

	语文	数学	英语
张三	100	100	100
李四	90	50	100
王五	60	70	80

- 1、创建二维数组，3行3列
- 2、统计考试成绩，让每行的3列相加，统计出总和

	0	1	2
0	<code>arr[0][0]</code> 1	<code>arr[0][1]</code> 2	<code>arr[0][2]</code> 3
1	<code>arr[1][0]</code> 4	<code>arr[1][1]</code> 5	<code>arr[1][2]</code> 6

实现一个加法函数，功能是 ， 传入两个整型数据，计算数据相加的结果，并且返回

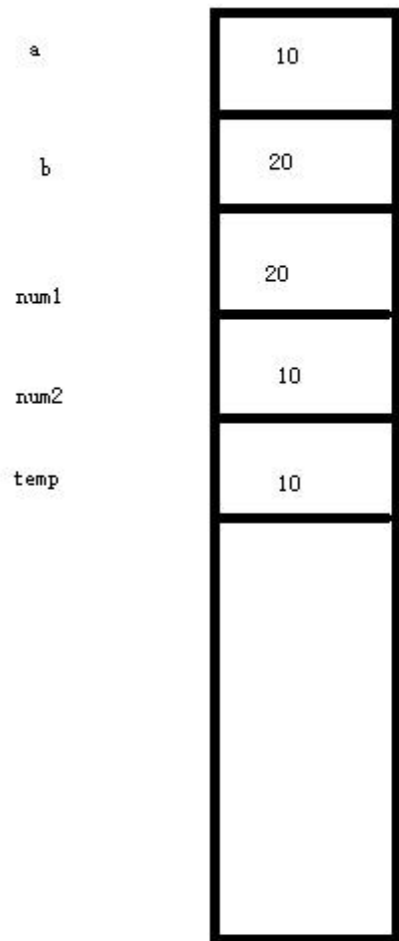
- 1、返回值类型            `int`
- 2、函数名                `add`
- 3、参数列表             `(int num1 , int num2 )`
- 4、函数体语句           `int sum = num1+ num2;`
- 5、return 表达式        `return sum;`

语法：

```
返回值类型  函数名  参数列表
{
    函数体语句

    return 表达式
}
```

```
int add(int num1,int num2)
{
    int sum = num1 + num2;
    return sum;
}
```



值传递的时候，  
形参发生任何的改变  
都不会影响实参

teacher\_A



student_A	student_B	student_C	student_D	student_E
-----------	-----------	-----------	-----------	-----------

teacher\_B



student_A	student_B	student_C	student_D	student_E
-----------	-----------	-----------	-----------	-----------

teacher\_C



student_A	student_B	student_C	student_D	student_E
-----------	-----------	-----------	-----------	-----------

```
struct teacher
{
    int id;
    string name;
    int age;
    struct student stu;
}
```



```
struct student
{
    string name;
    int age;
    int score;
}
```

银行理财 6%

要求：要么存满2年以上，要么存够10万以上RMB

老师有5万存款，存3年 可以

老师有15万存款，存1个月 可以

老师有5万，存6个月 不可以



银行理财 6%

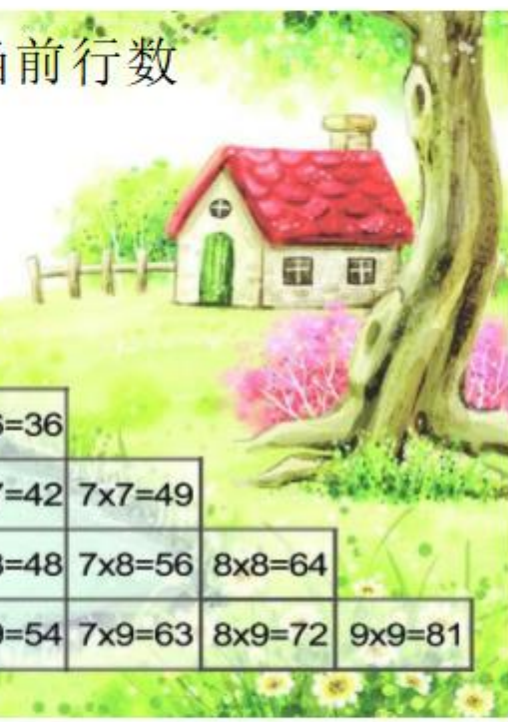
- 1、必须存满2年
- 2、必须存10万以上

逻辑与：两个条件都为真，结果才为真

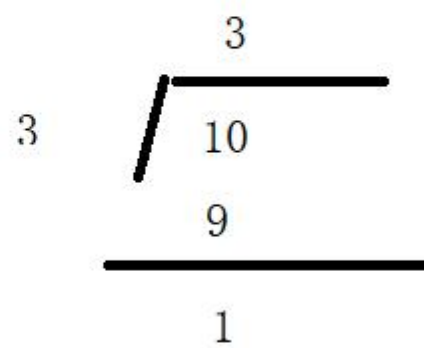
列数 \* 行数 = 计算结果

列数 ≤ 当前行数

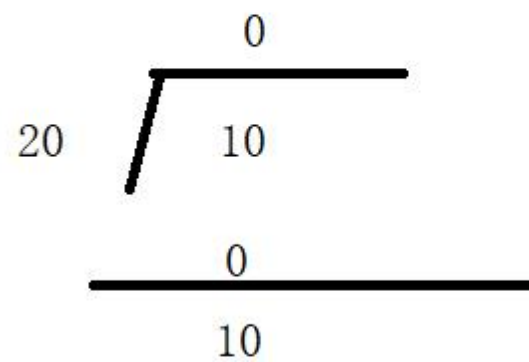
1	1x1=1								
2	1x2=2	2x2=4							
3	1x3=3	2x3=6	3x3=9						
4	1x4=4	2x4=8	3x4=12	4x4=16					
5	1x5=5	2x5=10	3x5=15	4x5=20	5x5=25				
6	1x6=6	2x6=12	3x6=18	4x6=24	5x6=30	6x6=36			
7	1x7=7	2x7=14	3x7=21	4x7=28	5x7=35	6x7=42	7x7=49		
8	1x8=8	2x8=16	3x8=24	4x8=32	5x8=40	6x8=48	7x8=56	8x8=64	
9	1x9=9	2x9=18	3x9=27	4x9=36	5x9=45	6x9=54	7x9=63	8x9=72	9x9=81
	1	2	3	4	5	6	7	8	9








$$10 \% 3 = 1$$



Handwritten long division of 10 by 3. The divisor 3 is on the left. The dividend 10 is written inside the division bar. The quotient 3 is written above the bar. The product 9 is written below the dividend 10. A horizontal line is drawn below 9, and the remainder 1 is written below that line.



