer: " << manufacturer << endl;
        // error: 'float Automobile::engine_cc' is private
        // cout << "Eng Cap: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
        cout << "Max Passan: " << max_passangers << endl;
    }
};

int main() {
    PassangerVehicle pv(2, 100, "Honda", 125, 4, 60);
    // error: 'int Vehicle::wheels_count' is private
    // cout << "Wheel Count: " << pv.wheels_count << endl;
    // error: 'float Vehicle::max_speed' is protected
    // cout << "Max Speed: " << pv.max_speed << endl;
    // error: string Vehicle::manufacturer is inaccessible
    // cout << "Manufacturer: " << pv.manufacturer << endl;
    // error: 'float Automobile::engine_cc' is private
    // cout << "Engine Capacity: " << pv.engine_cc << endl;
    // error: 'int Automobile::gear_count' is protected
    // cout << "Gear Count: " << pv.gear_count << endl;
    pv.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
PassangerVehicle constructor called
Gear Count: 4
Max Passan: 60
```

Vehicle	Automobile	PassangerVehicle	pv (object in main)
wheels_count	wheels_count	wheels_count	wheels_count
max_speed	max_speed	max_speed	max_speed
manufacturer	manufacturer	manufacturer	manufacturer
vehicle_print	vehicle_print	vehicle_print	vehicle_print
	engine_cc	engine_cc	engine_cc
	gear_count	gear_count	gear_count
	automobile_print	automobile_print	automobile_print
		max_passangers	max_passangers
		print	print

Vehicle
↑
Automobile
↑
PassangerVehicle

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void automobile_print() {
        // error: ‘int Vehicle::wheels_count’ is private
        // cout << "Wheel Count: " << wheels_count << endl;
        cout << "Max Speed: " << max_speed << endl;
        cout << "Manufacturer: " << manufacturer << endl;
        cout << "Engine Capacity: " << engine_cc << endl;
        cout << "Gear Count: " << gear_count << endl;
    }
};
```

```
int main() {
    Vehicle v(2, 100, "Honda");
    cout << "Wheel Count: " << v.wheels_count << endl;
    cout << "Max Speed: " << v.max_speed << endl;
    cout << "Manufacturer: " << v.manufacturer << endl;

    Automobile a(2, 100, "Honda", 125, 4);
    cout << "Wheel Count: " << a.wheels_count << endl;
    cout << "Max Speed: " << a.max_speed << endl;
    cout << "Manufacturer: " << a.manufacturer << endl;
    cout << "Engine Capacity: " << a.engine_cc << endl;
    cout << "Gear Count: " << a.gear_count << endl;

    return 0;
}
```

Vehicle	v (object in main)	Automobile	a (object in main)
wheels_count	?	wheels_count	?
max_speed	?	max_speed	?
manufacturer	?	manufacturer	?
vehicle_print	?	vehicle_print	?
		engine_cc	?
		gear_count	?
		automobile_print	?

Vehicle
↑
Automobile
↑
PassangerVehicle

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print();
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
void Vehicle::print(bool in_single_line = true) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
int main() {
    Vehicle vehicle(2, 100, "Honda");
    vehicle.print();
    vehicle.print(true);
    vehicle.print(false);
    return 0;
}
```

Guess the output of this code

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print();
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
void Vehicle::print(bool in_single_line = true) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
int main() {
    Vehicle vehicle(2, 100, "Honda");
    // error: call of overloaded ‘print()’ is ambiguous
    vehicle.print();
    vehicle.print(true);
    vehicle.print(false);
    return 0;
}
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print();
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
int main() {
    Vehicle vehicle(2, 100, "Honda");
    vehicle.print();
    vehicle.print(true);
    vehicle.print(false);
    return 0;
}
```

Guess the output of this code

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print();
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
int main() {
    Vehicle vehicle(2, 100, "Honda");
    vehicle.print();
    vehicle.print(true);
    vehicle.print(false);
    return 0;
}
```

Vehicle constructor called
Wheel Count: 2
Max Speed: 100
Manufacturer: Honda
2 100 Honda
2
100
Honda

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << max_speed << endl << manufacturer << endl;
        cout << engine_cc << endl << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
100
Honda
125
4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << max_speed << endl << manufacturer << endl;
        cout << engine_cc << endl << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print(true);
    return 0;
}
```

