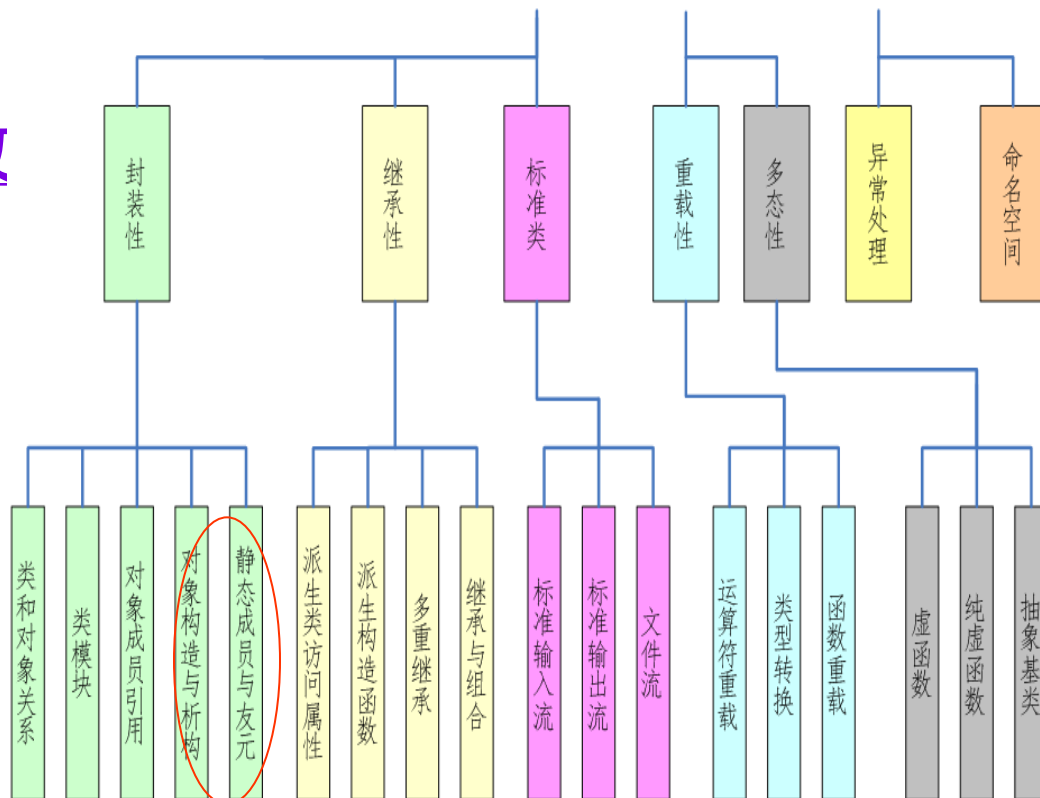


第3讲 类和对象（下）

参考教材第14章的内容

1. 对象浅拷贝问题
2. 对象动态建立和释放
3. 对象数组
4. 对象指针
5. 共用数据的保护
6. 静态成员

C++的OOP程序 = 对象 + 消息 + 工具

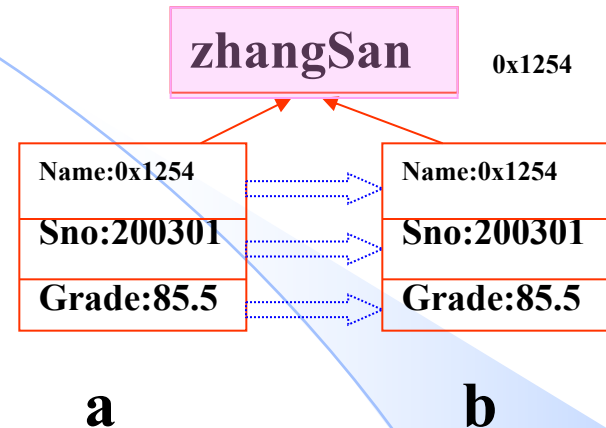


3.1 对象的浅拷贝问题

□ 何谓浅拷贝

● struct变量的复制

```
struct student {
    char *p;
    int sno;
    float grade;
} a={ "ZhangSan" , 200301, 85.5}, b;
b = a;
```



□ 问题引入：将一个结构体变量的各成员值赋给另一个结构变量的各成员时。指针类成员拷贝的只是地址，使得不同结构变量中指针成员指向同一块内存，造成了隐含的空间共享。

```

1 #include <iostream>
2 #include <string.h>
3 using namespace std;
4 class Strings {
5 public:
6     Strings(char *s);
7     ~Strings();
8     void Print();
9     void Set(char *s);
10 private:
11     int length;
12     char *str;};
13 Strings::Strings(char *s)
14 {length = strlen(s);
15  str=new char[length+1];
16  strcpy(str, s);
17  cout << "构造函数被调用！" << endl;}
18 Strings::~Strings()
19 {delete []str;
20  cout << "析构函数被调用！" << endl;}
21 void Strings::Set(char *s)
22 {length=strlen(s);
23  delete []str;
24  str=new char[length+1];
25  strcpy(str, s);}
26 void Strings::Print()
27 {cout << str << endl; cout << "length=" << length << endl;}
28 void main()
29 {Strings S1("Hello!");
30  Strings S2(S1);
31  S1.Print();
32  S2.Print();
33  S1.Set("New String!");
34  S1.Print();
35  S2.Print();}

```

自动窗口		
名称	值	类型
S1	{ length=6 str=0x00427A30 }	Strings
length	6	int
str	0x00427A30 "Hello!"	char*
*S1.str	72 'H'	char
S2	{ length=6 str=0x00427A30 }	Strings
length	6	int
str	0x00427A30 "Hello!"	char*
*S2.str	72 'H'	char

自动窗口 局部变量 线程 模块 监视 1

```

3 using namespace std;
4 class Strings {
5 public:
6     Strings(char *s);
7     ~Strings();
8     void Print();
9     void Set(char *s);
10 private:
11     int length;
12     char *str;};
13 Strings::Strings(char *s)
14 {length = strlen(s);
15  str=new char[length+1];
16  strcpy(str, s);
17  cout << "构造函数被调用！" << endl;}
18 Strings::~~Strings()
19 {delete []str;
20  cout << "析构函数被调用！" << endl;}
21 void Strings::Set(char *s)
22 {length=strlen(s);
23  delete []str;
24  str=new char[length+1];
25  strcpy(str, s);}
26 void Strings::Print()
27 {cout << str << endl; cout << "length=" << length << endl;}
28 void main()
29 {Strings S1("Hello!");
30  Strings S2(S1);
31  S1.Print();
32  S2.Print();
33  S1.Set("New String!");
34  S1.Print();
35  S2.Print();}

```

自动窗口		
名称	值	类型
S1	{ length=11 str=0x00427A30 }	Strings
length	11	int
str	0x00427A30 "New String!"	char*
*S1.str	78 'N'	char
S2	{ length=6 str=0x00427A30 }	Strings
length	6	int
str	0x00427A30 "New String!"	char*
*S2.str	78 'N'	char

构造函数被调用！

Hello!

length=6

Hello!

length=6

New String!

length=11

New String!

length=6

Microsoft Visual C++ Debug Library



Debug Error!