Handwritten long division of 20 by 3. The divisor 3 is on the left. The dividend 20 is written inside the division bar. The quotient 0 is written above the bar. The product 0 is written below the dividend 20. A horizontal line is drawn below 0, and the remainder 10 is written below that line.

```
for(...)  
{  
      
      
      
    continue; 执行到本行，就不再执行后面的代码了，而执行下一次循环  
      
      
}
```

continue; 执行到本行，就不再执行后面的代码了，而执行下一次循环

1   xxxxxx

2   xxxxxx

     goto FLAG;

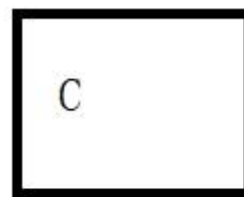
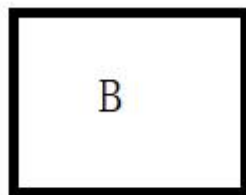
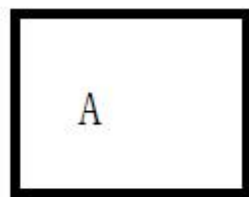
3   xxxxxx

4   xxxxxx

FLAG:



5   xxxxxx



1、先判断A和B谁重

A重	让A和C比较
A重	结果是A最重
C重	结果是C最重
B重	让B和C比较
B重	结果是B最重
C重	结果是C最重

- 1、先输出1到100 这些数字
- 2、从这100个数字中找到特殊数字，改为“敲桌子”

特殊数字

7的倍数	$(7\ 14\ 21\ 28\dots) \% 7 = 0$
个位有7	$(7, 17, 27, 37\dots) \% 10 = 7$
十位有7	$(70, 71, 72, 73\dots) / 10 = 7$

1、将所有的三位数进行输出 （100~999）

2、在所有三位数中找到水仙花数

水仙花数      153

获取个位       $153 \% 10 = 3$       对数字取模于10 可以获取到个位

获取十位       $153 / 10 = 15$        $15 \% 10 = 5$       先整除于10，得到两位数，再取模于10，得到十位

获取百位       $153 / 100 = 1$       直接整除于100，获取百位

判断       $\text{个位}^3 + \text{十位}^3 + \text{百位}^3 = \text{本身}$



```
int arr[5] = {300, 350, 200, 400, 250}
```

```
int max = 400
```



访问数组中每个元素，如果这个元素比我认定的最大值要大，更新最大值

```
int start = 0; //起始元素下标
```

```
int end = sizeof(arr)/sizeof(arr[0]) - 1; //末尾元素下标
```



0 1 2 3 4



start和end下标元素进行互换

int temp



```
int temp = arr[start];
```

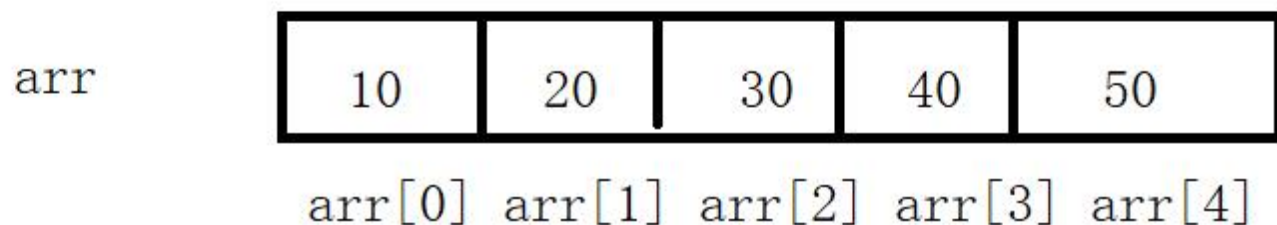
```
arr[start] = arr[end]
```

```
arr[end] = temp;
```

```
start++; end--;
```

如果start < end  
执行互换

数组特点：  
放在一块连续的内存空间中  
数组中每个元素都是相同数据类型

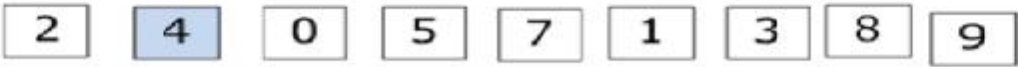


↑  
下标，我们可以通过下标访问数组中的元素

排序轮数

对比次数

0



8

1



7

2



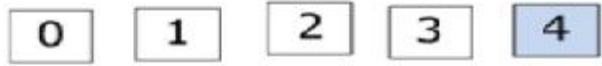
6

3



5

4



4

5



3

6



2

7



1



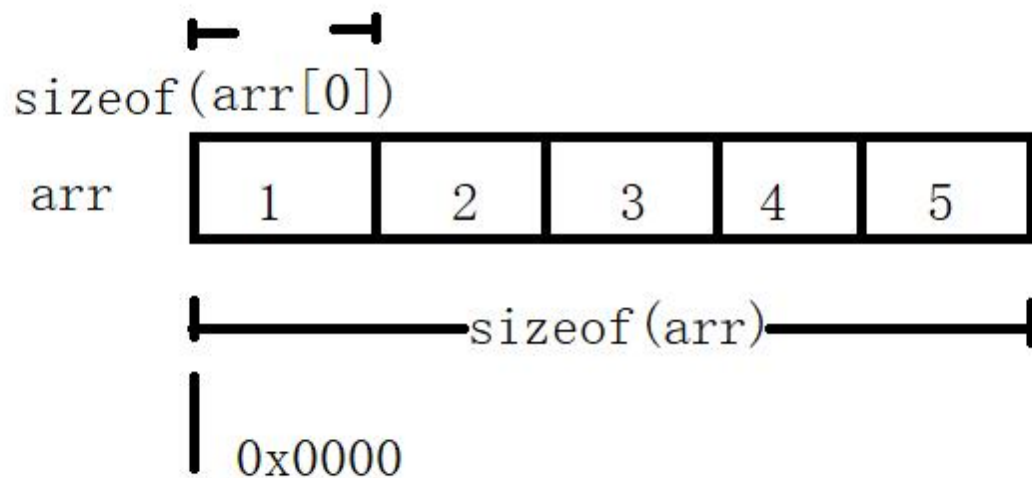
- 1. 比较相邻的元素。如果第一个比第二个大，就交换他们两个。
- 2. 对每一对相邻元素做同样的工作，执行完毕后，找到第一个最大值。
- 3. 重复以上的步骤，每次比较次数-1，直到不需要比较

