河北师范大学软件学院
Software College of Hebei Normal University

C++

第六讲 类和对象（二）

基础课教研室C++ 课程组

上一讲教学目标

- 理解C++中类的概念
- 掌握C++中类的定义与访问

本讲教学目标

- 掌握C++中类的构造函数
- 掌握C++中类的析构函数
- 掌握C++中类的拷贝构造函数

1 构造函数

2 析构函数

3 拷贝构造函数

构造函数

- 对象的初始化

概念：**在定义对象时为对象赋初值。**

注意：初始化就是开辟内存单元同时对数据成员给出明确的值。

构造函数

- 不能在定义类时初始化成员

```
class Time
{
public:
    void set(int h, int m, int s);
    void display();
private:
    int m_iHour = 0;
    int m_iMinute = 0;
    int m_iSec = 0;
};
```

构造函数

- 如果一个类中的数据成员为public，则可在定义对象时对数据成员初始化（不推荐）

```
class Time
{
public:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};
```

```
int main(void)
{
    Time t1 = {12,56,30};
    cout << t1.m_iHour << ":"
         << t1. m_iMnute << ":"
         << t1. m_iSec << endl;
     return 0;
}
```

如果数据成员的访问属性非public,则不能采用此法

构造函数

- 下面这种方法不是初始化

```
#include <iostream>
using namespace std;
class Time {
public:
    void init(int aHour,
    int aMin, int aSec);
    void display();
private:
    int m_iHour, m_iMinute,
        m_iSec;
};
void Time::init(int aHour,
       int aMin, int aSec)
{
    m_iHour   = aHour;
    m_iMinute = aMin;
    m_iSec    = aSec;
}
```

```
void Time::display()
{
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;

}

int main(void)
{
    Time time;
    time.init(12,30,15);
    time.display();

    return 0;
}
```

构造函数

- C++采用构造函数为对象初始化

功能：初始化对象
函数名：与类名相同
返回值：不指定返回值（不能写void ）
位置：通常作为public成员
内容：任意，通常只包含成员赋值语句
调用方法：**通常**创建对象时自动调用

```
Time time;
Time time = Time(); //OK
Time *p = new Time();
```

构造函数

- **无参构造函数**
- 有参构造函数
- 构造函数与参数初始化列表
- 默认构造函数

构造函数 – 无参构造函数

格式：

```
[类名::]类名()
{
  … …
}
```

调用：

```
类名   对象名；
```

[注意]

- 在实例化对象时自动调用
- 对象名后不能含有括号"类名 对象名( )"

构造函数 – 无参构造函数

- 下面这种方法不是初始化

```
class Time
{
public:
    Time();
    void display();
private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};

Time::Time()
{
    m_iHour   = 0;
    m_iMinute = 0;
    m_iSec    = 0;
}
```

```
void Time::display()
{
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;
}
int main(void)
{
    Time t1;
    t1.display();
    Time t2 = Time();
    t2.display();
    Time *pT = new Time;
    pT->display();

    delete pT;
    pT = NULL;
    return 0;
}
```

构造函数 – 无参构造函数

[注意]

在构造函数中最好只是对数据成员赋初值，不提倡加入与初始化无关的语句

```
Time::Time()
{
    m_iHour = 0;
    m_iMinute = 0;
    m_iSec = 0;
    cout << "Invoke Constructor" << endl;
}
```

构造函数

- 无参构造函数
- **有参构造函数**
- 构造函数与参数初始化列表
- 默认构造函数

构造函数 – 有参构造函数

格式：

```
[类名::]类名(参数表 )
{
  … …
}
```

调用：

```
类名 对象名 ( 实参1,实参2,… ) ;
```

```
类名 对象名 = 类名(实参1,实参2,… ) ;
```

```
new 类名(实参1,实参2,… ) ;
```

构造函数 – 有参构造函数

```
class Time
{
public:
    Time(int aHour,
     int aMinute, int aSec);
    void display();

private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};

Time::Time(int aHour,
    int aMinute, int aSec)
{
    m_iHour   = aHour;
    m_iMinute = aMinute;
    m_iSec    = aSec;
}
```

```
void Time::display()
{
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;

}
int main(void)
{
    Time t1(12,30,20);
    t1.display();

    Time t2 = Time(0,0,0);
    t2.display();

    Time *pT = new Time(1, 2, 3);
    pT->display();
    delete pT;
    return 0;
}
```