Program: e:\vc-exercise\test2\debug\test2.exe

HEAP CORRUPTION DETECTED: after Normal block (#186) CRT detected that the application wrote to memory af

(Press Retry to debug the application)

终止(A)

重试(R)

忽略(I)

这都是“浅拷贝”闯的祸

```

6   Strings(char *s);
7   Strings(Strings &p);
8   ~Strings();
9   void Print();
10  void Set(char *s);
11  private:
12      int length;
13      char *str;};
14 Strings::Strings(char *s)
15 {length = strlen(s);
16  str=new char[length+1];
17  strcpy(str, s);
18  cout << "构造函数被调用！" << endl;}
19 Strings::Strings(Strings &p)
20 { length = strlen(p.str);
21  str = new char[length+1];
22  strcpy(str, p.str);
23  cout << "拷贝构造函数被调用！" << endl;}
24 Strings::~~Strings()
25 {delete []str;
26  cout << "析构函数被调用！" << endl;}
27 void Strings::Set(char *s)
28 {length=strlen(s);
29  delete []str;
30  str=new char[length+1];
31  strcpy(str, s);}
32 void Strings::Print()
33 {cout << str << endl; cout << "length=" << length << endl;}
34 void main()
35 {Strings S1("Hello!");
36  Strings S2(S1);
37  S1.Print();
38  S2.Print();
39  S1.Set("New String!");
40  S1.Print();

```

名称	值	类型
S1	{ length=6 str=0x00427A30 }	Strings
length	6	int
str	0x00427A30 "Hello!"	char*
S2	{ length=6 str=0x00427A78 }	Strings
length	6	int
str	0x00427A78 "Hello!"	char*

自动窗口 局部变量 线程 模块 监视 1

C:\WINDOWS\system32\cmd.exe

```

构造函数被调用！
拷贝构造函数被调用！
Hello!
length=6
Hello!
length=6
New String!
length=11
Hello!
length=6
析构函数被调用！
析构函数被调用！
请按任意键继续. . .

```

提问：请写出系统缺省拷贝构造函数代码

构建深拷贝构造函数

3.2 对象的动态构建与释放

- 用new和delete运算符动态申请和释放内存;
- 如果定义了Box类, 可动态地建立一个对象:
Box *pt=new Box(12,15,18);
- 系统会开辟一段内存空间来存放一个Box类无名对象; 同时调用该类的构造函数, 以使该对象初始化. 如果内存量不足而无法构建, 则返回一个0指针值;
- 不再使用由new建立的对象时, 可以用delete运算符予以释放。如: delete pt;
- 执行delete时, 在释放内存空间前, 自动调用析构函数;
- 提问: New 和malloc()区别? Delete 和free()区别? 无名对象如何引用?

3.3 对象数组

静态分配数组：

类名 数组名[常量表达式] = {初始参数};

动态分配数组：

类名 * 指针名 = new 类名[表达式];

释放动态数组：

delete [] 指针名;

定义对象数组时，编译器会调用每一个数组分量的构造函数，若数组定义没有提供初始参数，则调用的是无参构造函数。

```
Date arrDays[30];
```

```
Date * parrDays = new Date[10];
```

```
...
```

```
delete [] parrDays;
```

delete parrDays的区别?

□ 数组初始化格式：类名 对象名[长度]={构造函数（实参列表1），构造函数（实参列表2），...}

□ 例: Student Stud[2]={Student(1001,18,87), Student(1002,19,76)}

对象数组的使用方法

```

1  #include <iostream>
2  using namespace std;
3  class Box
4  {public:
5      Box(int h=10,int w=12,int len=15): height(h),width(w),length(len){ }
6      //声明有默认参数的构造函数，用参数初始化表对数据成员初始化
7      int volume();
8  private:
9      int height;
10     int width;
11     int length;};
12
13 int Box::volume()
14 {return(height*width*length);}
15
16 int main()
17 { Box a[3]={
18     Box(10,12,15),
19     Box(15,18,20),
20     Box(16,20,26)
21 };
22 cout<<"volume of a[0] is"<<a[0].volume()<<endl; //调用a[0]的volume函数
23 cout<<"volume of a[1] is"<<a[1].volume()<<endl; //调用a[1]的volume函数
24 cout<<"volume of a[2] is "<<a[2].volume()<<endl; //调用a[2]的volume函数
25 }

```

监视 1

名称	值
a	{Length=3}
[0]	{ height=10 width=12 length=15 }
height	10
length	15
width	12
[1]	{ height=15 width=18 length=20 }
height	15
length	20
width	18
[2]	{ height=16 width=20 length=26 }
height	16
length	26

