第2讲 C++的输入/输出和非面向对象的特性

目录

CONTENTS

1 由C结构体到C++类

2 简单的C++程序

3 命名空间

4 输入输出

5 动态内存分配*

6 引用

7 内联函数

8 带默认参数值的函数

9 const修饰符

1 C语言结构体与C++语言结构体

```
#include <stdio.h>

struct Employee
{
    int     m_nID
};

int main()
{
    Employee Employee1;
    Employee1.m_nID = 1001;

    printf("员工编号
    %d\n",Employee1.m_nID);
    return 0;
}
```

```
#include <iostream>
using namespace std;
struct Employee
{
    int     m_nID;
     void setID(int id){m_nID=id;}
};
int main()
{
    Employee Employee1;
    Employee1.setID(1001);
    Employee1.m_nID=1002;
    cout<<"员工编号
    "<<Employee1.m_nID<<"\n";
    return 0;
}
```

两者的区别

- C的结构体内不允许有函数存在，C++允许有成员函数。
- C的结构体对成员变量的访问权限只能是public，而C++允许public, protected, private三种。
- C语言的结构体是不可以继承的，C++的结构体是可以从其他的结构体或者类继承过来的。

1 C++结构体与C++类

```
#include <iostream>
using namespace std;
struct Employee
{
    int     m_nID;
    void setID(int id){m_nID=id;}
};

int main()
{
    Employee Employee1;
    Employee1. setID(1001);
     Employee1.m_nID=1002;
    cout<<“员工编号
    “,Employee1.m_nID)<<"\n";
    return 0;
}
```

```
#include <iostream>
using namespace std;
class Employee
{
public:
    int     m_nID;
     void setID(int id){m_nID=id;}
};
int main()
{
    Employee Employee1;
    Employee1.setID(1001);
    Employee1.m_nID=1002;
    cout<<"员工编号
    "<<Employee1.m_nID<<"\n";
    return 0;
}
```

两者的区别

- C++结构体成员变量及成员函数默认的访问级别是public，而C++类的内部成员变量及成员函数的默认访问级别是private。
- C++结构体的继承默认是public，而C++类的继承默认是private。

2 简单的C++程序

```
//simple.cpp
#include <iostream>

int main()
{
    using namespace std;
    int a;   //声明变量
    a = 10; //变量赋值
   cout<<“变量a的值是“; //输出信息
   cout<<a;
    system(“pause”);
    return 0;
}
```

预处理

使用命名空间**std**

输出流对象“cout”和插入运算符（也称输出运算符）“<<”

等待输入

暂停

3 命名空间

- <iostream>和<iostream.h>是两个不同的文件，后缀为 .h 的头文件C++标准已明确提出不支持。
- 当使用<iostream.h>时，相当于在 C 中调用库函数，使用的是全局命名空间，也就是早期的C++实现；
- 当使用<iostream>时，该头文件没有定义全局命名空间，必须使用 using namespace std 。

3 命名空间

- 命名空间是C++的一个特性，旨在解决各种对象重名的问题，并使代码组织更加方便。
- 将同一作用域的变量、函数、类用一个自定义的空间名字区分。

3.1 定义空间

```
#include <iostream>
using namespace std;
namespace space1 {
    void f()
    {
        cout<<"I am in space 1 \n";
    }
}
namespace space2 {
    void f()
    {
        cout<<"I am in space 2 \n";
    }
}
```

```
namespace name{
    //variables, functions, classes
}
```

```
int main()
{
    space1::f();
    space2::f();
}
```

::作用域运算符，如果没有表示全局

3.2 using用法

using 声明以后的程序中如果出现了未指明命名空间的 f，就使用 space1::f

```
#include <iostream>
using namespace std;
namespace space1 {
    void f()
    {
        cout<<"I am in space 1 \n";
    }
}
namespace space2 {
    void f()
    {
        cout<<"I am in space 2 \n";
    }
}
```

```
using namespace space1;
int main()
{
    f();
}


int main()
{
    using namespace space1;
    f();
}
```