构造函数 – 有参构造函数

- 由于构造函数可含有参数，因此构造函数可以重载

```
class Time {
public:
    Time(int aHour,
       int aMinute, int aSec);
    Time();
    void display();
private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};

Time::Time(int aHour,
    int aMinute, int aSec)
{
     m_iHour   = aHour;
     m_iMinute = aMinute;
     m_iSec    = aSec;
}
```

```
Time::Time() {
     m_iHour   = 0;
     m_iMinute = 0;
     m_iSec    = 0;
}
void Time::display() {
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;

}
int main(void) {
    Time Time(12,30,20);
    Time.display();
    Time time;
    time.display();

    return 0;
}
```

构造函数 – 有参构造函数

- 重载构造函数时一定要防止产生二义性。

```
Time();                    // 声明无参构造函数
Time(int ,int);            // 有两个参数的构造函数
Time(int ,int=10, int=10); // 有一个参数不是默认参数
… …

Time time1;
Time time2(15);            // 调用第三个
Time time3(15, 30); // 调用第二个还是第三个？ Error
```

构造函数

- 无参构造函数
- 有参构造函数
- **构造函数与参数初始化列表**
- 默认构造函数

构造函数 – 初始化列表

```
Time::Time(int aHour, int aMinute, int aSec)
    : m_iHour(aHour), m_iMinute(aMinute), m_iSec(aSec)
{
    ...
}
```

构造函数 – 初始化列表

[注意]

- const成员、引用成员及无默认构造函数子对象成员必须使用初始化列表初始化

```
class ConstRef
{
public:
    ConstRef(int aVal);
private:
    int   m_iVal;
    const int m_ci;
    int   &m_r;
};
```

```
ConstRef::ConstRef(int aVal)
    :m_ival(aVal)
{
    m_ci = aVal;   // Error
    m_r  = m_iVal; // Error
}
```

构造函数 – 初始化列表

```
class ConstRef
{
public:
    ConstRef(int aVal);

private:
    int m_iVal;
    const int m_ci;
};
```

```
ConstRef::ConstRef(int aVal)
    : m_iVal(aVal), m_ci(aVal)
{}

int main(void)
{
    ConstRef obj(4);

    return 0;
}
```

初始化列表只在实现时出现

构造函数 – 初始化列表

- 构造函数初始化列表不能指定初始化次序，必须按照成员声明的顺序编写构造函数初始化列表。
- 尽量避免使用成员初始化成员。

```
class Point
{
public:
    Point(int aVal)
        :m_iY(aVal), m_iX(m_iY) // Error
    { }
private:
    int m_iX;
    int m_iY;
};
```

构造函数

- 无参构造函数
- 有参构造函数
- 构造函数与参数初始化列表
- **默认构造函数**

构造函数 – 默认构造函数

- 概念：初始化对象时不提供参数时所调用的构造函数

```
int main(void)
{
    Time time1;
    Time time2(); // Error
    Time &r = *new Time;
    delete Time;
    return 0;
}
```

思考：
创建对象若不提供参数调用什么类型的构造函数?

构造函数 – 默认构造函数

- **自定义的构造函数**
- 由系统创建的构造函数
- 含默认参数的构造函数

构造函数 – 默认构造函数

```
class Time {
public:
    Time();
    void display();
private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};

Time::Time() {
    m_iHour   = 0;
    m_iMinute = 0;
    m_iSec    = 0;
}
```

```
void Time::display() {
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;
}
int main(void) {
    Time t1;
    t1.display();
    Time t2 = Time();
    t2.display();
    Time * p = new Time;
    p->display();
    delete p;  p = NULL;
    return 0;
}
```

构造函数 – 默认构造函数

- 自定义的构造函数
- **由系统创建的构造函数**
- 含默认参数的构造函数

构造函数 – 默认构造函数

- 类体中若没有构造函数，则系统会生成一个默认构造函数。

```
class Time {
public:
    void display();
private:
    int m_iHour;
    int m_iMinute;
    int m_iSec;
};
void Time::display() {
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;
}
```

```
int main(void)
{
    Time localTime;
    localTime.display();
    return 0;
}
```

