**void指针，**

* **能够接受任何数据类型的特殊指针。**
* **使用void\*指针之前，必须显式地将它转换成某种数据类型的指针后使用**

cout**<<\*(double\*)pv**; //L5：正确，输出9

**begin && end 指针**

<iterator>头文件的两个函数，用于确定指向数组首元素和尾元素后一位置的指针，方便遍历数组。

int a[] = { 1,2,3,4,5,6,7,8,9,10 };

for (int \*p = **begin**(a); p != **end**(a); p++)

cout << \*p << ",";

cout << endl;

**new 和delete**

（new能够自动计算要分配的内存类型的大小，不用sizeof &&不需要进行类型转换）

p2=new int(10); //分配一个int类型大小的内存区域，并将10存入其中 **delete p2;**

**p3=new int[10]; //分配能够存放10个整数的数组区域**

if(!p3) 分配不成功

**delete []p3; //释放p3指向的数组**

**智能指针**会负责自动释放所指向的对象，无须调用delete进行回收。

* + auto\_ptr ：动态分配对象以及当对象不再需要时自动执行清理，构造时获取对某个对象的所有权，在析构时释放该对象，不能同时拥有同一个对象；

auto\_ptr< string> p1(new string("There is only one point to me."));

auto\_ptr<string> p2;

p2 = p1; //p1不再指向任何对象，其所指对象由p2指向

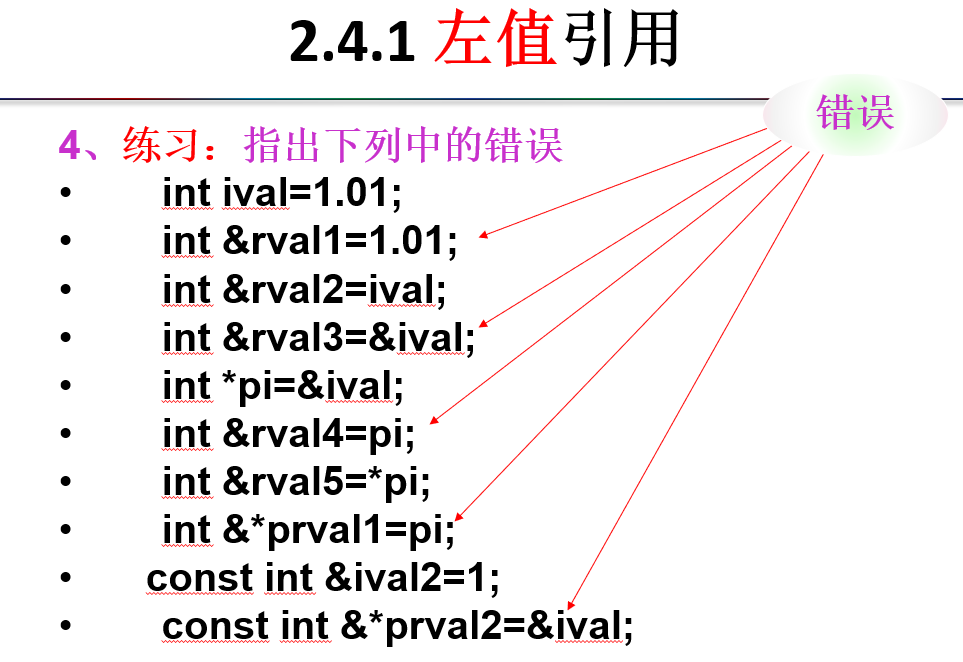
* + shared\_ptr：作用有如同[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88)，最后一个这样的指针被销毁，也就是一旦某个对象的引用计数变为0，这个对象会被自动删除。

shared\_ptr<int> p1 (new int(9));

shared\_ptr<int> p2(p1);

* + unique\_ptr：它持有对对象的独有权——两个unique\_ptr不能指向一个对象，不能进行复制操作只能进行移动操作
  + **#include<memory>**

**“左值引用”即“变量的别名” 相当于const 隐式指针**



**右值引用**

* + 其重要特点就是**只能绑定到即将销毁的对象上**，比如常量或表达式。通过右值引用可以方便地将它引用的资源“移动”到另一个对象上。
* double &lr2=r+10; //错误，引用只能是变量
* double &&rr=r+10; //正确，rr为表“r+10”计算结果，即20

const 类型 常量名称 = 常量;

