李海龙的C++

[在此处键入文档摘要。摘要通常为文档内容的简短概括。在此处键入文档摘要。摘要通常为文档内容的简短概括。]

[键入文档副标题]

目录

[第一节. 必备基础 3](#_Toc39568427)

[第1节. 书写Hello,World 3](#_Toc39568428)

[第2节. 变量 4](#_Toc39568429)

[第3节. 常量 4](#_Toc39568430)

[第4节. 标识符（关键字） 5](#_Toc39568431)

[第5节. 标识符命名规则 5](#_Toc39568432)

[第二节. 数据类型 5](#_Toc39568433)

[第6节. 整型、实型所占内存 5](#_Toc39568434)

[第7节. 实型 6](#_Toc39568435)

[第8节. 字符型 6](#_Toc39568436)

[第9节. 转义字符 7](#_Toc39568437)

[第10节. 字符串型 7](#_Toc39568438)

[第11节. 布尔类型 8](#_Toc39568439)

[第12节. 数据的输入？？？ 8](#_Toc39568440)

[第三节. 运算符 9](#_Toc39568441)

[第13节. 算术运算符 9](#_Toc39568442)

[第14节. 取模运算 10](#_Toc39568443)

[第15节. 递增递减 10](#_Toc39568444)

[第16节. 赋值运算 11](#_Toc39568445)

[第17节. 比较运算 12](#_Toc39568446)

[第18节. 逻辑运算 12](#_Toc39568447)

[第四节. 程序流程编号 13](#_Toc39568448)

[第19节. 选择结构??? 13](#_Toc39568449)

[第1部分. if语句 13](#_Toc39568450)

[第2部分. 三目运算符 19](#_Toc39568451)

[第3部分. Switch语句 20](#_Toc39568452)

[第20节. 循环结构 21](#_Toc39568453)

[第1部分. while语句 21](#_Toc39568454)

[第2部分. do…while语句 23](#_Toc39568455)

[第3部分. for循环 25](#_Toc39568456)

[第4部分. 嵌套循环 26](#_Toc39568457)

[第21节. 跳转语句 27](#_Toc39568458)

[第1部分. break语句 27](#_Toc39568459)

[第2部分. continue语句 28](#_Toc39568460)

[第3部分. goto语句 29](#_Toc39568461)

[第五节. 数组 29](#_Toc39568462)

[第22节. 一维数组 29](#_Toc39568463)

[第1部分. 定义方式 29](#_Toc39568464)

[第2部分. 数组名 30](#_Toc39568465)

[第3部分. 案例 31](#_Toc39568466)

[第4部分. 冒泡排序 34](#_Toc39568467)

[第23节. 二维数组 35](#_Toc39568468)

[第1部分. 定义方式 35](#_Toc39568469)

[第2部分. 数组名 37](#_Toc39568470)

[第3部分. 案例 37](#_Toc39568471)

[第六节. 函数 38](#_Toc39568472)

[第24节. 函数的定义 38](#_Toc39568473)

[第25节. 函数的调用 39](#_Toc39568474)

[第26节. 值传递 39](#_Toc39568475)

[第27节. 常见形式 40](#_Toc39568476)

[第28节. 函数的声明 42](#_Toc39568477)

[第29节. 函数的分文件编写 42](#_Toc39568478)

[第七节. 指针 43](#_Toc39568479)

[第30节. 指针的定义和使用 43](#_Toc39568480)

[第31节. 指针所占的内存空间 44](#_Toc39568481)

[第32节. 空指针 44](#_Toc39568482)

[第33节. 野指针 45](#_Toc39568483)

[第34节. const修饰指针 45](#_Toc39568484)

[第35节. 指针与数组 46](#_Toc39568485)

[第36节. 指针与函数（地址传递与值传递） 47](#_Toc39568486)

[第37节. 案例 48](#_Toc39568487)

[第八节. 结构体 49](#_Toc39568488)

[第38节. 结构体的定义和使用 49](#_Toc39568489)

[第39节. 结构体数组 50](#_Toc39568490)

[第40节. 结构体指针 51](#_Toc39568491)

[第41节. 结构体嵌套结构体 52](#_Toc39568492)

[第42节. 结构体作函数参数 53](#_Toc39568493)

[第43节. 结构体中const的使用案例 54](#_Toc39568494)

[第九节. 通讯录系统 55](#_Toc39568495)

[第44节. 通讯录的制作 55](#_Toc39568496)

[第十节. 内存模型 63](#_Toc39568497)

[第45节. 大纲 63](#_Toc39568498)

[第46节. 全局区 64](#_Toc39568499)

[第47节. 栈区 65](#_Toc39568500)

[第48节. 堆区 66](#_Toc39568501)

[第49节. new与delete 66](#_Toc39568502)

[第十一节. C++中的引用 67](#_Toc39568503)

[第50节. 引用的基本语法 67](#_Toc39568504)

[第51节. 引用的注意事项 68](#_Toc39568505)

[第52节. 引用作函数参数 69](#_Toc39568506)

[第53节. 引用作函数返回值 69](#_Toc39568507)

[第54节. 引用的本质 70](#_Toc39568508)

[第55节. 常量引用 71](#_Toc39568509)

[第十二节. 函数高级 72](#_Toc39568510)

[第56节. 函数的默认参数 72](#_Toc39568511)

[第57节. 函数的占位参数 72](#_Toc39568512)

[第58节. 函数重载 73](#_Toc39568513)

[第1部分. 基本语法 73](#_Toc39568514)

[第2部分. 注意事项 74](#_Toc39568515)

[第十三节. 类和对象 75](#_Toc39568516)

[第59节. 封装 75](#_Toc39568517)

[第1部分. 属性和行为作为整体 75](#_Toc39568518)

[第2部分. 案例 76](#_Toc39568519)

[第3部分. 访问权限 77](#_Toc39568520)

[第4部分. struct和class的区别 78](#_Toc39568521)

[第5部分. 成员属性私有化 79](#_Toc39568522)

[第6部分. 案例2 79](#_Toc39568523)

[第7部分. 案例3 82](#_Toc39568524)

[第60节. 对象特征 87](#_Toc39568525)

[第1部分. 构造函数和析构函数 88](#_Toc39568526)

[第2部分. 函数的分类以及调用 88](#_Toc39568527)

[第3部分. 拷贝构造函数调用时机 90](#_Toc39568528)

[第4部分. 构造函数调用规则 92](#_Toc39568529)

[第5部分. 深拷贝与浅拷贝 95](#_Toc39568530)

[第6部分. 初始化列表 97](#_Toc39568531)

[第7部分. 类对象作为类成员 98](#_Toc39568532)

[第8部分. 静态成员变量 99](#_Toc39568533)

[第9部分. 静态成员函数 100](#_Toc39568534)

[第10部分. 成员变量和成员函数分开存储 101](#_Toc39568535)

[第11部分. this指针的用途 102](#_Toc39568536)

[第12部分. 空指针访问成员函数 103](#_Toc39568537)

[第13部分. const修饰成员函数 104](#_Toc39568538)

## 必备基础

### 书写Hello,World

#include<iostream>

using namespace std;

int main()

{

cout<<"hello,world"<<endl;

system("pause");

return 0;

}

### 变量

//该视频须反复观看

//变量的作用：给一段指定的内存起名，方便操作这段内存

//如：int a=10; 某个内存中储存内容10，通过给该空间起名为a访问该空间，也可以通过该段内存的地址编号（通常是一段16进制的数）访问该内存

#include<iostream>

using namespace std;

int main()

{

//变量的定义:简单来说，变量是某段内存人为起的名字

//语法：数据类型 变量名 + 初始值

/\*c++中的数据类型有：

整型：短整型（short） 整型（int） 长整形（long） 长长整型（long long）

字符型（char）

浮点型：单精度型（float） 双精度型（double） 长双精度型（long double）

布尔型（bool）

\*/

int a = 10;

cout<<"a="<<a<<endl;

system("pause");

return 0;

}

### 常量

#include<iostream>

using namespace std;

#define Day 7 //1.宏常量

int main()

{

//Day = 8 //报错，宏常量不可修改

cout<<"一周有多少天？"<<Day<<"天"<<endl;

//2.const修饰变量

const int month = 12;

//month = 24 //报错，常量不可以被修改

cout<<"一年有多少个月？"<<month<<"个月份"<<endl;

system("pause");

return 0;

}

### 标识符（关键字）

作用：c++中预先保留的单词

使用原则：不要用关键字给常量或者变量起名字

### 标识符命名规则

给变量或者常量起名字时要注意：

标识符不能是关键字

标识符只能有由字母、数字、下划线组成

第一个字母不能是数字

标识符中字母区分大小写

## 数据类型

### 整型、实型所占内存

#include<iostream>

using namespace std;

int main()

{

cout<<"short类型所占用的内存空间为："<<sizeof(short)<<"字节"<<endl;

cout<<"int类型所占用的内存空间为："<<sizeof(int)<<"字节"<<endl;

cout<<"long类型所占用的内存空间为："<<sizeof(long)<<"字节"<<endl;

cout<<"long long类型所占用的内存空间为："<<sizeof(long long)<<"字节"<<endl;

cout<<"float类型所占用的内存空间为："<<sizeof(float)<<"字节"<<endl;

cout<<"double类型所占用的内存空间为："<<sizeof(double)<<"字节"<<endl;

system("pause");

return 0;

}

### 实型

#include<iostream>

using namespace std;

int main()

{

float f1=3.14f;//float数据的值后常跟着f

double d1=3.14;

cout<<f1<<endl;

cout<<d1<<endl;

cout<<"float sizeof:"<<sizeof(f1)<<endl;

cout<<"double sizeof："<<sizeof(d1)<<endl;

//科学计数法

float f2=2e3;

double d2=3e-2;

cout<<f2<<endl;//f2=2000

cout<<d2<<endl;//d2=0.03

system("pause");

return 0;

}

### 字符型

#include<iostream>

using namespace std;

int main()

{

char ch='a';

cout<<ch<<endl;