自动窗口 局部变量 线程 模块 监视 1

//定义对象数组

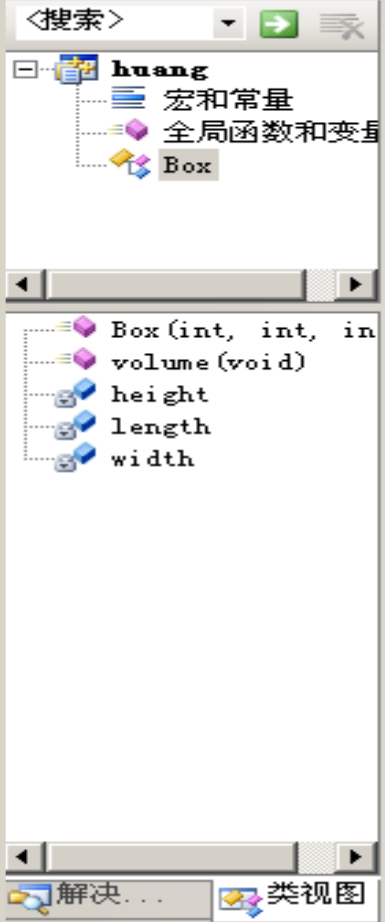
//调用构造函数Box，提供第1个元素的实参

//调用构造函数Box，提供第2个元素的实参

//调用构造函数Box，提供第3个元素的实参

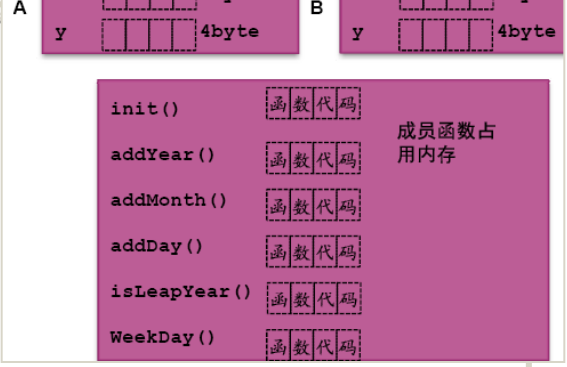
3.4 对象指针

对象的相关指针类型



```

1  #include <iostream>
2  using namespace std;
3  class Box
4  {public:
5      Box(int,int,int);
6      int volume();
7  private:
8      int height;
9      int width;
10     int length; };
11  Box::Box(int h,int w,int len)
12  {height=h;
13   width=w;
14   length=len;}
15  int Box::volume()
16  {return(height*
17
18  int main()
19  { Box box1(12,25,30);
20   cout<<"The volume of box1 is "<<box1.volume()<<endl;
21   Box box2(15,30,21);
22   cout<<"The volume of box2 is "<<box2.volume()<<endl;
23   return 0;}
        
```



对象函数指针

对象指针

对象数据成员指针

名称	值
&box1	0x0012ff58 {height=12 width=25 length=30 }
&(box1.height)	0x0012ff58
&(box1.width)	0x0012ff5c
&(box1.length)	0x0012ff60

指向对象的指针变量

➤ 对象空间的起始地址即：对象的指针；可定义同类型的指针变量,用来存放对象的指针.如：`Date A(1,1,1900);`
`Date * p = &A;`

全局范围)

```

1 #include <iostream>
2 using namespace std;
3 class Time
4 {public:
5   int hour;
6   int minute;
7   int sec;
8   void get_time();
9 };
10 void Time::get_time()
11 {cout<<hour<<":"<<minute<<":"<<sec<<endl;}
12
13 int main()
14 {Time *pt; //定义pt为指向Time类对象的指针变量
15   Time t1; //定义t1为Time类对象
16   pt=&t1; //将t1的起始地址赋给pt
17   cout<<(*pt).hour<<endl; //pt所指向的对象中的hour成员，即t1.hour
18   cout<<pt->hour<<endl; //pt所指向的对象中的hour成员，即t1.hour
19   (*pt).get_time(); //调用pt所指向的对象中的get_time函数，即t1.get_time
20   pt->get_time(); // 调用pt所指向的对象中的get_time函数，即t1.get_time
21 }
```

自动窗口

名称	值	类型
8&t1	0x0012F0E0 { hour=1241372 minute=1241344	Time*
hour	1241372	int
minute	1241344	int
sec	2045189024	int
*pt	{ hour=1241372 minute=1241344 sec=204518	Time
hour	1241372	int
minute	1241344	int
sec	2045189024	int
pt	0x0012F0E0 { hour=1241372 minute=1241344	Time*
hour	1241372	int
minute	1241344	int
sec	2045189024	int

自动窗口 局部变量 线程 模块 监视 1

```

C:\WINDOWS\system32\cmd.exe
1458808
1458808
1458808:1242096:2045189024
1458808:1242096:2045189024
请按任意键继续...
```

对象成员的指针变量

- 指向对象数据成员的指针变量。指向对象中某个数据成员的地址。定义方法和普通变量的指针变量方法相同。例：int *p1;
- 如果类的数据成员为公用，则可在类外通过对象成员的指针变量访问。例：p1=&t1.hour; cout<<*p1<<endl.

- 指向对象成员函数的指针。指向对象中某个成员函数的入口地址；和普通函数的指针变量定义方法不同
- 定义形式：数据类型 (类名::*指针变量名)(参数表列);
- 例如：void (Time::*p2)();
- 要求3方面匹配：①函数参数类型和个数；②函数返回值类型；③所属类
- 赋值形式：指针变量名=&类名::成员函数名;
- 引用形式：(对象.*指针变量) (参数) ;

```

1  #include <iostream>
2  using namespace std;
3  class Time
4  {public:
5      Time(int,int,int);
6      int hour;
7      int minute;
8      int sec;
9      void get_time();
10 };
11 Time::Time(int h,int m,int s)
12 {hour=h;
13  minute=m;
14  sec=s;
15 }
16 void Time::get_time()
17 {cout<<hour<<":"<<minute<<":"<<sec<<endl;}
18
19 int main()
20 {Time t1(10,13,56);
21  int *p1=&t1.hour;
22  cout<<*p1<<endl;
23  t1.get_time();
24  Time *p2=&t1;
25  p2->get_time();
26  void (Time::*p3)();
27  p3=&Time::get_time;
28  (t1.*p3)();
29 }

```

//声明公有成员函数

//定义公有成员函数

//定义Time类对象t1

//定义指向整型数据的指针变量p1，并使p1指向t1.hour

//输出p1所指的数据成员t1.hour

//调用对象t1的成员函数get_time

//定义指向Time类对象的指针变量p2，并使p2指向t1

//调用p2所指向对象(即t1)的get_time函数

//定义指向Time类公用成员函数的指针变量p3

//使p3指向Time类公用成员函数get_time

//调用对象t1中p3所指的成员函数(即t1.get_time())

