程序设计基础及语言 II
--C++大学教程

东南大学

计算机科学与工程学院

李慧颖

huiyingli@seu.edu.cn

课前简介

- 程序设计基础与语言 II
- 本学期学时：
  - 1~8周
  - 32+24+32
- 平时成绩：作业+上机实验+到课率
- 期末考试：拟第10周考试
- 总成绩：30%*平时成绩+70%*期末考试成绩

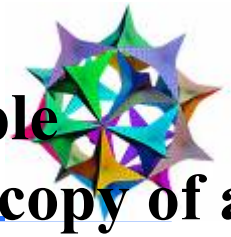
课前简介

授课内容

- Ch09: Classes重点
- Ch10: Classes重点, 10.9-10不作要求
- Ch11: 掌握运算符重载原理及11.10、13
- Ch12: Inheritance继承, 重点
- Ch13: Polymorphism多态, 重点, 13.7-8不作要求
- Ch14: Template模板
- Ch15: Stream Input/Output按需自学
- Ch16: Exception异常处理
- Ch17: File文件, 重点, 要求掌握17.2-5

Chapter 9

Classes: A Deeper Look, Part 1



- How to use a preprocessor wrapper to prevent multiple definition errors caused by including more than one copy of a header file in a source-code file.
- To understand class scope and accessing class members via the name of an object, a reference to an object or a pointer to an object.
- To define constructors with default arguments.
- How destructors are used to perform "termination housekeeping" on an object before it is destroyed.
- When constructors and destructors are called and the order in which they are called.
- The logic errors that may occur when a public member function of a class returns a reference to private data.
- To assign the data members of one object to those of another object by default memberwise assignment.

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.1 Introduction

- (1) 如何避免头文件被重复include至同一源文件?
  preprocessor wrapper(预处理器封装), 条件编译
- (2) 如何访问类的公有成员函数和数据成员?
  class scope(类作用域)
- 三种方式

① name of an object (对象名)

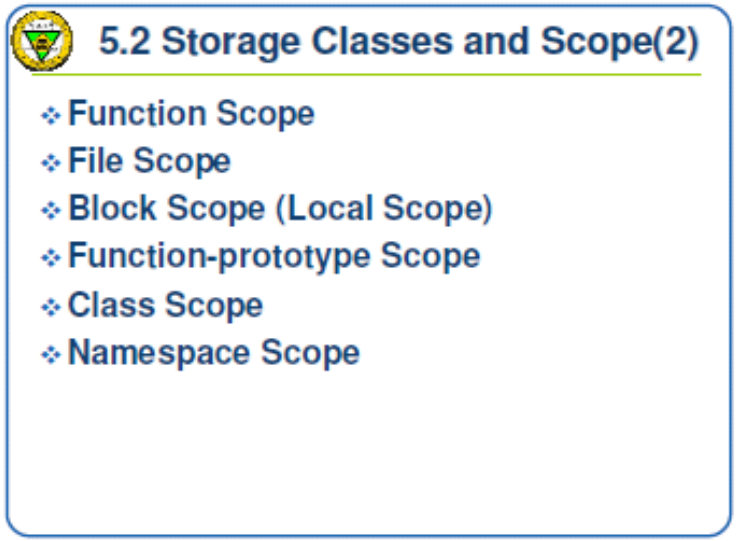
② reference to an object (对象引

③ pointer to an object (对象指针

- 两种操作符

① dot (.) member selection operator

② arrow (->) member selection operator



9.1 Introduction

- Access functions(访问函数) & Utility function (工具函数, also called a helper function)
- Arguments & default arguments for constructors
- Destructor(析构函数)
- Reference to private data
- 如何从类现有的对象生成新的对象?

  default memberwise assignment

  default copy constructor

Topics

- 9.1 Introduction
- **9.2 Time Class Case Study**
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.2 Time Class Case Study --预处理

- *1. Edit*
- *2. Preprocess* 宏、文件包含、条件编译
- *3. Compile* 编译错误(语法等)
- *4. Link*
- *5. Load*
- *6. Execute*

Runtime Error(运行时错误)

Fatal Error(致命错误)

Logic Error(逻辑错误)

9.2 Time Class Case Study
--预处理器指令

- 预处理器定义
  - #define指令
  - #define PI 3.1415
  - #define FLAG
- 条件编译
  - #ifdef / #ifndef
  - #else
  - #endif
  - #undef

