

第四章 包含动态内存申请的类

主讲教师: 同济大学电子与信息工程学院 陈宇飞 同济大学电子与信息工程学院 龚晓亮



景

- 含动态内存申请的构造与析构函数
- 构造函数与析构函数的调用时机
- 对象的动态建立和释放
- 对象的赋值与复制

4.1 含动态内存申请的构造与析构函数



- 使用:
 - 有数据成员需要动态内存申请的情况下,可在构造函数中申请空间, 在析构函数中释放空间
 - 在没有数据成员需要动态内存申请的情况下,一般不需要定义析构函数
 - 在有数据成员需要动态内存申请的情况下,也可以不定义析构函数而通过其他方法释放(但不提倡)

• 常规做法:系统自动调用析构函数(非显式)



```
class Time {
 private:
  int hour;
  int minute;
   int sec;
   char *s;
 public:
  Time(); //构造函数
  ~Time();//析构函数
Time::Time()
  hour = 0;
  minute = 0;
   sec = 0;
```

```
//接左侧
  s = new char[80];//动态申请
Time::~Time()
  delete s; //释放
int main()
  Time t1;
```

• 不提倡: 调用Release函数(显式)



```
class Time {
 private:
   int hour;
  int minute;
  int sec;
  char *s;
 public:
  Time(); //构造函数
  Release(); //定义成员函数
Time::Time()
  hour = 0;
  minute = 0;
   sec = 0;
```

```
//接左侧
  s = new char[80];//动态申请
Time::Release()
  delete s: //释放
int main()
  Time t1;
  t1. Release();
```

```
#include <iostream>
                                 int main()
using namespace std;
class Time {
                                    Time *t1 = new Time; //申请16字节
                                    cout << "main begin" <<endl;</pre>
 private:
     int hour, minute, sec;
                                    delete t1;
                                    cout << "main end" <<endl;</pre>
     char *s;
 public:
    Time();
                                    t1 2000
                                                       hour
                                          2100
    \simTime();
                                                   2100
                                                                 3000
                                                       minute
                                                        sec
                                                   2115
                                                                     ???
                                                       3000
Time::Time() {
                                                           构造函数 | 3079
    hour = 0;
                                      系统回收
    minute = 0;
    sec = 0;
                                    t1 2000
                                                       hour
                                           2100
                                                   2100
    s = new char[80]://申请
                                                                 3000
                                                      minute
                                                       sec
                                                   2115
                                                                     ???
                                                       3000
Time::~Time()
    delete s; //释放 }
```



main begin main end



景

- 含动态内存申请的构造与析构函数
- 构造函数与析构函数的调用时机
- 对象的动态建立和释放
- 对象的赋值与复制

4.2 构造函数与析构函数的调用时机



• 构造函数:

自动对象(形参)

静态局部对象

静态全局/外部全局对象

动态申请的对象

• 析构函数:

自动对象(形参)

