# 基本语法

菜鸟教程 [Reference](http://www.runoob.com/cplusplus/cpp-basic-syntax.html)

## 数据类型

## 指针

**指针**是一个变量，其值为另一个变量的地址，即，内存位置的直接地址。

指向类的数据成员指针：

指向类，所指数据成员必须是共有的，用途不大

指向类成员函数的指针：

类并不是对象，有时候可将其视为对象使用，可以声明并使用指向对象成员函数的指针。

例如：void(A::\*pafn)(void)：表示他指向无参数和无返回值的类A的成员。

使用：

pafn=A::fa;

A x;

A \*px=&x;

x.fa();

对象调用：(x.\*pafn)();

指针调用：(px->\*pafn)();

一般说明方法： type(A::\*pointer)(list);

pointer=A::fun;

函数名表示地址，在数组中，数组名表示首地址。

Test \*one{new Test(3),new Test(4);},

one[0]表示第一个地址，要把它当指针用。\*one[0]可表所指内容。

## 引用

引用变量是一个别名，也就是说，它是某个已存在变量的另一个名字。

1. **int**    i=10;
2. **int**&    r = i;

引用通常用于函数参数列表和函数返回值。

## 书写规范

1.C++以.cpp扩展名；/\* \*/或//注释；endl和“\n”都是换行；

2.类声明放在头文件中。较大的类采用一个头文件一个实现文件

3.

//Point.h

#if!defined(POINT\_H)

#define PONT\_H

class Point

{

private:

int x,y;

public:

Point();

Point(int ,int);

~Point(){…….};

}

#endif

//main\_test.c

#include <Point.h>

void main()

{

}

## 语法错误

class Point

{

int x=2;(类体内，只有对象才有意义)

}

void main()

{

cout<<Point.x; //x是类的私有成员。可使用public的函数Point.show();

}

2.显示调用构造器：Point a.Point(1,2);

3.Point d,\*p;p=&d;p.d->x;p只能指向对象而不是对象成员

4. 不能直接给类数据成员赋值,因为类对象不能直接访问类私有成员。区别java

**变量**

变量初始化：int a(5) 等同a=5;

## const常量

const修饰符和预处理程序。C++建议用const，如

#define BUFSIZE 100。在置换过程很多隐藏错误，还有类型问题。BUFSIZE究竟整形还是浮点数？等效const int BUFSIZE=100;常量不能改变，必须初始化

# 第二章 面向对象

## 2.1 类&对象

### C到C++的演变

C++允许结构中定义函数。实际上类在 C++ 中就是 **struct** 的一种扩展，允许了更高级的继承和虚函数。

1. **struct** Point {
2. **private**:
3. **int** a,b;
4. **public**:
5. Point(){};
6. Point(**int** x,**int** y){
7. a=x;b=y
8. };
9. **void** show(){..};
10. };

struct成员函数默认public，class中则是private

### 实例化方式

Point a,b;

Point \*p=&a

Point &r=b; //定义引用

1.对象和引用使用“.”访问对象成员.指针用“->”,例如this

2.Point a; 程序自动调用构造器来分配内存。

3.区别于java，C++私有成员在类外就不能访问，即便是对象，但可以a=b这样赋值

Point a[2]对象数组。可以Point a[2]={ Point(),Point(1,2 ) }

1. **class** Coordinate
2. {
3. **public**:
4. **int** m\_iX;
5. **int** m\_iY;
6. };
7. **void** main(){
8. Coordinate coord[3]; //栈上
9. coord[1].m\_iX=10;
10. Coordinate \*p=**new** Coordinate[3]; //堆中
11. p[0].m\_iX=10;
12. p->m\_iX=10;
13. delete []p;
14. p=NULL;
15. }

### 不完全类声明

We know在C中如果两个结构体互相包含，则需要对其中一个结构体进行不完整声明

1. **struct** B;    //对结构体B进行不完整声明
3. //结构体A中包含指向结构体B的指针
4. **struct** A
5. {
6. **struct** B \*partner;
7. //other members;
8. };

C++中不完全类声明

1. **class** A;
2. **class** B{
3. **public**:
4. A\* a; //（不能A a）
5. }

**好处**

不必要的#include   会增加编译时间.