//ch='abcde';//错误，单引号里只能有一个字符

//ch="a";//错误，不可以使用双引号

cout<<(int)ch<<endl;//查看字符a所对应的ASCII码

ch = 97 ;//可以直接对数据型变量进行赋值 //a对应97，A对应65

cout<<ch<<endl;

system("pause");

return 0;

}

### 转义字符

#include<iostream>

using namespace std;

int main()

{

//转义字符:作用是显示一些不能显示出来的ASCII字符

cout<<"Hello,world!\n";// \n是换行符

cout<<"\\"<<endl;// 输入\\输出\

cout<<"李海龙李海龙牛牛\thello,world!"<<endl; //水平制表符（8个位置）

cout<<"aa\thello,world!"<<endl;//一个\t表示将\t前的内容设置为8个字符宽，如果\t前超过8个字符，则设置为16个字符宽，以此类推

system("pause");

return 0;

}

### 字符串型

//作用：用于表示一串字符

#include<iostream>

using namespace std;

#include<string> //使用C++风格字符串时一定要使用该头文件

int main()

{

//C语言风格

char a[] ="李文";

cout<<"a="<<a<<endl;

//C++语言风格

string b="李海龙";

cout<<"b="<<b<<endl;

system("pause");

return 0;

}

### 布尔类型

//作用：代表真或加的值

#include<iostream>

using namespace std;

int main()

{

//1、创建bool数据类型

bool flag=true; //true代表真

cout<<flag<<endl;

flag=false; //false代表假

cout<<flag<<endl;

flag=6;

cout<<flag<<endl;//本质上 0代表假 其余字符代表真

cout<<"布尔数据所占的内存空间为："<<sizeof(flag)<<"字节"<<endl;

system("pause");

return 0;

}

### 数据的输入？？？

//作用：从键盘上获取数据

#include<iostream>

using namespace std;

int main()

{

bool flag=false;

cout<<"请给布尔数据flag赋值："<<endl;

cin>>flag;

cout<<"flag的值为："<<flag<<endl;

布尔类型中，除了0为假，其余都为真???

system("pause");

return 0;

}

## 运算符

### 算术运算符

#include<iostream>

using namespace std;

int main()

{

int a1=10; //C++中规定：两个整数相除结果依然是整数，将小数部分舍去，及向下取值

int b1=6;

cout<<"a1+b1="<<a1+b1<<endl;

cout<<"a1-b1="<<a1-b1<<endl;

cout<<"a1\*b1="<<a1\*b1<<endl;

cout<<"a1/b1="<<a1/b1<<endl;

/\*

int a2=3;

int b2=0;

cout<<"a2/b2="<<a2/b2<<endl; //该操作为非法操作

\*/

double a3=5; //可直观理解为double比float更精准，double内存更大

double b3=2;

cout<<"a3/b3="<<a3/b3;

system("pause");

return 0;

}

### 取模运算

#include<iostream>

using namespace std;

int main()

{

int a4=8; //int不能改为double，float等

int b4=20;

cout<<"余数为："<<a4%b4<<endl;

/\*

double e1=3.14;

double e2=2;

cout<<"余数为："<<e1%e2<<endl; //该存在方法为非法操作，取模运算不能有小数

\*/

system("pause");

return 0;

}

### 递增递减

#include<iostream>

using namespace std;

int main()

{

//1、前置递增

int a=10;

++a;

cout<<"a="<<a<<endl;

//2、后置递增

int b=10;

b++;

cout<<"b="<<b<<endl;

//3、前置递增和后置递增的区别

//前置递增：先让变量+1，然后进行表达式的运算

int c=10;

int d=++c\*10;

cout<<"d="<<d<<endl;

//后置递增：先进行表达式的运算，然后让变量+1

int e=10;

int f=e++\*10;

int g=e\*10;

cout<<"f="<<f<<endl;

cout<<"g="<<g<<endl;

system("pause");

return 0;

}

### 赋值运算

#include<iostream>

using namespace std;

int main()

{

//赋值运算

// =

int a=10;

a=2;

cout<<"a="<<a<<endl;

// +=

int b=10;

b+=2;

cout<<"b="<<b<<endl;

// -=

int c=10;

c-=2;

cout<<"c="<<c<<endl;

// \*=

int d=10;

d\*=2;

cout<<"d="<<d<<endl;

// /=

int e=10;

e/=2;

cout<<"e="<<e<<endl;

system("pause");

return 0;

}

### 比较运算

#include<iostream>

using namespace std;

int main()

{

//比较运算符

int a=10;

int b=2;

//==

cout<<(a==b)<<endl;

//!= //不等于

cout<<(a!=b)<<endl;

//>

cout<<(a>b)<<endl;

//<

cout<<(a<b)<<endl;

//>=

cout<<(a>=b)<<endl;

//<=

cout<<(a<=b)<<endl;

system("pause");

return 0;

}

### 逻辑运算

#include<iostream>

using namespace std;

int main()

{

//逻辑运算符

//在C++中 除了0 都为真

//非 ！

int a=10;

cout<<!a<<endl; //0

cout<<!!a<<endl;//1

//与 &&

int b=10;

int c=10;

cout<<(b&&c)<<endl;//1

int d=10;

int e=0;

cout<<(d&&e)<<endl;//0

int f=0;

int g=0;

cout<<(f&&g)<<endl;//0

//或 ||

int h=10;

int i=10;

cout<<(h||i)<<endl;//1

int j=0;

int k=10;

cout<<(j||k)<<endl;//1

system("pause");

return 0;

}

## 程序流程编号

### 选择结构???

#### if语句

#include<iostream>

using namespace std;

int main()

{

while(1)

{

cout<<"请输入您的高考分数：";

double score=0;

cin>>score;

if(score>=700&&score<=750)

{

cout<<"恭喜您被清华大学录取。"<<endl;

}

else if(score<700&&score>586)

{

cout<<"恭喜您被南京理工大学录取。"<<endl;

}

else if(score>0&&score<=586)

{

cout<<"恭喜您被汶川中学录取。"<<endl;

}

else

{

cout<<"输入无效。"<<endl;

}

cout<<"输入0退出程序，输入任意键继续使用"<<endl;//为什么？？？

double select;

cin>>select;

if(select==0)

{

break;

}

else if(select==1)

{

cout<<"欢迎您继续使用。"<<endl;

}

}

//选择结构 多行if语句

#include<iostream>

using namespace std;

int main()

{

//用户输入分数，如果分数大于586，视为考上南京理工大学，在屏幕上输出，如果不大于586，则输出恭喜你喜提阿师专录取通知书

//1、输入考试分数

int score=0;

cout<<"请输入你的高考分数"<<endl;

cin>>score;

//2、提示用户输入分数

cout<<"请你的高考分数为："<<score<<endl;

//3、判断

if(score>586)

{

cout<<"恭喜你考上南京理工大学"<<endl;

}

else

{

cout<<"恭喜你喜提阿师专录取通知书"<<endl;

}

system("pause");

return 0;

}

//选择结构 多条件if语句

#include<iostream>

using namespace std;

int main()

{

//一般，score在左

//1、提示用户输入分数

int score=0;

cout<<"请输入你的高考分数"<<endl;

cin>>score;

//2、显示用户输入的分数

cout<<"你输入的分数为："<<score<<endl;

//3、判断

if(score>750)

{

cout<<"输入分数无效"<<endl;

}

else if(score>700)

{

cout<<"恭喜你考上清华大学"<<endl;

}

else if(score>650)

{

cout<<"恭喜你考上电子科技大学"<<endl;

}

else if(score>586)

{

cout<<"恭喜你考上南京理工大学"<<endl;

}

else if(score>0)

{

cout<<"就你这成绩还是去工地上搬砖吧!"<<endl;

}

else

{

cout<<"输入分数无效"<<endl;

}

system("pause");

return 0;

}

//选择结构 嵌套if语句

#include<iostream>

using namespace std;

int main()

{

//一般，score在左

//1、提示用户输入分数

int score=0;

cout<<"请输入你的高考分数"<<endl;

cin>>score;

//2、显示用户输入的分数

cout<<"你输入的分数为："<<score<<endl;

//3、判断

if(score>750)

{

cout<<"输入分数无效"<<endl;

}

else if(score>586)

{

cout<<"恭喜你考上大学"<<endl;

if(score>700)

{

cout<<"你被清华大学录取"<<endl;

}

else if(score>650)

{

cout<<"你被电子科技大学录取"<<endl;

}

else

{

cout<<"你被南京理工大学录取"<<endl;

}

}

else if(score>0)

{

cout<<"就你这成绩还是去工地上搬砖吧!"<<endl;

}

else

{

cout<<"输入分数无效"<<endl;

}

//选择结构 练习题

#include<iostream>

using namespace std;

int main()

{

int a=0;

int b=0;

int c=0;

cout<<"请输入大猪的重量："<<endl;

cin>>a;

cout<<"请输入二猪的重量："<<endl;

cin>>b;

cout<<"请输入三猪的重量："<<endl;

cin>>c;

if(a>b)

{

if(a>c)

{

cout<<"大猪是最重的"<<endl;

}

else if(a<c)

{

cout<<"三猪是最重的"<<endl;

}

else if(a=c)

{

cout<<"大猪和三猪都是最重的"<<endl;

}

}

else if(a<b)

{

if(b>c)

{

cout<<"二猪是最重的"<<endl;

}

else if(b<c)

{

cout<<"三猪是最重的"<<endl;

}

else if(b=c)

{

cout<<"二猪和三猪都是最重的"<<endl;

}

}

else if(a=b)

{

if(a>c)

{

cout<<"大猪和二猪都是最重的"<<endl;

}

else if(a<c)

{

cout<<"三猪是最重的"<<endl;

}

else if(a=c)

{

cout<<"它们都是最重的"<<endl;

}

}

system("pause");

return 0;

}

#### 三目运算符

#include<iostream>

using namespace std;

int main()

{

//三目运算符: ? :

//创建三个变量a b c

int a=10;

int b=20;

int c=0;

c=(a>b?a:b);//将a和b作比较，将变量大的值赋值给变量c

cout<<"c="<<c<<endl;//C++中三目运算符返回的是变量，可以继续赋值,例子如下

(a<b?a:b)=100;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

system("pause");

return 0;