静态局部对象

静态全局/外部全局对象

动态申请的对象

: 函数中变量定义时

: 第一次调用时

:程序开始时

: new时

: 函数结束时

: 程序结束时

: 程序结束时

: delete时

```
main begin
#include <iostream>
                                        Time::~Time()
                         Time Begin
using namespace std;
                                            cout << "Time End" << end1;</pre>
                         fun
class Time {
                         Time End
  private:
                         continue
                                        void fun()
     int hour;
                         Time Begin
                                            Time t1;
     int minute;
                         fun
                                            cout << "fun" <<endl;</pre>
                         Time End
     int second;
                         main end
  public:
                                        int main()
    Time (int h=0, int m=0, int s=0);
     \simTime():
                                            cout << "main begin" <<endl;</pre>
                                            fun();
Time::Time(int h, int m, int s)
                                            cout << "continue" << endl;</pre>
                                            fun();
   hour = h;
                                            cout << "main end" << endl;</pre>
   minute = m;
   second = s:
   cout << "Time Begin" << endl;</pre>
                                           1、函数调用时分配空间结束时回收空间
                                           2、函数多次调用则多次分配/回收空间
```

```
main begin
#include <iostream>
                                       Time::~Time()
                        Time Begin
using namespace std;
                                           cout << "Time End" << endl;</pre>
                        fun
class Time {
                        continue
  private:
                                       void fun()
                        fun
     int hour;
                                           static Time t1;
                        main end
     int minute;
                                           cout << "fun" <<endl;
                        Time End
     int second;
  public:
                                       int main()
    Time(int h=0, int m=0, int s=0);
    ~Time();
                                           cout << "main begin" <<endl;</pre>
                                           fun();
Time::Time(int h, int m, int s)
                                           cout << "continue" << endl;</pre>
                                           fun();
   hour = h;
                                           cout << "main end" << endl;</pre>
   minute = m;
   second = s:
                                                      1、函数第1次调用时分配
   cout << "Time Begin" << endl;</pre>
                                                      2、后续函数调用不分配
                                                      3、全部程序结束后回收
```

```
Time Begin
#include <iostream>
                                          Time::~Time()
                         main begin
using namespace std;
                                              cout << "Time End" << endl;
                         fun begin
class Time {
                         fun end
  private:
                                          Time t1;
                         main end
     int hour;
                                          void fun()
                         Time End
     int minute;
     int second;
                                              cout << "fun begin" <<endl;</pre>
  public:
                                              cout << "fun end" <<endl;</pre>
     Time (int h=0, int m=0, int s=0);
     \simTime():
                                          int main()
Time::Time(int h, int m, int s)
                                              cout << "main begin" <<endl;</pre>
                                              fun();
   hour = h;
                                              cout << "main end" <<endl;</pre>
   minute = m;
   second = s:
   cout << "Time Begin" << endl;</pre>
                                                         1、main开始前分配
                                                         2、main结束后回收
```

```
main begin
                                         Time::~Time()
#include <iostream>
                          Time Begin
                                             cout << "Time End" << endl;
using namespace std;
                          new end
class Time {
                          Time End
  private:
                                         int main()
                          main end
     int hour;
     int minute;
                                              cout << "main begin" <<endl;</pre>
     int second;
                                              Time *t1 = new Time;
  public:
                                              cout << "new end" <<endl;</pre>
     Time (int h=0, int m=0, int s=0);
                                              delete t1;
     \simTime():
                                              cout << "main end" <<endl;</pre>
Time::Time(int h, int m, int s)
   hour = h;
   minute = m;
   second = s:
   cout << "Time Begin" << endl;</pre>
                                                              1、new时分配
                                                              2、delete时回收
```

```
main begin
#include <iostream>
                           new end
using namespace std;
                           main end
class Time {
  private:
     int hour;
     int minute;
     int second;
  public:
     Time (int h=0, int m=0, int s=0);
     \simTime():
Time::Time(int h, int m, int s)
   hour = h;
   minute = m;
   second = s:
   cout << "Time Begin" << endl;</pre>
```

```
Time::~Time()
 cout << "Time End" << endl;
int main()
   cout << "main begin" <<endl;</pre>
  Time *t1 = (Time *) malloc
              (sizeof(Time));
   cout << "new end" <<endl;</pre>
   free(t1):
   cout << "main end" <<endl:</pre>
```