Guess the output

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << max_speed << endl << manufacturer << endl;
        cout << engine_cc << endl << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: no matching function for call to
    // 'Automobile::print(bool)
    automobile.print(true);
    return 0;
}
```

- Functions/methods can be overloaded within the same scope only
- Here Vehicle and Automobile are two differenct classes and hence different scopes
- When we define print function in Automobile then all the print functions of the Vehicle class are inaccessible in Automobile class

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    void print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        cout << max_speed << endl << manufacturer << endl;
        cout << engine_cc << endl << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print();
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor
called
100
Honda
125
4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    void print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print(bool in_single_line) {
        if(in_single_line) {
            cout << max_speed << " " << manufacturer;
            cout << " " << engine_cc << " " << gear_count;
            cout << endl;
        }
        else {
            cout << max_speed << endl << manufacturer;
            cout << endl << engine_cc <<endl << gear_count;
            cout << endl;
        }

    }
};
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print(true);
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
100 Honda 125 4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    void print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
    float wheels_count = 16;
protected:
    int gear_count;
    int max_speed = 1236; // supersonic
public:
    string manufacturer = "Magic Corp.";
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print(bool in_single_line) {
        if(in_single_line) {
            cout << wheels_count << " ";
            cout << max_speed << " " << manufacturer;
            cout << " " << engine_cc << " " << gear_count;
            cout << endl;
        }
        else {
            cout << wheels_count << endl;
            cout << max_speed << endl << manufacturer;
            cout << endl << engine_cc <<endl << gear_count;
            cout << endl;
        }

    }
};
```

```
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print(true);
    return 0;
}
```

```
Vehicle constructor called
Automobile constructor called
16 1236 Magic Corp. 125 4
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    void print();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
void Vehicle::print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
    float wheels_count = 16;
protected:
    int gear_count;
    int max_speed = 1236;
public:
    string manufacturer = "Magic Corp.";
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print(bool in_single_line) {
        if(in_single_line) {
            // error:‘int Vehicle::wheels_count’ is private
            // cout << Vehicle::wheels_count << " ";
            cout << Vehicle::max_speed << " ";
            cout << Vehicle::manufacturer;
            cout << " " << engine_cc << " " << gear_count;
            cout << endl;
        } else {
            // error:‘int Vehicle::wheels_count’ is private
            // cout << Vehicle::wheels_count << endl;
            cout << Vehicle::max_speed << endl;
            cout << Vehicle::manufacturer;
            cout << endl << engine_cc << endl << gear_count;
            cout << endl;
        }
    }
};
```

```
int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    automobile.print(true);
    cout << automobile.manufacturer << endl;
    cout << automobile.Vehicle::manufacturer << endl;
    return 0;
}
```

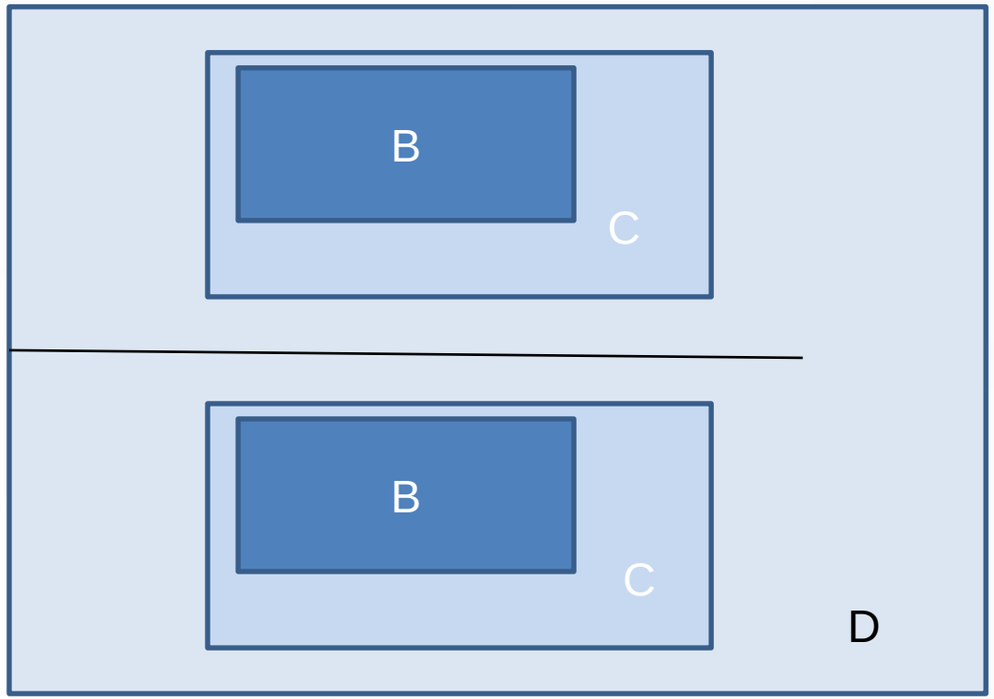
```
Vehicle constructor called
Automobile constructor called
100 Honda 125 4
Magic Corp.
Honda
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        Vehicle::print(false);
        cout << engine_cc << endl << gear_count << endl;
    }
};

int main() {
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: no matching function for call to
    // ‘Automobile::print(bool)
    // automobile.print(true);
    automobile.Vehicle::print(true);
    automobile.print();
    return 0;
}
```