排序总轮数 = 元素个数 - 1;  
每轮对比次数 = 元素个数 - 排序轮数 - 1;

一维数组名称用途:

1. 可以统计整个数组在内存中的长度 `sizeof(arr)`
2. 可以获取数组在内存中的首地址 `cout<< arr <<endl;`

```
int arr[5] = {1, 2, 3, 4, 5};
```

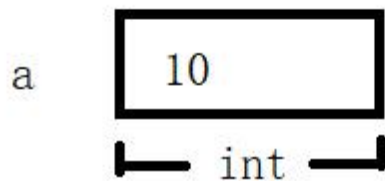


语法：数据类型    变量名 = 变量初始值

```
int a = 10;
```

数据类型存在意义：

给变量分配合适的内存空间



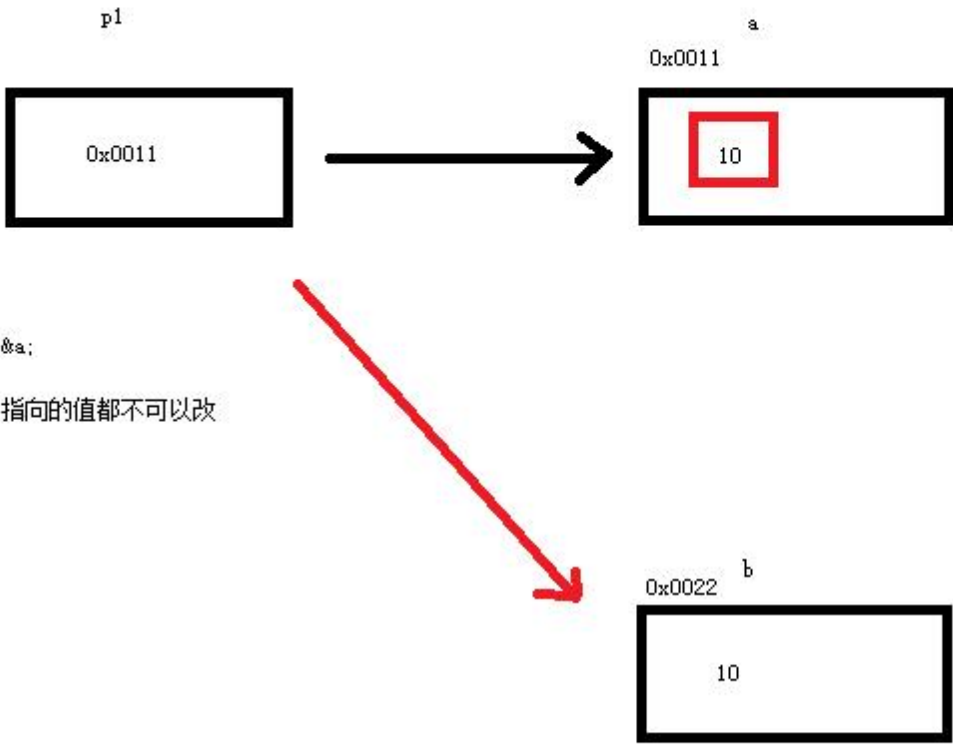
3. `const`即修饰指针，又修饰常量

```
int a = 10;
int b = 10;
int * p = &a;
```

```
const int * const p = &a;
```

