

# 期末习题课

# 习题课内容

- ▶ 面向对象程序设计练习
- ▶ 综合练习

# 面向对象程序设计 ( OOP )

- ▶ 面向对象程序设计是一种以数据为中心、基于数据抽象的程序设计范式。
- ▶ 一个面向对象程序由一些对象构成；
- ▶ 对象的特征由相应的类来描述；
- ▶ C++的类是一种用户自定义类型，定义形式如下：

```
class <类名>
```

```
{ <成员描述> };
```

- ▶ 类的成员描述要根据具体问题分析而得，成员包括：
  - ▶ 数据成员：描述具体问题的状态或信息
  - ▶ 成员函数：对状态或者信息的操作

# 面向对象程序设计 vs 过程式程序设计

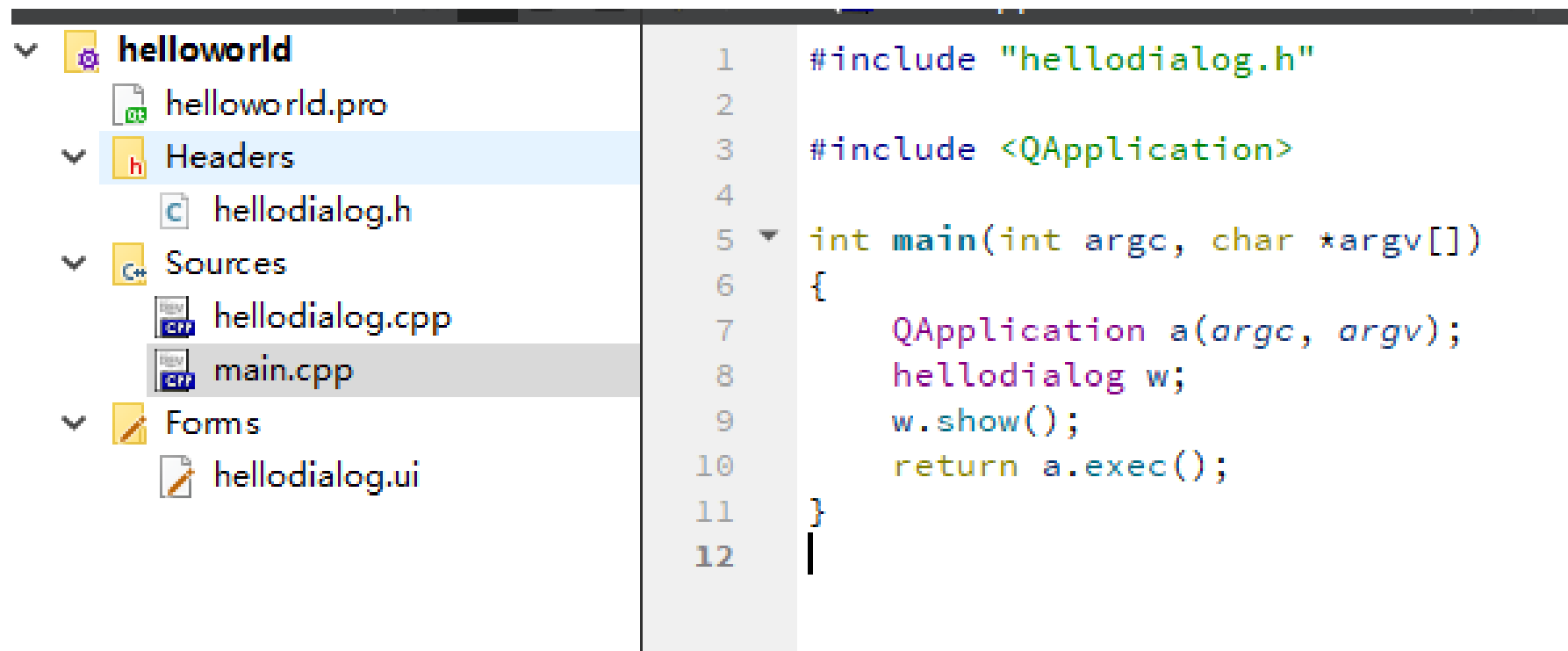
- ▶ 过程式程序设计：自顶向下、逐步精化
  - ▶ 单入单出的流程控制结构
  - ▶ 功能的分解及封装—**函数定义**
- ▶ 面向对象程序设计：
  - ▶ 数据和数据的操作封装在一起—**类的定义**
  - ▶ 以对象为中心，系统通过对象的交互完成特定的功能