malloc仅仅是分配内存,不会调用构造函数 free不会调用析构函数

```
void fun()
#include <iostream>
                                                 Time t1(15), t2(16);
using namespace std;
                                                 cout << "fun" <<endl:</pre>
class Time {
  private:
                                             int main()
    int hour, minute, second;
                                                 cout << "main begin" <<endl;</pre>
  public:
    Time (int h=0, int m=0, int s=0);
                                                  fun();
                                                 cout << "main end" <<endl;</pre>
    \simTime();
                                                                     main begin
Time::Time(int h, int m, int s)
                                                                     Time Begin15
    hour = h;
                                                                      Time Begin16
    minute = m;
                                                                     fun
    second = s;
                                                                      Time End16
    cout<< "Time Begin" << hour <<endl;</pre>
                                                                      Time End15
                                                                     main end
Time::~Time()
                                                                t1, t2都是自动变量
    cout << "Time End" << hour << endl;</pre>
                                                                   构造: t1, t2
                                                                   析构: t2, t1
```

```
void fun()
#include <iostream>
                                                static Time t1(15), t2(16)
using namespace std;
                                                cout << "fun" <<endl;</pre>
class Time {
  private:
    int hour, minute, second;
                                            int main()
                                                cout << "main begin" <<endl;</pre>
  public:
    Time (int h=0, int m=0, int s=0);
                                                 fun();
    \simTime();
                                                 cout << "main end" <<endl;</pre>
                                                                    main begin
Time::Time(int h, int m, int s)
                                                                    Time Begin15
    hour = h;
                                                                    Time Begin16
    minute = m;
                                                                    fun
    second = s;
                                                                    main end
    cout<< "Time Begin" << hour <<endl;</pre>
                                                                    Time End16
                                                                    Time End15
Time::~Time()
                                                            t1, t2都是静态局部变量
    cout << "Time End" << hour << endl;</pre>
                                                                 构造: t1, t2
                                                                 析构: t2, t1
```

```
#include <iostream>
using namespace std;
class Time {
  private:
    int hour, minute, second;
  public:
    Time (int h=0, int m=0, int s=0);
    \simTime():
Time::Time(int h, int m, int s)
    hour = h;
    minute = m;
    second = s;
    cout<< "Time Begin" << hour <<endl;</pre>
Time::~Time()
    cout << "Time End" << hour << endl;</pre>
```

```
Time t1(15), t2(16);
int main()
  cout << "main" <<endl;</pre>
                      Time Begin15
                       Time Begin16
                       main
                      Time End16
                       Time End15
```

t1, t2都是全局变量 构造: t1, t2 析构: t2, t1

```
void fun()
#include <iostream>
                                                  Time t1(15);
using namespace std;
                                                   static Time t2(16);
class Time {
                                                   cout << "fun" <<endl;</pre>
  private:
    int hour, minute, second;
  public:
                                               int main()
                                                   cout << "main begin" <<endl;</pre>
    Time (int h=0, int m=0, int s=0);
    \simTime();
                                                   fun();
                                                   cout << "main end" <<endl;</pre>
Time::Time(int h, int m, int s)
    hour = h;
    minute = m;
                                              main begin
    second = s;
                                              Time Begin15
    cout<< "Time Begin" << hour <<endl;</pre>
                                              Time Begin16
                                              fun
Time::~Time()
                                              Time End15
    cout << "Time End" << hour << endl;</pre>
                                                             t1, t2是不同性质的变量
                                              main end
                                              Time End16
```

```
#include <iostream>
                                             int main()
using namespace std;
                                                 Time *t1, *t2;
class Time {
                                                 t2=new Time(16);
                                                 t1=new Time(15);
 private:
   int hour, minute, second;
                                                 cout << "main begin" <<endl;</pre>
 public:
                                                 delete t2;
   Time (int h=0, int m=0, int s=0);
                                                 cout << "main end" <<endl;</pre>
   \simTime();
                                                 delete t1;
                                                                 Time Begin16
Time::Time(int h, int m, int s)
                                                                 Time Begin15
    hour = h;
                                                                 main begin
    minute = m;
                                                                 Time End16
    second = s;
                                                                 main end
    cout<< "Time Begin" << hour <<endl;</pre>
                                                                 Time End15
Time::~Time()
                                                 动态申请的变量,按new的顺序调
    cout << "Time End" << hour << endl:</pre>
                                                         按delete的顺序调析构,
                                                 不遵循栈规则
```



目录

- 含动态内存申请的构造与析构函数
- 构造函数与析构函数的调用时机
- 对象的动态建立和释放
- 对象的赋值与复制

4.3 对象的动态建立和释放



• C++方法: Time *p1, *p2; 申请: p1 = new(nothrow) Time; if (p1==NULL) { ... } $p2 = new(nothrow) Time(); if (p2==NULL) { ... }$ //new Time()创建对象时候,系统除了执行默认构造函数会 执行的那些操作外,还会为基本数据类型和指针类型的成员 用0赋初值 $p3 = new(nothrow) Time[2]; if (p3==NULL) { ... }$ 释放: delete pl; delete p2; delete []p3;

```
#include<iostream>
using namespace std;
class Time
private:
  int hour;
  int minute;
  int second;
public:
  void display();
void Time::display()
  cout << hour << endl;
  cout << minute << endl;</pre>
  cout << second << endl;</pre>
```

```
int main()
{ Time* p1 = new(nothrow) Time;
   if (p1 == NULL) \{ return -1; \}
  p1->display(); //随机值
   delete p1;
  Time* p2 = new(nothrow) Time()/;
   if (p2 == NULL) \{ return -1; \}
  p2->display(); //0, 0, 0
   delete p2;
  Time* p3 = new(nothrow)(Time[2]);
   if (p3 == NULL) \{ return -1; \}
  p3[0]. display(); //随机值
  p3[1]. display(); //随机值
  delete []p3;
  return 0;
```



```
-842150451
-842150451
-842150451
-842150451
842150451
842150451
842150451
```