特点：指针的指向和指针指向的值都不可以改

```
*p = 20; //错误
p = &b;  //错误
```



1. const修饰指针 —— 常量指针

```
int a = 10;
int b = 10;
int * p = &a;
```

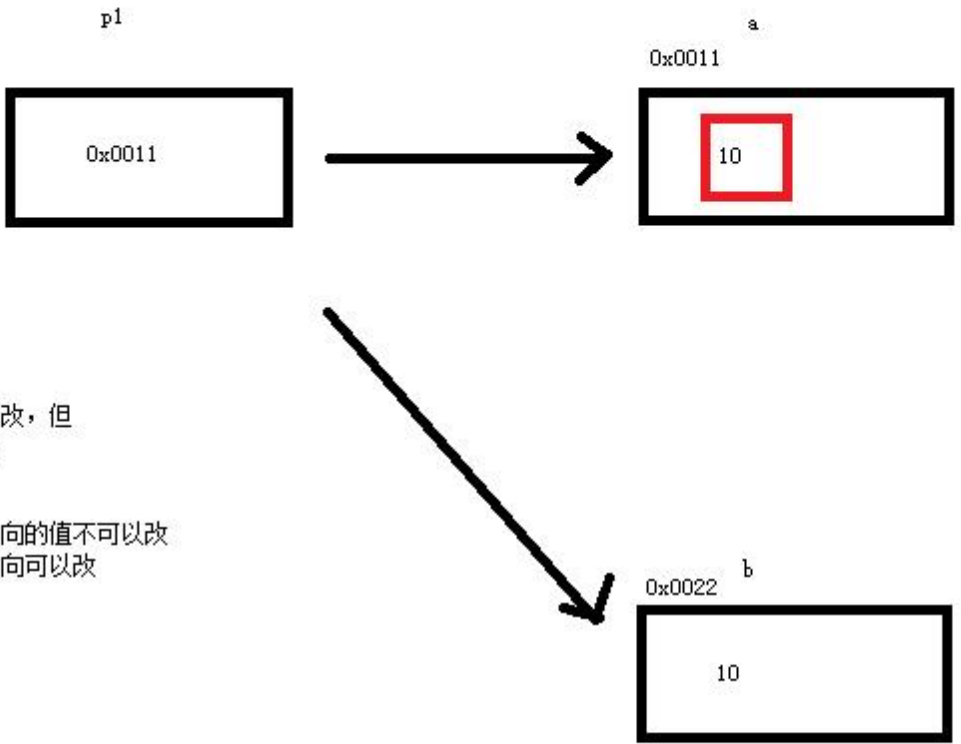
```
const int * p = &a;
```

常量指针

特点：指针的指向可以修改，但是指针指向的值不可以改

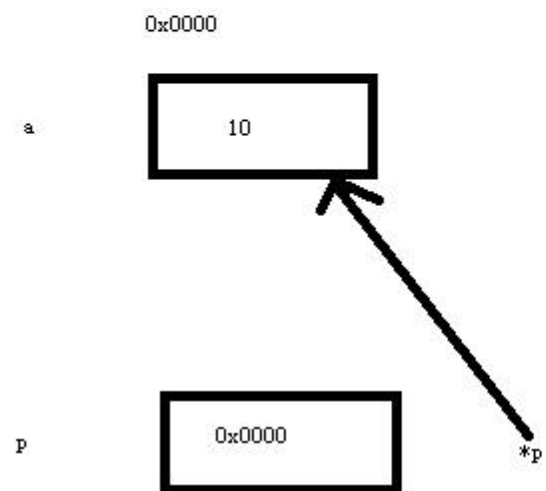
\*p = 20; 错误，指针指向的值不可以改

p = &b; 正确，指针指向可以改





```
int a = 10;
```



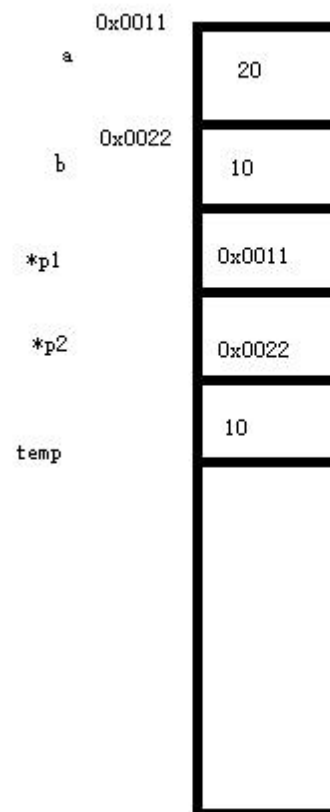
可以通过指针来保存一个地址

```
void swap02(int *p1 , int *p2)
{
    int temp = *p1;
    *p1 = *p2;
    *p2 = temp;
}
```

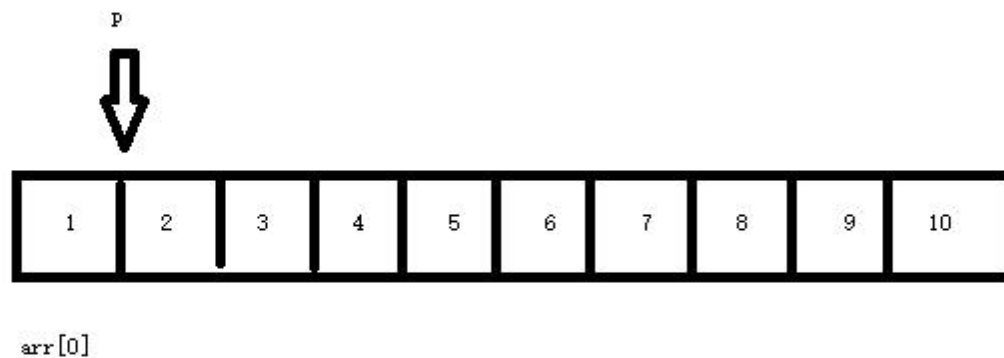
```
int a = 10;
int b = 20;
```

//2、地址传递

```
swap02(&a, &b);
cout << "a = " << a << endl;
cout << "b = " << b << endl;
```



```
int arr[10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
  
cout << "第一个元素为: " << arr[0] << endl;  
  
int * p = arr; //arr就是数组首地址  
  
cout << "利用指针访问第一个元素: " << *p << endl;  
  
p++; //让指针向后偏移4个字节  
  
cout << "利用指针访问第二个元素: " << *p << endl;
```



`int * p`

0x0000

在 32位操作系统下： 占用4个字节空间，64位下占8个字节。

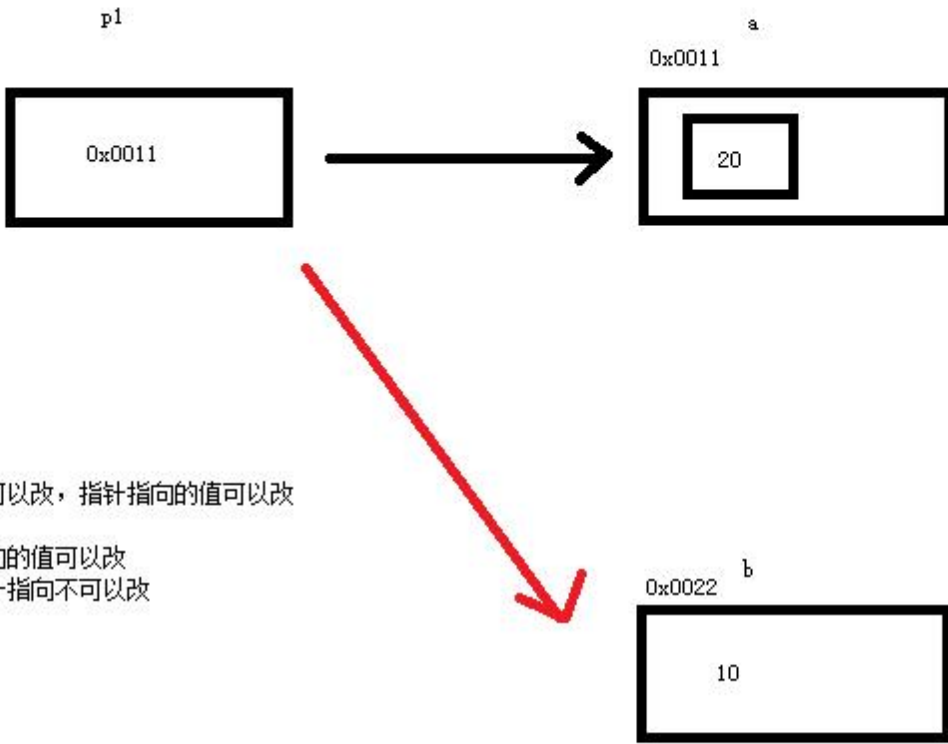
2. const修饰常里 — 指针常里

```
int a = 10;  
int b = 10;  
int * p = &a;
```