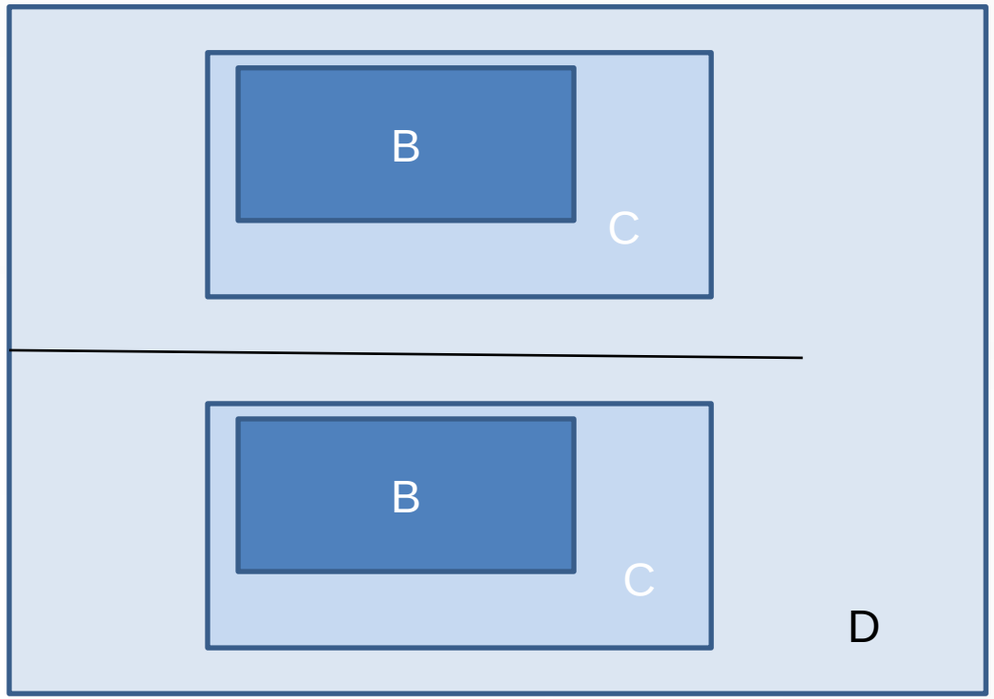
```
Vehicle constructor called
Automobile constructor called
2 100 Honda
2
100
Honda
125
4
```

```
class B {
protected:
B() { cout << "B "; }
~B() { cout << "~B "; }
};

class C : public B {
protected:
C() { cout << "C "; }
~C() { cout << "~C "; }
};

class D : private C {
C var;
public:
D() { cout << "D " << endl; }
~D() { cout << "~D "; }
};

int main() {
D d;
return 0;
}
```



Output: ??