4.3 对象的动态建立和释放



- C++中一般不建议使用C方法动态申请
 - · C方式动态内存申请和释放时不会调用构造和析构函数
 - 模块02例中,struct中有string类,则malloc/free会出错

```
#include<iostream>
                                   int main()
using namespace std;
                                     Time* p1 = new(nothrow) Time;
class Time
                                                  //调用构造函数
                                      if (p1 == NULL) \{ return -1; \}
                                      pl->display();
private:
  int hour;
                                      delete p1;
  int minute;
  int second;
                                      Time* p2 = (Time*)malloc(sizeof(Time));
                                                  //未调用构造函数
public:
  Time() { cout << "called!\n"; }
                                      if (p2 == NULL) \{ return -1; \}
  void display();
                                      p2->display();
                                                                called!
                                      free (p2);
                                                                 -842150451
void Time::display()
                                                                 -842150451
                                      retuen 0;
  cout << hour << endl;
                                                                 -842150451
  cout << minute << endl;</pre>
                                                                 -842150451
  cout << second << end1;</pre>
                                                                 -842150451
                                                                 -842150451
```



見录

- 含动态内存申请的构造与析构函数
- 构造函数与析构函数的调用时机
- 对象的动态建立和释放
- 对象的赋值与复制



• 基本概念

	<u>赋值</u>	复制
含义	将一个对象的所有 <mark>数据成员</mark> 的值对应赋值给另一 个对象的数据成员	建立一个新对象,其值与某个已有对象完全相同
形式	//执行语句 类名 对象名1,对象名2; 对象名1=对象名2;	//定义语句 类名 对象名(已有对象名) 类名 对象名=已有对象名
实现	将对象2的全部数据成员的值对应赋给对象1的全部数据成员,不包括成员函数(整体内存拷贝)	建立新对象时自动调用拷贝构造函数 () 浅拷贝:缺省拷贝构造函数 (内存拷贝)

思考: 1) 若对象数据成员无动态分配的数据,结果是否与预期一致? 若不一致如何解决?

深拷贝: 拷贝构造函数重载

2) 若对象数据成员是指针及动态分配的数据呢?

1907 JA

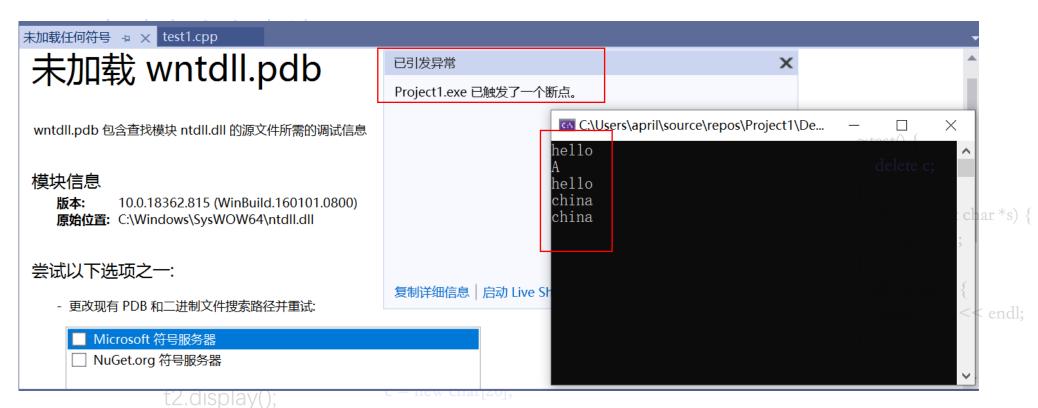
- 对象的赋值
 - 有动态内存申请

```
int main()
   test t1("hello"), t2:
   tl.display();
   t2. display();
   t2=t1;
   t2. display();
   t1. set ("china");
   tl. display();
   t2. display();
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
                                ~test() {
using namespace std;
                                  delete c;
class test {
private:
                                void set(const char *s)
    int a;
    int b;
                                  strcpy(c, s);
   char *c;
public:
                                void display()
    test(const char *s="A")
       a=0; b=0;
                                  cout << c << endl;
        c = new char[20];
        strcpy(c, s);
```