类体中若含有构造函数则系统不会生成默认构造函数

系统创建的默认构造函数就是函数体为空的无参构造函数
Time::Time( ) {}

构造函数 – 默认构造函数

- 自定义的构造函数
- 由系统创建的构造函数
- **含默认参数的构造函数**

构造函数 – 默认构造函数

```
class Time {
public:
    Time(int aHour=0,
       int aMinute=0, int aSec=0);
    void display();
private:
    int m_iHour;  int m_iMinute;
    int m_iSec;
};
Time::Time(int aHour,
  int aMinute, int aSec) {
    m_iHour   = aHour;
    m_iMinute = aMinute;
    m_iSec    = aSec;
}
```

```
void Time::display() {
    cout << m_iHour << ":"
       << m_iMinute << ":"
       << m_iSec << endl;

}
int main(void)
{
    Time time2;
    time2.display();
    Time time1(12, 30, 20);
    time1.display();

    return 0;
}
```

构造函数 – 默认构造函数

[注意]

- 在声明构造函数时，形参名可以省略（不推荐）

```
Time(int = 10, int =10, int =10);
```

- 一个类只能有一个默认构造函数

```
Time ();    // 声明无参构造函数
Time (int aX=10,int aY=10,int aZ=10);
Time box1; // Error
```

- 推荐使用默认参数的默认构造函数，以提高代码重用。

构造函数 – 默认构造函数

- 一个类中只能含有一个默认构造函数

```
Time(int aX= 10 ,int aY= 10,int aZ= 10);
Time();

… …

Time   time1;              // 调用第一个还是第二个？
Time   time2(15,30); // 调用第一个还是第二个？
```

默认构造函数只能出现一次

构造函数 – 默认构造函数

总结：

- 构造函数的函数名与类名相同，不写返回值，因而不能指定包括void在内的任何返回值类型。
- 构造函数通常为public
- 构造函数是成员函数，它可在类体内或类体外实现。
- 构造函数可以重载，重载时防止产生“二义性”。

构造函数 – 默认构造函数

- 最好用参数初始化列表实现构造函数
- 类外实现构造函数时初始化列表在实现时出现！
- 无参构造函数和全部参数都含有默认值的构造函数都是默认构造函数
- 一个类只能含有一个默认构造函数
- 习惯上要求定义一个含默认参数的默认构造函数，以增强类的通用性

1 构造函数

2 析构函数

3 拷贝构造函数

析构函数

功能：在销毁类的对象前执行清除工作

格式：

```
[类名::]~类名 ( )
{
  ....
}
```

注：析构函数的函数名要和类名相同

```
class Student
{public:
    Student(...);
    ~Student();
    void display()const;
private:
    int    m_iNum;
    string m_strName;
    char   m_cSex;
};
Student::~Student()
{ cout<<"Destructor "<<endl;}
… …
```

析构函数

注意：

- 函数名与类名相同，且函数名前加~
- 没有参数、不能被重载
- 不指定返回值
- 常定义为public
- 对象生命期结束时自动调用
- 如果一个类没有定义析构函数，系统会默认产生一个

析构函数

```
class Student
{
public:
  Student(int aNum,string aName,char aS);
  ~Student();
  void display();
private:
  int    m_iNum;
  string m_strName;
  char   m_cSex;

};

Student::Student(int aNum,string aName,
                 char aS)
:m_iNum(aNum),m_strName(aName),m_cSex(aS)
{
  cout << "num:"<< m_iNum << " ";
  cout << "Constructor called" << endl;
}
```

```
Student::~Student()
{
  cout << "num:"<< m_iNum << " ";
  cout << "Destructor called" <<
endl;
}

int main(void)
{
  Student stud1(1,"Guangyu",'m');
  Student stud2(2,"Zhangfei",'m');

  return 0;
}
```

```
num:1 Constructor called
num:2 Constructor called
num:2 Destructor called
num:1 Destructor called
```

析构函数

思考：

通常不需人为定义析构函数，什么时候必须定义析构函数？

一般，当类中含有指针成员，并且在构造函数中用指针指向了一块堆中的内存，则必须定义析构函数释放该指针申请的动态空间。

析构函数

```
class MyString
{
public:
    MyString();
    void display()const;
private:
    char *m_pstr;
};

MyString::MyString()
{
    m_pstr = new char[1000];
    strcpy(m_pstr, "hello");
}

void MyString::display()const
{
    cout << m_pstr << endl;
}
```

```
/*
MyString::~MyString()
{
    //系统生成的
}
*/

int main(void)
{
    MyString * pStr
             = new MyString;
    pStr->display();
    delete pStr;

    return 0;
}
```