监视 1		
名称	值	类型
p1	0x0012F0D0	int*
*p1	10	int
p2	0x0012F0D0 { hour=10 minute=13 sec=56 }	Time*
hour	10	int
minute	13	int
sec	56	int
p3	error: cannot obtain value	void*

```

1 #include <iostream>
2 using namespace std;
3 class Time
4 {public:
5     Time(int,int,int);
6 private:
7     int hour;
8     int minute;
9     int sec;
10    void get_time(); };
11 Time::Time(int h,int m,int s)
12 {hour=h;
13  minute=m;
14  sec=s; }
15 void Time::get_time()
16 {cout<<hour<<":"<<minute<<":"<<sec<<endl;}
17
18 int main()
19 {Time t1(10,13,56);
20  int *p1=&t1.hour;
21  cout<<*p1<<endl;
22  t1.get_time();
23  Time *p2=&t1;
24  p2->get_time();
25  void (Time::*p3)();
26  p3=&Time::get_time;
27  (t1.*p3)();
28  return 0;}

```

输出

显示以下输出(S): 生成

```

1>----- 已启动全部重新生成: 项目: test2, 配置: Debug Win32 -----
1>正在删除项目“test2”(配置“Debug|Win32”)的中间文件和输出文件
1>正在编译...
1>1. cpp
1>\1. cpp (20) : error C2248: 'Time::hour' : cannot access private member declared in class 'Time'
1>      \1. cpp (7) : see declaration of 'Time::hour'
1>      \1. cpp (4) : see declaration of 'Time'
1>\1. cpp (22) : error C2248: 'Time::get_time' : cannot access private member declared in class 'Time'
1>      \1. cpp (10) : see declaration of 'Time::get_time'
1>      \1. cpp (4) : see declaration of 'Time'
1>\1. cpp (24) : error C2248: 'Time::get_time' : cannot access private member declared in class 'Time'
1>      \1. cpp (10) : see declaration of 'Time::get_time'
1>      \1. cpp (4) : see declaration of 'Time'
1>\1. cpp (26) : error C2248: 'Time::get_time' : cannot access private member declared in class 'Time'
1>      \1. cpp (10) : see declaration of 'Time::get_time'
1>      \1. cpp (4) : see declaration of 'Time'
1>生成日志保存在“file://e:\vc-exercise\test2\test2\Debug\BuildLog.htm”
1>test2 - 4 个错误, 0 个警告
===== 全部重新生成: 0 已成功, 1 已失败, 0 已跳过 =====

```

Line21, 27 没有出错, Why?

□ 从面向对象的思想来看, 直接用指针访问对象的成员变量等于绕开了对象直接访问成员变量, 破坏了对对象的封装



对象成员函数的指针绑定问题

正确的写法是：

```
class Student
{
public:
    float SomeFunc(int ID, char * nam);
    ...
};

void main()
{
    Student stu;
    float (Student::*pFunc)(int, char *); //定义指针变量
    pFunc = &(Student::SomeFunc); //赋值时仍未与对象绑定
    (stu.*pFunc)(1, "Zhang"); //直到使用时才与对象绑定
}
```

1.类型匹配

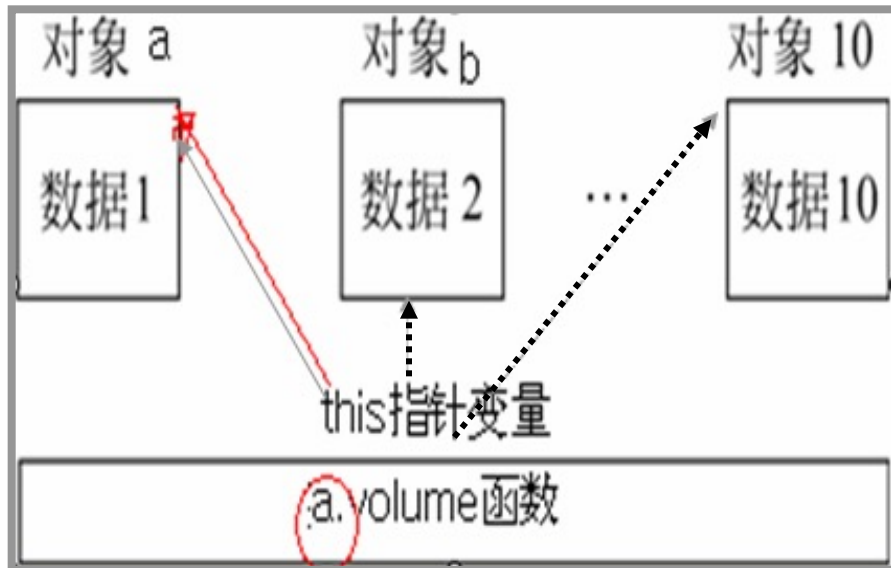
3.对象绑定

```
class Student
{
public:
    float SomeFunc(int ID, char * nam);
    ...
};

void main()
{
    Student stu;
    float (*pFunc)(int, char *);
    pFunc = stu.SomeFunc; // 错误! 不能这样写
    (*pFunc)(1, "Zhang"); // 错误! 不能这样写
}
```


对象成员函数的THIS 指针

- this 指针：每个成员函数都包含一个特殊的指针，即this。其值是当前被调用的成员函数所在对象的起始地址；
- 例:当a.volume()时，系统把对象a的起始地址赋给this指针。
height*width*length 编译时即为：(this->height)*(this->width)*(this->length)，相当于：(a.height)*(a.width)*(a.length)



```
Date & Date::add_day()
```

```
{ //实现日期加1天的代码
```

```
...
```

```
return *this;
```

```
}
```

```
void main()
```

```
{ Date today(1,1,1900);
```

```
today.add_Day().Print();
```

```
}
```

为什么这个函数可以返回对象的引用？



3.5 共用数据的保护



- 常变量:在程序运行中不能改变的变量, `const int a=3;`
- 量变量和符号常量关系: `const`比`#define`更灵活, 有类型;

`const` 数据类型名 变量名 = 初值表达式;

- 定义自身内容不可更改的普通变量

`const` 数据类型名 * 指针变量名;

- 定义指向的内存空间不可更改的指针变量 (自身指向地址可以更改)

数据类型名 * `const` 指针变量名 = 初值表达式;

- 定义自身内容 (即指向的地址) 不可更改的指针变量 (常指针)

常对象

类名 `const` 对象名(参数表列) ;

- 定义自身内容不可更改的对象（常对象必须通过构造函数进行初始化，而且不能更改；除了隐式调用的构造和析构函数以外，不能调用常对象的非`const`成员函数）

`const` 类名 * 指针变量名 ;