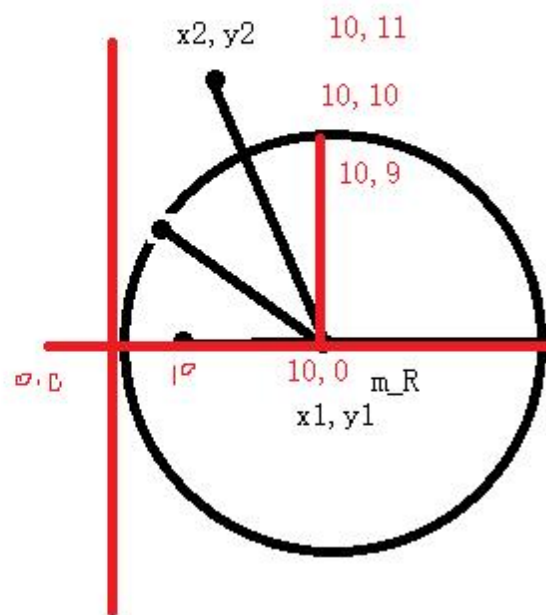
```
int * const p = &a;  
指针常里
```

特点：指针的指向不可以改，指针指向的值可以改

\*p = 20; 正确，指向的值可以改  
p = &b; 错误，指针指向不可以改



## 点和圆关系判断



点到圆心的距离 == 半径 点在圆上

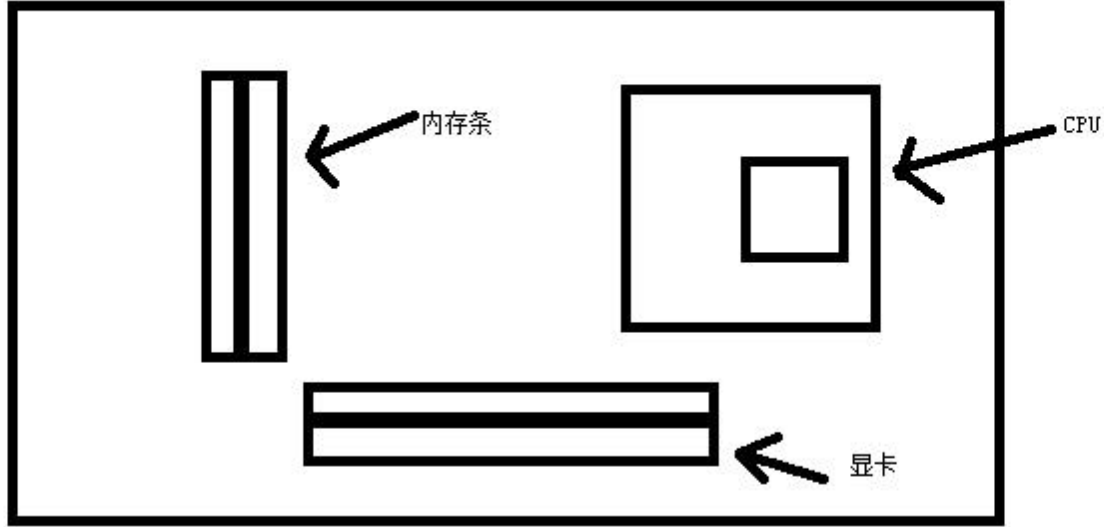
点到圆心的距离 > 半径 点在圆外

点到圆心的距离 < 半径 点在圆内

点到圆心的距离 ???

$$(x1 - x2)^2 + (y1 - y2)^2$$

和  $m_R^2$  对比



抽象出每个零件的类

```
class CPU 抽象类
{
    //抽象计算函数
    virtual void calculate() = 0;
}

class VideoCard 抽象类
{
    //抽象显示函数
    virtual void display() = 0;
}

class Memory 抽象类
{
    //抽象存储函数
    virtual void storage() = 0;
}
```

电脑类

```
class Computer
{
    构造函数中传入三个零件指针

    提供工作的函数
    {
        调用每个零件工作的接口
    }
}
```

具体零件厂商  
Inter 厂商

```
class IntelCpu :public CPU
{
    void calculate()
    {
        cout << "Intel的CPU开始计算了!"
    }
}
```

Lenovo厂商  
也需要提供三个零件

测试阶段 组装三台不同的电脑

```

class Animal
{
public:
    //虚函数
    virtual void speak()
    {
        cout << "动物在说话" << endl;
    }
};

```

```

//猫类
class Cat :public Animal
{
public:
    //重写 函数返回值类型 函数名 参数列表 完全相同
    virtual void speak()
    {
        cout << "小猫在说话" << endl;
    }
};

```

当子类重写父类的虚函数

子类中的虚函数表 内部 会替换成 子类的虚函数地址

Animal类内部结构

vfptr  
↓  
vftable 表内记录虚函数的地址

@Animal::speak

Cat 类内部结构

vfptr  
↓  
vftable

@Cat::speak

vfptr - 虚函数(表)指针

v - virtual

f - function

ptr - pointer

vftable - 虚函数表

v - virtual

f - function

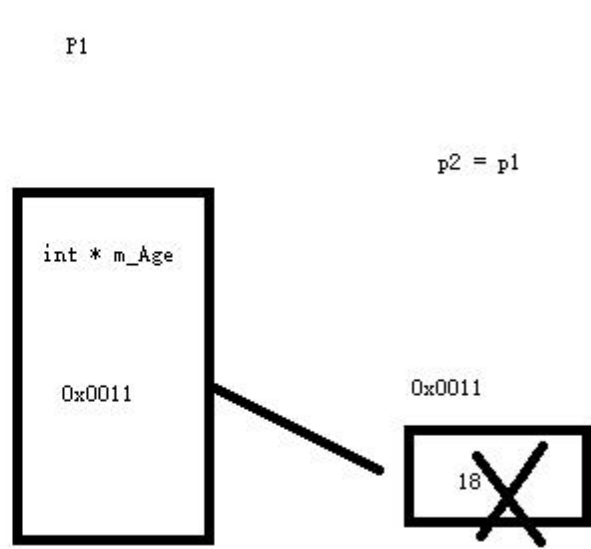
table - table

当父类的指针或者引用指向子类对象时候，发生多态

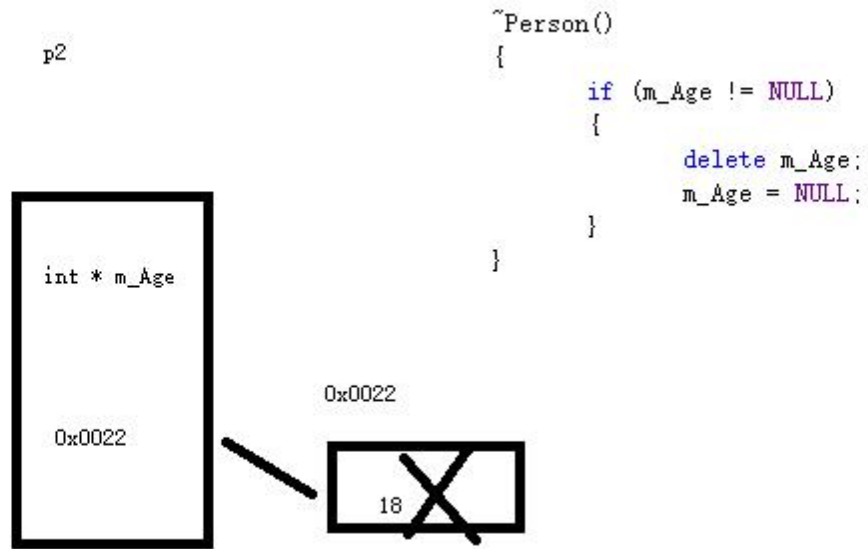
```
Animal & animal = cat;
```

```
animal.speak();
```