析构函数

```
class MyString
{
public:
    MyString();
    ~MyString();
    void display()const;
private:
    char *m_pstr;
};
MyString::MyString()
{
    m_pstr = new char[1000];
    strcpy(m_pstr, "hello");
}

MyString::~MyString()
{
   delete []m_pstr;
}
```

```
void MyString::display()const
{
    cout << m_pstr << endl;
}

int main(void)
{
    MyString * pStr
             = new MyString;
    pStr->display();
    delete pStr;

    return 0;
}
```

析构函数

```
class MyString
{
public:
    MyString(char *ap = "china");
    ~MyString();
    void display()const;
private:
    char *m_pstr;
};

void MyString::display() const
{
    cout << m_pstr << endl;
}
MyString::MyString(char *ap)
{
    m_pstr = new char[strlen(ap)+1];
    strcpy(m_pstr, ap);
}
```

```
MyString::~MyString ()
{
   delete []m_pstr;
}

int main(void)
{
    MyString str1("japan");
    str1.display();

    MyString str2;
    str2.display();

    return 0;
}
```

析构函数

- 如果要定义析构函数，通常也需要定义拷贝构造函数和赋值运算符的重载函数

1 构造函数

2 析构函数

3 拷贝构造函数

拷贝构造函数

❖ **拷贝构造函数**：用已存在的对象创建一个同类对象

➢ 格式1：**类名 对象名2（对象名1）**

```
Time t1(12,30,20);
Time t2(t1);
```

➢ 格式2：**类名 对象名2 = 对象名1**

```
Time t3 = t1;
```

拷贝构造函数

- 格式：

```
<类名>::<类名>(const <类名>& <引用名>)
{
    <函数体>;
}
```

拷贝构造函数

- 调用时间：当用已存在的对象初始化另一个同类对象时自动调用的构造函数。

[注意]

若类体内没有拷贝构造函数，则系统会生成一个默认拷贝构造函数，通常不需定义

拷贝构造函数

```
class Data
{public:
    Data(int aY=2009, int aM=9, int aD=1);
    ~Data();
    void Print();
private:
    int m_iYear,m_iMonth,m_iDay;
};
Data::Data(int aY,int aM,int aD)
{
    m_iYear = aY;
    m_iMonth = aM;
    m_iDay = aD;
    cout << "Constructor called.“
      << endl;
}
/* 系统自动生成
Data::Data(const Data&data)
{
  m_iYear = data.m_iYear;
  m_iMonth = data.m_iMonth;
  m_iDay = data.m_iDay;
} */
```

```
Data::~Data()
{    cout << "Destructor called." << endl; }
void Data::Print()
{
    cout << m_iYear <<"-" << m_iMonth <<"-"
        << m_iDay << endl;
}
Data foo(Data Q)
{
    Data R(Q);
    return R;
}
int main(void)
{
    Data m_iday1(2009,10,15);// 普通构造函数
    Data m_iday2(m_iday1);   // 拷贝构造函数
    Data m_iday3 = m_iday2;  // 拷贝构造函数
    Data m_iday4;            // 普通构造函数
    m_iday3 = m_iday4;  // 不调用任何构造函数
    Data s = foo(m_iday2);
   // 参数传递、执行过程中、函数返回时调用三次拷贝构造函数，调用结束后,马上调用三次析构函数

    return 0;
}
```

拷贝构造函数

既然系统已经给我们提供了默认的拷贝构造函数，那么我们是否就不需要去定义拷贝构造函数了？

拷贝构造函数

```
class MyString
{
public:
    MyString(char *ap = “china”);
    /*MyString(const MyString &r);*/
    ~MyString();
    void display()const;
private:
    char *m_pstr;
};
void MyString::display()const
{
    cout << m_pstr << endl;
}

MyString::MyString(char *ap)
{
    m_pstr = new char[strlen(ap)+1];
    strcpy(m_pstr, ap);
}
```