- 定义指向的对象内存空间不可更改的类指针变量（自身指向地址可以更改）

类名 * `const` 指针变量名 = 初值表达式 ;

- 定义指向的地址不可更改的类指针（其指针值始终保持为其初值，不可更改，即只能指向一个对象）

`const` 类名 & 对象名 = 初值表达式 ;

- 定义一个对象的常引用（代表另一个对象，但其值为只读）

```

1.  class Date
2.  {
3.      int d, m, y;
4.  public:
5.      Date(int dd=0, int mm=0, int yy=0);
6.      Date(Date & D); //也可声明为Date(const Date & D);
7.      int Day() const { return d; } // 常成员函数
8.      bool isLeapYear() const;      // 常成员函数
9.      void Print() const { cout<<d<<" "<<m<<" "<<y<<endl; }
10.     ...
11. };

12. void main()
13. {
14.     Date DDay(6,6,1944);
15.     Date const D1 = DDay; //常对象D1,也可写成const Date D1 = DDay
16.     const Date & D2 = DDay; //常引用D2,代表DDay,但不分配新对象
17.     DDay.Print();
18. }

```

类的常成员

```
class 类名
{
    const 数据类型名 变量名;
}
```

- 在类声明中定义自身内容不可更改的成员变量
常成员变量必须通过参数初始化表赋初值

```
class Date
{
    int m, y;
    const int d;
    ...
};

Date::Date(int dd, int mm, int yy)
    :d(dd)
{
    m = mm; y = yy; // d不允许通过赋值语句赋初值
}
```

普通const变量可以在定义时初始化常量值，const成员变量如何初始化常量值呢？

数据类型名 成员函数名(参数表列) const;

- 声明常成员函数，函数体不能更改类成员变量

```
数据类型名 类名::成员函数名(参数表列) const
{ ... }
```

- 定义常成员函数，函数体不能更改类成员变量

数据成员	非 const 成员函数	const 成员函数
非 const 的数据成员	可以引用,也可以改变值	可以引用,但不可以改变值
const 数据成员	可以引用,但不可以改变值	可以引用,但不可以改变值
const 对象的数据成员	不允许引用和改变值	可以引用,但不可以改变值

```

1.  class Date
2.  {
3.      int d, m, y;
4.  public:
5.      Date(int dd=0, int mm=0, int yy=0);
6.      int Day() const { return d; } // 常成员函数
7.      bool isLeapYear() const;      // 常成员函数
8.      ...
9.      const int a; // 常成员变量
10.     const int & d_ref; // 一个常引用作为成员
11. };

12. Date::Date(int dd, int mm, int yy)
13.     :d(dd), m(mm), y(yy), a(mm), d_ref(d) //初始化常成员变量
14. {}

15. bool Date::isLeapYear() const
16. { return (y%100==0) ? (y%400==0) : (y%4==0);
17. }

```

注意：每个构造函数都要加上对常成员变量的初始化表！

常函数参数和返回值



```

1.  class Date
2.  {
3.      int d, m, y;
4.  public:
5.      Date(int dd=0, int mm=0, int yy=0);
6.      int Day()    const { return d; } // 常成员函数
7.      int Month()  const { return m; } // 常成员函数
8.      int Year()   const { return y; } // 常成员函数
9.      ...
10. };
11. int Greater(const Date & D1, const Date & D2)
12. {
13.     if ( D1.Year() != D2.Year() ) return D1.Year()-D2.Year();
14.     else if ( D1.Month() != D2.Month() ) return D1.Month()-D2.Month();
15.     return D1.Day()-D2.Day();
16. }
17. const Date & MaxDay(const Date & D1, const Date & D2)
18. { return Greater(D1,D2)>0 ? D1 : D2; }
19. void main()
20. {   Date day1(1,1,1900), day2(2,3,2000);
21.     const Date & max_day = MaxDay(day1, day2);
22. }
  