堆区内存重复释放  
程序崩溃！



解决方案：  
利用深拷贝 解决  
浅拷贝带来的问题

```
int a = 10;
int b = 10;

if( a == b )
{

    cout << "a 和 b相等" <<endl;

}

Person p1;
Person p2;

if( p1 == p2 )
{

    cout << "p1 和 p2 相等 " <<endl;

}

if( p1 != p2 )
{

    cout << "p1 和 p2 不相等" <<endl;

}
```

对于内置数据类型，编译器知道如何进行运算

```
int a = 10;
int b = 10;
int c = a + b;
```

```
class Person
{
public:

    int m_A;
    int m_B;
}
```

```
Person p1;
p1.m_A = 10;
p1.m_B = 10;
```

```
Person p2;
p2.m_A = 10;
p2.m_B = 10;
```

```
Person p3 = p1 + p2;
```

通过自己写成员函数，实现两个对象相加属性后返回新的对象

```
Person PersonAddPerson(Person &p)
{
    Person temp;
    temp.m_A = this->m_A + p.m_A;
    temp.m_B = this->m_B + p.m_B;
    return temp;
}
```

编译器给起了一个通用名称

通过成员函数重载+号

```
Person operator+ (Person &p)
{
    Person temp;
    temp.m_A = this->m_A + p.m_A;
    temp.m_B = this->m_B + p.m_B;
    return temp;
}
```

```
Person p3 = p1.operator+(p2);
```

简化为

```
Person p3 = p1 + p2;
```

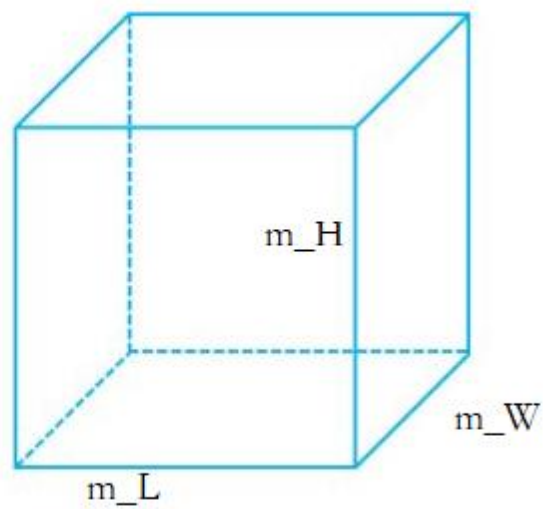
通过全局函数重载+

```
Person operator+ (Person &p1, Person &p2)
{
    Person temp;
    temp.m_A = p1.m_A + p2.m_A;
    temp.m_B = p1.m_B + p2.m_B;
    return temp;
}
```

```
Person p3 = operator+ (p1, p2)
```

简化为

```
Person p3 = p1 + p2;
```



```
class Cube  
{  
public:
```

```
    //行为
```

```
    //设置获取长宽高
```

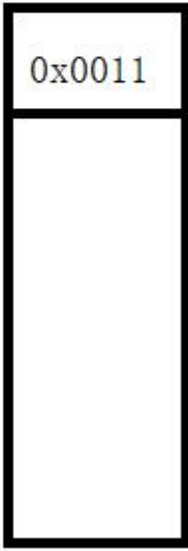
```
    获取立方体面积
```

```
    获取立方体体积
```

```
private:  
    //属性  
    m_L  
    m_W  
    m_H  
}
```

栈

int \* p



堆区

0x0011



局部变量、  
const修饰的局部变量（局部常量）



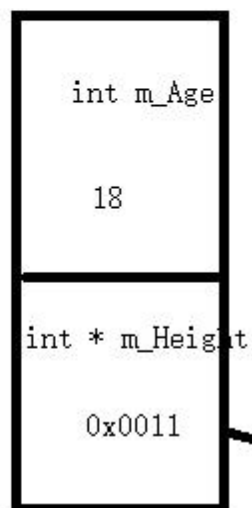
不在全局区中

全局变量  
静态变量 static关键字  
常量  
字符串常量  
const修饰的全局变量（全局常量）



全局区

p1



Person p2(p1)  
如果利用编译器提供的  
拷贝构造函数，会做浅  
拷贝操作

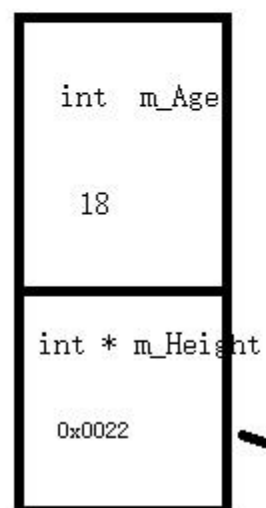
堆区

0x0011



浅拷贝带来的问题就是  
堆区的内存重复释放

p2



```
~Person()  
{
```

```
    //析构代码，将堆区开辟数据做释放操作
```

```
    if (m_Height != NULL)
```

```
    {
```

```
        delete m_Height;
```

```
        m_Height = NULL;
```

```
    }
```

```
    cout << "Person的析构函数调用" << endl;
```

```
}
```