3.3 namespace嵌套用法

```
#include <iostream>
using namespace std;
namespace space1 {
    void f()
    {
        cout<<"I am in space 1 \n";
    }
    namespace space2 {
        void f()
        {
            cout<<"I am in space 2 \n";
        }
    }
}
```

```
int main()
{
    space1::space2::f();
    space1::f();
}
```

```
using namespace space1;
int main()
{
    space2::f();
    f();
}
```

3.4 同名空间合并

允许相同的空间名字存在，相同空间名会被编译器合并成一个空间名

```
#include <iostream>
using namespace std;
namespace space1 {
    void f()
    {
            cout<<"I am in space 1 \n";
    }
}
namespace space1 {
    void g()
    {
        cout<<"I am in space 1 too \n";
    }
}
```

```
#include <iostream>
using namespace std;
namespace space1 {
    void f()
    {
            cout<<"I am in space 1 \n";
    }


    void g()
    {
        cout<<"I am in space 1 too \n";
    }
}
```

4 C++的输入输出

在C中进行输入输出操作时,常使用函数printf()和scanf()。例如:

```
int i;
float f;
 .....
scanf( " %d " ， &i);
printf( " %f " ， f);
```

4 C++的输入输出

在C++中,上面的程序段可以写为:

```
int  i;
float  f;
  …
cin >> i;
cout << f;
```

4.1 C++的输出

- cout是输出流对象，在程序中用于代表标准输出设备，通常指屏幕。

- 运算符<<表示将右方变量的值写到输出流对象cout中，即显示在屏幕上。

- 运算符<<允许用户连续输出一连串数据,也可以输出表达式的值：

```
cout<<a+b<<c;
```

cout例子

```
#include <iostream>
using namespace std;

int main()
{
   cout<<123<<"\n";
   cout<<12.3<<456<<endl;
   return 0;
}
```

endl 是C++标准库的 输入输出操纵子，用于C++流对象，表示换行。

endl到底是什么？
如何配合cout？

4.2 C++的输入

- cin是输入流对象，在程序中用于代表标准输入设备，通常指键盘。

- 运算符>>表示将从输入流对象(即键盘)读取的数值传送给右方指定的变量。

- 运算符>>允许用户连续输入一连串数据：

```
cin>>a>>b>>c;
```

要求输入时用空白符分隔 (空格、回车或Tab键)

cin例子

```
#include <iostream>
using namespace std;

int main()
{
   int m,n;
   cin>>m>>n;
   cout<<m<<n<<endl;
   return 0;
}
```

cin例子

```
#include <iostream>
using namespace std;
int main()
{
   char c[10];
   cin>>c;
   cout<<c<<endl;
   return 0;
}
```

如果输入内容为123 456，输出结果？

C++输入/输出操纵子

操纵符	I/O	含义
endl	O	输出新的一行，等于“\n”
ends	O	输出结尾值，等于“\0”
flush	O	强制立即执行输出操作
dec	I/O	十进制I/O
hex	I/O	十六进制I/O
oct	I/O	八进制I/O
setbase(int i)	O	**以i为基数输出（i=0,8,10,16, 0为缺省值）**
setfill(char c)	I/O	用c填满所设栏宽的空位
setw(int w)	I/O	设置数据流宽度为w
setprecion(int pr)	I/O	设置小数点后的位数为pr

【例】操作符hex、dec和oct的使用

```
#include <iostream>
using namespace std;
int main()
{
    int i = 20;
    cout<<hex<<i<<' '<<dec<<i<<' '<<oct<<i<<endl;
    return 0;
}
```

程序的运行结果如下：

```
14 20 24
```

【例】操作符

```
#include <iostream>
#include <iomanip>
using namespace std;
void main(void)
{
    cout << setfill('$') << setw(10) << 10 << endl;
    cout << setfill('#') << setw(10) << 10 << endl;
}
```

输出结果是:

```
$$$$$$$$10
########10
```

【例】精度控制

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    double a = 345.67,b=1.23;
    cout<<1.23456789<<endl;
    cout<<setprecision(2)<<a<<endl;
    cout<<setiosflags(ios::fixed)<<setprecision(1)<<a<<endl;
    return 0;
}
```

setprecision(n)可控制输出流显示浮点数的数字个数。
如果与setiosflags(ios::fixed) 合用，可以控制小数点右边的数字个数。

程序的运行结果如下：