9.2 Time Class Case Study --条件编译

- 一般情况下，源程序中所有的行都参加编译;但是有时希望其中一部分内容只在满足一定条件才进行编译，也就是对一部分内容指定编译的条件，这就是“条件编译”.

```
#ifdef FLAG
cout << “代码1” << endl;
#endif

#ifndef FLAG
cout << “代码2” << endl;
#endif
```

```
#ifdef FLAG
cout << “代码1” << endl;
#else
cout << “代码2” << endl;
#endif
```

9.2 Time Class Case Study

- 如何避免头文件被重复引用?

```
// a.h
class A{
   int a;
};
```

```
// b.h
#include "a.h"
class B{
   double  b;
};
```

```
// test.cpp
#include "a.h"
#include "b.h"

int main()
{
   A a;
   B b;
   return 0;
}
```

Test.cpp预处理include结果

```
class A{ int a;}

class A{ int a;}
class B{ double b;}

int main()
{
   A a;
   B b;
   return 0;
}
```

error C2011: 'A' : 'class' type redefinition

9.2 Time Class Case Study

```
// test.cpp
#include <string>
#include <iostream>

int main()
{
   return 0;
}
```

```
#include <istream>
#include <istream>
```

9.2 Time Class Case Study

```
// a.h
#ifndef A_H
#define A_H
class A{
    int a;
};
#endif
```

```
// b.h
#include "a.h"
class B{
    double b;
};
```

```
// test.cpp
#include "a.h"
#include "b.h"

int main()
{
    A a;
    B b;
    return 0;
}
```

Test.cpp预处理include结果

```
#ifndef A_H
#define A_H
class A{ int a;}
#endif

#ifndef A_H
#define A_H
class A{ int a;}
#endif

class B{ double b;}
```

9.2 Time Class Case Study

Test.cpp预处理include结果

```
#ifndef A_H
#define A_H
class A{ int a;}
#endif

#ifndef A_H
#define A_H
class A{ int a;}
#endif

class B{ double b;}
```

Preprocessor Wrapper

预处理器封装

```
#ifndef A_H
#define A_H

#endif
```

- 建议所有头文件均使用预处理器封装, 以避免重复的头文件引用
- 预处理器定义:
  FILENAME_H

程序解读（P293）

9.2 Time Class Case Study

- Time sunset;
- Time arrayOfTimes[5];
- Time &dinnerTime = sunset;
- Time *timeptr2 = &sunset;

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- **9.3 Class Scope and Accessing Class Members**
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.3 Class Scope and Accessing Class Members

- **Class Scope 类作用域**

  The name of a class member (数据成员/成员函数）

- **1.类内: accessible by all of that class's member functions and can be referenced by name**
- **2.类外: public class members are referenced through one of the handles(句柄) on an object**
  - **an object name (对象名)**
  - **a reference to an object (对象引用)**
  - **a pointer to an object (对象指针)**

```
#include <iostream> // P375. fig9.4
using namespace std;
class Count { ..... };
int main() {
    Count counter;
    Count &counterRef = counter;
    Count *counterPtr = &counter;

    counter.setX( 1 );
    counter.print();

    counterRef.setX( 2 );
    counterRef.print();

    (*counterPtr).setX( 3 );
    (*counterPtr).print();

    counterPtr->setX( 4 );
    counterPtr->print();

    return 0;
}
```

How to Access

1. using the object's name
   dot member selection operator
2. Using a reference
   dot member selection operator
3. Using a pointer with * operator
   dot member selection operator
4. Using a pointer directly
   Arrow member selection operator

```
class Count
{
public:
   void setX( int value ) { x = value; }
   void print( ) { cout << x << endl; }
private:
   int x;
};
```

1 2 3 4

9.3 Class Scope and Accessing Class Members

- public class members are referenced through one of the handles(句柄) on an object
- object name (对象名): 点操作符
- reference to an object (对象引用): 点操作符
- pointer to an object (对象指针): 箭头操作符

How to Access

```
#include <iostream>
using namespace std;
int a = 0;
class Test {
public:
    Test ( int a ){
        int b; b = 99;
        this->a = a; Test::a = a; this->b = b; Test::b = b;
        ::a = 100;
    }
   void displayMessage(){ cout << a << " " << b << " " << ::a; }
private:
    int a, b;
};
int main() {
    Test test(98); test.displayMessage();
    return 0;
}
```

