

南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

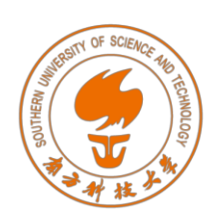
# C/C++ Program Design

## CS205

**Prof. Shiqi Yu (于仕琪)**

yusq@sustech.edu.cn

<http://faculty.sustech.edu.cn/yusq/>



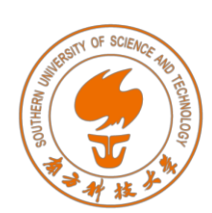
南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Improve Your Source Code



# Suggestions to your Project 3

- Use size\_t for mat.cols and mat.rows
- Use memcpy() to copy data. Element assignment has a lower efficiency.
- Use 1D array (float\*) nor 2D array (float\*\*) for matrix data.
- Redundant computation in loops
- Do parameter checking in functions: null pointers, dimension matching in matrix operations, etc 检查参数
- Do not bind the create matrix function with file I/O.
- File name: head.h, source1.c, source2.c, source3.c
- Good implementation VS good homework.



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Derived Classes



# Inheritance

- Inherit members (attributes and functions) from one class
  - Base class (parent)
  - Derived class (child)
- C++ supports multiple inheritance and multilevel inheritance

```
class Base
{
    public:
    int a;
    int b;
};
class Derived: public Base
{
    public:
    int c;
};
```

```
class Derived: public Base1, public Base2
{
    ...
};
```



# Constructors

- To instantiate a derived class object
  - Allocate memory
  - Derived constructor is invoked
    - ✓ Base object is constructed by a base constructor
    - ✓ Member initializer list initializes members
    - ✓ To execute the body of the derived constructor

```
class Derived: public Base
{
public:
    int c;
    Derived(int c): Base(c - 2, c - 1), c(c)
    {
        ...
    }
};
```



# Destructors

- The destructor of the derived class is invoked first,
- Then the destructor of the base class.



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Access Control

访问控制





# Member Access

- Public members
  - Accessible anywhere
- Private members
  - Only accessible to the members and friends of that class

```
class Person {  
    private:  
        int n; // private member  
    public:  
        // this->n is accessible  
        Person() : n(10) {}  
        // other.n is accessible  
        Person(const Person& other) : n(other.n) {}  
        // this->n is accessible  
        void set(int n) {this->n = n;}  
        // this->n and other.n are accessible  
        void set(const Person& other) {this->n = other.n;}  
};
```



# Member Access

```
// a non-member non-friend function
void compare(Base& b, Derived& d)
{
    // b.n++; // Error
    // d.n++; // Error
}
```

- Protected members
  - Accessible to the members and friends of that class
  - Accessible to the members and friends of the **derived** class

```
class Base
{
protected:
    int n;
private:
    void foo1(Base& b)
    {
        n++; // Okay
        b.n++; // Okay
    }
};
```

```
class Derived : public Base
{
    void foo2(Base& b, Derived& d)
    {
        n++; //Okay
        this->n++; //Okay
        //b.n++; //Error.
        d.n++; //Okay
    }
};
```



# Public Inheritance

- Public members of the base class
  - Still be public in the derived class
  - Accessible anywhere
- Protected members of the base class
  - Still be protected in the derived class
  - Accessible in the derived class only
- Private members of the base class
  - Not accessible in the derived class



# Protected Inheritance

- **Public** members and **protected** members of the base class
  - Be **protected** in the derived class
  - Accessible in the derived class only
- Private members of the base class
  - Not accessible in the derived class



# Private Inheritance

- **Public** members and **protected** members of the base class
  - Be **private** in the derived class
  - Accessible in the derived class only
- Private members of the base class
  - Not accessible in the derived class



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Virtual Functions



# Virtual Functions

- Let's look at the example first, what will be the output?

```
class Person
{
public:
    void print()
    {
        cout << "Name: " << name << endl;
    }
};

class Student: public Person
{
public:
    void print()
    {
        cout << "Name: " << name;
        cout << ". ID: " << id << endl;
    }
};
```

```
Person * p = new Student();
p->print(); // call Person::print()?
```

virtual.cpp



# Virtual Functions

- But if we define `print()` function as a virtual function, the output will be different.
- **Static** binding: the compiler decides which function to call
- **Dynamic** binding: the called function is decided at runtime.
- Keyword `virtual` makes the function virtual for the base and all derived classes.

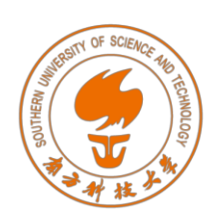




# Virtual Destructors

- If a virtual destructor is not virtual, only the destructor of the base class is executed in the follow examples.

```
Person * p = new Student("xue", "2020");  
p->print();  
...  
...  
delete p; //if its destructor is not virtual
```



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Inheritance and Dynamic Memory Allocation



# Question

- If a base class uses dynamic memory allocation, and redefines a copy constructor and assignment operator
- Case 1: If no dynamic memory allocation in the derived class, no special operations are needed
- Case 2: if dynamic memory is allocated in the derived class, you should redefine a copy constructor and an assignment operator.



