# hello world

#include <iostream>

int main(int argc, const char \* argv[]){

// 标准的命名空间std

std::cout << “hello world”;

}

# 标准库

对象使用.

std::vector<>;

std::cout;

std::endl;

# 类

## 定义类

**NPC.h文件中定义函数**

使用宏定义防止多次加载

#ifndef \_\_xxx

#define \_\_xxx

class Rect{}

定义类

class NPC

{

NPC（）； //构造方法,与类名相同

~NPC（）； //析构方法

Rect \* a; //引用另外的类

private:

protected:

public:

string Name；

void move(int a); //定义一个方法

}

#endif //宏定义

## 实现类

**NPC.cpp文件实现方法**

#include “NPC.h”

NPC::NPC(){} //实现构造方法

NPC::~NPC(){ //析构方法

delete this->a; //释放空间a

}

void NPC::move(int a){ //::成员操作符

std::cout << “1”;

}

## 调用

**main.cpp**

#include “NPC.h”

int main()

{

NPC n1 // 创建一个类，分配空间

n1.move(1) //调用类中的方法

#-------- 动态生成一个对象

NPC \* n2；

n2=&n1 // 可以指向一个已经创建的对象

n2=new NPC() // new一个对象

#-------------

NPC n3(1) //调用构造函数

NPC \* n3 = new NPC(1)

}

1. 构造方法
2. 析构方法

# 方法重载

在同一个类中有相同的名称，但参数不同(包含类型不同)，系统会根据参数不同，调用不同的方法

class A

{

void B(int a){};

void B(int a, int b){}; //重载

}

# 五、类的继承

class NPC{}

#include “NPC.h”

class Other:NPC{} //继承NPC

class Other:NPC,NPC2 {} //继承多个

class Other:public NPC {} //公有继承

多层继承会依次调用各个层的构造方法

# 六、虚函数-抽象类-不懂

虚函数永远执行子类中的方法（方法重载）

实现父类指针调用子类方法

class NPC2:NPC

{

virtual void fire(int a); //虚函数定义

virtual void fire(int a)=0; //纯虚函数，子类必须定义

}

class Sound{

virtual void play()=0;

}

void sound::play(){}

#include “sound.h”

# 七、操作符的重载

class NPC

{

void operator++(); //重载++操作符

}

void NPC::operator++(){} //重新实现++操作符

# 八、模板-泛型编程

## 模板函数

//定义一个模板函数

// 函数名getMax

template<class T>

T getMax(T a, T b) {

return a>b?a:b;

}

int main()

{

// 不用特殊定义属性，模板函数会自动返回T类型的值，输入T类型的变量（T类型，是根据你的输入自动变换的）

std::cout<<getMax(1,2)<<std::endl;

std::cout<<getMax(“a”,”b”)<<std::endl;

}

## 模板类-不懂

模板定义的变量是可变的，其他的都一样

template<class T>

class LinkList{

struct node{

T \* data;

}

}

# 九、C++11 新特性

## 1、auto 自动实现类型转换，

auto 可以指向任何函数，实现类型转换【弱类型语言】

auto 定义的就是，任意类型语言，可以放任何东西

## 2、增强的for循环

**[可以理解成for...in]**

for(auto tmp:allNPC){

//tmp是allNPC中的每一个值

tmp->move(30,40) //移动

}

## lambda 匿名函数

[] 使用外部变量

auto a = **[**=赋值方式、&引用方式**](**参数**){**代码块**}**

a(); //调用