File Scope

Block Scope

Class Scope

98 99 100

this指针是类的一个自动生成、自动隐藏的私有成员，它存在于类的非静态成员函数中，指向被调用函数所在的对象。全局仅有一个this指针，当一个对象被创建时，this指针就存放指向对象数据的首地址。

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- **9.4 Separating Interface from Implementation**
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.4 Separating Interface from Implementation

- Class Code分解为

  Interface + Implementation

- 但是Interface依然包括inline函数和私有数据成员等信息
- Proxy Class (Ch10.10)

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- **9.5 Access Functions and Utility Functions**
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.5 Access Functions and Utility Functions

- Access function, 访问函数
- ① Read or Display data
  - 例如: GradeBook.displayMessage()
- ② Predicate functions, 判定函数
  - Test the truth or falsity of conditions
  - 例如: 模板类vector
  - – bool empty(); —— 判定函数
- 为public成员函数, 供类用户使用

9.5 Access Functions and Utility Functions

- Utility functions(工具函数), or Helper function(工具函数)
- • private成员函数
- • 用于支撑其它成员函数, 不供类用户使用

- 例子: 输入销售人员的月销售额, 统计并输出年销售额
- • SalesPerson类
- • double sales[12]

9.5 Access Functions and Utility Functions

```
1.  // Fig. 9.5: SalesPerson.h
2.  // SalesPerson class definition.
3.  // Member functions defined in SalesPerson.cpp.
4.  #ifndef SALESP_H
5.  #define SALESP_H
6.
7.  class SalesPerson
8.  {
9.  public:
10.    SalesPerson(); // constructor
11.    void getSalesFromUser(); // input sales from keyboard
12.    void setSales( int, double ); // set sales for a specific month
13.    void printAnnualSales(); // summarize and print sales
14. private:
15.    double totalAnnualSales(); // prototype for utility function
16.    double sales[ 12 ]; // 12 monthly sales figures
17.  }; // end class SalesPerson
18.
19. #endif
```

访问函数

工具函数

9.5 Access Functions and Utility Functions

```
1.  // Fig. 9.6: SalesPerson.cpp
2.  // Member functions for class SalesPerson.
3.  #include <iostream>
4.  using std::cout;
5.  using std::cin;
6.  using std::endl;
7.  using std::fixed;
8.
9.  #include <iomanip>
10. using std::setprecision;
11.
12. #include "SalesPerson.h" // include SalesPerson class definition
13.
14. // initialize elements of array sales to 0.0
15. SalesPerson::SalesPerson()
16. {
17.    for ( int i = 0; i < 12; i++ )
18.       sales[ i ] = 0.0;
19. } // end SalesPerson constructor
```

```
21.  // get 12 sales figures from the user at the keyboard
22.  void SalesPerson::getSalesFromUser()
23.  {
24.     double salesFigure;
25.
26.     for ( int i = 1; i <= 12; i++ )
27.     {
28.        cout << "Enter sales amount for month " << i << ": ";
29.        cin >> salesFigure;
30.        setSales( i, salesFigure );
31.     } // end for
32.  } // end function getSalesFromUser
33.
34.  // set one of the 12 monthly sales figures; function subtracts
35.  // one from month value for proper subscript in sales array
36.  void SalesPerson::setSales( int month, double amount )
37.  {
38.     // test for valid month and amount values
39.     if ( month >= 1 && month <= 12 && amount > 0 )
40.        sales[ month - 1 ] = amount; // adjust for subscripts 0-11
41.     else // invalid month or amount value
42.        cout << "Invalid month or sales figure" << endl;
43.  } // end function setSales
```

```
45.  // print total annual sales (with the help of utility function)
46.  void SalesPerson::printAnnualSales()
47.  {
48.     cout << setprecision( 2 ) << fixed
49.          << "\nThe total annual sales are: $"
50.          << totalAnnualSales() << endl; // call utility function
51.  } // end function printAnnualSales
52.
53.  // private utility function to total annual sales
54.  double SalesPerson::totalAnnualSales()
55.  {
56.     double total = 0.0; // initialize total
57.
58.     for ( int i = 0; i < 12; i++ ) // summarize sales results
59.        total += sales[ i ]; // add month i sales to total
60.
61.     return total;
62.  } // end function totalAnnualSales
```