}

//比较三只小猪的体重

#include<iostream>

using namespace std;

int main()

{

int a=0;

int b=0;

int c=0;

int d=0;

int e=0;

cout<<"请输入大猪的体重："<<endl;

cin>>a;

cout<<"请输入二猪的体重："<<endl;

cin>>b;

cout<<"请输入三猪的体重："<<endl;

cin>>c;

d=(a>b?a:b);

e=(d>c?d:c);

cout<<"体重最重的是："<<e<<"kg"<<endl;

system("pause");

return 0;

}

#### Switch语句

#include<iostream>

using namespace std;

int main()

{

//switch语句

//给电影打分：9~10 经典 7~8 非常好 5~6一般 5以下 烂片

//1、提示用户给电影打分

int score=0;//只能是int或者char

cout<<"请给电影打分:";

//2、让用户开始打分

cin>>score;

cout<<"您的评分为："<<score<<"分"<<endl;

//3、根据用户的打分结果输出相应评价

switch(score)

{

case 10:

cout<<"您认为该电影是经典电影"<<endl;

break;//退出当前分支

case 9:

cout<<"您认为该电影是经典电影"<<endl;

break;

case 8:

cout<<"您认为该电影非常棒"<<endl;

break;

case 7:

cout<<"您认为该电影非常棒"<<endl;

break;

case 6:

cout<<"您认为该电影一般"<<endl;

break;

case 5:

cout<<"您认为该电影一般"<<endl;

break;

default:

cout<<"您认为这是一部烂片"<<endl;

}

/\*if 和 switch 语句的区别

switch缺点：数据类型只能判断整型或字符型，不能判断一个区间，不能是实型、浮点型

switch优点：结构清晰，执行效率高

\*/

system("pause");

return 0;

}

### 循环结构

#### while语句

#include<iostream>

using namespace std;

int main()

{

//while循环

//打印0~9这十个数字

/\*笨方法

int num = 0;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

num++;

cout<<num<<endl;

\*/

int num = 0;

while(num<10) //写循环时，避免进入死循环

{

cout<<num<<endl;

num++;

}

system("pause");

return 0;

}

练习题

#include<iostream>

using namespace std;

#include<ctime>//设置time系统时间头文件

int main()

{

//while循环练习：系统随机生成一个1~100的数字，用户猜对了过关，猜错了则提示用户猜大了亦或是猜小了，直至用户猜对

//添加随机数种子，利用当前系统时间生成随机数，防止每次生成的数字一样

srand((unsigned int)time(NULL));

//1、系统随机生成一个数

int num=rand()%100+1;

//2、玩家进行猜测

cout<<"请输入您猜测的数字"<<endl;

//3、判断玩家是否猜对

while(1)

{

int val=0;

cin>>val;

if(num>val)

{

cout<<"您猜测的数过小"<<endl;

}

else if(num<val)

{

cout<<"您猜测的数过大"<<endl;

}

else if(num=val)

{

cout<<"恭喜您回答正确"<<endl;

break;//退出循环

}

}

system("pause");

return 0;

}

#### do…while语句

#include<iostream>

using namespace std;

int main()

{

//do...while语句:先执行，再循环 while语句：先循环，再执行

//输入0~9这10个数字

int num = 0 ;

do

{

cout<<num<<endl;

num++;

}

while(num<10);

system("pause");

return 0;

}

练习：寻找水仙花数

#include<iostream>

using namespace std;

int main()

{

//1、打印所有的三位数

int num=100;

do

{

int a=0;//个位

int b=0;//十位

int c=0;//百位

a=num%10;//获取个位数字

b=num/10%10;//获取十位数字

c=num/100;//获取百位数字

if(a\*a\*a+b\*b\*b+c\*c\*c==num)

{

cout<<num<<endl;

num++;

}

while(num<1000);

//2、从所有三位数中找到水仙花数

//如果是水仙花,则打印

cout<<"水仙花数有"<<num<<endl;

system("pause");

return 0;

}

#### for循环

#include<iostream>

using namespace std;

int main()

{

//for循环

//打印0~9

for(int i=0;/\*起始表达式，创建起始值，不参加循环\*/i<10;/\*条件表达式\*/i++/\*末尾表达式\*/)

{

cout<<i<<endl;

}

system("pause");

return 0;

}

练习：敲桌子案例

#include<iostream>

using namespace std;

int main()

{

//敲桌子：从1到100，如果数字个位数含7，或十位数含7，或该数字是7的倍数，则打印“敲桌子”，其余数字正常打印

//1、先输入1~100

for(int i=1;i<100;i++)

{

//2、寻找特殊数字

if(i%7==0||i%10==7||i/10==7)//如果是特殊数字，则打印“敲桌子”

{

cout<<"敲桌子"<<endl;

}

else //如果不是特殊数字，则正常打印

{

cout<<i<<endl;

}

}

system("pause");

return 0;

}

#### 嵌套循环

#include<iostream>

using namespace std;

int main()

{

//利用嵌套循环实现10\*10的#图

//打印一行#图

//外层执行一次，内层执行一周

for(int i=1;i<=10;i++)//外层循环

{

for(int j=1;j<=10;j++)//内层循环

{

cout<<"# ";

}

cout<<endl;

}

system("pause");

return 0;

}

#include<iostream>

using namespace std;

int main()

{

for(int i=1;i<10;i++)

{

for(int j=1;j<=i;j++)

{

{

cout<<j<<"\*"<<i<<"="<<i\*j<<"\t";

}

}

cout<<endl;

}

system("pause");

return 0;

}

### 跳转语句

#### break语句

#include<iostream>

using namespace std;

int main()

{

//break的使用时机

//1、出现在switch语句中

cout<<"请选择副本的难度"<<endl;

cout<<"1.简单 "<<"2.中等 "<<"3.困难 "<<endl;

int select=0;//创建选择结构的变量

cin>>select;//等待用户输入

switch(select)

{

case 1:

cout<<"您选择的是普通难度"<<endl;

break;

case 2:

cout<<"您选择的是中等难度"<<endl;

break;

case 3:

cout<<"您选择的是困难难度"<<endl;

break;

default:

cout<<"输入无效"<<endl;

break;

}

//2、出现在循环语句中

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

{

if(i>5)

{

break;

}

cout<<i<<endl;

}

//3、出现在嵌套语句中

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

{

for(int j=0;j<10;j++)

{

if(j==5)

{

break;

}

cout<<"\* ";

}

cout<<endl;

}

system("pause");

return 0;

}

#### continue语句

#include<iostream>

using namespace std;

int main()

{

//continue语句

for(int i=0;i<=100;i++)

{

if(i%2==0)//如果是奇数则输出，偶数不输出

{

continue;//可以筛选条件，执行到此就不再向下执行，执行下循环一次，而break则不会继续执行下一次循环

}

cout<<i<<endl;

}

system("pause");

return 0;

}

#### goto语句

#include<iostream>

using namespace std;

int main()

{

//goto语句:尽量不使用goto语句，方便阅读

cout<<"1"<<endl;

cout<<"2"<<endl;

goto FLAG;//忽略3和4，直接到5

cout<<"3"<<endl;

cout<<"4"<<endl;

FLAG:

cout<<"5"<<endl;

cout<<"6"<<endl;

system("pause");

return 0;

}

## 数组

### 一维数组

#### 定义方式

#include<iostream>

using namespace std;

int main()

{

//数组

/\*

1.数组类型 数组名[数组长度]

2.数组类型 数组名[数组长度]={值1，值2...}

3.数组类型 数组名[]={值1，值2...}

\*/

//1.数组类型 数组名[数组长度]

//数组名不要和变量名重合

int arr[5];

//给数组中的元素进行赋值

arr[0]=10;//数组元素的下标是从0开始

arr[1]=20;

arr[2]=30;

arr[3]=40;

arr[4]=50;

//访问数据元素

cout<<arr[0]<<endl;

cout<<arr[1]<<endl;

cout<<arr[2]<<endl;

cout<<arr[3]<<endl;

cout<<arr[4]<<endl;

//2.数组类型 数组名[数组长度]={值1，值2...}

int arr2[5]={1,2,3,4,5};//如果在初始化数据时没有全部填写，会用0代替

cout<<arr2[0]<<endl;

cout<<arr2[1]<<endl;

cout<<arr2[2]<<endl;

cout<<arr2[3]<<endl;

cout<<arr2[4]<<endl;

//亦或是

for(int i=0;i<5;i++)

{

cout<<arr[i]<<endl;

}

//3.数组类型 数组名[]={值1，值2..

//定义数组时，必须要有初始长度

int arr3[]={90,80,70,60};

for(int i=0;i<4;i++)

{

cout<<arr3[i]<<endl;

}

system("pause");

return 0;

}

#### 数组名

#include<iostream>

using namespace std;

int main()

{

/\*一维数组名称的用途：

1.可以统计整个数组内存中的长度

2.可以获取数组在内存中的首地址

\*/

int arr[5]={1,2,3,4,5};

cout<<"整个数组所占内存："<<sizeof(arr)<<endl;

cout<<"每个元素占用内存空间为："<<sizeof(arr[0])<<endl;

cout<<"数组中的元素个数为："<<sizeof(arr)/sizeof(arr[0])<<endl;

cout<<"该数组在内存中的首地址为："<<(int)arr<<endl;//输入(int)将十六进制转换为十进制

cout<<"该数组中第一个元素地址为："<<(int)&arr[1]<<endl;//&不可少

/\*arr=100;\*/ //数组名是常量，不可以进行赋值操作

system("pause");

return 0;

}

#### 案例

#include<iostream>

using namespace std;

int main()

{

//比较五只小猪的体重的最大值：

//1.创建五只小猪体重的数据

int max=0;

cout<<"请输入每只小猪的体重"<<endl;

int arr[5];

cout<<"请输入第一只小猪的体重："<<endl;

cin>>arr[0];

cout<<"请输入第二只小猪的体重："<<endl;

cin>>arr[1];

cout<<"请输入第三只小猪的体重："<<endl;

cin>>arr[2];

cout<<"请输入第四只小猪的体重："<<endl;

cin>>arr[3];

cout<<"请输入第五只小猪的体重："<<endl;

cin>>arr[4];