```
1.23457
3.5e+002
345.7
```

5 动态内存分配与释放*

- C语言使用函数malloc()和free()动态分配内存和释放动态分配的内存。
- C++使用运算符new和delete更好、更简单地进行内存的分配和释放。

malloc和new的区别是什么？

new运算符使用说明

在C++中，new运算符的使用方法有如下三种类型：

（1）**new 类型名**

（2）**new 类型名(初值)**

（3）**new 类型名[表达式]**

示例

```
//分配一个整数内存空间，该空间存储的初始值为5
int *p = new int(5);

//分配一个整数内存空间，但没有进行初始化
int *q = new int;

//分配10个整数的内存空间（r[0]～r[9]），并返回该
  空//间的首地址(r = &r[0])
int *r = new int[10];
```

delete运算符使用说明

delete运算符用于释放new运算符所分配的空间，以使得该空间能够被重新使用。delete运算符的使用方法有两种形式：

（1）**delete 指针变量名**

（2）**delete []指针变量名**

其中第2种形式用于释放数组对象空间。

6 引用

引用是C++很有特色的一个概念。简单来说，引用就是另一个变量的别名；也就是说，引用和它所指的变量是同一个实体。

引用的主要用途是作为函数的参数和返回值使用，在作为参数方面，它可以起到与指针参数相同的作用，但使用较指针参数要简便。

6.1 引用的声明

声明一个引用的格式如下:

类型 & 引用名 = 已定义的变量名;

例如:

```
int i = 5;
int &j = i;
```

```
#include<iostream>
using namespace std;
int main()
{ int i;
  int &j=i;
  i=30;
  cout<<"i="<<i<<" j="<<j<<"\n";
  j=80;
  cout<<"i="<<i<<" j="<<j<<"\n";
  cout<<"变量i的地址:"<<&i<<"\n";
  cout<<"引用j的地址:"<<&j<<"\n";
  return 0;
}
```

程序执行结果如下:

```
i=30 j=30
i=80 j=80
变量i的地址:0012FF7C
引用j的地址:0012FF7C
```

对变量声明一个引用，编译系统不给它单独分配存储单元，i和j都代表同一变量单元。

6.2 引用的使用

(1)在声明一个引用时，必须立即对它进行初始化，即声明它代表哪一个变量，不能声明完成后再赋值。

例如下述声明是错误的。

```
int i;
int &j;  //错误
j=i;
```

应该是：

```
int i;
int &j=i;
```

6.2 引用的使用

(2) 为引用提供的初始值，可以是一个变量或另一个引用。

例如:

```
int i=5;          //定义整型变量i
int &j1=i;     //声明j1是整型变量i的引用(别名)
int &j2=j1;    //声明j2是整型引用j1的引用(别名)
```

这样定义后，变量i有两个别名：j1和j2。

6.2 引用的使用

(3) 指针是通过地址间接访问某个变量，需要书写间接运算符“*”；

引用是通过别名直接访问某个变量。每次使用引用时，可以不用书写间接运算符“*”，因而使用引用可以简化程序。

```
#include<iostream>
using namespace std;
int main()
{
  int i=15;     //定义整型变量i,赋初值为15
  int *iptr=&i; //定义指针变量iptr,将变量i的地址赋给iptr
  int &rptr=i;  //声明变量i的引用rptr,rptr是变量i的别名
  cout<<"i is "<<i<<endl;           //输出i的值
  cout<<"*iptr is "<<*iptr<<endl;   //输出*iptr的值
  cout<<"rptr is "<<rptr<<endl;     //输出rptr的值
  return 0;
}
```

6.2 引用的使用

(4) 引用在初始化后不能再被重新声明为另一个变量的引用(别名)。

例如:

```
int i,k;      //定义i和k是整型变量
int &j=i;     //声明j是整型变量i的引用(别名)
j=&k;         //错误,企图重新声明j
              //是整型变量k的引用(别名)
```

6.2 引用的使用

(5)引用符号&仅在声明引用时出现。其他场合使用的“&”都是地址操作符。

例如:

```
int j=5;
int& i=j;        //声明引用i，“&”为引用符号
i=123;           //使用引用i，不带引用符号
int *pi=&i;     // 在此“&”为地址操作符
cout<<&pi;  // 在此“&”为地址操作符
```

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

关于引用的描述中，错误的说法是

A 引用可作为函数参数

B 引用可作为函数返回值

C 引用可先声明后赋值

D 同一变量可有多个引用

提交

6.3 引用的用途

- 建立引用的目的是为某一个变量起一个别名，然后通过操作该别名，来修改其所代表的变量的值。
- 引用的主要用途是作函数的参数以及函数的返回类型。
- 在作为函数参数方面，它可以起到与指针参数相同的作用，但其使用更简便。

（1）引用作为函数参数

① 用变量做函数参数
② 用指针做函数参数
③ 用引用做函数参数

```
void swap(int m,int n)     //变量做函数参数
{
    int temp;
    temp=m; m=n;  n=temp;
}
int main()
{
    int a=5,b=10;
    cout<<"a="<<a<<"b="<<b<<endl;
    swap(a,b);
    cout<<"a="<<a<<"b="<<b<<endl;
    return 0;
}
```

```
void swap(int* m,int* n)  //指针做函数参数
{
    int temp;
    temp=*m;  *m=*n; *n=temp;
}
int main()
{
    int a=5,b=10;
    cout<<"a="<<a<<"b="<<b<<endl;
    swap(&a,&b);
    cout<<"a="<<a<<"b="<<b<<endl;
    return 0;
 }
```

```
void swap(int& m,int& n) //引用做函数参数
{
    int temp;
    temp=m;  m=n; n=temp;
}
int main()
{
    int a=5,b=10;
    cout<<"a="<<a<<"b="<<b<<endl;
    swap(a,b);
    cout<<"a="<<a<<"b="<<b<<endl;
    return 0;
 }
```

（2）使用引用返回函数值

使用引用可以返回函数值，采用这种方法可以将该函数调用放在赋值运算符的左边。

```
#include<iostream>
using namespace std;
 int a[ ]={1,3,5,7,9};
 int& index(int i)
 {   return a[i] ;}