TO TO THE PART OF THE PART OF

- 对象的赋值
 - 有动态内存申请 //上例运行结果:

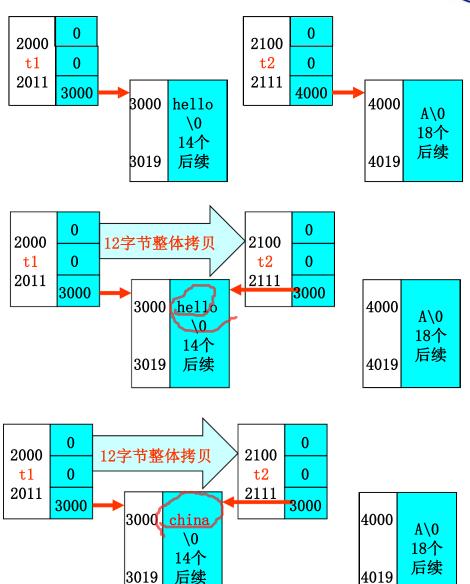


//有动态内存申请时,执行结果错且有错误弹窗

• 对象的赋值

• 有动态内存申请

```
int main()
    test t1("hello"), t2;
    tl. display();
                       hello
    t2. display();
    t2=t1;
    t2. display();
                        hello
    t1. set ("china");
    tl. display();
                        china
    t2. display();
                        china
```





• 对象的赋值

• 有动态内存申请

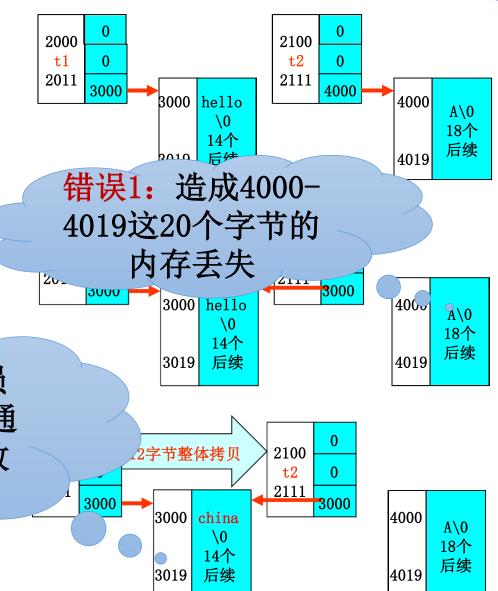
```
int main()
    test t1("hello"), t2;
    t1. display();
                       hello
    t2. display();
    t2=t1;
    t2. display();
                        hello
    t1. set ("china");
    tl. display();
                        china
    t2. display();
                        china
```





- 对象的赋值
 - 有动态内存申请

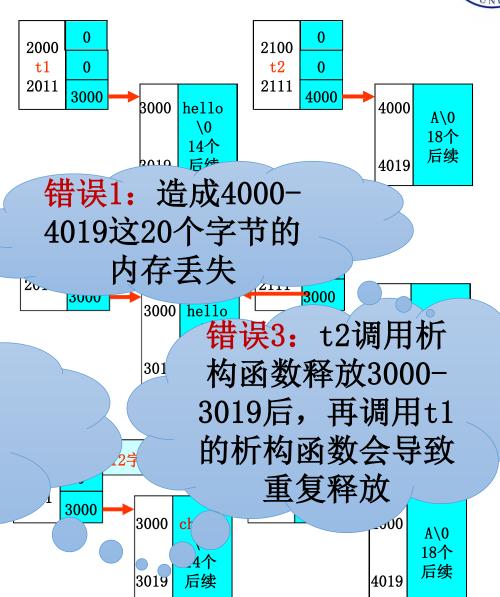
```
int main()
   test t1("hello"), t2;
   t1. display(); hello
   t2. display();
   t2=t1;
   t2. display 错误2: t1/t2的c成员
   t1. set ("
             同时指向一块内存,通
   t1. di
             过t1的c修改,会导致
   t2. dis
               t2的c值同时改变
```