# oop样例：使用现有的类定义对象

```
#include <QApplication> // 管理应用程序资源的类
#include <QDialog> //对话框的类
#include <QLabel> //标签的类
int main(int argc, char *argv[])
{
    QApplication a(argc,argv); // 定义一个Qt程序对象
    QDialog w; // 定义一个对话框对象
    w.resize(400,200); //调用成员函数，设定对话框的尺寸（单位是像素）
    QLabel label(&w); // 在对话框中，定义一个标签对象
    label.move(100,100); // 调用成员函数，将标签移动到相应坐标位置
    label.setText(“你好！南京大学！”); // 调用成员函数，设定标签文本内容
    w.show(); //调用成员函数，显示对话框
    return a.exec(); // Qt程序对象进入事件循环，等待运行期间产生的事件，比如鼠标按键
}
```



# oop样例：面向对象编程



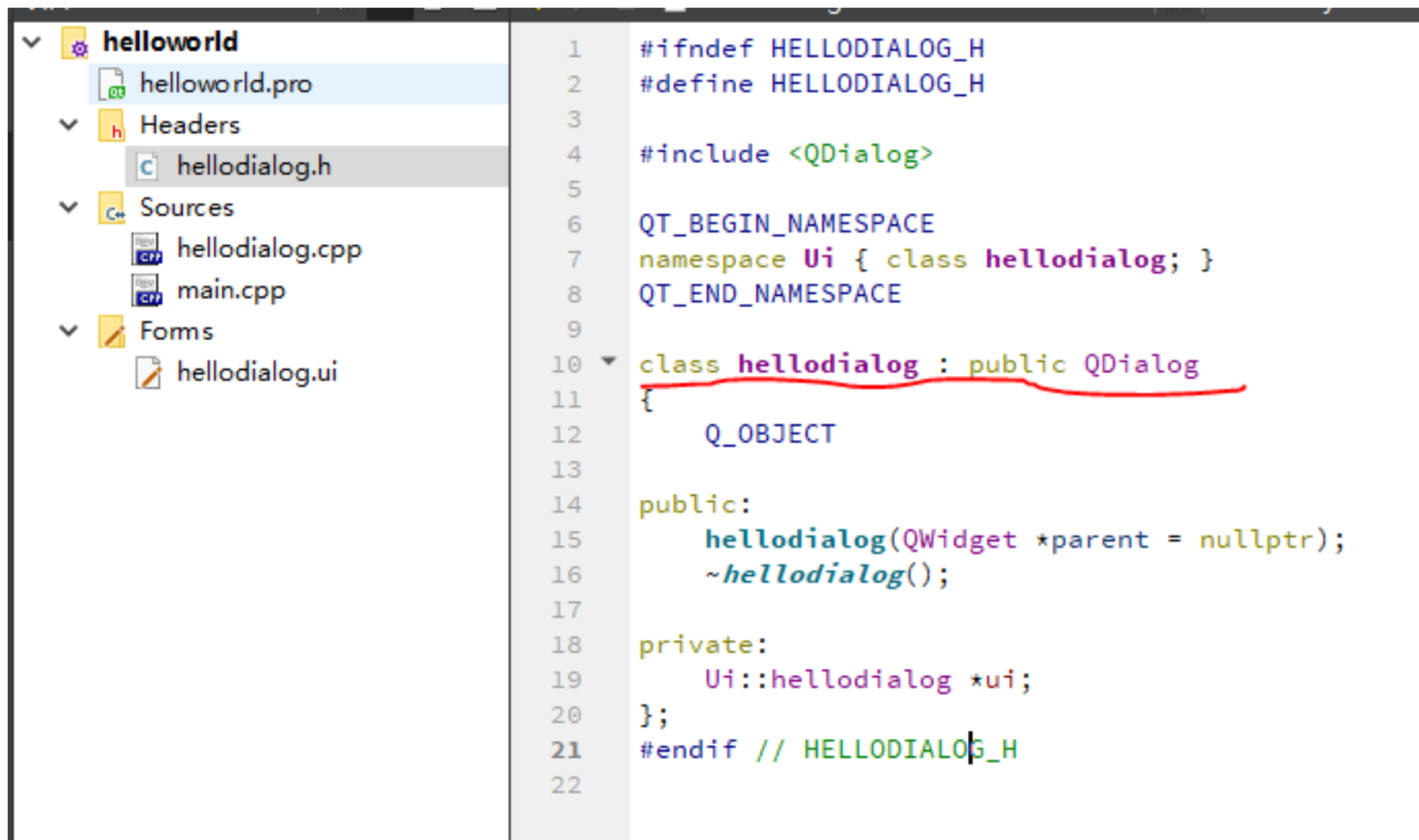
The screenshot displays the Qt IDE interface for a project named 'helloworld'. The left sidebar shows the project's file structure:

- helloworld
  - helloworld.pro
  - Headers
    - hellodialog.h
  - Sources
    - hellodialog.cpp
    - main.cpp
  - Forms
    - hellodialog.ui

The main editor window shows the content of 'main.cpp' with line numbers 1 through 12:

```
1 #include "hellodialog.h"
2
3 #include <QApplication>
4
5 int main(int argc, char *argv[])
6 {
7     QApplication a(argc, argv);
8     hellodialog w;
9     w.show();
10    return a.exec();
11 }
12
```










# oop 样例：面向对象编程



The screenshot displays a Qt IDE interface. On the left, a project explorer shows the structure of a project named 'helloworld'. The project contains a 'helloworld.pro' file, a 'Headers' directory with 'hellodialog.h', a 'Sources' directory with 'hellodialog.cpp' and 'main.cpp', and a 'Forms' directory with 'hellodialog.ui'. The 'hellodialog.h' file is selected, and its source code is shown in the main editor area on the right. The code is a C++ header file for a Qt widget class named 'hellodialog'. It includes the 'QDialog' header and uses the 'Ui' namespace. The class 'hellodialog' is defined as a public subclass of 'QDialog'. It inherits 'Q\_OBJECT' and implements a public constructor 'hellodialog(QWidget \*parent = nullptr)' and a destructor '~hellodialog()'. A private member variable 'ui' of type 'Ui::hellodialog' is declared. The code is enclosed in a preprocessor guard 'ifndef HELLODIALOG\_H' and 'endif // HELLODIALOG\_H'. The line 'class hellodialog : public QDialog' is underlined in red.

```
1  #ifndef HELLODIALOG_H
2  #define HELLODIALOG_H
3
4  #include <QDialog>
5
6  QT_BEGIN_NAMESPACE
7  namespace Ui { class hellodialog; }
8  QT_END_NAMESPACE
9
10 class hellodialog : public QDialog
11 {
12     Q_OBJECT
13
14 public:
15     hellodialog(QWidget *parent = nullptr);
16     ~hellodialog();
17
18 private:
19     Ui::hellodialog *ui;
20 };
21 #endif // HELLODIALOG_H
22
```

# oop样例：面向对象编程

 <b>helloworld</b>	1	<code>#include "hellodialog.h"</code>
 helloworld.pro	2	<code>#include "ui_hellodialog.h"</code>
▼  Headers	3	
 hellodialog.h	4	<code>hellodialog::hellodialog(QWidget *parent)</code>
▼  Sources	5	<code>: QDialog(parent)</code>
 hellodialog.cpp	6	<code>, ui(new Ui::hellodialog)</code>
 main.cpp	7	<code>{</code>
▼  Forms	8	<code>ui-&gt;setupUi(this);</code>
 hellodialog.ui	9	<code>}</code>
	10	
	11	<code>hellodialog::~hellodialog()</code>
	12	<code>{</code>
	13	<code>delete ui;</code>
	14	<code>}</code>
	15	
	16	