 int main()
 {
     cout<<index(2)<<endl;
     index(2)=25;
     cout<<index(2);
     return 0;
 }
```

6.4 引用和指针的区别

- 引用和指针都可以通过一个变量访问另一个变量，但访问时的语法形式不同。引用采用的是直接访问形式，而指针采用的是间接访问形式。
- 当作为函数参数使用时，引用所对应的实参是某个变量名，而指针所对应的实参是某个变量地址。引用在作为函数参数使用时，其效果与指针相同，但使用更方便。
- 引用在定义时被初始化，其后不能被改变（即不能再成为另一个变量的别名）；而指针则可以再通过赋值的方式，指向另一个变量。

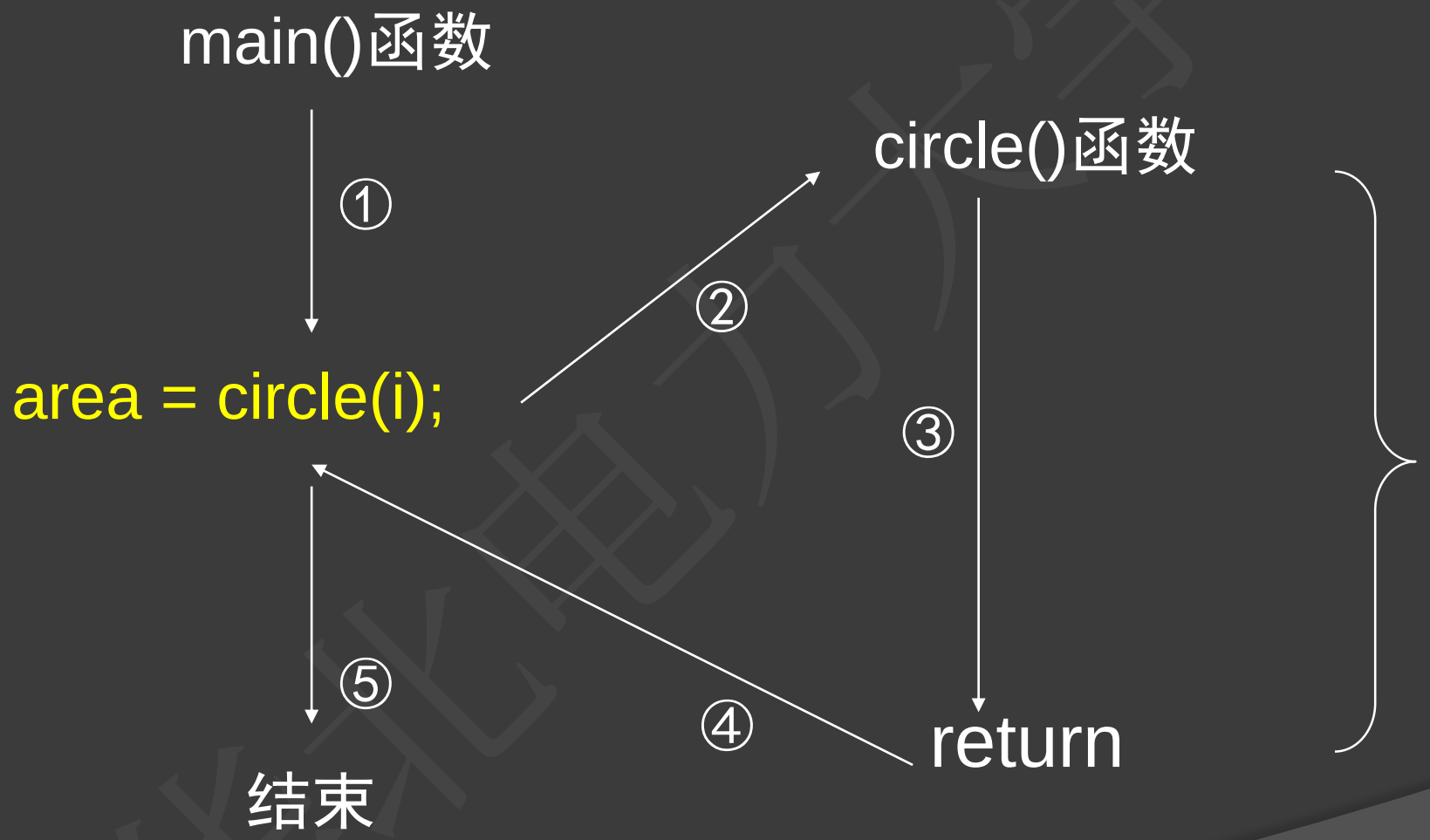
7 内联函数

- 内联函数又称为内置函数。当程序中出现对内联函数的调用时，C++编译器直接将函数体中的代码插入到调用该函数的语句处，同时用实参来代替形参。

- 使用内联函数的好处是减少了函数调用所产生的额外开销，可以提高程序运行的效率。通常情况下，对于要频繁调用的函数，如果其函数体中的代码很短，可以将其定义为内联函数。

```
#include<iostream>
 using namespace std;
 inline float circle(float r)
 {  return 3.1416*r*r; }
 int main()
 {
    float area;
    for (int i=1;i<=3;i++)
    {
        area = circle(i); //调用内联函数
        cout<<"r="<<i<<"area=" << area <<endl;
     }
     return 0;
  }
```

7.1 普通函数和内联函数的区别



普通函数调用过程示意图

内联函数的调用过程

每当程序中出现对内联函数的调用时，
C++编译器使用函数体中的代码替代函数调用表达式:

area = circle(i);

用area=3.1416*i*i替代

这样能加快代码的执行，减少调用开销。

7.2 内联函数的注意事项

- 在内联函数的函数体中，不能有复杂的控制语句，如for语句和switch语句等。

- 使用内联函数时，其函数体在使用位置被直接展开，这是一种以空间换时间的手段。如果内联函数较长，调用的位置又很多，就会大大加长程序代码的长度，造成大的空间开销。因此，只有很短（如1～5条语句）且调用频繁的函数才考虑定义为内联函数。

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

关于内联函数，下列说法错误的是

A 减少了函数调用所产生的时间开销

B 提高程序运行的效率

C 频繁调用的函数可将其定义为内联函数

D 减少了程序代码的空间开销

提交

8 带有默认参数的函数

- 函数调用通常要传递一个特定的参数值。
- 程序员可把该参数指定为默认参数，并且可以为该参数提供默认值。当函数调用中省略某一实参数时，默认参数值自动作为相应参数值传递给被调用函数。
- 在C++中，允许在函数声明或函数定义时给函数的形参指定默认值。
- 在进行带默认参数函数的调用时，如果给出了实参，则将实参传递给形参；如果省略了实参，则将默认值传递给实参。

8 带有默认参数值的函数

函数原型说明为：

int special(int x=5，float y=5.3)；

以下的函数调用都是允许的:

special( )；　　// x=5，　y=5.3

special(25)；　　// x=25，　y=5.3

special(100，79.8)；　// x=100，y=79.8

8.1 注意事项

(1)在声明函数时,所有指定默认值的参数都必须出现在不指定默认值的参数的右边,否则出错。

例如：