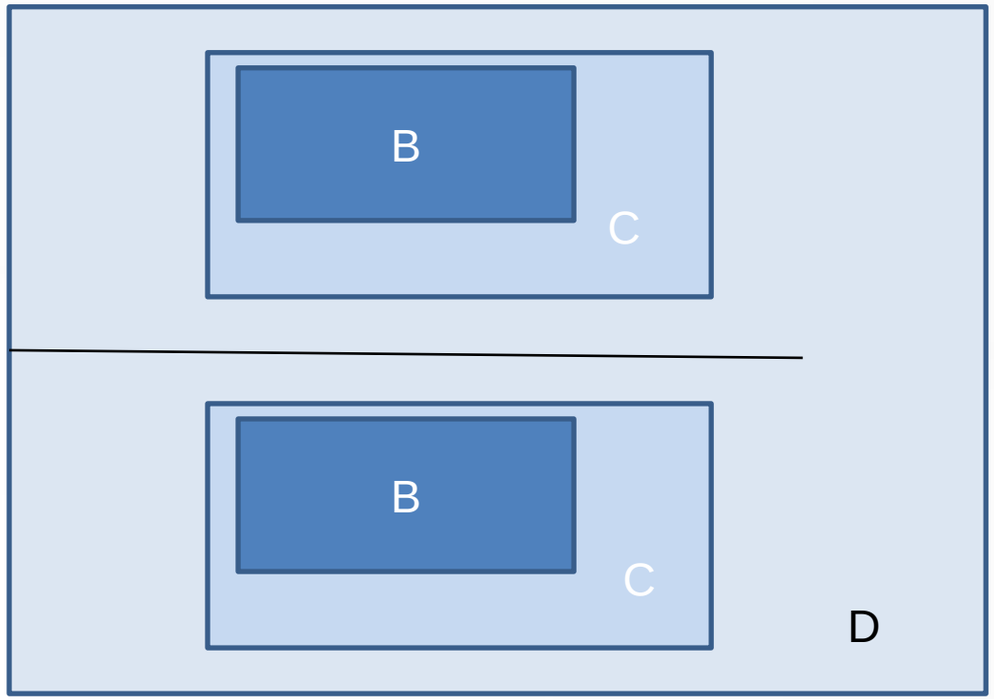
```
class B {
protected:
B() { cout << "B "; }
~B() { cout << "~B "; }
};

class C : public B {
protected:
C() { cout << "C "; }
~C() { cout << "~C "; }
};

class D : private C {
C var;
public:
D() { cout << "D " << endl; }
~D() { cout << "~D "; }
};

int main() {
D d;
return 0;
}
```



Output: Error

var can not access constructor which is protected

```
class B {
protected:
B() { cout << "B "; }
~B() { cout << "~B "; }
};

class C : public B {
public:
C() { cout << "C "; }
~C() { cout << "~C "; }
};

class D : private C {
C var;
public:
D() { cout << "D " << endl; }
~D() { cout << "~D "; }
};

int main() {
D d;
return 0;
}
```



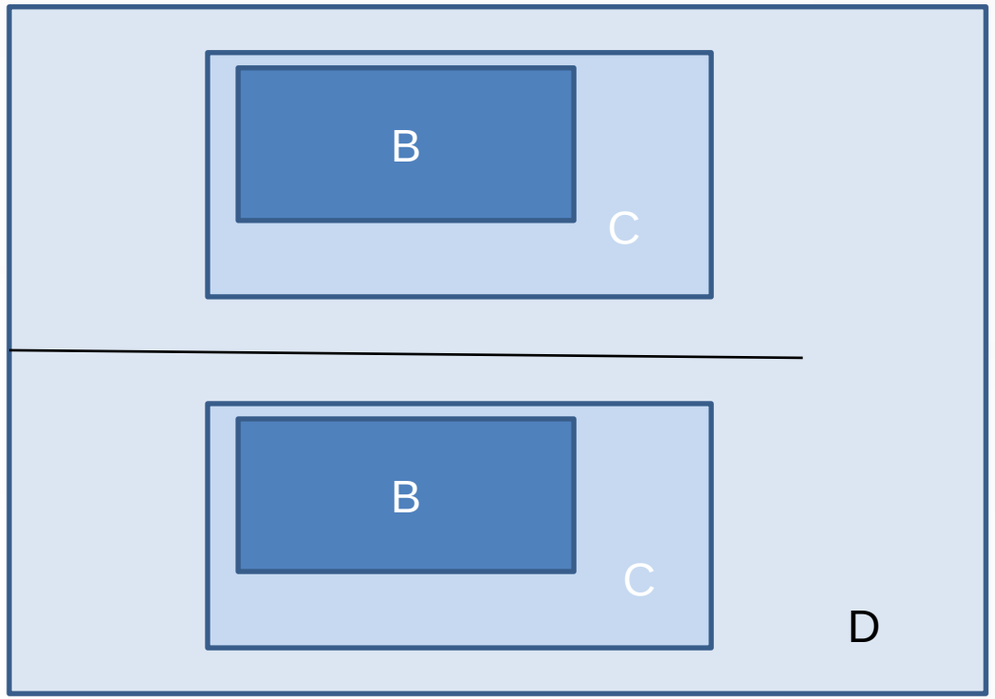
B C B C D
~D ~C ~B ~C ~B

```
class B {
protected:
B() { cout << "B "; }
~B() { cout << "~B "; }
};

class C : public B {
protected:
 int data=100;
public:
C() { cout << "C "; }
~C() { cout << "~C "; }
};

class D : private C {
C var;
public:
D() { cout << "D " << endl; }
~D() { cout << "~D "; }
void change_data();
void print();
};
```

```
int main() {
D d;
d.change_data();
d.print();
return 0;
}
```