//2.从数组中找到最大值

for(int i=1;i<6;i++)

{

if(max<arr[i])

{

max=arr[i];

}

}

//3.打印最大值：

cout<<"体重最重的是:"<<max<<endl;

system("pause");

return 0;

}

//另解：称小猪

#include<iostream>

using namespace std;

int main()

{

int max=0;

int arr[6];

cout<<"请分别输入6只小猪的体重:"<<endl;

cin>>arr[0]>>arr[1]>>arr[2]>>arr[3]>>arr[4]>>arr[5];

for(int i=0;i<6;i++)

{

if(max<arr[i])

{

max=arr[i];

}

}

cout<<"最重的小猪有"<<max<<"kg"<<endl;

system("pause");

return 0;

}

//元素逆置

#include<iostream>

using namespace std;

int main()

{

//实现数组元素逆置

//1.创建数组

int arr[5]={1,3,2,5,4};

cout<<"元素逆置前的结果：";

for(int i=0;i<5;i++)

{

cout<<arr[i];

}

/\*2.实现逆置

a.记录起始下标的位置

b.记录结束下标的位置

c.起始下标与结束下标位置互换

d.起始位置++，结束位置--

e.循环执行a操作，直到起始位置>=结束位置\*/

int start=0;

int end=sizeof(arr)/sizeof(arr[0])-1;

while(start<end)

{

//实现元素互换

int temp=arr[start];

arr[start]=arr[end];//"三角变换"

arr[end]=temp;

start++;end--;

}

cout<<endl;

//3.打印逆置后的数组

cout<<"逆置后的数字元素为：";

for(int i=0;i<5;i++)

{

cout<<arr[i];

}

cout<<endl;

system("pause");

return 0;

}

#### 冒泡排序

#include<iostream>

using namespace std;

int main()

{

//利用冒泡排序实现升序序列

int arr[9]={4,2,8,0,5,7,1,3,9};

cout<<"排序前的元素为：";

for(int i=0;i<9;i++)

{

cout<<arr[i]<<" ";

}

cout<<endl;

//开始冒泡排序

//每轮对比数=总元素个数-排序轮数-1

for(int i=0;i<9-1;i++)//排序总轮数=总元素个数-1

{

//内层循环对比：次数=元素个数-当前轮数-1

for(int j=0;j<9-1-i;j++)

{

//如果第一个数字比第二个数字大，实现交换两个数字

if(arr[j]>arr[j+1])

{

int temp=arr[j];

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

cout<<"排序后的元素为：";

for(int j=0;j<9;j++)

{

cout<<arr[j]<<" ";

}

cout<<endl;

system("pause");

return 0;

}

### 二维数组

#### 定义方式

#include<iostream>

using namespace std;

int main()

{

/\*二维数组的定义方式

1.数据类型 数组名[行数][列数]

2.数据类型 数组名[行数][列数]={{数据1，数据2}，{数据3，数据4}}

3.数据类型 数组名[行数][列数]={数据1，数据2，数据3，数据4}

4.数据类型 数组名[][列数]={数据1，数据2，数据3，数据4}

\*/

//1.数据类型 数组名[行数][列数]

/\*int arr[2][3];//先行再列

arr[0][0]=1;

arr[0][1]=2;

arr[0][2]=3;

arr[1][0]=4;

arr[1][1]=5;

arr[1][2]=6;\*/

/\*

cout<<arr[0][0]<<endl;

cout<<arr[0][1]<<endl;

cout<<arr[0][2]<<endl;

cout<<arr[1][0]<<endl;

cout<<arr[1][1]<<endl;

cout<<arr[1][2]<<endl;

\*/

//外层循环打印行数，内层循环打印列数

/\*for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

cout<<arr[i][j]<<endl;

}

}\*/

//2.数据类型 数组名[行数][列数]={{数据1，数据2}，{数据3，数据4}}

int arr2[2][3]=

{

{1,2,3},

{4,5,6}

};

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

cout<<arr2[i][j]<<" ";

}

cout<<endl;

}

//3.数据类型 数组名[行数][列数]={数据1，数据2，数据3，数据4}

/\*int arr3[2][3]={1,2,3,4,5,6};

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

cout<<arr3[i][j]<<" ";

}

cout<<endl;

}

\*/

//4.数据类型 数组名[][列数]={数据1，数据2，数据3，数据4}

/\*

int arr4[][3]={1,2,3,4,5,6};

for(int i=0;i<2;i++)

{

for(int j=0;j<3;j++)

{

cout<<arr3[i][j]<<" ";

}

cout<<endl;

}

\*/

system("pause");

return 0;

}

#### 数组名

#include<iostream>

using namespace std;

int main()

{

int arr[2][3]=

{

{1,2,3},

{4,5,6}

};

cout<<"二维数组占用的内存空间为："<<sizeof(arr)<<endl;

cout<<"二维数组第一行占用的内存为："<<sizeof(arr[0])<<endl;

cout<<"二维数组第一个元素占用内存为："<<sizeof(arr[0][1])<<endl;;

cout<<"二维数组元素的个数为："<<sizeof(arr)/sizeof(arr[0][0])<<endl;

cout<<"二维数组占用的行数为："<<sizeof(arr)/sizeof(arr[0])<<endl;

cout<<"二维数组占用的列数为："<<sizeof(arr[0])/sizeof(arr[0][0])<<endl;

cout<<"二维数组的首地址为："<<arr<<endl;//16进制

cout<<"二维数组的首地址为："<<(int)arr<<endl;//10进制

cout<<"二维数组的第一行首地址为："<<arr<<endl;

cout<<"二维数组的第二个元素首地址为："<<&arr[0][1]<<endl;

system("pause");

return 0;

}

#### 案例

#include<iostream>

using namespace std;

#include <string>

int main()

{

//二维数组案例-统计成绩

//1.创建二维数组

int scores[3][3]=

{

{100,100,100},

{90,50,100},

{60,70,80}

};

//2.统计每个人的总分数

string names[3]={"张三","李四","王二麻子"};

for(int i=0;i<3;i++)//外层控制行数

{

int sum=0;

for(int j=0;j<3;j++)//内层控制列数

{

sum+=scores[i][j];

cout<<scores[i][j]<<" ";

}

cout<<names[i]<<i+1<<"的总分数为："<<sum;

cout<<endl;

}

system("pause");

return 0;

}

## 函数

### 函数的定义

#include<iostream>

using namespace std;

/\*

语法：

返回值类型 函数名 参数列表

{

函数体语句

return 表达式

}

\*/

int add (int num1,int num2)

{

int sum=num1+num2;

return sum;

}

/\*对比

int main()

{

system("pause");

return 0;

}

\*/

### 函数的调用

#include<iostream>

using namespace std;

//函数定义的时候，num1和num2并没有实际数据，它只是形式上的参数，简称形参。

int add(int num1,int num2)

{

int sum=num1+num2;

return sum;

}

int main()

{

//main函数中调用add函数

int a=100;

int b=2000;

//函数调用的语法：函数名称（参数）

//a,b简称为实参（实际参数）

//当调用函数的时候，实参的值会传递给形参

int c=add(a,b);

cout<<"c="<<c<<endl;

system("pause");

return 0;

}

### 值传递

#include<iostream>

using namespace std;

//值传递

//定义函数，实现两个数字进行交换函数

//如果函数不需要返回值，声明是可以写void

void swap(int num1,int num2)

{

cout<<"交换前："<<endl;

cout<<"num1="<<num1<<endl;

cout<<"num2="<<num2<<endl;

int temp=num1;

num1=num2;

num2=temp;

cout<<"交换后："<<endl;

cout<<"num1="<<num1<<endl;

cout<<"num2="<<num2<<endl;

//return; //返回值不需要的时候，可以不写return

}

int main()

{

int a=10;

int b=20;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

//当我们做值传递的时候，函数形参发生改变，并不会影响实参

swap(a,b);

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

system("pause");

return 0;

}

### 常见形式

#include<iostream>

using namespace std;

/\*

常见函数有4种样式：

1、无参无返

2、有参无返

3、无参有返

4、有参有返

\*/

//1、无参无返

void test01()

{

cout<<"这是test01"<<endl;

//不需要写return;

}

//2、有参无返

void test02(int a)

{

cout<<"这是test02 a="<<a<<endl;

}

//3、无参有返

int test03()

{

cout<<"这是test03 ";

return 1000;

}

//4、有参有返

int test04(int a)

{

cout<<"这是test04 a="<<a<<endl;

return a;

}

int main()

{

//无参无返的调用

test01();

//有参无返的调用

test02(100);

//无参有返的调用

int num1=test03();

cout<<"num1="<<num1<<endl;

//有参有返的调用

int num2=test04(10000);

system("pause");

return 0;

}

### 函数的声明

#include<iostream>

using namespace std;

//函数声明

//比较函数，实现两个整型数字进行比较，返回较大的值

//提前告诉编译器函数的存在，可以利用函数的声明

//函数的声明:

int max(int a,int b);

//声明可以写多次，但是定义只能写一次

int main()

{

int a=10;

int b=20;

cout<<max(a,b)<<endl;

system("pause");

return 0;

}

//定义

int max(int a,int b)

{

return a>b?a:b;

}

### 函数的分文件编写

//1、创建 .h后缀名的头文件 //文件名无所谓

//2、创建 .cpp后缀名的源文件

//3、在头文件中写函数声明 //#include<iostream> using namespace std;也要写

//4、在源文件中写函数的定义 // #include"swap.h"要写 #include<iostream> using namespace std;也要写

## 指针

### 指针的定义和使用

//指针的作用：指针就是某个内存低地址编号

//指针与内存的区别：指针储存的是内存的地址，变量储存的是内存的值

//须重复观看该节视频内容

//将p理解为指针（地址），\*p理解为地址的内容(如：a)

#include<iostream>

using namespace std;

int main()

{

//1、定义指针

int a=0;