```
int fun(int i, int j=5, int k);
```

可改为：

```
int fun(int i, int k, int j=5);
```

```
#include <iostream>
using namespace std;
void init ( int x=5,int y=10);
int main()
{
    init(100,80);  //x=100，y=80
    init(25);      //相当于init(25,10)，结果为x=25,y=10
    init();        //相当于init(5,10)，结果为x=5,y=10
    return 0;
}
void init(int x,int y)
{
    cout<<"x: "<<x<<"\t y: "<<y<<endl;
}
```

8.1 注意事项

(2) 在函数调用时，若某个参数省略，则其后的参数皆应省略而采用默认值。

也就是说，不允许某个参数省略后，再给其后的参数指定参数值。

special(, 21,5); 

8.1 注意事项

(3)在函数原型中默认参数可以不包含参数的名字。

```
#include<iostream>
 using namespace std;
 void write( int =5);
 void main( )
 {
   write( );
 }
 void write(int a)
 { cout<<a; }
```

8.1 注意事项

(4)默认值只能出现一次，如果定义的函数在函数调用之后，则在调用位置之前必须给出函数声明。这时对形参的默认值的指定在函数声明中进行，而在后面的函数定义说明中不能再次给出默认值。

如果函数的定义在函数调用之前，则直接在函数定义的函数说明中给出带默认参数形参的默认值。

```
void DisplayInfo(int nLanType = 0);
int main()
{……
   DisplayInfo();
   ……
}
void DisplayInfo(int nLanType)
{……
}
```

单选题 2分

设置

此题未设置答案，请点击右侧设置按钮

C++中，下面设置的默认参数正确的是

A void fun(int x=0, int y, int z)

B short fun(int x=0, int y=0, int z)

C float fun(int x=0, int y, int z=0)

D double fun(int x, int y=0, int z=0)

提交

9 const关键字*

- const关键字对变量进行修饰，该变量的值不可修改的。
- const用于修饰函数参数时，说明该函数参数不能被该函数所修改。

9.1 const修饰变量

```
#include <iostream>
using namespace std;
int main()
{
    const int a = 10; //必须赋初值
    int const b = 20;
    //错误 a = 20;
    //错误 b = 10;
    return 0;
}
```

9.2 常量指针和常指针

- 常量指针：指针所指向内容为常量

```
const int *np = new int(5);
*np=6;//错误
```

- 常指针：指针内容为常量

```
int* const np = new int(5);
np=&a;//错误
```

- 指向常量的常指针：两者都为常量

```
const int* const np = new int(5);
```

9.3 const修饰函数参数

指向常量的指针

常引用

```
#include <iostream>
using namespace std;
double sub(const double *dp, const double &rd)
{   // *dp = 100.5;  //不能修改指向常量的指针所指的内容
    // rd = 12.7;    //不能修改常引用所指变量的值
    return *dp - rd;
}
int main()
{   double d1, d2;
    d1 = 33.3;
    d2 = 11.1;
    cout<<d1<<'-'<<d2<<'='<<sub(&d1, d2)<<endl;
    d1 = 44.4;
    d2 = 33.3;
    cout<<d1<<'-'<<d2<<'='<<sub(&d1, d2)<<endl;
    return 0;
}
```