访问函数

工具函数

- 一般情况下，类的数据成员和在类内部使用的成员函数应该指定为private，只有提供给外界使用的成员函数才能指定为public
- 操作一个对象，只能通过访问对象类中的public成员来实现

9.5 Access Functions and Utility Functions

```
1. // Fig. 9.7: fig09_07.cpp
2. // Demonstrating a utility function
3. // Compile this program v
4.
5. // include SalesPerson cl
6. #include "SalesPerson.h
7.
8. int main()
9. {
10. SalesPerson s; // create
11.
12. s.getSalesFromUser(); /
13. s.printAnnualSales(); //
14. return 0;
15. } // end main
```

```
Enter sales amount for month 1: 5314.76
Enter sales amount for month 2: 4292.38
Enter sales amount for month 3: 4589.83
Enter sales amount for month 4: 5534.03
Enter sales amount for month 5: 4376.34
Enter sales amount for month 6: 5698.45
Enter sales amount for month 7: 4439.22
Enter sales amount for month 8: 5893.57
Enter sales amount for month 9: 4909.67
Enter sales amount for month 10: 5123.45
Enter sales amount for month 11: 4024.97
Enter sales amount for month 12: 5923.92

The total annual sales are: $60120.59
```

- 控制逻辑应封装在类的成员函数中

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- **9.6 Time Class Case Study: Constructors with Default Arguments**
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.6 Time Class Case Study: Constructors with Default Arguments

- 默认实参应在函数名第一次出现时设定
- 全局函数
  - 函数原型
  - 无函数原型时的函数头部
- 类成员函数
  - 类定义

9.6 Time Class Case Study: Constructors with Default Arguments

- 构造函数可以指定默认实参

- 缺省构造函数
  - 不带参数的构造函数
  - 或所有参数都有默认值的构造函数
- 总结: 不指定实参即可调用的构造函数, 称为缺省构造函数

程序解读（P303）

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- **9.7 Destructors**
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

9.7 Destructors—需求

- new 运算符, 申请内存(Ch10.6)
- delete 运算符, 释放内存

6.11 Function Call Stack and Activation Records(函数调用栈和活动记录)

程序占用内存区域的分布:

- 代码区(code area)
  - 被编译程序的执行代码部分
- 全局数据区(data area)
  - 常量、静态全局量和静态局部量等
- 堆区(heap)
  - 动态分配的内存, new / delete
- 栈区(stack)
  - 函数数据区, 函数形参和局部变量等自动变量所使用的内存区域

9.7 Destructors—需求

```
1.  class test {
2.  public:
3.      void init() { … new … }
4.      void useTheMemory() { …… }
5.      void cleanup() { … delete … }
6.  };
```

- 用户在使用test类时, 必须首先调用init函数, 使用完后调用cleanup函数!
  - 忘了调用init, use wrong memory
  - 忘了调用cleanup, Memory Leakage

9.7 Destructors—需求

- 希望有函数能在创建对象时由系统自动调用，有函数在对象销毁时自动调用：
  - constructor (ctor), 构造函数
  - destructor (dtor), 析构函数

```
1.  class test{
2.  public:
3.      test() { ... new ... }
4.      void useTheMemory() { ..... }
5.      ~test() { ... delete ... }
6.  };
```

9.7 Destructors—定义

```
10.  class CreateAndDestroy
11.  {
12.  public:
13.      CreateAndDestroy( int, string ); // constructor
14.      ~CreateAndDestroy(); // destructor
```

Line 14: ~CreateAndDestroy ()

- 析构函数, 另一种特殊的成员函数
  - 与类同名, 类名前带 ~
  - 无参数, 无返回类型 (包括void)
  - public, 公有函数
- 用于进行类对象销毁时的清理工作

9.7 Destructors—定义

- A class's destructor is called implicitly when an object is destroyed.

- If the programmer does not explicitly provide a destructor, the compiler creates an "empty" destructor.