//指针定义的语法： 数据类型\*指针变量名：

int \* p;//指针的数据类型与变量的数据类型必须一样 //\*用于取内容

//让指针记录变量a的地址(建立指针与变量的关系)

p=&a;//&为取址符号

cout<<"a的地址为："<<&a<<endl;

cout<<"指针p="<<p<<endl;

//2、使用指针

//可以通过解引用的方式找到指针指向的内存

//指针前加\*代表解引用，找到指针指向的内存

\*p=1000;

cout<<"a="<<a<<endl;//指针可修改内存

cout<<"\*p="<<\*p<<endl;//通过指针访问内存

system("pause");

return 0;

}

### 指针所占的内存空间

#include<iostream>

using namespace std;

//int\*是指针的标志

int main()

{

//所有数据类型的指针在32位操作系统下占4字节空间大小，在64位操作系统下占8字节空间大小

int a=10;

int \*p;p=&a;//可以简化为int \*p=&a

cout<<"sizeof(int\*)="<<sizeof(int\*)<<endl;//cout<<sizeof(p);也可以

cout<<"sizeof(float\*)="<<sizeof(float\*)<<endl;

cout<<"sizeof(double\*)="<<sizeof(double\*)<<endl;

cout<<"sizeof(long long\*)="<<sizeof(long long\*)<<endl;

system("pause");

return 0;

}

### 空指针

#include<iostream>

using namespace std;

int main()

{

//空指针:指针变量p指向内存中编号为0的空间的指针

//1、空指针用于给指针变量进行初始化

int \*p=NULL;

//2、空指针不能进行访问

//空指针不能进行访问（0~255之间的内存编号是系统占用的，因此不可以访问）

//\*p=100;

system("pause");

return 0;

}

### 野指针

//该视频须多次观看

#include<iostream>

using namespace std;

int main()

{

//野指针:指针变量指向非法空间

//在程序中，尽量避免野指针

int \*p=(int \*)520;//(int \*)表编号示后面的内容是地址

cout<<(int)p<<endl;//可以访问地址

cout<<\*p<<endl;//不可以访问地址上的值

\*p=10;//错误操作。

//空指针和野指针都不是我们申请的空间，不要随意访问。

system("pause");

return 0;

}

### const修饰指针

#include<iostream>

using namespace std;

int main()

{

//1.const修饰指针

int a=10;

int b=10;

const int \*p=&a;//指针指向的值不能改，指针的指向可以改

//\*表示找内容 const修饰\*后表示找的内容不能变

//\*p=20; 错误 p=&b; 正确

//2.const修饰常量

int \*const p2=&a;//指针的指向不可以改，指针指向的值可以改

//p2是一个地址，const修饰p2后使得指向的地址不能改变

//\*p2=100; 正确 p2=&b; 错误

//3.const修饰指针和常量

const int \*const p3=&a;//指针的指向和指针指向的值都不能修改

//\*p3=100; 错误 p3=&b; 错误

system("pause");

return 0;

}

### 指针与数组

#include<iostream>

using namespace std;

int main()

{

//指针和数组

//利用指针访问数组中的元素

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

cout<<"第一个元素是："<<arr[0]<<endl;

int \* p=arr;//arr就是数组的首地址

cout<<"利用指针访问第一个元素："<<\*p<<endl;

p++;//让指针向后偏移4个字节

cout<<"利用指针访问第二个元素："<<\*p<<endl;

cout<<"利用指针访问所有元素。"<<endl;

int \*p2=arr;

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

{

//cout<<"第"<<i+1<<"个元素是："<<arr[i]<<endl;

cout<<"第"<<i+1<<"个元素是："<<\*p2<<endl;//也可以写作\*p2++;

p2++;

}

system("pause");

return 0;

}

### 指针与函数（地址传递与值传递）

#include<iostream>

using namespace std;

//实现两个数字进行交换

void swap01(int a,int b)

{

int temp=a;

a=b;

b=temp;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

}

void swap02(int \*p1,int \*p2)

{

int temp=\*p1;

\*p1=\*p2;

\*p2=temp;

}

int main()

{

//指针和函数

//1、值传递

int a=10;

int b=20;

swap01(a,b);

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

//2、地址传递

//如果是地址传递，可以修饰实参

swap02(&a,&b);

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

system("pause");

return 0;

}

### 案例

#include<iostream>

using namespace std;

//冒泡排序的函数 参数1 数组的首地址 参数2 数组的长度

void bubbleSort(int \* arr,int len)

{

for(int i=0;i<len-1;i++)

{

for(int j=0;j<len-1-i;j++)

{

if(arr[j]>arr[j+1])

{

int temp=arr[j];//疑问：传入的arr是指针，为什么arr[]是变量？

arr[j]=arr[j+1];

arr[j+1]=temp;

}

}

}

}

//打印数组

void printArray(int \*arr,int len)

{

for(int i=0;i<len;i++)

{

cout<<arr[i]<<endl;

}

}

int main()

{

//案例描述：封装一个函数，利用冒泡排序，实现对整型数组的升序

//1、创建数组

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

//求数组的长度

int len=sizeof(arr)/sizeof(arr[0]);

//2、创建函数，实现冒泡排序

bubbleSort(arr,len);

//3、打印排序后的数组

printArray(arr,len);

system("pause");

return 0;

}

## 结构体

### 结构体的定义和使用

#include<iostream>

using namespace std;

#include<string>

//结构体属于用户自定义的数据类型，允许用户存储不同的数据

//系统自带的数据类型有：整型、字符型等

//1、创建学生数据类型 学生包括（姓名、年龄、分数）

//自定义数据类型就是一些类型集合组成的一个类型

//语法： struct 结构体名 {结构体成员列表};

struct student

{

//成员列表

//姓名

string name;

//年龄

int age;

//分数

int score;

}s3; //2.3 在定义结构体时顺便创建结构体变量

//2、通过学生类型创建具体学生

//2.1 struct student s1

//2.2 struct student s2={成员1.成员2...}

//2.3 在定义结构体时顺便创建结构体变量

int main()

{

//2.1 struct student s1

//结构体创建时关键字struct可以不写

struct student s1; //struct student是数据类型 s1是变量名

//给s1属性赋值，通过 . 访问结构体变量中的属性

s1.name="张三";

s1.age=18;

s1.score=100;

cout<<"姓名："<<s1.name <<" 年龄:"<<s1.age <<" 分数："<<s1.score<<endl;

//2.2 struct student s2={成员1.成员2...}

struct student s2={"李四",19,80};//struct可省略

cout<<"姓名："<<s2.name <<" 年龄:"<<s2.age <<" 分数："<<s2.score<<endl;

//2.3 在定义结构体时顺便创建结构体变量（见前面） 不常用

s3.name="王五";

s3.age=20;

s3.score=90;

cout<<"姓名："<<s3.name <<" 年龄:"<<s3.age <<" 分数："<<s3.score<<endl;

system("pause");

return 0;

}

### 结构体数组

#include<iostream>

using namespace std;

#include<string>

//结构体数组

//1、定义结构体

struct student

{

string name;

int age;

int score;

};

int main()

{

//2、创建结构体数组

struct student stuArray[3]=

{

{"张三",18,100},

{"李四",20,93},

{"王五",30,99},

};

//3、给结构体数组中的元素修改赋值

stuArray[2].name="赵六";

stuArray[2].age =80;

stuArray[2].score=56;

//4、遍历结构体数组

for(int i=0;i<3;i++)

{

cout<<"姓名："<<stuArray[i].name<<" 年龄："<<stuArray[i].age<<" 分数："<<stuArray[i].score <<endl;;

}

system("pause");

return 0;

}

### 结构体指针

#include<iostream>

using namespace std;

#include<string>

//结构体指针

//定义 学生 结构体

struct student

{

string name;

int age;

int score;

};

int main()

{

//1、创建 学生 结构体变量

struct student s={"张三",17,100};

//2、通过指针指向结构体变量

struct student \*p=&s;//不能使用int数据类型的指针

//3、通过指针访问结构体变量中的数据

p->name ="李海龙";

cout<<"姓名："<<p->name<<" 年龄:"<<p->age <<" 分数："<<p->score<<endl;

//想通过结构体指针访问结构体中的属性，需要利用 ->

system("pause");

return 0;

}

### 结构体嵌套结构体

#include<iostream>

using namespace std;

#include<string>

//定义 学生 结构体

struct student

{

string name;

int age;

int score;

};

//定义 老师 结构体

struct teacher

{

int id;//老师的ID

string name;//老师的姓名

int age;//老师的年龄

struct student stu;//老师辅导的学生

};

int main()

{

//结构体嵌套结构体

//创建老师

teacher t;

t.id=10000;

t.name="老王";

t.age=66;

t.stu.name="小李";

t.stu.age=19;

t.stu.score =98;

cout<<"老师的姓名："<<t.name<<" 老师的职工编号："<<t.id<<" 老师的年龄："<<t.age<<endl;

cout<<" 老师辅导学生的姓名："<<t.stu.name<<" 老师辅导学生的年龄："<<t.stu.age<<" 老师辅导学生的分数："<<t.stu.score<<endl;

system("pause");

return 0;

}

### 结构体作函数参数

#include<iostream>

using namespace std;

#include<string>

//定义学生的结构体

struct student

{

string name;

int age;

int score;

};

//打印学生函数信息

//1.值传递

void printstudent1(struct student s)

{

cout<<"子1函数中 姓名："<<s.name<<" 年龄："<<s.age<<" 分数："<<s.score<<endl;

}

//2.地址传递

void printstudent2(struct student \*p)//引入指针变量接收地址&s

{

cout<<"子2函数中 姓名："<<p->name<<" 年龄："<<p->age<<" 分数："<<p->score<<endl;

}

int main()

{

//结构体作为函数的参数

//将 学生 传入到一个参数中，打印学生身上的所有信息

//创建结构体变量

struct student s;

s.name="张三";

s.age =20;

s.score =90;

struct student \*p=&s;

printstudent1(s);

printstudent2(&s);//地址传递，传递地址

cout<<"main函数中 姓名："<<s.name<<" 年龄："<<s.age<<" 分数："<<s.score<<endl;

system("pause");

return 0;

}

### 结构体中const的使用案例

#include<iostream>

using namespace std;