```

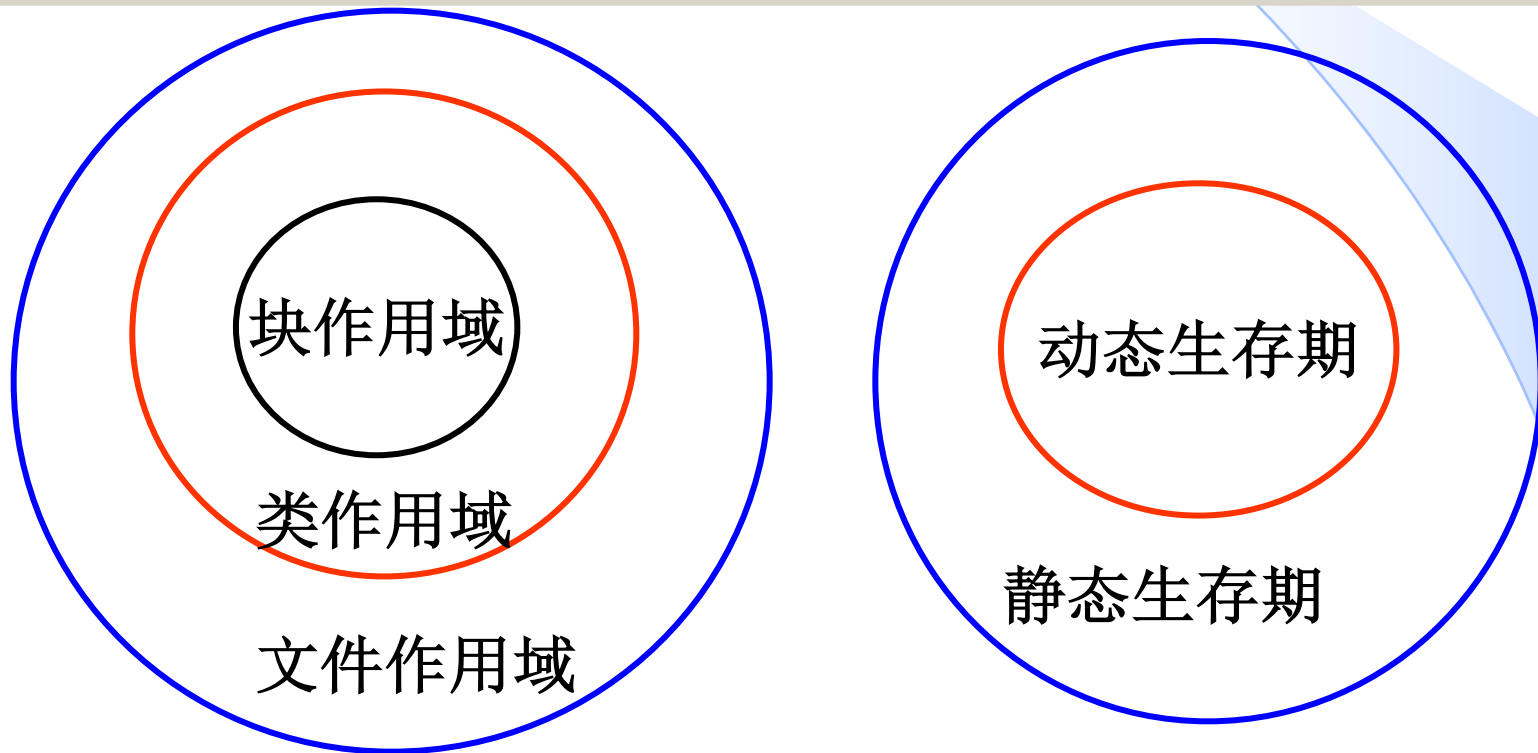


CONST共用数据保护机制的小结

- 为提高效率，C++函数常通过引用来传递类对象，若要确保传入的对象内容不被改变，可以使用**常引用**；
- 如果返回值是一个对象，也可以通过返回引用来提高效率，若要确保返回的对象内容不被改变，可以返回**常引用**；
- 如果要确保对象内容不被改变，比常对象更好的方法是对对象的**常引用**；
- 比常对象和常成员变量更好的方法是定义**常成员函数**，用常成员函数作为对象属性的只读访问接口；
- 对于**常对象**，除了隐式调用的构造和析构函数以外，不能调用常对象的非const成员函数)

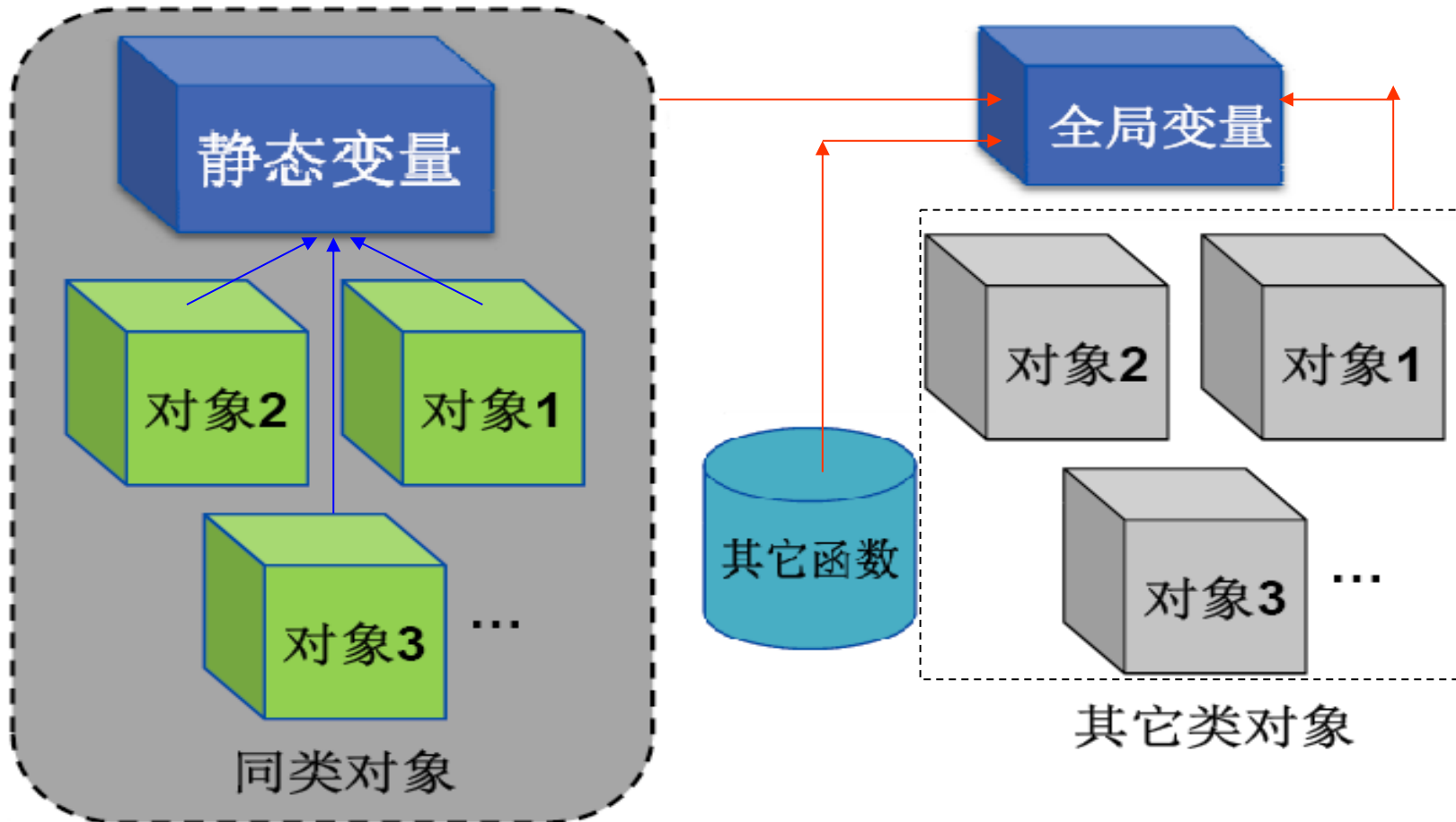
3.6 静态成员

- 问题引入：假设有一个学籍管理软件，将一个学生作为一个对象，对象中“学号”数据成员，如何实现每个对象中“学号”自动编号？该问题涉及到同类对象之间通信问题
- 变量类型：全局变量，局部静态变量，局部动态变量



变量的作用域和生存期

静态成员和全局变量



类的静态数据成员

- ❑ 对象属性：如果有n个同类的对象，那么每一个对象都有相同的数据成员，不同对象数据成员各不相同；
- ❑ 类属性：描述类的所有对象共同特征的数据项，对任何同类对象，其属性值是相同的，即类的静态数据成员；
- ❑ 静态数据成员的用途是实现同类对象之间数据共享。