混乱随意的#include可能导致循环#include,可能出现编译错误

### 局部类

定义在一个函数内部的类，这个类只能在这个函数内部使用

1. **int** main()  {
2. **class** c4 {
3. **public**:
4. **int** a;
5. **void** foo() {a = 4;}
6. };
7. **class** c4 ff;
8. ff.foo();
9. cout << ff.a << endl;
10. **return** 0;
11. }

### 嵌套类

一个类中定义另一个类。（同java的内部类？）

## 2.2 多态性

### 2.2.1 虚函数

[Reference](https://blog.csdn.net/haoel/article/details/1948051)

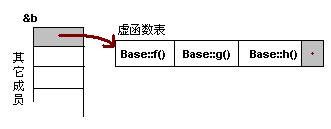
虚函数（Virtual Function）主要作用是实现多态。

虚函数表

虚函数是通过一张虚函数表（Virtual Table）来实现，简称为vtbl。

1. **class** Base{
2. **public**:
3. **virtual** **void** f(){
4. cout<<"Base::f() is called"<<endl;
5. }
6. **virtual** **void** g(){…}
7. **virtual** **void** h(){…}
8. };

**Base b;**

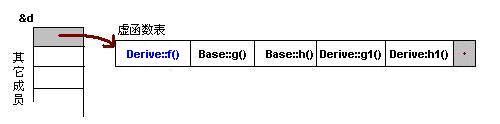


* 编译器会为每个有虚函数的类创建一个虚函数表，该虚函数表将被该类的所有对象共享。
* 通过对象实例的地址vptr得到这张虚函数表，然后就可以遍历并调用其中函数。

派生类

Derive \*d;

1. **class** Derive:**public** Base{
2. **public**:
3. **void** f(){
4. cout<<"Derive::f() is called"<<endl;
5. }
6. };



注意vtable只存储virtual修饰的方法。

#### 纯虚函数

C++版的抽象类。在基类中没有定义，但要求任何派生类都要定义自己的实现方法。

1. **virtual** **void** funtion1()=0

**new**

1.new用于建立生存期可控的动态对象，返回对象指针。

Point \* p=new Point;Point \* p2=new Point(1,2);

delete p;delete p2;

2.指针名 =new 结构名

3.动态变量

.double \*p;p=new double[3]；delete p;

### 2.2.3 重载&重写

同java，overwrite/overload用于实现多态。

## 2.3 继承

单一继承.多重继承

class Rectangel:public Point，private Oval{

void test(){show();}

}

继承方式 基类 派生类

public private 不可访问

protected protected

public public

private private和protected变成private，public变成protected

在派生类中可以void show(){Point::show}方式来访问

**二义性**

类Rectangel的基类Point，Oval都有show（）。c.show()产生二义性。可以

r.A::show()来区分，Point属于Rectangel的被支配者。

Rec若和基类重名则自动覆盖了。

**string对象**

要#include<string>

初始化：1.string str=”ab”或str[0]=’a’,str[1]=’b’同样cout<<str[0]来单个输出

2.string str(“how are you!”);

## 2.3 数据封装

封装是面向对象编程中的把数据和函数绑定在一起的一个概念，这样能避免受到外界的干扰和误用，从而确保了安全。

接口（抽象类）

## 2.4 接口

C++的接口通过抽象类实现

1. **class** Box
2. {
3. **public**:
4. // 纯虚函数
5. **virtual** **double** getVolume() = 0;
6. **private**:
7. **double** length;      // 长度
8. **double** breadth;     // 宽度
9. **double** height;      // 高度
10. };

## 2.4 函数

1.函数要有类型说明。否则void。函数要先声明或写在前面才能用。

int add(int ,int)不要参数名（函数原型）。声明只告诉编译器类型

2.参数传递：传值和传引用。可使用对象（传值，传的是备份，不影响原来的），对象指针，对象引用（C++推荐）

如：void swap(string,string)不改变原来的

3.返回值：

引用 int& max(int, int)

指针　int　＊　ｍａｘ（ｉｎｔ，ｉｎｔ）

对象 string max（int，int）

4.定义成员函数

除了在类中，也可在类外定义，但返回值，参数类表都要一样

int Pont::show(int a=1,int b=2)

{

x=a; y=b;

}

等价于int Point::show:x(1),y(2){..}

**this指针**

C++在类外定义成员函数可以访问类的私有成员。实际隐藏this指针

void Point::say(string str,(Point)\*this)

{

this->name=str;

}

说明：1.this指向调用它的对象

2.作用域运算符“：：”，之后内容表示属于这个类

### 构造函数

编译器给类默认生成，但自己定义了就不产生

### 复制构造函数

编译器默认产生的原型Point::Point(Point& t)；

若定义Point::Point(const Point& t) //推荐加const

{

x=t.x;y=t.y;

}

Point p1(1,2);

