# 一、C

## 1、引入文件

#include "good.h" //调用good.h,直接调用即可

#include "..\two\some.h" //调用上一目录的其他文件夹下的头文件

## 2、宏定义

#ifndef \_\_luck\_\_MainMenu\_\_

#define \_\_luck\_\_MainMenu\_\_

// luck项目名称，MainMenu文件名称

#endif

## 3、BASE

#### 变量

int 整形 %d

float/double 浮点型 %f

short 65535-长度

long

字符类型：char

每一个字符都有自己的编码ACSII，如果字符串按数字输出，将输出ACSII编码

char a = ‘12’ %c

scanf（“%d”）获取用户输入数据

#### 数组

int Array[20] //定义20长度的数组

int Arr[10]={0,1,....} //定义一个10个大小的数组,连续空间

int Array[]={1,2,3,4} //定义并赋值

printf(“%d”,Arr[1]) //输出

#### 字符串

Char name[]=”holle” //字符串，有\0

Char name[]={“h”,”o”,”l”,”l”,”e”} //字符串数组，没有\0

// \0字符串结束标准，字符串的内容要循环修改，将字符一个一个赋值，不能直接等于赋值

char name[20]={‘a’,’b’,’\0’}

#### 使用内置函数拷贝

//主要是因为，C语言中字符串是存放在数组中的，拷贝字符串需要把数组中每一个元素拷贝一遍

#include <string.h>

strcpy(a,b)

#### 函数

int getName(int a, int b)

{

return 0; 必须有返回值

}

## 4、scanf

scanf(“%s”,input); 字符串不用添加&

scanf(“%d”,&input);获取数字时要添加&

## 5、指针

**在1个等式上不能同时出现 \*, &;**

**当定义一个变量是指针类型时，它唯一多出来的属性是 \* ，可以使用\*获取变量中地址的内容，其他的变量属性是不变的，为了防止内存溢出，\*p声明时就应赋值**

定义：

int a=5

int \*p // \*p指针变量，4个字节

### 情况一

p=a // p保存的是a中的值，相当于普通变量

### 情况二

p=a;

printf(\*p) //这里会溢出，因为P变量存的值是5,\*p是找 内存中指向5的位置

### 情况三

\*p=a / p=&a //这时2种方式，都是获取a的地址，变量p中保存的都是地址

printf（\*p）

### 情况四

\*p=&a // 这里会溢出，不知道指到哪里了

### 情况五

int p, b=5;

p=&b; //a中存放地址

printf(\*p); //语法错误，只有指针类型才能使用 \*

printf(p); // 打印b的地址

### 情况六

int \*p, b=5

p = &b; // 存放b的地址，普通变量也可存放地址

\*p = 15 //给 b 赋值

### 情况七-指向数组的指针

int stu[2] = {1,2}, \*p

// 这3中写法都指向了数组stu，stu存放的是数组的首地址，

// stu[0]存放的是首元素，so，要使用 &stu[0]

p = &stu[0] / p=stu / \*p=stu

printf(\*(p+3)) // 指针下滑，指向第3个数

### 情况八-指向函数的指针

int func(int a){}

int (\* pfun)(int a)； // 定义一个函数指针

pfun=func; // 指向一个函数

pfun=&func; // 指向一个函数

printf（pfun（1）） // 调用函数

## 6、结构体

结构体的存在是为了节省代码，相同的变量，不用多次声明，通过结构体声明多个具有相同属性的变量

**结构体变量，结构体数组，结构体指针**

### 定义方式一

1. **声明结构体**

struct Player{

int a;

}

1. **定义结构体 p1**

struct Player p1; // p1包含Player中声明的所有变量

**3、使用结构体变量赋值**

p1.a = 1; // 给结构体的属性赋值

### 定义方式二

struct Player{

int a;

}p1,p2 // 定义了2个结构体变量（Player）:p1,p2

struct Player p3;

struct Player p4;

-----------------------------------------------------------------

// 可以这样定义，但这样定义将不能定义p1,p2之外的结构体变量

struct { //这个结构体并没有标明其标签

int a;

}p1,p2

### 定义方式三

typedef struct { // 用typedef创建新类型,新类型名为Player

int a;

}Player

Player p1; //定义一个结构变量

p1.a = 1;

---------------------------------------------------------

typedef struct \_Player{

int a;

}Player;

Player p1;

p1.a = 1; // 给结构体的属性赋值

struct \_Player p2 //结构体定义

### 指针使用结构体

typedef struct \_Player{

int a;

}Player;

Player \* nowp //声明一个**机构体**指针，这个指针（nowp）指向结构体（Player）

nowp=&p1 // 将结构体指针，指向p1

nowp->a // -> 指针的成员操作符与.意思相同

### 结构体数组-没确定使用方式

typedef struct \_player{

int a;

}Player;

Struct Player info[30] // 使用结构体Player 声明一个30容量的info

info[1].a=1;

## 7、联合体，结构体数组

typedef struct \_Play{

int a;

union { // 联合体，类似枚举类型,只保存联合体中最大的容量

int ID;

char name[20];

}job;

}

## 8、枚举类型

enum SYS{ // 枚举类型的值是根据顺序排列的

ADD=1, // 1

QUERY=2, // 2

}

enum SYS input; // 声明一个枚举变量input

scanf(“%d”, &input) //输入信息放到input中,提示用户输入1、2、3、4

if (input == ADD) {} // 枚举类型会自动管理相关变量

## 9、动态分配空间

# C语言-Player结构体动态分配空间

Player \* None=（Player \*）malloc(sizeof(Player));

#

Player \* None = new Player;

free //释放空间

## 10、文件操作

FILE \* f1; //定义一个文件指针

f1=fopen(“路径”, “w”)

## 11、杂项

变量存储内省

堆区 存放动态分配的数据

栈区 存放局部数据，局部变量

全局数据区 全局数据和静态数据，全局变量

程序代码区 程序的各个函数代码

变量存储类型

Auto自动类

register寄存器类 仅局部变量

static静态类

extern外部类

# 二、C++

## 1、hello world

#include <iostream>

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

// 标准的命名空间std

std::cout << “hello world”;

}

## 2、标准库

对象使用.

std::vector<>;

std::cout;

std::endl;

## 3、类

### （1）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

### （2）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”;

}

### （3）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

## 4、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

}

## 5、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

## 6、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”

## 7、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

## 8、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;

}

}

## 9、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

# C++内存管理对象

new 创建对象、delete删除对象

# 字符串

**字符串都是数组，char类型也是数组（这么理解）**

字符串的二维数组：C语言与其他语言不同

**普通定义-字符串常量中放的是变量**

char a[] = {“221”};

char a[2][10] 表示有2个字符串，每一个字符串的长度是10.

**指针定义-a中放的是地址**

char \*a[] = {“11”,”22”};

用处：

当在使用if的时候，