0x0022



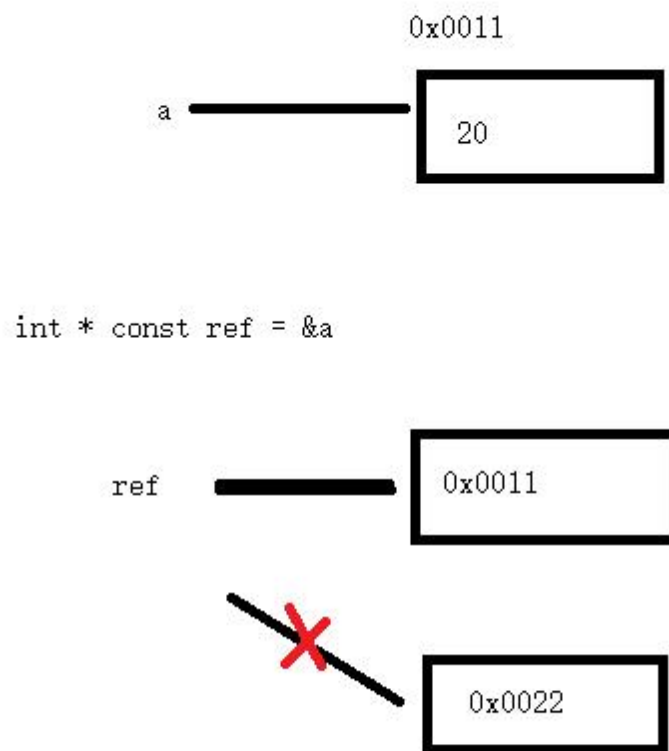
浅拷贝的问题 要利用深拷贝进行解决

```
1 //发现是引用, 转换为 int* const ref = &a;
2 void func(int& ref){
3     ref = 100; // ref是引用, 转换为*ref = 100
4 }
5 int main(){
6     int a = 10;
7
8     //自动转换为 int* const ref = &a; 指针常量是指针指向不可改, 也说明为什么引用不可更改
9     int& ref = a;
10    ref = 20; //内部发现ref是引用, 自动帮我们转换为: *ref = 20;
11
12    cout << "a:" << a << endl;
13    cout << "ref:" << ref << endl;
14
15    func(a);
16    return 0;
17 }
```

引用的本质 就是一个指针常量

引用一旦初始化后, 就不可以发生改变

←





```
int a = 10;
```

4个字节

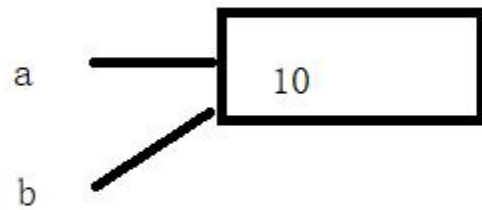
引用：给变量起别名

语法：数据类型 &别名 = 原名

```
int &b = a;
```

```
b = 20;
```

```
cout << a << endl; ??? 也是20
```

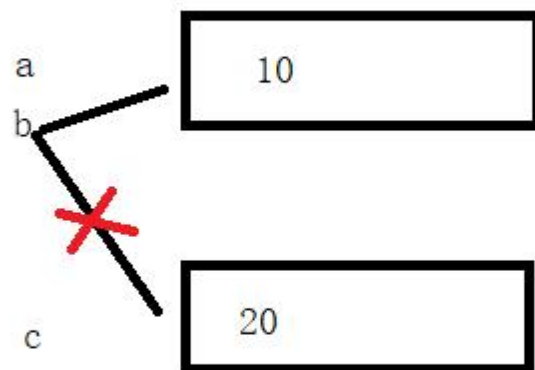


```
int a = 10;  
int &b = a;
```

1、引用必须要初始化

```
int &b; //错误的
```

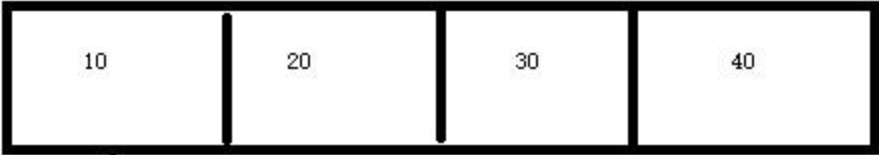
2、引用一旦初始化后，就不可以更改了



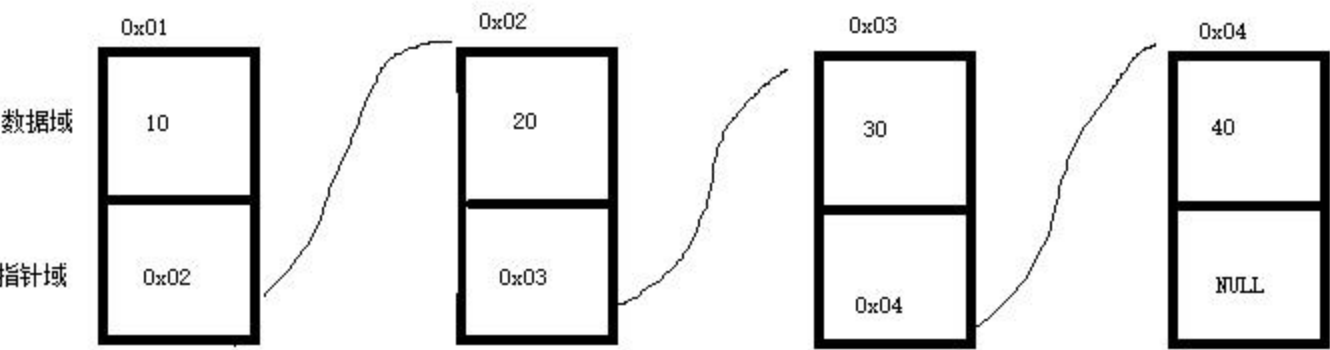
```
int a = 10;  
cout << a <<endl;
```

```
Person p;  
p.m_A = 10;  
p.m_B = 10;  
  
cout << p <<endl;
```