Output:

??




```
void D:: change_data()
  {
      data=100;  //update the data of inherited C
      var.data=500; //update the data of var C
   }
 void D:: print()
  {
      cout<<endl<<data<<endl;
      cout<<var.data<<endl;
  }
```

```
class B {
protected:
B() { cout << "B "; }
~B() { cout << "~B "; }
};

class C : public B {
protected:
 int data=100;
public:
C() { cout << "C "; }
~C() { cout << "~C "; }
};

class D : private C {
C var;
public:
D() { cout << "D " << endl; }
~D() { cout << "~D "; }
void change_data();
void print();
};
```

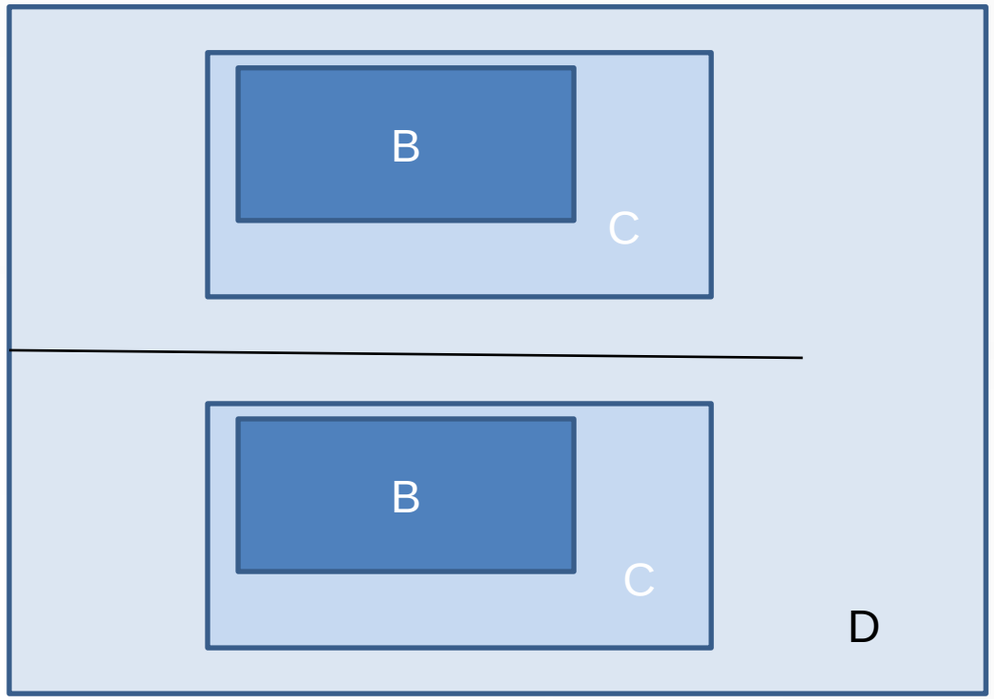
```
int main() {
D d;
d.change_data();
d.print();
return 0;
}
```

Output:

Error :

var.data is private and not accessible in change_data() and in print()

To allow modification of data using var, we can create pubilc method in class C which modifies data and call it using var




```
void D:: change_data()
  {
      data=100;  //update the data of inherited C
      var.data=500; //ERROR: access not allowed
   }
 void D:: print()
   {
      cout<<endl<<data<<endl;
      cout<<var.data<<endl; //ERROR: access not allowed
   }
```