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- **9.8 When Constructors and Destructors Are Called**
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- 9.10 Default Memberwise Assignment

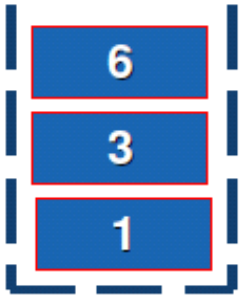
9.8 When Constructors and Destructors Are Called

- 全局对象: 在任何函数(含main)执行前, 构造; 在程序结束时, 析构.
- 局部对象
  - 自动变量: 对象定义时, 构造; 块结束时, 析构.
  - 静态变量: 首次定义时, 构造; 程序结束时, 析构.
- 多个全局和静态对象(均为静态存储类别)析构顺序恰好与构造顺序相反.
- 特例1: 调用exit函数退出程序执行时, 不调用剩余自动对象的析构函数.
- 特例2: 调用abort函数退出程序执行时, 不调用任何剩余对象的析构函数.

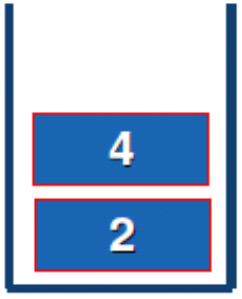
```
12.     CreateAndDestroy first( 1, "(global before main)" ); // global object
13.
14.     int main()
15.     {
16.        cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
17.        CreateAndDestroy second( 2, "(local automatic in main)" );
18.        static CreateAndDestroy third( 3, "(local static in main)" );
19.
20.        create(); // call function to create objects
22.        cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
23.        CreateAndDestroy fourth( 4, "(local automatic in main)" );
24.        cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
25.        return 0;
26.     } // end main
29.     void create( void )
30.     {
32.        CreateAndDestroy fifth( 5, "(local automatic in create)" );
33.        static CreateAndDestroy sixth( 6, "(local static in create)" );
34.        CreateAndDestroy seventh( 7, "(local automatic in create)" );
35.     } // end function create
```

1C
2C
3C
5C
6C
7C
7D
5D
4C
4D
2D
6D
3D
1D

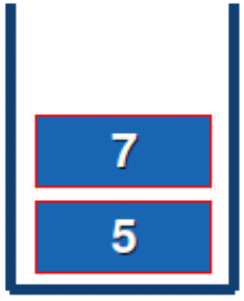
全局数据区



main 栈区



create 栈区



```
12.     CreateAndDestroy first( 1, "(global before main)" ); // global object
13.
14.     int main()
15.     {
16.        cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
17.        CreateAndDestroy second( 2, "(local automatic in main)" );
18.        static CreateAndDestroy third( 3, "(local static in main)" );
19.
20.        create(); // call function to create objects
22.        cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
23.        CreateAndDestroy fourth( 4, "(local automatic in main)" );
24.        cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
25.        return 0;
26.     } // end main
29.     void create( void )
30.     {
32.        CreateAndDestroy fifth( 5, "(local automatic in create)" );
33.        static CreateAndDestroy sixth( 6, "(local static in create)" );
34.        CreateAndDestroy seventh( 7, "(local automatic in create)" );
35.     } // end function create
```

exit(0);

1C
2C
3C
5C
6C
7C
7D
5D
4C
4D
2D
6D
3D
1D

```
12.     CreateAndDestroy first( 1, "(global before main)" ); // global object
13.
14.     int main()
15.     {
16.        cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
17.        CreateAndDestroy second( 2, "(local automatic in main)" );
18.        static CreateAndDestroy third( 3, "(local static in main)" );
19.
20.        create(); // call function to create objects
22.        cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
23.        CreateAndDestroy fourth( 4, "(local automatic in main)" );
24.        cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
25.        return 0;
26.     } // end main
29.     void create( void )
30.     {
32.        CreateAndDestroy fifth( 5, "(local automatic in create)" );
33.        static CreateAndDestroy sixth( 6, "(local static in create)" );
34.        CreateAndDestroy seventh( 7, "(local automatic in create)" );
35.     } // end function create
```

return 0; ↔ abort();

```
12.     CreateAndDestroy first( 1, "(global before main)" ); // global object
13.
14.     int main()
15.     {
16.        cout << "\nMAIN FUNCTION: EXECUTION BEGINS" << endl;
17.        CreateAndDestroy second( 2, "(local automatic in main)" );
18.        static CreateAndDestroy third( 3, "(local static in main)" );
19.
20.        create(); // call function to create objects
22.        cout << "\nMAIN FUNCTION: EXECUTION RESUMES" << endl;
23.        CreateAndDestroy fourth( 4, "(local automatic in main)" );
23.5    create();
24.        cout << "\nMAIN FUNCTION: EXECUTION ENDS" << endl;
           return 0;
26.     } // end main
29.     void create( void )
30.     {
           CreateAndDestroy fifth( 5, "(local automatic in create)" );
           static CreateAndDestroy sixth( 6, "(local static in create)" );
           CreateAndDestroy seventh( 7, "(local automatic in create)" );
       } // end function create
```