# oop样例：面向对象编程

过滤器

Stacked Widget

Frame

Widget

MDI Area

Dock Widget

QAxWidget

Input Widgets

Combo Box

Font Combo Box

Line Edit

Text Edit

Plain Text Edit

Spin Box

Double Spin Box

Time Edit

Date Edit

Date/Time Edit

Dial

Horizontal Scroll Bar

Vertical Scroll Bar

Horizontal Slider

Vertical Slider

Key Sequence Edit

Display Widgets

Label

Text Browser

Graphics View

Calendar Widget

你好！南京大学！

hellodialog

你好！南京大学！

名称

使用

文本

快捷方式

可选的

工具提示

Filter

对象

类

hellodialog

QDialog

label

QLabel

过滤器

hellodialog : QDialog

属性

值

QObject

objectName

hellodialog

QWidget

enabled

☒

geometry

X

0

Y

0

宽度

400

高度

200

sizePolicy

[Preferred, Preferred, 0, 0]

minimumSize

0 x 0

maximumSize

16777215 x 16777215

# oop思考：定义一个string类

- ▶ 需要定义什么样数据成员？
- ▶ 需要定义哪些参数的构造函数？
- ▶ 你能想到的接口有哪些？

string str: 生成空字符串

string s(str): 生成字符串为str的复制品

string s(str, strbegin, strlen): 将字符串str中从下标strbegin开始、长度为strlen的部分作为字符串初值

string s(cstr, char\_len): 以C\_string类型cstr的前char\_len个字符串作为字符串s的初值

string s(num ,c): 生成num个c字符的字符串

string s(str, stridx): 将字符串str中从下标stridx开始到字符串结束的位置作为字符串初值

```
string str1;           // 生成空字符串
```

```
string str2("123456789"); // 生成"1234456789"的复制品
```

```
string str3("12345", 0, 3); // 结果为"123"
```

```
string str4("012345", 5);  // 结果为"01234"
```

```
string str5(5, '1');       // 结果为"11111"
```

```
string str6(str2, 2);      // 结果为"3456789"
```

- ▶ 字符串的输出、遍历每个字符（正向、反向）
- ▶ 字符串的大小和容量
- ▶ 字符串比较
- ▶ string的字符插入、拼接、删除、替换、大小写转换、查找、截取、分割
- ▶ 每个接口都会有多个函数重载，以适应实际应用需求

## OOP练习：链表实现集合类IntSet

定义一个元素类型为int、元素个数不受限制的集合类Set

分析：

数据成员：链表的头指针、元素个数

成员函数：

- 1、构造函数、拷贝构造函数、析构函数
- 2、操作接口：集合的操作

## OOP 练习：链表实现集合类 IntSet

`bool is_empty() const;` //判断是否为空集。

`int size() const;` //获取元素个数。

`bool is_element(int e) const;` //判断e是否属于集合。

`bool is_subset(const Set& s) const;` //判断s是否包含于集合。

`bool is_equal(const Set& s) const;` //判断集合是否相等。

`void display() const;` //显示集合中的所有元素。

`Set& insert(int e);` //将e加入到集合中。

`Set& remove(int e);` //把e从集合中删除。

`Set union2(const Set& s) const;` //计算集合的并集。

`Set intersection(const Set& s) const;` //计算集合的交集。

`Set difference(const Set& s) const;` //计算集合的差。

## OOP 练习：链表实现集合类 IntSet

```
bool IntSet::is_element(int e) const    // 判断元素是否属于集合
{
    for (Node *p=head; p!=NULL; p=p->next) // for 循环遍历单链表
        if(p->value == e) return true;
    return false;
}

Set& Set::insert(int e)
{
    if(!is_element(e))
    {
        Node *p=new Node;
        p->value = e;      p->next = head;      head = p;
        count++;
    }
    return *this;
}
```