Inheritance

- Constructors, destructor, friend functions, friend classes, and static members of base class are not inherited by derived class.
- Friend fuctions and classes have access to protected members of the class along with private and public members of the class

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    friend void friend_fun1();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    friend void friend_fun2();
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        Vehicle::print(false);
        cout << engine_cc << endl << gear_count << endl;
    }
};
```

```
friend_fun1 called
Vehicle constructor called
Automobile constructor called
2
100
Honda
friend_fun2 called
Vehicle constructor called
Automobile constructor called
100
Honda
125
4
```

```
void friend_fun1() {
    cout << "friend_fun1 called\n";
    Automobile automobile(2, 100, "Honda", 125, 4);
    cout << automobile.wheels_count << endl;
    cout << automobile.max_speed << endl;
    cout << automobile.manufacturer << endl;
    // error: 'float Automobile::engine_cc' is private
    // cout << automobile.engine_cc << endl;
    // error: 'int Automobile::gear_count' is protected
    // cout << automobile.gear_count << endl;
}

void friend_fun2() {
    cout << "friend_fun2 called\n";
    Automobile automobile(2, 100, "Honda", 125, 4);
    // error: 'int Vehicle::wheels_count' is private
    // cout << automobile.wheels_count << endl;
    cout << automobile.max_speed << endl;
    cout << automobile.manufacturer << endl;
    cout << automobile.engine_cc << endl;
    cout << automobile.gear_count << endl;
}

int main() {
    friend_fun1();
    friend_fun2();
    return 0;
}
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void print(bool);
    friend void friend_fun1();
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::print(bool in_single_line) {
    if(in_single_line) {
        cout << wheels_count << " " << max_speed;
        cout << " " << manufacturer << endl;
    }
    else {
        cout << wheels_count << endl << max_speed;
        cout << endl << manufacturer << endl;
    }
}
```

```
class Automobile : public Vehicle{
    float engine_cc;
protected:
    int gear_count;
public:
    friend void friend_fun1();
    Automobile(int wheels_count, float max_speed,
        string manufacturer, float engine_cc,
        int gear_count):
    engine_cc(engine_cc),
    gear_count(gear_count),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    void print() {
        Vehicle::print(false);
        cout << engine_cc << endl << gear_count << endl;
    }
};
```

```
friend_fun1 called
Vehicle constructor called
Automobile constructor called
2
100
Honda
125
4

3
70
Bajaj
```

```
void friend_fun1() {
    cout << "friend_fun1 called\n";
    Automobile automobile(2, 100, "Honda", 125, 4);

    cout << automobile.wheels_count << endl;
    cout << automobile.max_speed << endl;
    cout << automobile.manufacturer << endl;
    cout << automobile.engine_cc << endl;
    cout << automobile.gear_count << endl;

    Vehicle vehicle(3, 70, “Bajaj“);

    cout << endl<<vehicle.wheels_count << endl;
    cout << vehicle.max_speed << endl;
    cout << vehicle.manufacturer << endl;

   //error: ‘class Vehicle’ has no member named ‘engine_cc’
   // cout << vehicle.engine_cc << endl;
}

int main() {
    friend_fun1();
    return 0;
}
```

```
class Vehicle {
    int wheels_count;
protected:
    float max_speed;
public:
    string manufacturer;
    Vehicle(int, float, string);
    void vehicle_print();
    ~Vehicle() {
        cout << "Vehicle Destructor called\n";
    }
};
Vehicle::Vehicle(int wheels_count, float max_speed,
    string manufacturer):
wheels_count(wheels_count),
max_speed(max_speed),
manufacturer(manufacturer){
    cout << "Vehicle constructor called\n";
}
void Vehicle::vehicle_print() {
    cout << "Wheel Count: " << wheels_count << endl;
    cout << "Max Speed: " << max_speed << endl;
    cout << "Manufacturer: " << manufacturer << endl;
}
```

```
class Engine {
    int cylinders_count;
protected:
    string cylinders_arrangement; //v-type, inline etc...
public:
    float engine_cc;
    Engine(int cylinders_count,
        string cylinders_arrangement, float engine_cc):
    cylinders_count(cylinders_count),
    cylinders_arrangement(cylinders_arrangement),
    engine_cc(engine_cc){
        cout << "Engine constructor called\n";
    }
    ~Engine() {
        cout << "Engine Destructor called\n";
    }
    void print() {
        cout << "cyl cnt: " << cylinders_count << endl;
        cout << "cyl arr: " << cylinders_arrangement;
        cout << endl << "eng cc: " << engine_cc << endl;
    }
};
```