- 对象的赋值
 - 有动态内存申请

```
int main()
   test t1("hello"), t2;
   tl. display();
               hello
   t2. display();
   t2=t1;
   t2. display 错误2: t1/t2的c成员
   t1. set ("
             同时指向一块内存,通
   t1. di
             过t1的c修改,会导致
   t2. dis
               t2的c值同时改变
```



- 对象的赋值
 - 有动态内存申请

```
int main()
                                         错误1:造成
  tl. display(); 如何解决?
t2. display();
                                        4000-4019这20个
                                        字节的内存丢失
                                               错误3: t2调用析
  t2. display();
                                               构函数释放3000-
                  错误2: t1/t2的c成员同
                                               3019后,再调用
                   时指向一块内存,通过
                                               t1的析构函数会
                  t1的c修改,会导致t2的
                                                导致重复释放
  tl. display();
                      c值同时改变
  t2. display();
               china
```

•解决方法:运算符重载!!(后续内容,此处了解即可)

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std:
class test {
private:
    int a; int b; char *c;
public:
    test(const char *s="A")
         a=0; b=0;
         c=new char[20];
         strcpy(c, s);
     test()
    { delete c; };
    void set(const char *s)
    { strcpy(c, s); }
```

```
void display()
 { cout << c << end1; }
 test & operator = (const test &t);
      //重载=的声明
test &test::operator=(const test &t)
      //重载=体外实现
\{ a = t.a; b = t.b; \}
  delete c; //释放原空间
  c=new char[20]; //申请新空间
  strcpy(c, t.c);
  return *this; //返回对象自身
int main()
```



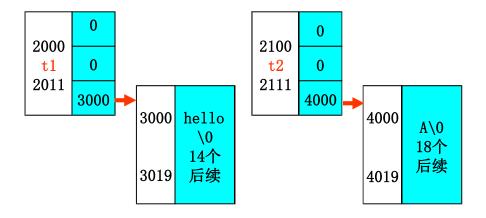
- 对象的赋值
 - 有动态内存申请 //上例运行结果(运算符重载为后续内容,此处仅了解)

```
test1.cpp ≠ X
                                                                                     g operator=(const test & t)
Project1
                                        → test
            #define CRT SECURE NO WARNINGS
          m#include <iostream>
           #include <cstring>
            using namespace std;
                                                              Microsoft Visual Studio 调试控制台
          □class test {
                                                              hello
            private:
                int a; int b; char* c;
                                                              hello
           public:
     8
                                                              china
                test(const char* s = "A")
                                                             hello
    10
    11
                   a = 0; b = 0;
                                                              C:\Users\april\source\repos\Project1\Debug\Project1.ex
                   c = new char[20];
    12
                                                                 (进程 14356)已退出,代码为 0。
在调试停止时自动关闭控制台,请启用"工具"->"选项"
"调试"->"调试停止时自动关闭控制台"。
    13
                   strcpy(c, s);
    14
                ~test() { delete c; };
    15
                                                              按任意键关闭此窗口...
                void set(const char* s) { strcpy(c, s); }
    16
                void display() { cout << c << endl; }</pre>
    17
               test& operator=(const test& t); //重载=的声明
    18
    19
            Itest& test: operator=(const test& t)//重载=体外空
```



- 对象的赋值
 - 有动态内存申请

```
int main()
   test t1("hello"), t2;
    t1. display(); hello
    t2. display(); A
    t2=t1;
    t2. display();
    tl. set ("china");
    tl. display();
    st2. display();
```



- 对象的赋值
 - 有动态内存申请

```
int main()
   test t1("hello"), t2;
    t1. display(); hello
    t2. display(); A
    t2=t1;
    t2. display();
                  hello
    tl. set ("china");
    tl. display();
    t2. display();
```