constexpr 类型 常量名=常量表达式

**必须初始化、**

**const中可以有变量名，但constexpr的表达式中不能有变量**

constexpr 在编译时进行初始化，const 在运行时初始化

**const int \* const cpc=&ci; //指向常量的指针常量**

* **const对象的地址只能赋值给指向const对象的指针**

当指针本身被限定为常量时，称指针为顶层const；当所指的对象被限定为常量，而指针本身未被限定时，称指针为底层const；

**指针和所指对象两者都被限定为常量时，则指针为顶层const，所指对象为底层const。**

**顶层const其实是指不可被修改的常量对象**

**所有声明为const的引用都是底层const**

const double d=9.0 //ic为顶层const

const int &ri = i; //ri为底层const

* **（1）复制顶层 const 不受影响。**
* **（2）底层cons的复制是受限制的。**
  + - **（1）用const限定的参数，可以接受const和非const类型的参数（非const可以转换成const)**
    - **（2）若函数参数为非const类型，则不能接受const类型的实参。**

**auto和decltype的功能**

**从变量或表达式中自动推断出数据类型**

int i;

const int \*const p=&i;

const int ic = i,&rc=ic;

auto pc = &ic; // const int \*pc=&ic,对常量对象取地址是底层const

auto rrc = rc; //int rrc，**忽略引用的const**

auto ric = ic; //int ric, **忽略顶层const**

auto pp = p; //const int \*p, 忽略顶层const

* + 用auto设置一个引用类型时，初始值中的顶层const会被保留，而引用字面常量时需要指定为const引用。
  + 一条auto**可以同时定义多个变量，但数据类型只能有一种**。
  + **在用auto定义变量时，表达式的类型是清楚而明确的**。

deltype(表达式1) 变量=表达式2；

**从表达式1的结果类型定义变量，并用表达式2的值初始化量。**

auto将对象定义为指向数组第一个元素类型的指针，decltype采用与数组完全相同的类型变定义数组。

int a[] = { 1,2,3,4,5,6,7,8,9,10 };

auto p1 = a; // 等价： int \*p1

decltype(a) p2; // 等价： int p2[10]

类型转换：

C: char \* cp = (char \*) p;

C++: char \* cp = char \* (p);

**C++中强制类型转换的几种形式**

**cast-name<type>(expression);**

其中的cast-name可以为：

**static\_cast :静态转换**

**dynamic\_cast：动态转换**

**const\_cast ：常量转换**

**reinterpret\_cast：用于不相关的类型转换，如将int转换成指针等。**

例：

double b=-67.89;

int c=b;

int c=static\_cast<int>(b);

**void sqr(const &x) {**

**const\_cast<int &>(x)=x\*x; //const\_cast去掉了x的const限制后修改了x**

**}**

**函数原型描述了函数的接口**

int f(); //该函数没有参数

int f(void); //该函数不需要参数

引用作为参数传递的是实参变量本身

默认参数 int f(int i1,int i2=2,int i3=0); //正确 **（某个参数开始指定默认值，它右边的所有参数都必须指定默认。）**

**缺省参数的函数：**f(2) 则i1=2，其余未初始化的参数i2=2，i3=0；

可以用表达式作为默认参数，只要表达式可以转换成形参所需要的类型即可。但是，局部变量不能作为默认参数值。

**函数重载**

**每个函数的参数表唯一**

顶层const参数不影响实参的传入，易产生二义性

即，顶层const形参，可以接受const和非const的同类型实参

* + 底层const则是可区别的，能够正常使**用**

**用一个函数的代码替换函数调用叫内联。用inline 关键字声明,**

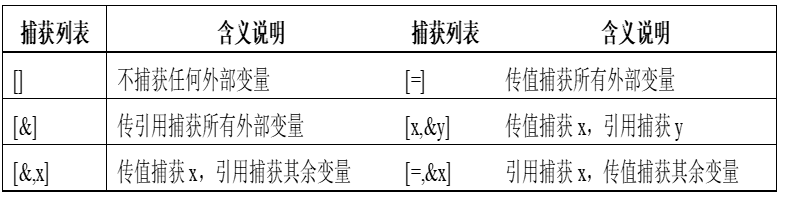
**内联函数和C中的#define相似，**

* **内联函数不能有复杂的结核.如含有for,while,switch等结构的复杂程序.**
* **递归函数是不能用来做内联函数的**

lamada表达式实质上是一种基于模板（第7章）的**匿名内联函数**，因此也称为lamada函数。

[capture](parameters) [mutable] ->return\_type {statement}

**捕获列表 形参表 取消lambda的常量性 返回类型 参数表**



**在默认情况下，lambda表达式捕获的值变量具有const特征**

**auto f2 = [a, &b](int c)mutable->int {return b += ++a + c; };**

**sort(a6, a6 + 4, [](double a, double b) { return a<b; });**

**名字空间的定义**

**namespace XX**

**{**

**members;**

**}**

**namespace\_name::identifier**

ABC::count=1; //访问ABC空间中的count

int **ABC::min**(int a,int b){

return a>b?a:b;