```
class Automobile : public Vehicle{
    Engine e;
protected:
    int gear_count;
public:
    Automobile(int wheels_count, float max_speed,
        string manufacturer, int cylinders_count,
        string cylinders_arrangement, float engine_cc,
        int gear_count):
    gear_count(gear_count),
    e(cylinders_count, cylinders_arrangement,
        engine_cc),
    Vehicle(wheels_count, max_speed, manufacturer) {
        cout << "Automobile constructor called\n";
    }
    ~Automobile() {
        cout << "Automobile Destructor called\n";
    }
    void print() {
        vehicle_print();
        e.print();
        cout << "Gear Count: " << gear_count << endl;
    }
};

int main() {
    Automobile a(4, 200, "Tata", 4, "V-shape", 1200, 6);
    a.print();
    return 0;
}
```

```
Vehicle constructor called
Engine constructor called
Automobile constructor called
Wheel Count: 4
Max Speed: 200
Manufacturer: Tata
cyl cnt: 4
cyl arr: V-shape
eng cc: 1200
Gear Count: 6
Automobile Destructor called
Engine Destructor called
Vehicle Destructor called
```

Different types of inheritance

```
class A {
protected:
    int num1 = 1;
};

class B {
protected:
    int num2 = 2;
};

class C: public A, public B {
protected:
    int num3 = 3;
public:
    void print() {
        cout << num1 << num2 << num3;
    }
};

int main() {
    C c;
    c.print();
    return 0;
}
```

```
class A {
protected:
    int num1 = 1;
public:
    A() {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num2 = 2;
public:
    B() {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
```

```
class C: public A, public B {
protected:
    int num3 = 3;
public:
    C() {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        cout << num1 << num2 << num3 << endl;
    }
};

int main() {
    C c;
    c.print();
    return 0;
}
```

```
A constructor called
B constructor called
C constructor called
123
C destructor called
B destructor called
A destructor called
```

```
class A {
protected:
    int num1;
public:
    A(int n): num1(n) {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num2 = 2;
public:
    B(int n): num2(n) {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
    void print() {
        cout << num2 << endl;
    }
};
```

```
class C: public B, public A {
protected:
    int num3 = 3;
public:
    B b;
    C(int num1, int num2, int num3, int num4):
    A(num2),
    B(num1),
    num3(num3),
    b(num4) {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        cout << num1 << num2 << num3;
    }
};

int main() {
    C c(1, 3, 2, 4);
    c.print();
    c.b.print();
    return 0;
}
```

```
B constructor called
A constructor called
B constructor called
C constructor called
3124
C destructor called
B destructor called
A destructor called
B destructor called
```

```
class A {
protected:
    int num1;
public:
    A(int n): num1(n) {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num2;
public:
    B(int n): num2(n) {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
    void print() {
        cout << num2 << endl;
    }
};
```

```
class C: public B, public A {
protected:
    int num3;
public:
    B b;
    C(int num1, int num2, int num3, int num4):
    A(num2),
    B(num1),
    num3(num3),
    b(num4) {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        cout << num1 << num2 << num3;
    }
};

int main() {
    C c(1, 3, 2, 4);
    c.print();
    c.b.print();
    return 0;
}
```

```
B constructor called
A constructor called
B constructor called
C constructor called
3124
C destructor called
B destructor called
A destructor called
B destructor called
```

```
class A {
protected:
    int num1;
public:
    A(int n): num1(n) {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num2;
public:
    B(int n): num2(n) {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
    void print() {
        cout << num2 << endl;
    }
};
```

```
class C: public B, public A {
protected:
    int num3 = 1;
    int num4;
public:
    B b;
    C(int n1, int n2, int n3, int n4):
    A(n2),
    B(n1),
    num4(num3 * 2),
    num3(n3),
    b(n4) {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        cout << num1 << num2 << num3 << num4;
    }
};
int main() {
    C c(1, 3, 2, 4);
    c.print();
    c.b.print();
    return 0;
}
```

```
B constructor called
A constructor called
B constructor called
C constructor called
31244
C destructor called
B destructor called
A destructor called
B destructor called
```