❑例：

```
class student
{public:
    int set-info( );
    private:
        static int num;
        int number;
        char name[10]
        .....};
```

```
student::student(char * c)
{ strcpy(name,c);
    num++;
    number=num;}
int student::num=0;
int main()
{ student, stu1,stu2,stu3,stu4;
    .....}
```

```

1  #include <iostream>
2  using namespace std;
3  class Box
4  {public:
5      Box(int=10,int=10,int=10);
6      int volume();
7  private:
8      static int number;
9      int height;
10     int width;
11     int length; };
12
13 Box::Box(int h,int w,int len)
14 {height=h;
15  width=w;
16  length=len; }
17 int Box::number=0;
18 int Box::volume()
19 {return(height*width*length); }
20
21 int main()
22 {Box box1(10,20,20);
23  Box box2(15,30,25);
24  Box box3=box1;
25  cout<<"The volume of box1 is "<<box1.volume()<<endl;
26  cout<<"The volume of box2 is "<<box2.volume()<<endl;
27  return 0; }

```

监视 1

名称	值
box1	{number=0 height=10 width=20 ...}
number	0
height	10
width	20
length	20
box2	{number=0 height=15 width=30 ...}
number	0
height	15
width	30
length	25
box3	{number=0 height=10 width=20 ...}
number	0
height	10
width	20
length	20
&box1.number	0x00419148 int Box::number
&box2.number	0x00419148 int Box::number
&box3.number	0x00419148 int Box::number

3个对象的number地址为啥相同?

自动窗口

局部变量

线程

模块

监视 1

调用堆栈

断点

类的静态数据成员初始化

- 记住：类的静态成员属于整个类，不属于任何一个具体对象，所以它们和全局变量一样，在所有本类对象产生之前就被分配，它们的初始化也应独立于所有成员函数以外。

```
class Date {
    int d, m, y;
    static int def_d, def_m, def_y;
public:
    ...
};

int Date::def_d = 1;
int Date::def_m = 1;
int Date::def_y = 1900;
```

静态成员为普通变量时的初始化写法

```
class Date {
    int d, m, y;
    static Date default_day;
public:
    ...
};

Date Date::default_day(1,1,1900);
```

静态成员为对象时的初始化写法

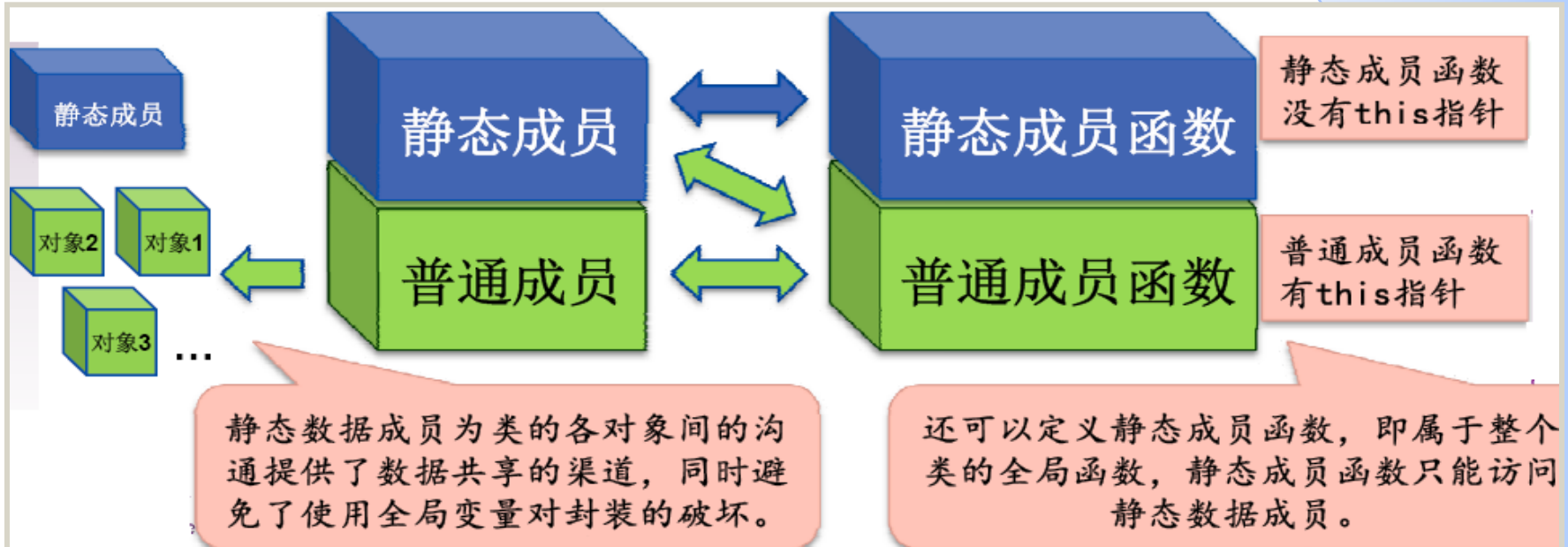
□ Date::date(int dd,int mm,int yy):default_day(1,1,1900)对吗?

类的静态成员函数

```
class Date {
    int d, m, y;
    static Date default_day;
public:
    static void set_default(int d, int m, int y);
};

void Date::set_default(int d, int m, int y)
{ default_day.d = d; default_day.m = m;
  default_day.y = y; }
```

或者，还可以干脆声明一个不属于任何对象的**静态成员函数**来专门负责静态成员的初始化



3.6 类的静态成员一小结

1. 由于静态成员实质上是全局变量，所以不随对象产生而分配，也不随对象销毁而释放，在类声明时已经分配内存，在`main()`函数结束后才释放
2. 静态成员为所有类对象所共有，每一个类对象的成员函数都可以访问它们
3. 静态数据成员的初始化不能在构造函数、初始化参数表中进行，只能在类外进行：

类型 类名::静态成员变量名 = 初始值;

4. 在类外访问静态成员变量时（`public`属性），一般使用“类名::静态成员名”方式，因为静态成员属于整个类，而不是某个具体对象
5. 静态成员函数没有`this`指针，不能访问非静态数据成员