数组



链表

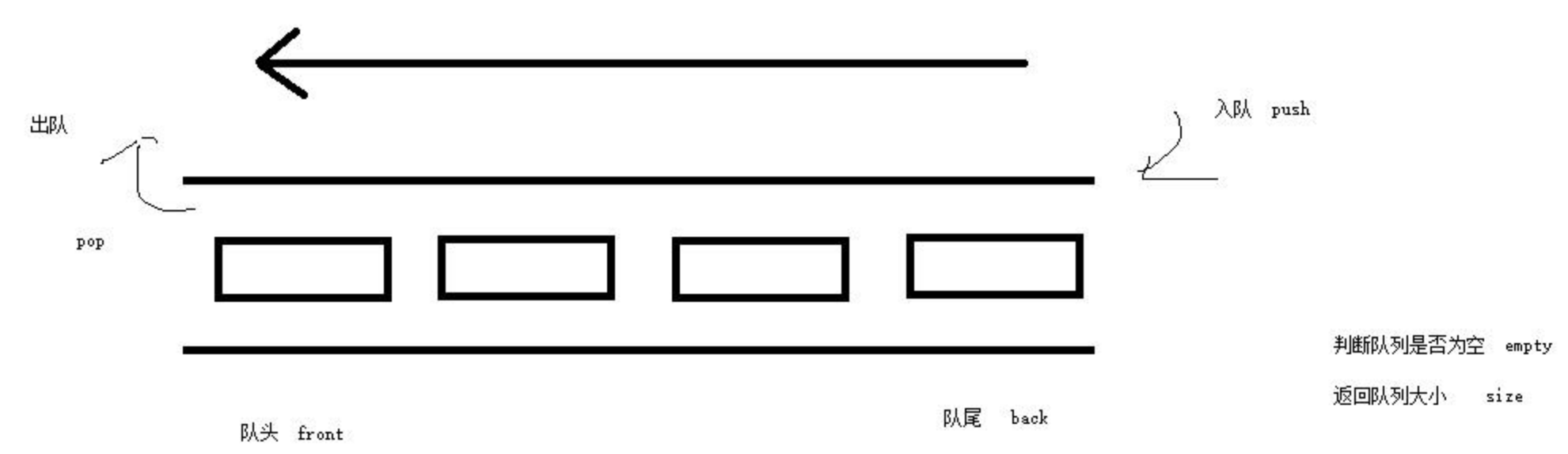


结点



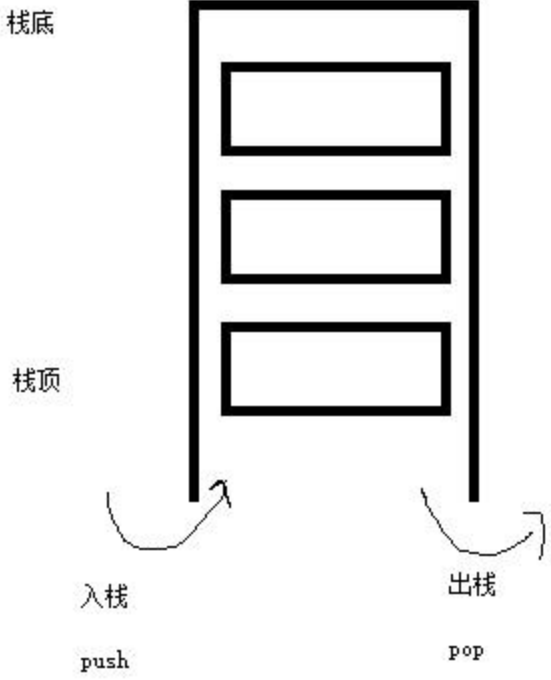
优点：  
可以对任意位置进行快速插入或删除元素

缺点：  
容器遍历速度，没有数组快  
占用空间比数组大



队列 Queue符合 先进 先出      只有队头和队尾能被外界访问，因此不允许有遍历行为

栈容器 符合先进后出



栈不允许有遍历行为

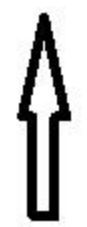
栈可以判断容器是否为空吗？ 可以 `empty`

栈可以返回元素个数吗？ 可以 `size`

10	20	30	40
----	----	----	----



v.begin()

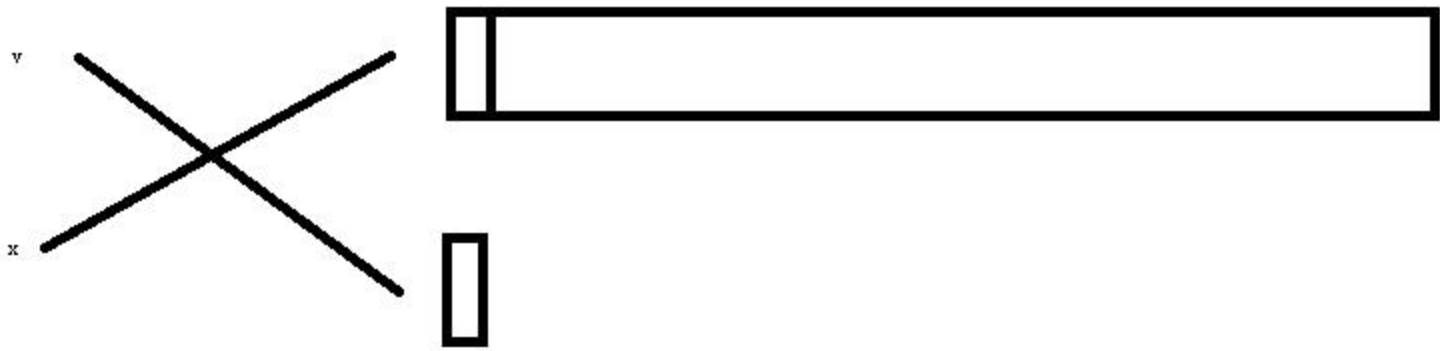


v.end()

`vector<int>(v).swap(v);`

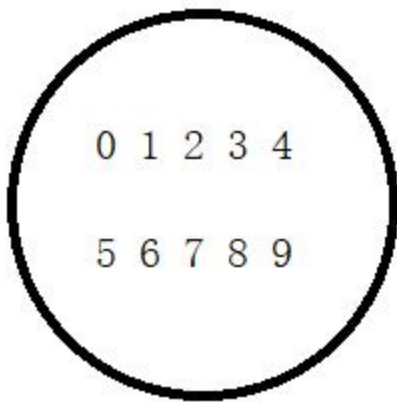
`vector<int>(v)`//匿名对象

`.swap(v);`     容器交换

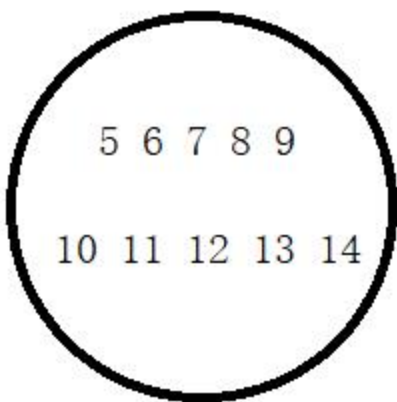




V1



V2



V1和V2容器 交集: 5 6 7 8 9

V1和V2容器 并集: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

V1和V2容器 差集: 0 1 2 3 4

V2和V1容器 差集: 10 11 12 13 14

```
class MyArray
{
private:
    //数组    T * pAddress;

    //容量    m_Capacity

    //大小    m_Size;
```

- public:
- 构造函数（容量）
- 拷贝构造
- operator=
- 利用下标的方式访问数组中的元素 ？
- 尾插法
- 尾删法
- 获取数组容量
- 获取数组大小
- 析构



```
T * = new T[5];
```