```
//解决方法:运算符重载!!
  test &test::operator=(const test &t)
     a = t.a; b = t.b;
     delete c; //释放原空间
     c=new char[20]; //申请新空间
     strcpy(c, t.c);
     return *this; //返回对象自身
                     5000 hello
                        \0
2000
             2100
                        14个
2011
             2111
  3000
      3000 hello
         \0
         14个
```

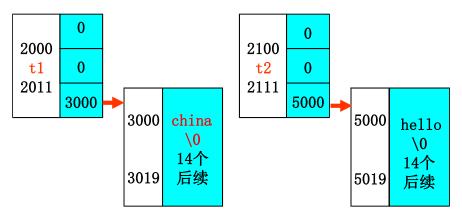
3019

后续



- 对象的赋值
 - 有动态内存申请

```
int main()
   test t1("hello"), t2;
   tl. display(); hello
    t2. display(); A
    t2=t1;
    t2. display();
                    hello
    t1. set ("china");
    tl. display();
                    china
    t2. display();
                  hello
```





- •对象的复制
 - 拷贝构造函数/复制构造函数
 - 形式: 类名(const 类名 &引用名)
 - 用一个对象的值去初始化另一个对象
 - 允许体内实现或体外实现
 - 复制构造函数和普通构造函数(可能多个)的地位平等,调用其中一个后就不再调用其它构造函数
 - · 若不定义复制构造函数,则系统自动定义一个,参数为const型引用, 函数体为对应成员内存拷贝(浅拷贝)
 - 若定义了复制构造函数,则系统缺省定义的消失(可做深拷贝)

- 对象的复制
 - 浅拷贝

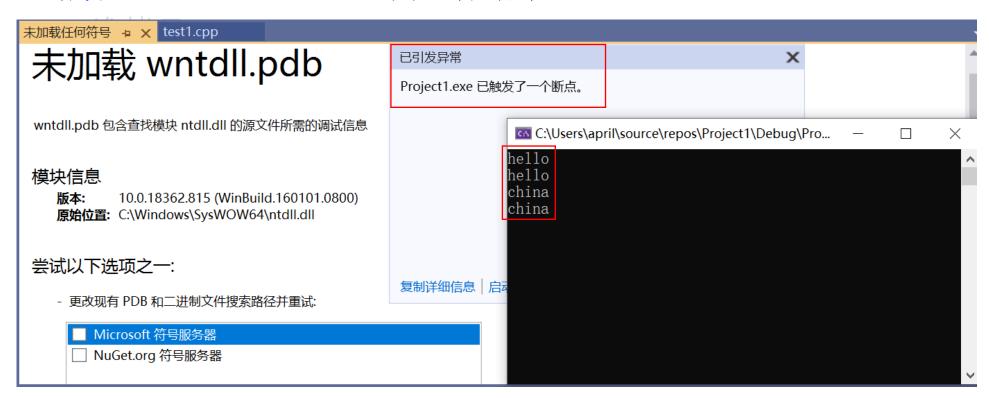
```
int main()
{
    test t1("hello"), t2(t1);
    t1. display();
    t2. display();
    t1. set("china");
    t1. display();
    t2. display();
}
```

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <cstring>
using namespace std;
class test {
                             ~test() {
private:
                                  delete c;
   int a;
   int b;
                             void set(const char *s)
  char *c;
                                  strcpy(c, s);
public:
   test(const char *s="A")
                             void display() {
       a=0; b=0;
                                  cout << c << endl;
       c=new char[20];
       strcpy(c, s);
```



- 对象的复制
 - 浅拷贝

//上例运行结果:

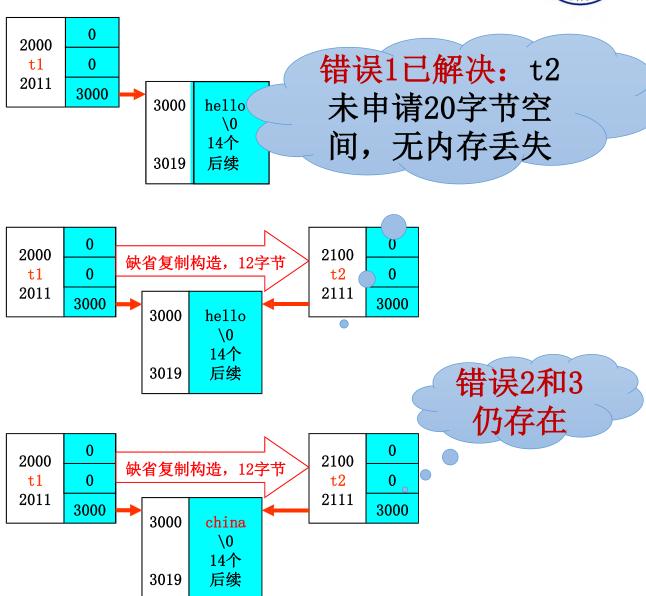


//有动态内存申请时,执行结果错且有错误弹窗



- 对象的复制
 - 浅拷贝

```
int main()
{    test t1("hello"), t2(t1);
    t1. display(); hello
    t2. display(); hello
    t1. set("china");
    t1. display(); china
    t2. display(); china
}
```



- 对象的复制
 - 浅拷贝

```
如何解决?

{ test t1("hello"), t2(t1);
    t1. display(); hello
    t2. display(); hello
    t1. set("china");
    t1. display(); china
    t2. display(); china
```



• 深拷贝 ---动态成员不是简单赋值,而是重新动态分配空间

```
#define _CRT_SECURE_NO WARNINGS
                                       test(const test &t);
#include <iostream>
                                          //复制构造函数的声明
                                       ~test() { delete c: }
#include <cstring>
using namespace std;
                                       void set(const char *s)
class test {
                                       { strcpy(c, s); }
private:
                                       void display()
    int a;
                                        { cout << c << end1; }
    int b;
    char *c;
                                    test::test(const test &s)
public:
                                         //复制构造的体外实现
   test(const char *s="A")
                                        a=s. a; b=s. b;
        a=0; b=0;
                                        c=new char[20];
        c=new char[20];
                                        strcpy(c, s.c);
        strcpy(c, s);
                                     int main()
                                     {•••}
```



- 对象的复制
 - 深拷贝

//上例运行结果:

```
test1.cpp ≠ X
Project1
                                               (全局范围)
             #define CRT SECURE NO WARNINGS
           ⊟#include <iostream>
             #include <cstring>
            using namespace std;
           □class test {
                                                                亟 Microsoft Visual Studio 调试控制台
                                                                                                                                X
            private:
                 int a:
                                                               hello
                int b:
                                                               hello
     9
                char* c;
                                                               china
     10
            public:
                                                               hello
                 test(const char* s = "A")
     11
                                                                C:\Users\april\source\repos\Project1\Debug\Project1.ex
     12
                                                              e (进程 20248)已退出,代码为 0。
要在调试停止时自动关闭控制台,请启用"工具"->"选项"->"调试"->"调试停止时自动关闭控制台"。
按任意键关闭此窗口...
     13
                     a = 0: b = 0:
                     c = new char[20]:
     14
     15
                     strcpy(c, s);
     16
                 test(const test& t): //复制构造函数的声明
     17
                 ~test() { delete c; }
     18
                 void set(const char* s) { strcpy(c, s); }
     19
                 void display() { cout << c << endl. }</pre>
```



• 对象的复制

• 深拷贝

```
2000
                                                              2100
                                                               t2
                                           t1
int main()
                                          2011
                                                              2111
                                             3000 hello
                                                                       4000
                                                                          hello
     test t1("hello"), t2(t1);
                                                       14个
                                                                           14个
     tl. display();
                        hello
                                                       后续
     t2. display();
                            hello
     t1. set ("china");
     tl. display();
                                                              2100
                             china
                                          2000
                                                               t2
                                           t1
     t2. display();
                                          2011
                                                              2111
                            hello
                                                  3000
                                                                      4000 hello
                                                      china
                                                                           \0
                                                       14个
                                                                           14个
                                                       后续
```



- 对象的赋值与复制(小结)
 - 对象的赋值发生在执行语句时,对象的复制发生在定义语句时
 - 赋值的操作是整体内存拷贝,复制的操作是自动调用拷贝构造函数
 - 若对象数据成员是指针及动态分配的数据,则可能导致不可预料的 后果甚至报错
 - ▶ 解决方法:

赋值:运算符重载(后续章节)

复制: 拷贝构造函数重载(深拷贝)



总结

- 含动态内存申请的构造与析构函数
- 构造函数与析构函数的调用时机
- 对象的动态建立和释放
- 对象的赋值与复制