}

**条件编译**

**#ifdef 标识符**

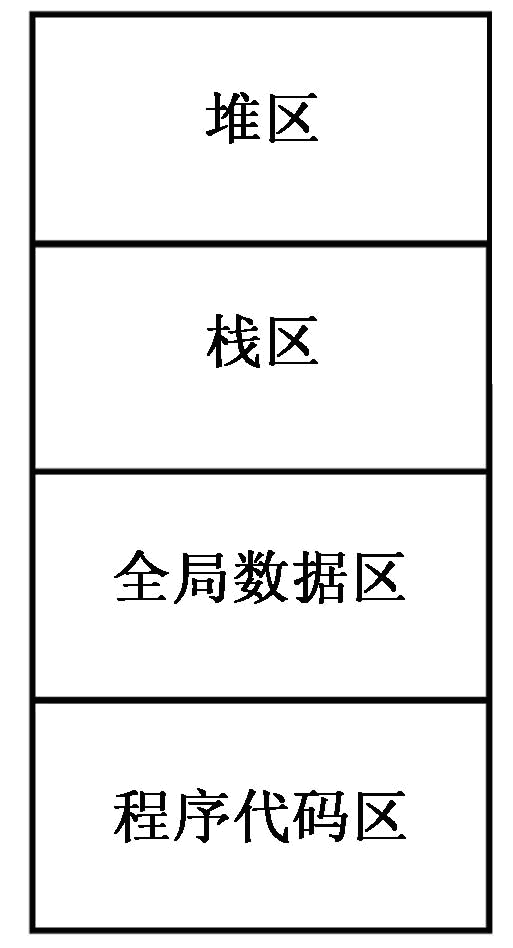
**语句组1**

**[#else**

**语句组2**

**]**

**#endif**

****

* + **堆区的数据由程序员管理，程序员可用new或malloc分配其中的存储单元给指针变量，用完之后，由程序员用delete或free将其归还系统，以便其他程序使用。**

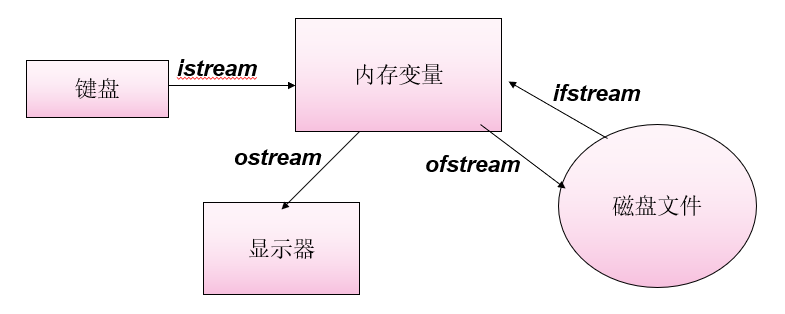
int &f1(int x){

int temp=x;

return temp;

}

函数返回的临时变量的生存期短于函数调用本身的生存期！

****

**#include<fstream>**

３、文件操作方法

（1）在程序中包含头文件fstream

#include <fstream>;

（2）定义文件流变量

ifstream inData; //定义输入文件流变量

ofstream outData; 　　　　//定义输出文件流变量

（3）将文件流变量与磁盘文件关联起来

inData.open(filename，mode)

outData.open(filename，mode)

（4）用文件流（<<或>>）读/写文件数据

inData>>x>>y;

outData<<x<<y;

（5）关闭文件

void main(){

ofstream outData("d:\\data.txt"); //在C盘根目录下建立文件data.txt

ifstream inData; 　　//定义inData为输入数据的文件

int x,a[10];

for (int i=0;i<10;i++){

cin>>x;

outData<<x<<" "; //outData将x写入文件data.txt，数据间用空白间隔

}

outData.close();

inData.open(“d:\\data.txt”);　//以读方式打开C:\data.txt文件

int j=0;

while (!inData.eof()) 　//从文件中读数据，直到遇到文件结束符

inData>>a[j++]; 　//从文件中将数据读入到数组a中

inData.close(); 　//关闭文件

int s=0;

for(int i=0;i<10;i++){

s+=a[i];

cout<<a[i]<<“ ”; 　//输出数组a，该数组中的数据来源于文件

}

**作用域限定符::**

**类与对象**

**外类定义成员函数**

r\_type class\_name::f\_Name(T1 p1, T2 p2,…);

double getSalary() const; //不能通过它修改name和salary

* + 在面向对象程序设计中**用任何数据类型定义的变量都可以称为对象**。

**构造函数：**

* 1. 构造函数与类同名，并且没有返回类型；
  2. 构造函数可以被重载；
  3. 构造函数由系统自动调用，不允许在程序中显式调用；

d.Desk(1,2,3,4); //错误，构造函数不能被显式调用

* 1. 构造函数不能被声明为const函数。

**Desk(int, int);** //构造函数声明

**Desk::Desk(int w, int h)** { //构造函数定义

width = w; length = h;

cout << "call constructor !" << endl;