Point p2(p1); //就可以复制

### 析构函数

编译器默认产生，自动调用。形如：

~Point(){};

delete释放动态数组:delete [] p数组也要掉析构

说明：构造函数，复制构造函数，析构函数是类的基本成员

### 内联函数

加快程序执行速度，例如

inline int isnumber(char c){…..}

## 2.5 友元friend

友元提供了一种 普通函数或者类成员函数 访问另一个类中的私有或保护成员的机制。

好处：按需增加接口，但破坏封装性。

友元类

1. **class** B{
2. **public**:
3. B(**int** \_b){ b=\_b;};
4. **friend** **class** C;//声明友元类C
6. **private**:
7. **int** b;
8. };
10. **class** C {//实现友元类C
11. **public**:
12. **int** getB\_b(B \_classB){
13. **return** \_classB.b;//访问友元类B的私有成员
14. };
15. };

18. **int** main(**int** argc, **char** \*argv[]){
19. B \_classB(3);
20. C \_classC;//定义友元类实例
21. std::cout<<\_classC.getB\_b(\_classB);
22. **return** 0;
23. }

友元函数

1. **class** Demo{
2. **public**:
3. Demo(**int** \_a){ a=\_a;};
4. **friend** **int** display(A &\_classA);//友元函数
6. **private**:
7. **int** a;
8. };
10. **int** display(A &\_d){
11. **return** \_d.a;//通过对象名访问私有变量
12. }
14. **int** main(**int** argc, **char** \*argv[]){
15. Demo \_d(3),\_d2(4);
16. std::cout<<display(\_d1)+display(\_d2);//友元函数只是普通函数，可以在任意地方调用
17. **return** 0;
18. }

# 第三章 进阶

## 3.1文件和流

输入输出格式控制：

自动识别浮点数，并用最短格式输出，例如：

int a;double b;

cin>>a>>b; //23.23时...

单个字符：舍去空格,直到对字符 char a,b,c;cin>>a>>b>>c;

字符串：没有结束位，加"\0"，防止后面的乱码

char \*p=new char[5];

for(int i=0;i<4;i++) cin>>\*(p+i);

p[4]=‘\0’;cout<<p;

字符数组：用数组名整体读入 char a[30];cin>>a;

对于字符指针，尽量动态分配地址，也能逐个赋值。例如：

char \*a;

st=new char[strlen（n）+1];

## 3.2 模板

## 3.3 多线程

## Socket

1. #include <sys/socket.h>

scoket🡪bind🡪listen🡪accept🡪read/write🡪close

## 3.4动态内存

## 3.5 编译指令

编译指令在编译程序之前进行，分析处理C++的不是程序一部分。

作用：易读，易改，易移植和便于调试

### 文件包含（嵌入指令）

#include “c:\hello.h” ；“ ”一般自己定义的头文件。<xxx.h>系统的；C++不需要<.h>了。

说明：编译指令不是C++一部分，所以这里是“\”。若在程序中，则

char fname[]=”\\user\\test.h”

<cmath>代替<math.h>，#include <iostream> 标准库函数。但必须

using namespace std;如cout和cin<<x<<y;输入2 4<CR>2和4之间有空格

### 宏定义#define

#define MAX(a,b) ((a)>(b)? \

(a)(b) )

说明：结束不能加分号”;”如换行用“\”。不想影响其他部分用#undef删除

**条件编译。**

#if,#else,#elseif,#endif

形式:#if true/false

….

#else

…..#error(#error常在else中使编译器显示错误信息)

#endif

**defined操作符**

判断标识符是否被#define定义。若用于调试。下如

#define DEBUG //调试结束删除即可

void main()

{

#if defined(DEBUG)

…..

#endif

}

#ifdef,#infndef等同

#if!defined(HEAD\_H)---------------#ifndef HEAD\_H

#define HEAD\_H #define HEAD\_H

…….. ……

#endif #endif

编译指令一个占一行。一个头文件可能在多个文件中使用，造成变量和和类的重复定义。为了避免多个文件重复编译一个头文件，可在头文件Point.h中定义

#if!defined(POINT\_H)

#define PONT\_H

class Point{

…..

}

#endif

**编译器：**编译成.obj，链接程序（Link）处理成.exe

test.cpp----test.obj---test.exe

## 3.6 POSIX

1. #include <pthread.h>  //在Linux下编写多线程程序需要包含的头文件

POSIX线程（POSIX threads），简称Pthreads，是线程的POSIX标准。该标准定义了创建和操纵线程的一整套API。在类Unix操作系统（Unix、Linux、Mac OS X等）中，都使用Pthreads作为操作系统的线程。Windows操作系统也有其移植版pthreads-win32。

遵循这个标准的好处是软件可以跨平台。  
所以windows也支持就很容易理解了，那么多优秀的开源软件，支持了这个这些软件就可能有windows版本，就可以完善丰富windows下的软件。

# 第四章 实践

window7 hello World

1. #include "stdafx.h"
2. #include <iostream>
3. **using** **namespace** std;
5. **int** main()
6. {
7. cout << "Hello World";
8. **return** 0;
9. }