1C
2C
3C
5C
6C
7C
7D
5D
4C
5C
7C
7D
5D
4D
2D
6D
3D
1D

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- **9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member**
- 9.10 Default Memberwise Assignment

9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member

```
1.  int& test()
2.  {
3.      static int ret = 0; // 必须为静态变量
4.      ret++;
5.      cout << ret << endl;
6.      return ret;
7.  }

8.  int main(){
9.      test(); test(); test();
10.     int& ret_ref = test();
11.     ret_ref = 100;
12.     test() = 200;
13.     test();
14.     return 0;
15. }
```

```
1
2
3
4
101
201
```

以test返回的引用作为左值

9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member

- 类成员函数同样可以返回引用, 并且可以是私有数据成员的引用, 但应避免这种用法.

```
class Time
{
public:
   Time( int = 0, int = 0, int = 0 );
   void setTime( int, int, int );
   int getHour();
   int &badSetHour( int );
private:
   int hour;
   int minute;
   int second;
}; // end class Time
```

私有数据成员

```
int &Time::badSetHour( int hh )
{
   hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
   return hour;
} // end function badSetHour
```

```
int main()
{
   Time t;
   int &hourRef = t.badSetHour( 20 );

   cout << "Valid hour before modification: "
        << hourRef;
   hourRef = 30;
   cout << "\n Invalid hour after modification: "
        << t.getHour();

   // Dangerous: Function call that returns
   // a reference can be used as an lvalue!
   t.badSetHour( 12 ) = 74;

   cout << t.getHour() << endl;
   return 0;
} // end main
```

- Returning a reference or a pointer to a private data member breaks the encapsulation of the class.

Topics

- 9.1 Introduction
- 9.2 Time Class Case Study
- 9.3 Class Scope and Accessing Class Members
- 9.4 Separating Interface from Implementation
- 9.5 Access Functions and Utility Functions
- 9.6 Time Class Case Study: Constructors with Default Arguments
- 9.7 Destructors
- 9.8 When Constructors and Destructors Are Called
- 9.9 Time Class Case Study: A Subtle Trap - Returning a Reference to a private Data Member
- **9.10 Default Memberwise Assignment**

9.10 Default Memberwise Assignment

基本数据类型

- int a = b; // 初始化

  a = b; // 赋值运算

抽象数据类型

- 对象之间的赋值运算
- 用一个对象对另一个对象初始化(缺省拷贝构造)
- 缺省情况下, 逐个成员赋值!

9.10 Default Memberwise Assignment

- The assignment operator (=) can be used to assign an object to another object of the same type.

- Example:

```
Date date1, date2;
date2 = date1;
```

9.10 Default Memberwise Assignment

- By default, such assignment is performed by memberwise assignment (按成员赋值/逐个成员赋值).
- Example: // 缺省赋值运算

```
Date date1, date2; // year, month, day
date2 = date1;
```

即:

```
date2. year = date1.year;
date2. month = date1.month;
date2. day = date1.day;
```

9.10 Default Memberwise Assignment

1.
```
void copyDate( Date date2 );
Date date1;
copyDate( date1 ); // Date date2 = date1
```

2.
```
Date getDate();
Date date2 = getDate();
```

- 均为传值方式, 编译器提供default copy constructor (缺省拷贝构造函数), 将原对象的每个数据成员的值拷贝至新对象的相应成员, 即 Memberwise Copy (Assignment).

程序解读（P313）

Summary

- 条件编译
- 访问成员函数的三种方式(句柄+操作符)
- 成员函数的作用域：class scope
- 访问函数和工具函数
- 带默认实参的构造函数
- 构造函数和析构函数被调用的顺序
- 破坏类的封装的一种做法：返回对私有数据成员的引用
- 利用一个对象初始化另一个对象(拷贝构造函数)