## OOP 练习：链表实现集合类 IntSet （续）

```
IntSet IntSet::union(const IntSet& s) const // 求并集
{
    IntSet set(s); // 定义拷贝构造函数
    Node *p=head;
    while (p!=NULL) // while循环遍历单链表
    {
        if (!set.is_element(p->value))
            set.insert(p->value);
        p=p->next;
    }
    return set;
}
```



## OOP 练习：链表实现集合类IntSet（续）

```
IntSet IntSet::intersection (const IntSet& s) const // 求交集
{
    IntSet set; //空集合
    Node *p=head;
    while (p!=NULL)    // while循环遍历单链表
    {
        if ( s.is_element(p->value))
            set.insert(p->value);
        p=p->next;
    }
    return set;
}
```

## OOP 练习：链表实现集合类IntSet（续）

```
IntSet IntSet::difference (const IntSet& s) const // 求差集 当前集合-s
{
    IntSet set; // 定义拷贝构造函数
    Node *p=head;
    while (p!=NULL)    // while循环遍历单链表
    {
        if ( !s.is_element(p->value))
            set.insert(p->value);
        p=p->next;
    }
    return set;
}
```

# oop练习：变量的生存期，分析下面程序结果

```
class A
{ int m;
public:
    A(int x=10)
    { m=x;
      cout<<"cos object of m= "<<m<<endl;
    }
    void set(int x) { m=x;}
    A( A &a)
    { m=a.m;
      cout<<"copy cos"<<endl;
    }
    ~A()
    { cout<<"des object of m= "<<m<<endl;}
};
```

```
A f( A aa)
{ aa.set(20);
  return aa;
}

A a(40);
int main()
{
    A a1(30);
    A a2(a1);
    A a3;
    a3=f(a2);
    return 0;
}
```

## 练习：变量的生存期，分析下面程序结果

- cos object of m= 40 → 全局变量：A a(40)；
- cos object of m= 30 → main函数中的局部变量：A a1(30)；
- copy cos → main函数中的局部变量：A a2(a1)；
- cos object of m= 10 → main函数中的局部变量：A a3；
- copy cos → f函数中的局部变量aa：由实参a2拷贝构造而成
- copy cos → return aa；f函数中的局部变量aa拷贝构造临时对象
- des object of m= 20 → f函数调用结束，f函数中的局部变量aa 消亡
- des object of m= 20 → 完成 a3=f(a2)；临时对象消亡
- des object of m= 20 → main函数调用结束，main函数中的局部变量 a3 消亡
- des object of m= 30 → main函数调用结束，main函数中的局部变量 a2 消亡
- des object of m= 30 → main函数调用结束，main函数中的局部变量 a1 消亡
- des object of m= 40 → 程序运行结束，全局变量 a 消亡

## 练习：实现登录函数

```
struct Info{
    char name[20];
    char psw[20];
};

const int NUM = 6;

Info list[NUM] ={{"ada", "123"}, {"coco", "234"}, {"lily", "456"},
                 {"may", "111"}, {"tom", "222"}, {"jerry", "555"}}

int main()
{
    if(login()) //实现login
        cout<<"~~~~~welcome~~~~~\n";
    else
        cout<<"bye!!!\n";
    return 0;
}
```

```
You have 3 chances
input ID:coco
input password:222
try again
You have 2 chances
input ID:coco
input password:111
try again
You have 1 chances
input ID:coco
input password:222
bye!!!
```

```
You have 3 chances
input ID:ada
input password:222
try again
You have 2 chances
input ID:tom
input password:222
~~~~~welcome~~~~~
```

# 练习：指针的类型

判断下列对指针类型数据操作的对错，并指出错误的原因：

```
double x;  
int *pi;  
pi++;  
pi = &x;  
pi = &a;  
double *pf = &x;  
int a[10];  
char c;  
char *pc = &c;  
int *pa = &a;  
int ** p = new (int *)[10];
```

& 的用法：

单个&：

- 1、逻辑位操作--按位与
- 2、取地址
- 3、定义引用变量

两个&&

逻辑与操作符

