类

内联函数：

构造/析构函数：

初始化器（初始化const成员唯一机会）

嵌套层次（构造外到里，析构里到外）

复制构造函数：deep copy

存在构造函数，不会再默认生成（必须自己补充默认构造函数！）

访问控制：

private：类与友元

protected：类，子类与友元

public：所有

无标号：默认private

\*this指针：return \*this连锁调用；参见等号重载a=b=c;等价于a.operator=(b).operator=(c)

友元函数：给予友谊，所有成员均可访问，任意标号下即可声明

Static成员：类层次而非对象层次，所有属于一个类的对象共享该成员，常用作生成对象计数器；static成员函数无this指针，不能使用非static成员;外部初始化！

例：

class Newclass{

friend friendship();//友元函数

Private:

Int a;

Static count;//static成员作计数器

Public:

Void blank();//空函数

inline const string& shorterString(const string &s1, const string &s2)

{return s1.size() < s2.size() ? s1:s2;}

//内联函数Const函数：声明与定义都必须带const关键字

Newclass():a(0){count++;}

//构造函数：初始化器（initializer），把a初始化为0；计数器，每生成一个对象加1！

~ Newclass(){cout << “over” << endl;}//析构函数

}

int Newclass::count=0;//static成员的初始化！

Void Newclass::blank(){}//普通成员函数实现实例

Int main()

{

Newclass temp;//注意Newclass temp();是错误的！这是函数声明并且返回该对象！

Newclass \*ptr=&temp;

Ptr->blank();//指针调用（动态绑定必须用这个！）

Temp.blank();//函数名调用

}

重载函数

PS：

Main不重载

形参与返回同——重复声明

形参同，返回不同——错误

形参不同——重载（注意（引用或指针形参带/不带const）视为不同）

例：

Record lookup(Account&);

Record lookup(const Account&);//可以，两个不同函数

局部声明函数，将屏蔽外层作用域中的同名函数！

例：

Void print(double);//函数1

Void foo(int ival)

{

Void print(int);//函数2

Print(ival);//调用2

Print(3.14);//仍然调用2！1被屏蔽

}

形参不同

Record lookup(const Account&);

Record lookup(const Phone&);

Record lookup(const Name&);

操作符重载

四则运算（+用+=实现，其他类似）

[－](http://www.adintr.com/myarticle/operator.html) Carrot& Carrot::operator+=(const Carrot& rhs)

{   
    return \*this;   
}   
【注】1. C++ 允许 (x += 1) = 0 形式的赋值操作, operator+= 必须传回 \*this 的非常量引用.

const Carrot operator+(const Carrot& lhs, const Carrot& rhs)

{   
    Carrot result = lhs;  
    return result += rhs;   
}   
【注】1. 如果可能, 应考虑使用 operator+= 来实现 operator+ .   
         2. operator+ 不能返回引用, 应返回值类型.

[－](http://www.adintr.com/myarticle/operator.html) friend ostream& operator<<(ostream& \_OStr, const Carrot& rhs)

{   
    return \_OStr;   
}

[－](http://www.adintr.com/myarticle/operator.html) bool operator==(const Carrot& lhs, const Carrot& rhs)

[－](http://www.adintr.com/myarticle/operator.html) const Carrot Carrot::operator++(int)

{   
    Carrot oldValue = \*this;  
    ++(\*this);  
    return oldValue;  
}   
【注】1. 后置操作(postfix): carrot++ 调用时编译器自动生成一个 0 作为参数.

[－](http://www.adintr.com/myarticle/operator.html) Coca& Carrot::operator[](KeyType index)

friend Point operator+(Point& p1, Point& p2)

{

Point sum;

sum.x=p1.x+p2.x;

sum.y=p1.y+p2.y;

return sum;

}

friend Point operator-(Point& p1, Point& p2)

{

Point abs;

abs.x=p1.x-p2.x;

abs.y=p1.y-p2.y;

return abs;

}

friend Point operator\*(Point& p1, Point& p2)

{

Point pro;

pro.x=p1.x\*p2.x;

pro.y=p1.y\*p2.y;

return pro;

}

friend ostream& operator<<(ostream& out,const Point& p){

out << "(" << p.x << ", " << p.y << ")" ;

return out;

}

继承

动态绑定：在基类的函数前加上virtual关键字，在派生类中重写该函数，运行时将会根据对象的实际类型来调用相应的函数：

如果对象类型是派生类，调用派生类的函数，

如果对象类型是基类，调用基类的函数。

返回类型前加上virtual（构造函数不虚）

纯虚函数：函数名后加=0

抽象类：有虚函数的类

Print为虚函数

Employee e, \*ePtr = &e;//指向基类的基类指针

HourlyWorker h, \*hPtr = &h; //指向子类的子类指针

ePtr->print(); //调用基类函数  
hPtr->print(); //调用子类函数  
ePtr=&h; //隐式转换！  
ePtr->print(); //调用子类函数，多态性体现！

构造函数和赋值操作符不是虚函数！

例（不能运行，可以运行的去林倞那里找）：

class Shape {

public:

virtual double area() const { return 0.0; }

virtual double volume() const { return 0.0; }

// 纯虚函数示范

virtual void printShapeName() const = 0;

virtual void print() const = 0;

};

class Point : public Shape {

public:

Point( int = 0, int = 0 ); // default constructor

void setPoint( int, int );

int getX() const { return x; }

int getY() const { return y; }

virtual void printShapeName() const { cout << "Point: "; }

virtual void print() const;

private:

int x, y; // x and y coordinates of Point

};

int main()

{

Point point( 7, 11 ); // create a Point

point.printShapeName(); // 静态绑定

point.print();

Shape \*baseClassPtr=&point；

baseClassPtr->printShapeName();//动态绑定

baseClassPtr->print();

}

模板

函数模板

Template <class T1,typename T2>

Int foo(const T1&, const T2&);

特化

Template<> int foo<const char\*, const int\*>(const char\* const&v1, const int\* ii);

实例化

Foo<int,double>(a,b);

类模板

Template <class T>

class myqueue{…..};

实例化

Myqueue<int> queue\_int;

异常处理

Try{…}

Catch(int i){…throw;}

Catch(…){….}//捕获余下所有异常

命名空间

限定作用域，不要以分号结尾！

Namespace cpp{

Class fun{};

}

外部使用方法

Using cpp::fun；