#include<string>

//const的使用场景

//定义 学生 结构体

struct student

{

string name;

int age;

int score;

};

//将函数中的形参改为指针能减少内存空间，而且不会产生新的副本

void printstudent(const student \*s)

{

//加入const之后则不能修改s，可以防止误操作

// s->age=200;

cout<<"姓名："<<s->name<<" 年龄："<<s->age<<" 分数："<<s->score<<endl;

}

int main()

{

//创建 学生 结构体变量

struct student s={"张三",16,89};

//通过函数打印结构体变量的信息

printstudent(&s);

cout<<"张三的年龄为："<<s.age<<endl;

system("pause");

return 0;

}

## 通讯录系统

### 通讯录的制作

#include<iostream>

using namespace std;

#include<string>

#define MAX 1000

void menu()//菜单界面

{

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*1、添加联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*2、显示联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*3、删除联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*4、查找联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*5、修改联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*6、清空联系人\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*0、退出通信录\*\*\*\*\*\*"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

}

//设计联系人结构体

struct person

{

string name;

int age;

int sex;//1表示男 2表示女

string phone;

string addr;

};

//设计通讯录结构体

struct book

{

//通讯录中保存联系人的数组

struct person personarr[MAX];

//通讯录中当前记录联系人的个数

int size;

};

//1、添加联系人 函数

void addperson(book \*abs)

{

//判断通讯录是否已满，如果满了就不再添加

if(abs->size ==MAX)

{

cout<<"通讯录已满，无法添加"<<endl;

return;

}

else

{

//添加具体联系人

string name;

cout<<"请输入姓名：";

cin>>name;

abs->personarr [abs->size].name =name; //将输入的name存放到abs的personarr[i]的name中，其中i=abs->size;

//性别

cout<<"请输入性别（1代表男，2代表女）：";

while(1)

{

int sex=0;

cin>>sex;

if(sex==1||sex==2)//如果输入1或2，则退出循环

{

abs->personarr [abs->size ].sex =sex;

break;

}

cout<<"输入有误，请重新输入"<<endl;

}

//年龄

cout<<"请输入年龄：";

int age=0;

cin>>age;

abs->personarr [abs->size ].age =age;

//电话

cout<<"请输入联系电话：";

string phone;

cin>>phone;

abs->personarr [abs->size ].phone =phone;

//家庭住址

cout<<"请输入家庭住址：";

string addr;

cin>>addr;

abs->personarr [abs->size ].addr =addr;

//更新通讯录中的人数

abs->size ++;

cout<<"添加成功！";

system("pause");

system("cls");//清屏操作,清除该函数的所有内容

}

}

//2、显示联系人

void showperson(book \*abs)

{

//判断通讯录中人数是否为0。如果为0，提示为空。如果不为0，显示联系人

if(abs->size ==0)

{

cout<<"联系人为空。"<<endl;

}

else

{

for(int i=0;i<abs->size ;i++)

{

cout<<"姓名："<<abs->personarr [i].name <<"\t";

cout<<"性别："<<(abs->personarr [i].sex==1?"男":"女" )<<"\t";//三目运算，妙！

cout<<"年龄："<<abs->personarr [i].age <<"\t";

cout<<"联系电话："<<abs->personarr [i].phone <<"\t";

cout<<"家庭住址："<<abs->personarr [i].addr <<"\t";

cout<<endl;

}

}

system("pause");

system("cls");

}

//检测联系人是否存在，如果存在，返回联系人所在数组中的具体位置，不存在返回-1

//参数1代表通讯录 参数2代表对比姓名

int exist(book \*abs,string name)

{

for(int i=0;i<abs->size ;i++)

{

if(abs->personarr [i].name ==name)

{

return i;//找到了，返回这个人在数组中的编号

}

}

return -1;//没有找到，返回-1。

}

//3、删除联系人

void deleteperson(book \*abs,string name)

{

//ret=-1 未查到

//ret!=-1 找到了

int ret=exist(abs,name);

if(ret!=-1)

{

//查找到人，要进行删除操作。

for(int i=ret;i<abs->size ;i++)

{

//数据前移操作

abs->personarr[i]=abs->personarr[i+1];

}

cout<<"删除成功！";

abs->size --;//更新通讯录中的人员数

system("pause");

system("cls");

}

else

{

cout<<"查无此人。";

system("pause");

system("cls");

}

}

//4、封装 查找联系人 函数

void findperson(book \*abs)

{

cout<<"请输入您要查找的联系人姓名：";

string name;

cin>>name;

//判断指定的联系人是否存在

int ret=exist(abs,name);

if(ret!=-1)//找到联系人

{

for(int i=0;i<abs->size ;i++)

{

cout<<"姓名："<<abs->personarr [i].name <<"\t";

cout<<"性别："<<(abs->personarr [i].sex==1?"男":"女" )<<"\t";//三目运算，妙！

cout<<"年龄："<<abs->personarr [i].age <<"\t";

cout<<"联系电话："<<abs->personarr [i].phone <<"\t";

cout<<"家庭住址："<<abs->personarr [i].addr <<"\t";

cout<<endl;

}

}

else//未找到

{

cout<<"查无此人。";

}

system("pause");

system("cls");

}

//5、修改 联系人 函数

void modifyperson(book \*abs)

{

cout<<"请输入您要修改的联系人：";

string name;

cin>>name;

int ret=exist(abs,name);

if(ret!=-1)

{

string name;

cout<<"请输入修改后的姓名：";

cin>>name;

abs->personarr [ret].name =name;

while(1)

{

int sex=0;

cout<<"请选择修改后的性别(1代表男，2代表女)：";

cin>>sex;

if(sex==1||sex==2)

{

abs->personarr [ret].sex =sex;

break;

}

else

{

cout<<"输入有误，请重新输入";

}

}

int age=0;

cout<<"请输入修改后的年龄：";

cin>>age;

abs->personarr [ret].age =age;

int phone=0;

cout<<"请输入修改后的联系电话：";

cin>>phone;

abs->personarr [ret].phone =phone;

string addr;

cout<<"请输入修改后的家庭地址：";

cin>>addr;

abs->personarr [ret].addr =addr;

}

else

{

cout<<"查无此人。";

}

system("pause");

system("cls");

}

//6、封装 清空联系人 函数

void cleanperson(book \*abs)

{

cout<<"一旦清空将无法恢复，您确定清空吗？"<<endl;

cout<<"1、确定 2、取消 (选择序号) "<<endl;

int select=0;

cin>>select;

while(1)

{

switch(select)

{

case 1:

{

abs->size =0;

cout<<"通讯录已清空"<<endl;

system("pause");

system("cls");

return;

}

case 2:

system("pause");

system("cls");

return;

default:

cout<<"输入无效，请重新输入。";

}

}

}

int main()

{

//创建通讯录结构体变量

book abs;

//初始化通讯录中当前人员的个数

abs.size =0;

while(1)

{

menu();

int select=0;

cout<<"请选择编号："<<endl;

cin>>select;

switch(select)

{

case 1://1、添加联系人

addperson(&abs);//利用地址传递可以修饰实参

break;

case 2://2、显示联系人

showperson(&abs);

break;

case 3://3、删除联系人

{

cout<<"请输入删除联系人的姓名：";//case下面语句过多，须用{}将其括起来。

string name;

cin>>name;

deleteperson(&abs,name);

}

break;

case 4://4、查找联系人

findperson(&abs);

break;

case 5://5、修改联系人

modifyperson(&abs);

break;

case 6://6、清空联系人

cleanperson(&abs);

break;

case 0://0、退出通信录

cout<<"欢迎您下次使用"<<endl;

system("pause");

return 0;//结束函数

}

}

system("pause");

return 0;

}

## 内存模型

### 大纲

C++程序在执行时，将内存大方向分为4个区域。

代码区：

全局区：

栈区：

堆区：

在程序编译后，生成exe可执行程序，未执行该程序前分为两个区域。

代码区：

1.存放CPU执行的机器指令，及二进制数字。

2.代码区是共享的，共享的目的是对于被频繁执行的程序，只需在内存中有一份代码即可，节约了资源。

3.代码区是只读的，使其只读的原因是防止程序意外修改其指令。

总结：共享、只读

全局区：

1.全局区包含全局变量、静态变量和常量（全局常量，字符串常量等）。

2.常量区包含字符串常量和const修饰的常量。

3.const修饰的常量分为局部常量和全局常量。注：局部常量不在全局区。

4.该区域的数据在程序结束后由操作系统释放。

栈区：

由编译器自动分配释放，存放函数的参数值，局部变量等。

注意事项：不要返回局部变量的地址，栈区开辟的数据由编译器自动释放。

堆区：

由程序员分配释放，若程序员不释放，程序运行期间堆区内存不释放，程序结束时由操作系统回收。

在C++中主要利用new在堆区开辟内存

### 全局区

#include<iostream>

using namespace std;

int g\_a=10;//全局变量:存放在函数外面的变量

int g\_b=10;

const int c\_g\_a=10;//全局常量:const修饰的全局变量

const int c\_g\_b=10;

int main()

{

//全局区包含全局变量，静态变量和常量

//创建普通的局部变量

int a=10;//函数体内的变量都是局部变量

int b=10;

cout<<"局部变量a的地址为："<<(int)&a<<endl;

cout<<"局部变量b的地址为："<<(int)&b<<endl;

cout<<"全局变量g\_a的地址为："<<(int)&g\_a<<endl;

cout<<"全局变量g\_b的地址为："<<(int)&g\_b<<endl;

//静态变量:在普通变量前加static

static int s\_a=10;

static int s\_b=10;

cout<<"静态变量s\_a的地址为："<<(int)&s\_a<<endl;

cout<<"静态变量s\_b的地址为："<<(int)&s\_b<<endl;

//常量

//字符串常量（""之内的字符都叫做字符串）

cout<<"字符串常量的地址为："<<(int)"hello,world"<<endl;

//const修饰的变量

//const修饰的全局变量

cout<<"全局常量c\_g\_a的地址为："<<(int)&c\_g\_a<<endl;

cout<<"全局常量c\_g\_b的地址为："<<(int)&c\_g\_b<<endl;