3.7 类的静态成员一问答

(1) 如果静态数据成员被定义为私有的，是否可以在类外直接引用？如何引用？

答：不能，通过公共成员函数来引用。

(2) 类的静态数据成员的主要作用是什么？

答：实现同类对象之间的数据共享。

(3) 公用静态数据成员与全局变量的不同？

答：作用域不同，局限于类作用域。而全局变量在整个程序中；但两者的生命期相同。

(4) 静态成员函数是否可以访问类中非静态数据成员？

答：调用对象的成员函数时，系统会把对象地址赋给成员函数的this指针。而静态成员函数并不属于某一对象，没有this指针。由此决定了静态成员函数不能直接访问类中非静态成员。

```

1 #include <iostream>
2 using namespace std;
3 class Box
4 {public:
5     Box(int,int);
6     int volume();
7     static int height;
8     int width;
9     int length;};

```

C:\WINDOWS\system32\cmd.exe

```

10
10
10
3000
请按任意键继续. . .

```

//把height定义为公用的静态的数据成员

```

10
11 Box::Box(int w,int len) //通过构造函数对width和length赋初值
12 {width=w;
13     length=len;}

```

```

14
15 int Box::volume()
16 {return(height*width*length);}

```

```

17
18 int Box::height=10; //对静态数据成员height初始化

```

```

19
20 int main()
21 {Box a(15,20),b(20,30);
22     cout<<a.height<<endl; //通过对象名a引用静态数据成员
23     cout<<b.height<<endl; //通过对象名b引用静态数据成员
24     cout<<Box::height<<endl; //通过类名引用静态数据成员
25     cout<<a.volume()<<endl;} //调用volume函数，计算体积，输出结果

```



```

1 #include <iostream>
2 using namespace std;
3 class Student //定义Student类
4 {public:
5     Student(int n,int a,float s):num(n),age(a),score(s){ } //定义构造函数
6     void total();
7     static float average(); //声明静态成员函数
8 private:
9     int num;
10    int age;
11    float score;
12    static float sum; //静态数据成员
13    static int count; //静态数据成员
14    void Student::total() //定义非静态成员函数
15    {sum+=score; //累加总分
16     count++;} //累计已统计的人数
17    float Student::average() //定义静态成员函数
18    {return(sum/count);}
19    float Student::sum=0; //对静态数据成员初始化
20    int Student::count=0; //对静态数据成员初始化
21    int main()
22    {Student stud[3]={ //定义对象数组并初始化
23        Student(1001,18,70),
24        Student(1002,19,78),
25        Student(1005,20,98)};
26        int n;
27        cout<<"please input the number of students:";
28        cin>>n; //输入需要求前面多少名学生的平均成绩
29        for(int i=0;i<n;i++) //调用3次total函数
30            stud[i].total();
31        cout<<"the average score of "<<n<<" students is"<<Student::average()<<endl;
32        //调用静态成员函数
33        return 0;}

```

```

C:\WINDOWS\system32\cmd.exe
please input the number of students:3
the average score of 3 students is82
请按任意键继续. . .

```


第3次实验作业

本次练习必做 2 道题，提交方式同前面，第5周末之前交。

1. “五子棋”程序在第2次作业的基础上，增加新需求：(1)记录对弈双方整个过程的下棋“点数”（即总共下了多少步才结束）（提醒：使用类静态成员）。(2)按照五子棋的一般规则，结合前2次作业要求，完成整个程序编写、调试和测试。需要有程序的文件、类、函数和重要语句的注释。提交代码和运行测试结果。

■ 在3次作业基本需求基础上，自由发挥，增加功能，鼓励创新。

2. 建立一个对象数组，内放 10 个学生的数据（姓名、学号、成绩），建立一个函数max，用指向对象的指针做函数参数，在max函数中找出 10 个学生中成绩最高者，并输出其学号。根据学号，将对象的姓名、成绩等等信息使用拷贝构造函数，拷贝到一个新对象中。对对象进行打印输出。

第3次实验作业选择题3道

(1) 商店销售某一商品，每天公布统一折扣 (discount)。同时允许销售人员销售时灵活掌握价格(price)。在此基础上，对每一次购10件以上者，可享受9.8折扣优化。现已知当天的3名售货员的销售情况如下：

销售员号	销售件数	销货价格
101	5	23.5
102	12	24.5
103	100	21.5

编写程序，计算当日此商品总销售款 (sum)，及每件商品的平均售价。并打印计算结果。另外要求：要求使用静态数据成员和静态成员函数

(2) “XX公司人事管理系统”在第2次作业基础上，增加新需求：(1) 需要将“职工编号”由程序自动产生。每增一个职工，构造一个employee对象，将对象的“individualEmpNo”自动加1。(2) 在现有4个职工基础上，需新增20个职工。修改程序满足上述需求。

■ 阅读和参考教材代码，自主扩展功能和编写代码，注释要清晰。

(3) 将下述程序中的func()函数分别改为下列3种情况，上机运行结果

```
Date func(Date A)
{
    Date B(A);
    return B;
}
void main()
{
    Date today;
    today = func(today);
}
```

```
Date & func(Date A)
{
    Date B(A);
    return B;
}
void main()
{
    Date today;
    today = func(today);
}
```

```
Date func(Date A)
{
    return A;
}
void main()
{
    Date today;
    today = func(today);
}
```

```
1. class Date
2. {
3.     int d, m, y;
4. public:
5.     Date(int dd=0, int mm=0, int yy=0);
6.     Date(Date &D); //拷贝构造函数
7.     ~Date();
8.     ...
9. };
10. Date::Date(int dd, int mm, int yy)
11. : d(dd), m(mm), y(yy)
12. { cout << "Constructor called! Address=0x" <<
13.     hex << setw(8) << setfill('0') << this << endl;
14. }
15. Date::Date(Date &D)
16. { d = D.d; m = D.m; y = D.y;
17.     cout << "Copy constructor called! Address=0x" <<
18.         hex << setw(8) << setfill('0') << this << endl;
19. }
20. Date::~~Date()
21. { cout << "Destructor called! Address=0x" <<
22.     hex << setw(8) << setfill('0') << this << endl;
23. }
```

```
Date func(Date A)
{
    return Date(A);
}
void main()
{
    Date today;
    today = func(today);
}
```

Date(A)
临时对象
被保留到
这句话之
后再释放