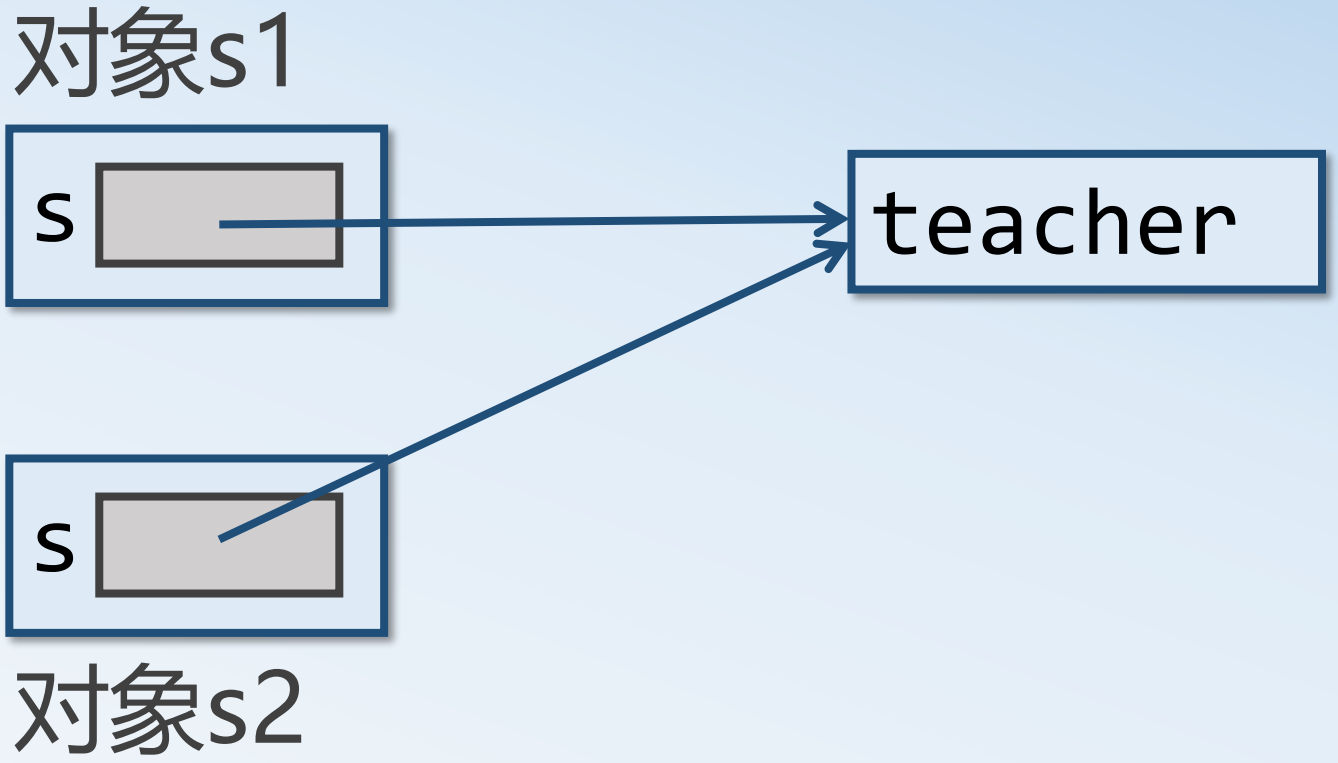
```
/*
MyString::MyString(const MyString &r)
{
    m_pstr = r.m_pstr;
} // 模拟默认拷贝构造函数
*/

MyString::~MyString()
{
   delete []m_pstr;
}

int main(void)
{
    MyString str1("teacher");
    MyString str2(str1);
    str2.display();
    return 0;
} // 运行时错误
```