//const修饰的局部变量

const int c\_l\_a=10;//c代表const g代表global l代表local

const int c\_l\_b=10;

cout<<"局部常量c\_l\_a的地址为："<<(int)&c\_l\_a<<endl;

cout<<"局部常量c\_l\_b的地址为："<<(int)&c\_l\_b<<endl;

//总结：1.全局变量和静态变量挨在一起，它们和常量有一点点间隙，但也在一个区域中。局部变量与局部常量在一个区域，它们和前三者不在一个区域。

system("pause");

return 0;

}

### 栈区

#include<iostream>

using namespace std;

//栈区数据注意事项---不要返回局部变量的地址

//栈区的数据由编译器管理开辟和释放

int\* func(int b)//形参也放在栈区

{

int a=10;//局部变量：存放在栈区，栈区的数据在函数执行完后自动释放

return &a;//返回局部变量的地址

}

int main()

{

//接收func函数的返回值

int\*p=func(20);

cout<<p<<endl;//第一次编译器防止误操作，顾没有出错

cout<<p<<endl;//乱码，数据出错

cout<<\*p<<endl;

cout<<\*p<<endl;

system("pause");

return 0;

}

/\*难点：为什么不能返回局部变量的地址，却能返回局部变量的值？

返回过程： 函数->寄存区->主函数

如果返回局部变量的值,如a=10;return a;将10返回寄存器，再返回主函数。主函数接收到数据10.同时a已经被释放。

而若返回指针，指针先被放入寄存器，在被主函数接收，同时a被释放，此时能输出a的地址（因为a的地址已经被主函数接收），但是不能访问a的内容，因为a已经被释放

\*/

### 堆区

#include<iostream>

using namespace std;

int \* func()

{

//利用new关键字可以将数据开辟到堆区

//指针本身也是局部变量，存放在栈上，指针指向的数据放在堆区

int\* p=new int(10);//创建好该数据后返回的是堆区的地址，因此创建int \*p接收该地址

return p;

}

int main()

{

//在堆区开辟数据

int \*p=func();

cout<<\*p<<endl;//主函数接收指针p，同时函数中的指针p被释放掉，但是p上面的内容存放在堆区，不会被释放，因此主函数中仍能访问\*p

cout<<\*p<<endl;

cout<<\*p<<endl;

cout<<\*p<<endl;//不会乱码

system("pause");

return 0;

}

### new与delete

#include<iostream>

using namespace std;

//1、new的基本语法

int \* func()

{

//在堆区创建一个整型数据10（也就是在堆区开辟一段内存的过程）

int\* p=new int (10);//new返回的是该数据类型的指针（地址编号） int a;

return p;

}

void test01()

{

int\* p=func();

cout<<\*p<<endl;

cout<<\*p<<endl;

//堆区的数据由程序员管理开辟和释放

//如果想要释放堆区的数据，利用关键字delete

delete p;//注：不是delete \*p.必须记住此搭配。

cout<<\*p<<endl; //该内存已经被释放，再次访问就是非法操作

}

//2、在堆区利用new开辟数组

void test02()

{

//在堆区创建10个元素的整型数据的数组的方法

int \*arr=new int[10];//10代表数组有10个元素，返回这段内存的首地址

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

{

arr[i]=i+100;//给10个元素赋值 100~109

}

for(int j=0;j<10;j++)

{

cout<<"arr[j]="<<arr[j];

cout<<endl;

}

//释放堆区的数组:必须要加[]

delete[]arr;//必须记住此固定搭配

}

int main()

{

test01();

int \*p=func();

test02();

system("pause");

return 0;

}

## C++中的引用

### 引用的基本语法

#include<iostream>

using namespace std;

int main()

{

//引用的语法： 数据类型 &别名 = 原名

int a=10;

int &b=a;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

b=100;

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

system("pause");

return 0;

}

### 引用的注意事项

#include<iostream>

using namespace std;

int main()

{

int a=10;

int c=20;

//1、引用必须要初始化

//int &b; 错误写法,必须要初始化,及常量指针必须初始化

//int &b=a; 正确写法

//2、引用一旦初始化就不能更改了

int &b=a;

//&b=c;错误写法

cout<<"a="<<a<<endl;

cout<<"b="<<b<<endl;

cout<<"c="<<c<<endl;

system("pause");

return 0;

}

### 引用作函数参数

#include<iostream>

using namespace std;

//引用传递，形参也会修饰实参。清楚其本质！

void swap(int &a,int &b)

{

int temp=a;

a=b;

b=temp;

}

int main()

{

int a=10;

int b=20;

swap(a,b);

cout<<"main函数中 a="<<a<<endl;

cout<<"main函数中 b="<<b<<endl;

system("pause");

return 0;

}

### 引用作函数返回值

#include<iostream>

using namespace std;

//引用作函数的返回值

//1、不要返回局部变量的引用

int& test01()

{

int a=10;//局部变量a存放在栈区

return a;//错误操作

}

//2、函数的调用可以作为左值

int& test02()

{

static int a=10;//静态变量，存放在全局区，全局区上的数据在程序结束后由系统释放

return a;

}

int\* test03()

{

static int a=10;

return &a;

}

int main()

{

/\*错误操作

int &ref=test01();//ref为a的别名

cout<<"ref="<<ref<<endl;//第一次结果正确，因为编译器作了保留

cout<<"ref="<<ref<<endl;//第二次结果错误，因为a的内存已经释放

\*/

int ref2=test02();//ref是a的别名，两者都是一模一样的变量

cout<<"ref2="<<ref2<<endl;

cout<<"ref2="<<ref2<<endl;

test02()=1000;//如果函数的返回值是引用，这个函数调用可以作为左值

cout<<"ref2="<<ref2<<endl;

cout<<"ref2="<<ref2<<endl;

//活学活用：返回值是指针，函数也可以作左值

int\* ref3=test03();

cout<<\*ref3<<endl;

\*test03()=1000;

cout<<\*ref3<<endl;

system("pause");

return 0;

}

### 引用的本质

#include<iostream>

using namespace std;

//引用的本质是一个指针常量!

void func(int &ref)

{

ref=100;

}

int main()

{

int a=10;

int &ref=a;//编译器内部自动转换为 int \* const ref =&a;指针常量指向的方向不变，也就说明了为什么引用初始化后就不可修改

ref=20;//内部发现ref是引用，自动帮我们转化成：\*ref=20;

cout<<"a="<<a<<endl;

cout<<"ref="<<ref<<endl;

func(a);

system("pause");

return 0;

}

### 常量引用

#include<iostream>

using namespace std;

//打印数据的函数

void showvalue(const/\*限制形参\*/ int & val)

{

//val=1000;错误操作

cout<<"val="<<val<<endl;

}

int main()

{

//常量引用

//使用场景：用来修饰形参，防止误操作

int a=10;

//int &ref=10;引用必须引用栈区或堆区上的数据，该操作错误

const int &ref=10;//加上const后，编译器将代码修改为 int temp=10; int &ref=temp; 因此原名无人知晓，由编译器起名，只能操作别名。

//ref=20;//加入const之后，变为只读，不可修改

int b=100;

showvalue(b);

system("pause");

return 0;

}

## 函数高级

### 函数的默认参数

#include<iostream>

using namespace std;

//函数的默认参数

int func(int a=20,int b=20,int c=30);

/\*注意事项

1、如果某个位置已经有了默认参数，那么从这个位置往后，从左到右都必须有默认值

2、函数的声明和函数的实现智能有一个有默认参数。

\*/

int main()

{

cout<<func(10,20,30)<<endl;//如果实参与形参的默认值不一样，以实参的值为准。

system("pause");

return 0;

}

/\*错误操作

int func(int a=20,int b=20,int c=30)

{

return a+b+c;

}

\*/

int func(int a,int b,int c)

{

return a+b+c;

}

### 函数的占位参数

#include<iostream>

using namespace std;

//占位参数语法：

//返回值类型 函数名（数据类型）{}

//占位参数还可以有默认参数

void func(int ,int=10)//目前阶段的占位参数，我们还用不到。

{

cout<<"666"<<endl;

}

int main()

{

func(10);

system("pause");

return 0;

}

### 函数重载

#### 基本语法

#include<iostream>

using namespace std;

//函数重载

//可以让函数名相同，提高复用性

//函数重载满足的条件：1、都在同一作用于下（main函数中）。 2、函数名称相同。 3、函数参数类型不同，或者个数不同，或者顺序不同

void func()

{

cout<<"func()的调用"<<endl;

}

void func(int a)

{

cout<<"func(int a)的调用"<<endl;

}

void func(double a,int b)

{

cout<<"func(double a,int b)的调用"<<endl;

}

void func(int a,double b)

{

cout<<"func(int a,double b)的调用"<<endl;

}

/\*

//注意事项：函数的返回值不可以作为函数重载的条件

int func(int a,double b)

{

cout<<"func(int a,double b)的调用"<<endl;

}

因为也可以出现二义性。

\*/

int main()

{

func();

func(10);

func(3.14,10);

//func(20,10);有多个函数都能被调用，出现错误。

system("pause");

return 0;

}

#### 注意事项

#include<iostream>

using namespace std;

//函数重载的注意事项

//1、引用作为重载的条件:有无const

void func(int &a)//int &a=10;不合法 int &a=a;合法

{

cout<<"func(int &a)的调用"<<endl;

}

void func(const int &a)//const int &a=10;合法

{

cout<<"func(const int &a)的调用"<<endl;

}

//2、函数重载碰到默认参数

void func2(int a,int b=10)

{

cout<<"func2(int a,int b=10)的调用"<<endl;

}

void func2(int a)

{

cout<<"func2(int a)的调用"<<endl;

}

int main()

{

int a=10;

func(a);

func(10);

//func2(10);//当函数重载碰到默认参数，出现二义性，报错。尽量避免该情况出现。

system("pause");

return 0;

}

## 类和对象

### 封装

#### 属性和行为作为整体

/\*设计一个圆类，求圆的周长

圆的周长的公式：c=2πr

\*/

#include<iostream>

using namespace std;

//圆周率

const double PI=3.14;

