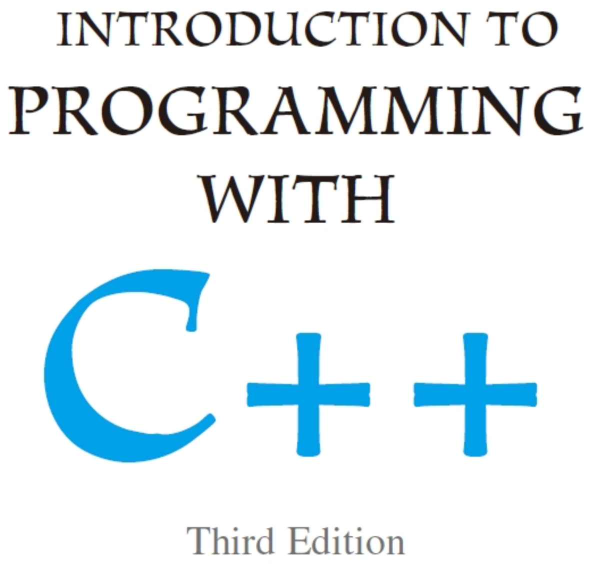
Introduction to Programming with C++

Chieh-Sen (Jason) Huang

Department of Applied Mathematics

National Sun Yat-sen University



INTRODUCTION TO
PROGRAMMING
WITH
C++

Third Edition

Contents are based on book by Y. Daniel Liang

Inheritance

- Suppose you are to define classes to model circles, rectangles, and triangles. These classes have many common features. What is the best way to design them to avoid redundancy? The answer is to use inheritance.

- Inheritance enables you to define a general class (i.e., a base class) and later extend it to more specialized classes (i.e., derived classes).

- `class Rectangle: public Shape` tells the compiler that the (Rectangle) class is derived from the base class (Shape).

- The `Rectangle` class Inherits `Shape` class **public(ly)**, so all public members in `Shape` are inherited as public members in `Rectangle`.

- Note that all private members in `Shape` can NOT be accessed in `Rectangle`.

Inheritance

- We can not access private members x and y in Shape from Rectangle.

```
class Shape { // List15_1a.cpp
public:
  Shape (double i = 0, double j = 0)
  { x = i;  y = j;}
  double getX() { return x;}
  double getY() { return y;}
private:
  double x, y;   // The starting point of a Shape
};
class Rectangle : public Shape   { //
public:
  Rectangle(double i, double j, double k, double l):Shape(i,j)
  { x = k; y = l; }
  double getX() { return x;}  // We use the same
  double getY() { return y;}  // names of member
  double area();              // functions as in Shape
private:
  double x,y;             // These are private members of Rectangle
};
double Rectangle::area()
{  return (x - Shape::getX()) * (y - Shape::getY());}
```

Inheritance

- **Protected** members on base class can be access from derivated class if we **public(ly)** inherits them. And they are now protected members.
- We use `Shape::x` to refer the `x` from `shape`.

```
class Shape { // List15_2a.cpp
public:
  Shape (double i = 0, double j = 0) { x = i;  y = j;}
  double getX() { return x;}
  double getY() { return y;}
protected:
  double x, y;   // The starting point of a Shape
};
class Rectangle : public Shape   { //
public:
  Rectangle(double i, double j, double k, double l)
  {Shape::x = i, Shape::y = j, x = k; y = l; }
  double area();               // functions as in Shape
private:
  double x,y;          // These are private members of Rectangle
};
double Rectangle::area()
{ return (x - Shape::getX()) * (y - Shape::getY());}
```

Inheritance

- Case study: Class `Circle` and `GeometricObject`

```
#include "GeometricObject.h"
#include "DerivedCircle.h"
#include "DerivedRectangle.h"
#include <iostream>
using namespace std;

int main() // List15_7.cpp
{
 GeometricObject shape;
 Circle circle(5);
 Rectangle rectangle(2, 3);
}
```

```
class GeometricObject
{
 public:
  GeometricObject();
  GeometricObject(const string& color,
                  bool filled);
  string getColor() const;
  void setColor(const string& color);
  bool isFilled() const;
  void setFilled(bool filled);
  string toString() const;

 private:
  string color;
  bool filled;
};
```

Inheritance

```
class Circle: public GeometricObject
{
 public:
  Circle();
  Circle(double);
  Circle(double radius,
   const string& color, bool filled);
  double getRadius() const;
  void setRadius(double);
  double getArea() const;
  double getPerimeter() const;
  double getDiameter() const;
  string toString() const;

 private:
  double radius;
};
}
```

```
class Rectangle: public GeometricObject
{
 public:
 Rectangle();
 Rectangle(double width, double height);
 Rectangle(double width, double height,
 const string& color, bool filled);
 double getWidth() const;
 void setWidth(double);
 double getHeight() const;
 void setHeight(double);
 double getArea() const;
 double getPerimeter() const;
 string toString() const;

 private:
 double width;
 double height;
};
```

- `class Circle: public GeometricObject` tells the compiler that the (circle) class is derived from the base class (GeometricObject).

- The Circle class Inherits GeometricObject class **publicly**, so all public members in GeometricObject are inherited as public members in Circle.

- Note that all private members in `GeometricObject` can NOT be accessed in `Circle`.

Inheritance

- The constructor `Circle(double radius, const string& color, bool filled)` is implemented by invoking the `setColor` and `setFilled` functions to set the color and filled properties.

- Note that These two public functions are defined the base class `GeometricObject` and are inherited in `Circle`. So, they can be used in the derived class.

```
Circle::Circle(double radius, const string& color, bool filled)
{
  setRadius(radius);
  setColor(color);
  setFilled(filled);
}
```

Inheritance

- Note that all private members in `GeometricObject` can NOT be accessed in `Circle`.
- You might attempt to use the data fields `color` and `filled` directly in the constructor as follows:

```
Circle::Circle(double radius, const string& c, bool f)
{
  this->radius = radius; // This is fine
  color = c; // Illegal since color is private in the base class
  filled = f; // Illegal since filled is private in the base class
}
```

Multiple Inheritance

- Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes.
- The constructors of inherited classes are called in the same order in which they are inherited.
- Child is protected inherited Father and Mother, therefore the proteced members are interited as protected members in Child class.

```
int main()
{            // List15_3a.cpp
  Child Baby;
  Baby.Show();
  cout << "[Father + Mother] "
       << Baby.GetMoney() << endl;
  return 0;
}
```

```
class Child:protected Father, protected Mother
{
 private:
  char m_szEmail[32];
 public:
  Child();
  inline char *GetEmail();
  void Show();
  long GetMoney();
};
```

Multiple Inheritance

```
class Father
{
 private:
  char m_szWife[32];
 protected:
  char m_szSurname[32];
  char m_szAddress[32];
  long m_lMoney;
 public:
  Father();
  inline char *GetSurname();
  inline char *GetAddress();
  inline long GetMoney();
};
```

```
class Mother
{
 private:
  char m_szHusband[32];
 protected:
  char m_szSurname[32];
  char m_szTelephone[32];
  long m_lMoney;
 public:
  Mother();
  inline char *GetSurname();
  inline char *GetTelephone();
  inline long GetMoney();
};
```

- All three classes have `GetMoney()` function.
- `Baby.GetMoney()` will call their parent's `GetMoney()` functions.

```
long Child::GetMoney()
{
 return Father::GetMoney() + Mother::GetMoney();
}
```