# Case 2

```
class MyMap: public MyString
{
    char * keyname;
public:
    MyMap(const char * key, const char * value)
    {
        ...
    }
    MyMap(const MyMap & mm): MyString(mm.buf_len, mm.characters)
    {
        //allocate memory for keyname
        //and hard copy from mm to *this
    }
    MyMap & operator=(const MyMap &mm)
    {
        MyString::operator=(mm);
        //allocate memory for keyname
        //and hard copy from mm to *this
        return *this;
    }
};
```



南方科技大学  
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Examples in OpenCV



# Derived cv::Mat\_

- Template matrix class derived from cv::Mat, a wrapper, more C++ style.

modules/core/include/opencv2/core/mat.hpp

```
2198     template<typename _Tp> class Mat_ : public Mat
2199     {
2200     public:
2201         typedef _Tp value_type;
2202         typedef typename DataType<_Tp>::channel_type channel_type;
2203         typedef MatIterator_<_Tp> iterator;
2204         typedef MatConstIterator_<_Tp> const_iterator;
2205
2206         //!< default constructor
2207         Mat_() CV_NOEXCEPT;
2208         //!< equivalent to Mat(_rows, _cols, DataType<_Tp>::type)
2209         Mat_(int _rows, int _cols);
2210         //!< constructor that sets each matrix element to specified value
2211         Mat_(int _rows, int _cols, const _Tp& value);
2212         //!< equivalent to Mat(_size, DataType<_Tp>::type)
2213         explicit Mat_(Size _size);
2214         //!< constructor that sets each matrix element to specified value
2215         Mat_(Size _size, const _Tp& value);
```



# cv::Matx

- A template class for small matrices whose type and size are known at compilation time.

modules/core/include/opencv2/core/matx.hpp

```
99  template<typename _Tp, int m, int n> class Matx
100  {
101  public:
102      enum {
103          rows      = m,
104          cols      = n,
105          channels    = rows*cols,
106  #ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
107          depth      = traits::Type<_Tp>::value,
108          type       = CV_MAKETYPE(depth, channels),
109  #endif
110          shortdim   = (m < n ? m : n)
111      };
112
113      typedef _Tp          value_type;
114      typedef Matx<_Tp, m, n> mat_type;
115      typedef Matx<_Tp, shortdim, 1> diag_type;
116
117      //! default constructor
118      Matx();
```



# cv::Vec

modules/core/include/opencv2/core/matx.hpp

```
template<typename _Tp, int cn> class Vec : public Matx<_Tp, cn, 1>
{
public:
    typedef _Tp value_type;
    enum {
        channels = cn,
#ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
        depth     = Matx<_Tp, cn, 1>::depth,
        type      = CV_MAKETYPE(depth, channels),
#endif
        _dummy_enum_finalizer = 0
    };

    //! default constructor
    Vec();

    Vec(_Tp v0); //!< 1-element vector constructor
    Vec(_Tp v0, _Tp v1); //!< 2-element vector constructor
    Vec(_Tp v0, _Tp v1, _Tp v2); //!< 3-element vector constructor
    Vec(_Tp v0, _Tp v1, _Tp v2, _Tp v3); //!< 4-element vector constructor
    Vec(_Tp v0, _Tp v1, _Tp v2, _Tp v3, _Tp v4); //!< 5-element vector constructor
```

`Vec<float, 3> xyz(1.2f, 2.3f, 3.4f);`





# Combined with typedef

modules/core/include/opencv2/core/matx.hpp

```
409  typedef Vec<uchar, 2> Vec2b;
410  typedef Vec<uchar, 3> Vec3b;
411  typedef Vec<uchar, 4> Vec4b;
412
413  typedef Vec<short, 2> Vec2s;
414  typedef Vec<short, 3> Vec3s;
415  typedef Vec<short, 4> Vec4s;
416
417  typedef Vec<ushort, 2> Vec2w;
418  typedef Vec<ushort, 3> Vec3w;
419  typedef Vec<ushort, 4> Vec4w;
420
421  typedef Vec<int, 2> Vec2i;
422  typedef Vec<int, 3> Vec3i;
423  typedef Vec<int, 4> Vec4i;
424  typedef Vec<int, 6> Vec6i;
425  typedef Vec<int, 8> Vec8i;
426
427  typedef Vec<float, 2> Vec2f;
428  typedef Vec<float, 3> Vec3f;
429  typedef Vec<float, 4> Vec4f;
430  typedef Vec<float, 6> Vec6f;
```

`Vec<float, 3> xyz(1.2f, 2.3f, 3.4f);`



`Vec3f xyz(1.2f, 2.3f, 3.4f);`



# Combined with typedef

```
221 typedef Matx<float, 1, 2> Matx12f;
222 typedef Matx<double, 1, 2> Matx12d;
223 typedef Matx<float, 1, 3> Matx13f;
224 typedef Matx<double, 1, 3> Matx13d;
225 typedef Matx<float, 1, 4> Matx14f;
226 typedef Matx<double, 1, 4> Matx14d;
227 typedef Matx<float, 1, 6> Matx16f;
228 typedef Matx<double, 1, 6> Matx16d;
229
230 typedef Matx<float, 2, 1> Matx21f;
231 typedef Matx<double, 2, 1> Matx21d;
232 typedef Matx<float, 3, 1> Matx31f;
233 typedef Matx<double, 3, 1> Matx31d;
234 typedef Matx<float, 4, 1> Matx41f;
235 typedef Matx<double, 4, 1> Matx41d;
236 typedef Matx<float, 6, 1> Matx61f;
237 typedef Matx<double, 6, 1> Matx61d;
238
239 typedef Matx<float, 2, 2> Matx22f;
240 typedef Matx<double, 2, 2> Matx22d;
241 typedef Matx<float, 2, 3> Matx23f;
242 typedef Matx<double, 2, 3> Matx23d;
```

```
Matx33f m(1, 2, 3,
          4, 5, 6,
          7, 8, 9);
cout << sum(Mat(m*m.t())) << endl;
```