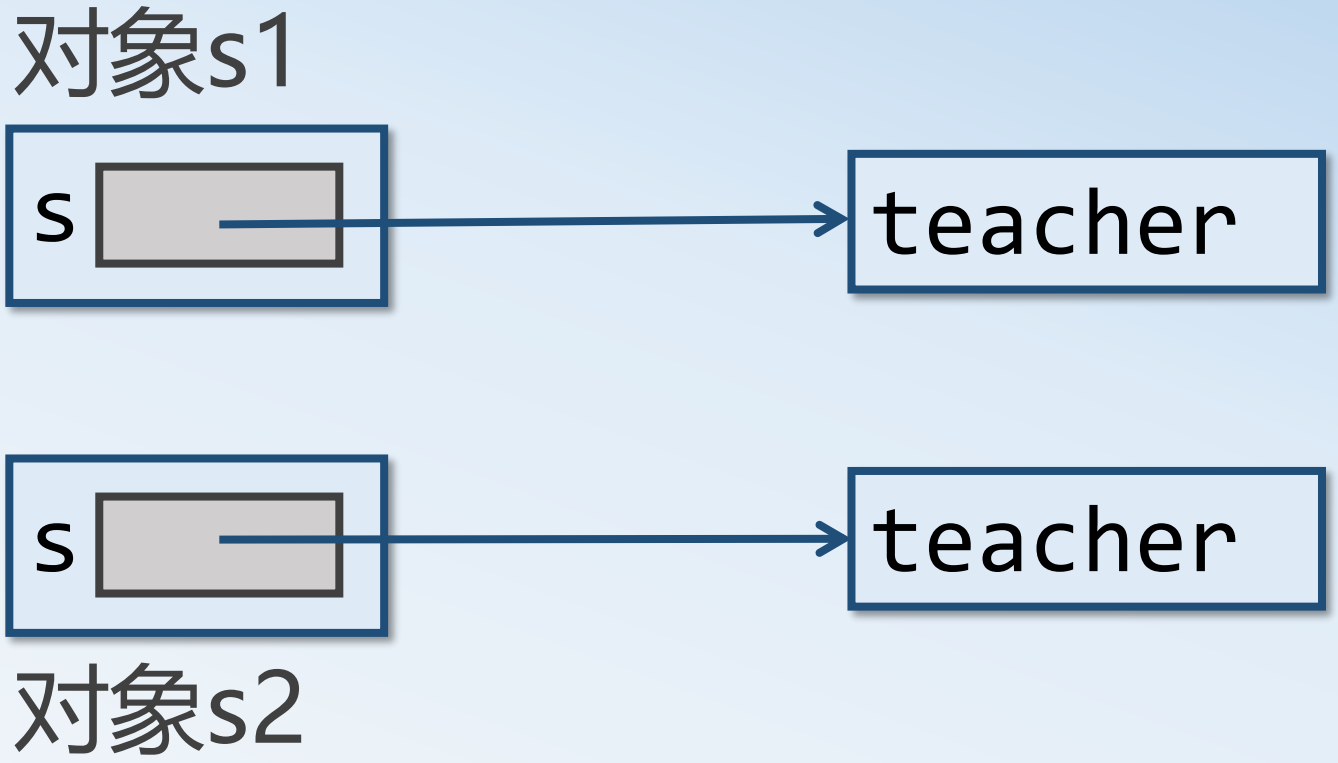
拷贝构造函数

- **浅拷贝**



拷贝构造函数

- **深拷贝**



拷贝构造函数

```
class MyString
{
public:
    MyString (char *ap = "china");
    MyString (const MyString &temp);
    ~MyString ();
    void display()const;
private:
    char *m_pstr;
};

void MyString::display()const
{
    cout << m_pstr << endl;
}
MyString::MyString(char *ap)
{
    m_pstr = new char[strlen(ap)+1];
    strcpy(m_pstr, ap);
}
```

```
MyString::MyString(const MyString &temp)
{
  if(temp.m_pstr)
  {
      m_pstr =
      new char[strlen(temp.m_pstr)+1];
      strcpy(m_pstr, temp.m_pstr);
   }
   else m_pstr = 0;
}
MyString::~MyString()
{    delete []m_pstr; }

int main(void)
{
    MyString str1("japan");
    MyString str2(str1);
    str2.display();
    return 0;
}
```

拷贝构造函数

❖**思考**：何时定义拷贝构造函数？

通常情况下，若一个类包含指针成员，并且通过该指针在构造函数中动态申请了空间，则必须为该类定义一个拷贝构造函数。

拷贝构造函数

- **作用**：用一个已经存在的对象初始化同类的另一个新对象，拷贝构造函数是一种特殊的构造函数，具有构造函数的所有特征。
- 对每个类，编译系统会自动生成一个拷贝构造函数，作为该类的公有成员。
- 拷贝构造函数的参数为const &
- 如果在构造函数中分配了堆区内存：
  - 需要定义析构函数
  - 需要定义拷贝构造函数
  - 需要定义赋值运算符的重载函数

本讲教学目标

- 掌握C++中类的构造函数
- 掌握C++中类的析构函数
- 掌握C++中类的拷贝构造函数


THANKS