```
double x;    int *pi;    pi++; //pi未初始化就自增
```

```
pi = &x; //类型不配
```

```
pi = &a; //a未定义
```

```
double *pf = &x;
```

```
int a[10]; char c; char *pc = &c;
```

```
int *pa = &a; //不需要&    int *pa = a;
```

```
int * * p = new (int *) [10]; // = new int * [10];
```

## 区分指针数组和指向数组的指针

`int * p[10]` 表示有10个元素的数组p，每个元素的类型是整型指针  
`p+1`, 表示指向下一个元素

`int (*p)[10]` 表示p是一个指向有10个整型元素数组的指针，也可以视为行指针，`p+1`, 表示指向下一行

# 练习：分析程序功能

```
int func(char *s1, const char *s2)
{ while (*s1 && *s2 && *s1 == *s2)
    { s1++;
      s2++;
    }
  return (*s1 - *s2);
}
```



↑  
s2



↑  
s1



## 练习：字符串替换find\_and\_replace

► 程序流程：穷举法（默认替换串不超过被替换串）

► 从原串的第一个字符开始：

► find: 看从这个字符开始是否和被替换的字符串相同，如果相同，说明找到，否则，从原串的下一个字符开始；

► replace: 将替换串拷贝的被替换串中，如果替换串小于被替换串，还需要将后面的字符串向前移动。

► 从原串的下一个字符开始，重复find、replace

思考：

有没有效率更高的流程？

# Knuth-Morris-Pratt 算法 ( KMP 算法 )

原串：abcabc**abcdef**； 查找串：abcdef

当第四个字符d不匹配的时候，之前匹配的字符为abc，  
可以向右滑动3位，从第二个a开始进行匹配。

KMP算法可以优化find环节的效率，利用前面的匹配过程信息，进行尽可能的向右滑动，避免无谓的比较。

KMP算法的关键问题就是计算查找串的某位置匹配失败的时候，可以向右滑动的位数。

<https://www.cnblogs.com/yjiyjige/p/3263858.html#!comments>

## 练习：数组前置和、后置和

假设有一个数组 $x[]$ ，它有 $n$ 个元素，每一个都大于0；

称 $x[0]+x[1]+\dots+x[i]$ 为前置和，而 $x[j]+x[j+1]+\dots+x[n-1]$ 为后置和。

试编写一个程序，求出 $x[]$ 中有多少组相同的前置和与后置和。

例如，如果 $x[]$ 的元素为3、6、2、1、4、5、2，

则，前置和有：3、9、11、12、16、21、23；

后置和有：2、7、11、12、14、20、23；

11、12、23这3对就是值相同的前置和与后置和，

因为：11=3+6+2（前置和）=2+5+4（后置和），

12=3+6+2+1（前置和）=2+5+4+1（后置和），

23是整个数组元素的和。

## 练习：前置和、后置和

分析: a数组的元素:

3	6	2	1	4	5	2
---	---	---	---	---	---	---

p1 ↑ p2 ↑

两种思路：

(1) 穷举：将所有的前置和、后置和计算出来，逐个比较；

(2) 利用两个下标变量，循环计算前置和、后置和

每次循环判断当前的前置和、后置和;

若：前置和=后置和，计数器+1；

前置和<后置和, p1++, 前置和+=a[p1];

后置和<前置后, p2--,后置后+=a[p2];

## 练习：函数的参数传递

```
void Swap(int x, int *y, int & z)
{
    int temp = x;
    x= *y ;
    *y = z ;
    z = temp ;
}
```

```
int main( )
{
    int i = 1, j = 2 , k = 3;
    Swap(i, &j, k);
    cout<<"i="<<i<<"    j="<<j<<"    k="<<k<<endl;
    return 0;
}
```

## 练习：静态局部变量，分析下面程序结果

```
#include <iostream>
using namespace std;
int f()
{   static int s=1;
    s=(7*s+19)%3;
    return s;
}
```

```
int main( )
{ int m;
  m = f( );
  cout<<"1、 m="<<m<<endl;
  m = f( );
  cout<<"2、 m="<<m<<endl;
  m = f( );
  cout<<"3、 m="<<m<<endl;
  return 0;
}
```