}

Desk d(3, 5);

**定义对象数组或用new创建动态对象时，也要调用构造函数。但定义数组对象时，必须有不需要参数的构造函数**

Desk(){ weight=high=width=length=0;} //无参构造函数为private

**默认构造函数**

**X()=default; // 11C++ 要求编译器创建合成的默认构造函数。**

* + 在数据成员的取值比较固定时，可以通过为构造函数参数提供缺省参数初始化它们。

**Point(int a=0, int b=0) { setPoint(a, b); } //L1**

Point p1(1,1); //L2 调用point(int ,int)构造函数

**static Point p2;** //L3 调用point(a,b)，a、b 默认为0

构造函数名(参数表)：成员1(初始值),成员2(初始值),…{

……

* + }将括号中的初始值参数的值赋给该括号前面的成员。

Tdate::Tdate(int m,int d,int y):month(m),day(d) {

year=y;

cout <<month <<"/" <<day <<"/" <<year <<endl;

}**初始化列表中成员初始化次序与它们在类中的声明次序相同，与初始列表中的次序无关。**

**类内初始值→构造函数初始化列表→构造函数体**

**必须采用初始化列表（或类内初始值）进行初始化的成员**

* + **常量成员，引用成员，类对象成员，派生类构造函数对基类构造函数的调用。**

**析构函数（destructor）是与类同名的另一个特殊成员函数，作用与构造函数相反，用于在对象生存期结束时，完成对象的清理工作。**

**class X**

**{**

**~X ( ) {……};**

**}**



1. **在某些情况下，必须编写析构函数才能够完成对象销毁前的资源清理工作。常见情况是用它来释放由构造函数分配的自由存储空间。**

**拷贝构造函数**

Person p1("张勇",21);

Person p2=p1; //调用默认拷贝构造函数

**将自己同类的对象的引用作为参数进行构造函数的初始化，该构造函数就是拷贝构造函数**

**移动即转移，即把一个在内存区域中的内容转移给另一个对象**

**解决临时对象复制时的系统开销，提高效率**

//L1的执行过程：

1. 调用f函数，执行f函数体内的代码；
2. 执行“return t；”将创建无名临时对象并返回调用语句；
3. 将无名对象拷贝给对b
4. 销毁无名临时对象。

对象移动技术直接将无名对象的内存内容转移给b，省去了第③④步的系统开销

* + 把对象的内存资源（即右值）绑定到要转移给的对象。
* class A {
* ……
* A(A&& o){……} // 移动构造函数
* A &operator=(A&& o) {……} // 移动赋值运算符
* };
  + **拷贝左值，移动右值**

**静态数据成员（static data member）**

* + - **被类的所有成员所共享**
    - **与类关联，而不与特定的对象关联**
    - **即便类没有任何对象时，就已经存在**
    - **生命期与程序相同**
* **在类外定义静态数据成员时，不能加上static限定词；**

**静态成员访问**

**（**1）通过类名访问(非静态成员不具有这种访问方式)

* 类名::静态数据成员名;
* 类名::静态成员函数名(参数表);
* （2）通过对象访问
* 对象名.静态成员名;
* 对象名.静态成员函数名(参数表)