```
class A {
protected:
    int num1;
public:
    A(int n): num1(n) {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num1;
public:
    B(int n): num1(n) {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
    void print() {
        cout << num1 << endl;
    }
};
```

```
class C: public B, public A {
protected:
    int num3 = 1;
    int num4;
public:
    B b;
    C(int n1, int n2, int n3, int n4):
    A(n2),
    B(n1),
    num4(num3 * 2),
    num3(n3),
    b(n4) {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        // error: reference to 'num1' is ambiguous
        cout << num1 << num1 << num3 << num4;
    }
};
int main() {
    C c(1, 3, 2, 4);
    c.print();
    c.b.print();
    return 0;
}
```

```
class A {
protected:
    int num1;
public:
    A(int n): num1(n) {
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};

class B {
protected:
    int num1;
public:
    B(int n): num1(n) {
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
    void print() {
        cout << num1 << endl;
    }
};
```

```
class C: public B, public A {
protected:
    int num3 = 1;
    int num4;
public:
    B b;
    C(int n1, int n2, int n3, int n4):
    A(n2),
    B(n1),
    num4(num3 * 2),
    num3(n3),
    b(n4) {
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
    void print() {
        cout << A::num1 << B::num1 << num3 << num4;
    }
};
int main() {
    C c(1, 3, 2, 4);
    c.print();
    c.b.print();
    return 0;
}
```

```
B constructor called
A constructor called
B constructor called
C constructor called
31244
C destructor called
B destructor called
A destructor called
B destructor called
```

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
```

```
class D: public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {
        // error: reference to num1 is ambiguous
        cout << num1 << endl;
    }
};

int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
```

```
class D: public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {
        // error: 'A' is an ambiguous base of 'D'
        cout << A::num1 << endl;
    }
};

int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
```

```
class D: public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {

        cout << B::num1 << " " << C::num1 << endl;
    }
};

int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
A constructor called
B constructor called
A constructor called
C constructor called
D constructor called
1 7
D destructor called
C destructor called
A destructor called
B destructor called
A destructor called
```

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: virtual public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: virtual public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
class D: public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {

        cout << B::num1 << " " << C::num1 << endl;
    }
};

int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
A constructor called
B constructor called
C constructor called
D constructor called
7 7
D destructor called
C destructor called
B destructor called
A destructor called
```

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: virtual public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: virtual public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
```

```
class E {
public:
    E(){
        cout << "E constructor called\n";
    }
    ~E() {
        cout << "E destructor called\n";
    }
};
class D: public E, public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {
        cout << B::num1 << " " << C::num1 << endl;
    }
};
int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
A constructor called
E constructor called
B constructor called
C constructor called
D constructor called
7 7
D destructor called
C destructor called
B destructor called
E destructor called
A destructor called
```

Virtual base class is constructed before any other class in multiple inheritance

```
class A {
public:
    int num1 = 1;
    A(){
        cout << "A constructor called\n";
    }
    ~A() {
        cout << "A destructor called\n";
    }
};
class B: public A {
public:
    B(){
        cout << "B constructor called\n";
    }
    ~B() {
        cout << "B destructor called\n";
    }
};
class C: virtual public A {
public:
    C(){
        cout << "C constructor called\n";
    }
    ~C() {
        cout << "C destructor called\n";
    }
};
class E: virtual public A {
public:
    E(){
        cout << "E constructor called\n";
    }
    ~E() {
        cout << "E destructor called\n";
    }
};
class D: public E, public B, public C {
public:
    D(){
        cout << "D constructor called\n";
    }
    ~D() {
        cout << "D destructor called\n";
    }
    void print() {
        cout << B::num1 << " " << C::num1;
        cout << " " << E::num1 << endl;
    }
};
int main() {
    D d;
    d.C::num1 = 7;
    d.print();
    return 0;
}
```

```
A constructor called
E constructor called
A constructor called
B constructor called
C constructor called
D constructor called
1 7 7
D destructor called
C destructor called
B destructor called
A destructor called
E destructor called
A destructor called
```

Virtual base class is constructed before any other class in multiple inheritance

Interesting reads

- Accessing protected members in a derived class trhough object of base class is not permitted
  - https://stackoverflow.com/questions/3247671/accessing-protected-members-in-a-derived-class
- Inheritance of constructors and destructor
  - https://stackoverflow.com/questions/14184341/c-constructor-destructor-inheritance