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/

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
- 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.

matrix/

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.

derive.cpp

Access Control

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

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;
    }
};
```

virtual.cpp

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

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
```

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: pubic 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

```
template<typename _Tp> class Mat_ : public Mat
{
public:
    typedef _Tp value_type;
    typedef typename DataType<_Tp>::channel_type channel_type;
    typedef MatIterator_<_Tp> iterator;
    typedef MatConstIterator_<_Tp> const_iterator;

    //! default constructor
    Mat_() CV_NOEXCEPT;
    //! equivalent to Mat(_rows, _cols, DataType<_Tp>::type)
    Mat_(int _rows, int _cols);
    //! constructor that sets each matrix element to specified value
    Mat_(int _rows, int _cols, const _Tp& value);
    //! equivalent to Mat(_size, DataType<_Tp>::type)
    explicit Mat_(Size _size);
    //! constructor that sets each matrix element to specified value
    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

```cpp
template<typename _Tp, int m, int n> class Matx
{
public:
    enum {
           rows     = m,
           cols     = n,
           channels = rows*cols,
#ifdef OPENCV_TRAITS_ENABLE_DEPRECATED
           depth    = traits::Type<_Tp>::value,
           type     = CV_MAKETYPE(depth, channels),
#endif
           shortdim = (m < n ? m : n)
         };

    typedef _Tp                           value_type;
    typedef Matx<_Tp, m, n>               mat_type;
    typedef Matx<_Tp, shortdim, 1> diag_type;

    //! default constructor
    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 construct
    Vec(_Tp v0, _Tp v1, _Tp v2, _Tp v3, _Tp v4); //!< 5-element vector c
```

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

Combined with typedef

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

```
typedef Vec<uchar, 2> Vec2b;
typedef Vec<uchar, 3> Vec3b;
typedef Vec<uchar, 4> Vec4b;

typedef Vec<short, 2> Vec2s;
typedef Vec<short, 3> Vec3s;
typedef Vec<short, 4> Vec4s;

typedef Vec<ushort, 2> Vec2w;
typedef Vec<ushort, 3> Vec3w;
typedef Vec<ushort, 4> Vec4w;

typedef Vec<int, 2> Vec2i;
typedef Vec<int, 3> Vec3i;
typedef Vec<int, 4> Vec4i;
typedef Vec<int, 6> Vec6i;
typedef Vec<int, 8> Vec8i;

typedef Vec<float, 2> Vec2f;
typedef Vec<float, 3> Vec3f;
typedef Vec<float, 4> Vec4f;
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

```cpp
typedef Matx<float, 1, 2> Matx12f;
typedef Matx<double, 1, 2> Matx12d;
typedef Matx<float, 1, 3> Matx13f;
typedef Matx<double, 1, 3> Matx13d;
typedef Matx<float, 1, 4> Matx14f;
typedef Matx<double, 1, 4> Matx14d;
typedef Matx<float, 1, 6> Matx16f;
typedef Matx<double, 1, 6> Matx16d;

typedef Matx<float, 2, 1> Matx21f;
typedef Matx<double, 2, 1> Matx21d;
typedef Matx<float, 3, 1> Matx31f;
typedef Matx<double, 3, 1> Matx31d;
typedef Matx<float, 4, 1> Matx41f;
typedef Matx<double, 4, 1> Matx41d;
typedef Matx<float, 6, 1> Matx61f;
typedef Matx<double, 6, 1> Matx61d;

typedef Matx<float, 2, 2> Matx22f;
typedef Matx<double, 2, 2> Matx22d;
typedef Matx<float, 2, 3> Matx23f;
typedef Matx<double, 2, 3> Matx23d;
```

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