**This指针：**

this是类成员函数中用于标识调用对象自引用的隐式指针参数，**代表调用成员函数的对象自身的地址**

**哪个对象在访问成员函数，this就是该对象所在内存区域的首地址**，并且不允许修改，所以被指定为const指针

**class X{……**

**f(**X \*const this，**…） //定义函数时不定义this，由系统生成**

**};**

**X a;**

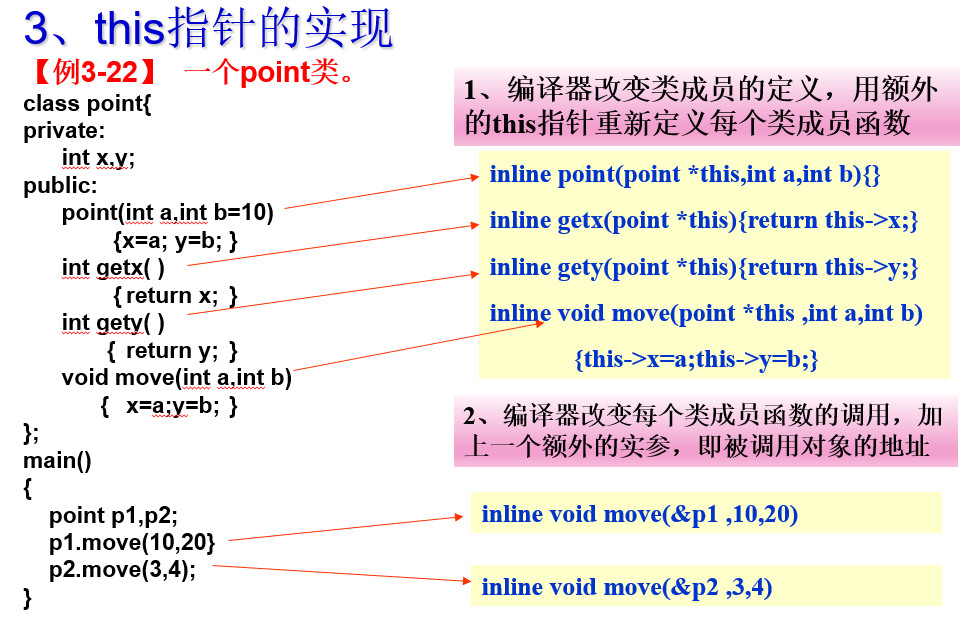
**访问this指针**

**X::f (……)**

**{**

**this->member**

**}**



**在类X的非const成员函数里，this的类型就时X \*。然而this并不是一个常规变量，不能给它赋值，但可以通过它修改数据成员的值。**

在类成员函数中，可以通过this指针返回对象的地址或引用

Point 为类

point p1(3, 3); //定义单个对象

point p[3]={ **{2,2},{3,3}, {4,4}** }; //L4，列表是3次调用构造函数的参数

point p2[3]; //L5

point\* pt; //L6

* 1. **普通函数（非类成员）接收参数对象后，在函数体内必须按照访问权限访问对象成员，即只能访问对象的公有成员。**
  2. **类成员函数可以访问本类参数对象的私有、保护、公有成员。**

**class X{**

**类名1 成员名1;**

**类名2 成员名2;**

**……**

**类名n 成员名n；**

**};**

* + 拥有对象成员的类必须对对象成员进行初始化。
  + 对象成员的初始化方式包括类内初始化或构造函数初始化列表
  + 当对象成员所在类有类内初始值或默认构造数函（包括系统自动生成的默认构造函数）时，可以省略对象成员的初始化代码，编译器会自动调用它们。
  + 若对象成员没有类内初始值，也没有默认构造函数，就必须在构造函数初始化列表中显式初始化对象成员，否则会产生错误。
* **X::X(参数表0)：成员名1（参数表1）,……成员名n(参数表n)**
* **{**
* **构造函数体**
* **}**

**3．各类对象的生存期**

**① 生存期与对象的构造次序和销毁次序密切相关。**

**② 局部对象和静态对象的构造次序与它们在块中的声明次序相同，即在块中先声明的就先构造，块即对象定义所在的一对{}所框定的代码区域。**

**③ 所有的全局对象在main之前构造，在main 结束之后销毁。**

**④ 对象数据成员（包括对象成员）的构造次序与其在类中的声明次序相同，而与它们在构造函数的初始化列表中的次序无关。**

**⑤ 在对象生存期结束时，具有相同生存期的对象将按与构造的相反次序销毁。**

**⑥ 非静态对象的生存期与其作用域是一致的，而静态对象的生存期则长于其作用域，程序结束时静态对象的生存期才结束。**

获得了访问该类的所有成员—包括public、protected、private类型特权的函数就是**友元**。

**class X{**

**……**

**friend T f(…); //声明f为X类的友元，f的形参通常是X类的对象**

**};**

**……**

**T f(…) { …… } //友元不是类成员函数，定义时不能用“X::f”限定函数名**

* **一个类可以是另一个类的友元，友元类的所有成员函数都是另一个类的友元函数，能够直接访问另一个类的所有成员（包括public、private和protected）。**

**可以指定类的某个成员函数是另一个类的友元，也就是友元成员函数**

【例】有两个类A和B，将类A的成员函数sum 定义成类B的友元成员函数，使它能够计算两个类数据成员的总和。

**friend int A::sum(B b);**

友元函数**并非类的成员函数**

友元**不具逆向性和传递性**

**接口与实现的分离：**

**………………**