//class代表设计一个类，类后面紧跟的就是类的名称

class circle

{

//访问权限

//公共权限

public:

//属性（一般用变量表示）

double m\_r;

//行为(一般用函数表示)

//获取圆的周长

double calculateC()

{

return 2\*PI\*m\_r;

}

};

int main()

{

//通过圆类创建具体的圆（对象）,也就是实例化（具体化）

circle c1;

//给圆对象 的属性进行赋值操作

c1.m\_r =10;

cout<<"圆的周长为："<<c1.calculateC ()<<endl;

system("pause");

return 0;

}

#### 案例

/\*

案例：

设计一个学生类，属性有姓名，学号。

可以给姓名和学号赋值，可以显示学生的姓名和学号

\*/

#include<iostream>

using namespace std;

#include<string>

//设计学生类

class student

{

public://公共权限

//属性（姓名、学号）

string m\_name;

int id;

//行为

//显示姓名和学号

void show()

{

cout<<"姓名："<<m\_name<<endl<<"学号："<<id;

}

//给姓名赋值

void setname(string name)

{

m\_name=name;

}

//活学活用：给学号进行赋值

};

int main()

{

//创建一个具体的学生(通过类实例化对象)

student s1;

//给s1对象的属性进行赋值操作

s1.m\_name ="李文";//通过变量对属性进行赋值

//或 s1.setname ("李文") ;通过函数对属性进行赋值

s1.id =666;

//显示学生信息

s1.show();

cout<<endl;

student s2;

s2.m\_name ="库里";

s2.id =30;

s2.show();

cout<<endl;

system("pause");

return 0;

}

/\*专业术语介绍：

类中的属性和行为，我们统一称为成员

属性也叫做成员属性或成员变量

行为也叫做成员函数或成员方法

\*/

#### 访问权限

#include<iostream>

using namespace std;

/\*访问权限有三种：

公共权限（public）：成员（成员属性和成员函数）类内可以访问，类外也可以访问。

保护权限（protected）：成员 类内可以访问，类外不可以访问。“儿子”也可以访问“父亲”中的保护内容

私有权限（private）：成员 类内可以访问，类外不可以访问。“儿子”不可以访问“父亲”的私有内容

\*/

class person

{

public://公共权限

string name;

protected://保护权限

string car;

private://私有权限

int password;

public:

void func()

{

name="张三";

car="法拉利";

password=666666;

}

};

int main()

{

//实例化锯腿对象

person p1;

p1.name ="李文";

/\*

p1.car ="拖拉机";//保护权限的内容在类外无法访问

p1.password =123456;//私有权限的内容在类外也无法访问

\*/

p1.func ();

system("pause");

return 0;

}

#### struct和class的区别

#include<iostream>

using namespace std;

//struct和class的区别：struct默认权限是公共（public）,class默认权限是私有（private）。其余方面两者几乎相同

class C1

{

int m\_a;//默认权限为私有

};

struct C2

{

int m\_b;//默认权限为公共

};

int main()

{

C1 c1;

C2 c2;

//c1.m\_a =100;操作错误

c2.m\_b =200;

system("pause");

return 0;

}

#### 成员属性私有化

#include<iostream>

using namespace std;

//struct和class的区别：struct默认权限是公共（public）,class默认权限是私有（private）。其余方面两者几乎相同

class C1

{

int m\_a;//默认权限为私有

};

struct C2

{

int m\_b;//默认权限为公共

};

int main()

{

C1 c1;

C2 c2;

//c1.m\_a =100;操作错误

c2.m\_b =200;

system("pause");

return 0;

}

#### 案例2

/\*案例描述：设计立方体类，求出立方体的面积和体积，分别利用全局函数和成员函数判断两个立方体的表面积和体积是否相等。

\*/

#include<iostream>

using namespace std;

//创建cube类

class cube

{

public:

void getL(double l)

{

L=l;

}

void getW(double w)

{

W=w;

}

void getH(double h)

{

H=h;

}

double size()

{

return 2\*(L\*W+L\*H+W\*H);

}

double v()

{

return L\*W\*H;

}

bool classsame(cube &c)

{

if(v()==c.L \*c.W \*c.H )

{

return true;

}

else

{

return false;

}

}

private:

double L;

double W;

double H;

};

//利用全局函数判断两个立方体面积是否相等

bool issame(cube &c1,cube &c2)

{

if(c1.size ()==c2.size ())

{

return true;

}

else

{

return false;

}

}

int main()

{

double l,w,h;

cube c1;

c1.getL (10);

c1.getW (10);

c1.getH (12);

cout<<"c1的面积："<<c1.size ()<<endl;

cout<<"c1的体积："<<c1.v ()<<endl;

cube c2;

c2.getL (11);

c2.getW (10);

c2.getH (10);

cout<<"c2的面积："<<c2.size ()<<endl;

cout<<"c2的体积："<<c2.v ()<<endl;

//利用全局函数判断

bool ret=issame(c1,c2);

{

if(ret)//注意注意!!!不要写作if(ret=true)

{

cout<<"全局函数判断：c1的面积与c2的面积相等。";

}

else

{

cout<<"全局函数判断：c1的面积和c2的面积不相等。";

}

cout<<endl;

}

//利用成员函数判断

ret=c1.classsame (c2);

if(ret)

{

cout<<"成员函数判断：c1和c2的体积相等的。"<<endl;

}

else

{

cout<<"成员函数判断：c1和c2的体积不相等"<<endl;

}

cout<<endl;

system("pause");

return 0;

}

#### 案例3

#include<iostream>

using namespace std;

/\*案例：

设计一个圆类、一个点类，计算点和圆的位置关系

\*/

//点类

class point

{

public:

//设置X

void setX(double x)

{

X=x;

}

//获取X

double giveX()

{

return X;

}

//设置Y

void setY(double y)

{

Y=y;

}

//获取

double giveY()

{

return Y;

}

private:

double X;

double Y;

};

//圆类

class circle

{

public:

//设置半径

void setR(double r)

{

R=r;

}

//获取半径

double giveR()

{

return R;

}

//设置圆心

void setCenter(point center)

{

Center=center;

}

//获取圆心

point giveCenter()

{

return Center;

}

private:

double R;

point Center;

};

//判断点和圆的关系

void isincircle(circle &c,point &p)

{

double dd=(c.giveCenter ().giveX ()-p.giveX ())\*(c.giveCenter ().giveX ()-p.giveX ())+(c.giveCenter ().giveY ()-p.giveY ())\*(c.giveCenter ().giveY ()-p.giveY ());

double rr=c.giveR ()\*c.giveR ();

if(dd<rr)

{

cout<<"点在圆内。"<<endl;

}

else if(dd==rr)//注意：==才是等号

{

cout<<"点在圆上。"<<endl;

}

else if(dd>rr) //无论如何，程序判断该项正确，其余两项错误。为什么？

{

cout<<"点在圆外。"<<endl;

}

}

int main()

{

circle c;//实例化圆

point center;//实例化圆心

c.setR (10);

center.setX (10);

center.setY (0);

c.setCenter (center);// 太TM重要！！！

//实例化点

point p;

p.setX (0);

p.setY (0);

//判断关系

isincircle(c,p);

system("pause");

return 0;

}

头文件：circle.h

#pragma once//必须记住格式

#include<iostream>

using namespace std;

#include"point.h"//circle类中包含point类，须将point类的头文件包含进来

class circle

{

public:

void setR(double r);//注意写分号;

double giveR();

void setCenter(point center);

point giveCenter();

private:

double R;

point Center;

};

头文件：point.h

#pragma once

#include<iostream>

using namespace std;

class point

{

public:

void setX(double x);

double giveX();

void setY(double y);

double giveY();

private:

double X;

double Y;

};

源文件：circle.cpp

#include"circle.h"

void circle::setR(double r)

{

R=r;

}

double circle::giveR()

{

return R;

}

void circle::setCenter(point center)

{

Center=center;

}

point circle::giveCenter()

{

return Center;

}

源文件：point.cpp

#include "point.h"

//只需要留下函数的实现

void point::setX(double x)//point:: 表示setX(double x)是在point作用域下的成员函数

{

X=x;

}

double point::giveX()

{

return X;

}

void point::setY(double y)

{

Y=y;

}

double point::giveY()

{

return Y;

}

main函数中的源文件：

using namespace std;//记住格式

#include"point.h"

#include"circle.h"

void isincircle(circle &c,point &p)

{

double d=(c.giveCenter ().giveX ()-p.giveX ())\*(c.giveCenter ().giveX ()-p.giveX ())+(c.giveCenter ().giveY ()-p.giveY ())\*(c.giveCenter ().giveY ()-p.giveY ());

if(d>(c.giveR ()\*c.giveR ()))

{

cout<<"点在圆外。"<<endl;

}

else if(d==(c.giveR ()\*c.giveR ()))

{

cout<<"点在圆上。"<<endl;

}

else if(d<(c.giveR ()\*c.giveR ()))

{

cout<<"点在圆内。"<<endl;

}

}

int main()

{

circle c;

point center;

c.setR (10);

center.setX (10);

center.setY (0);

c.setCenter (center);// 太TM重要！！！

point p;

p.setX (1);

p.setY (0);

isincircle(c,p);

system("pause");

return 0;

}

### 对象特征

#### 构造函数和析构函数

#include<iostream>

using namespace std;

//对象的初始化和清理

//1、构造函数 进行初始化操作

class person

{

public:

//1.1、构造函数：

//没有返回值，不用写void

//函数名 与 类名相同

//构造函数可以有参数，可以发生重载

//创建对象的时候，构造函数会自动调用，且只调用一次

person()

{

cout<<"person的构造函数的调用"<<endl;

}

//2、析构函数 进行清理操作

//没有返回值，不写void

//函数名和类相同 在名称前加~

//析构函数不可以有参数，不可以发生重载

//对象在销毁前会调用析构函数，而且只会调用一次

~person()

{

cout<<"person的析构欧函数的调用"<<endl;

}

};

//构造函数和析构函数必须有实现，如果不提供，编译器会提供一个空实现的构造和析构

void test01()

{

person p;//构造函数不用写()，不用人为调动。由编译器自动调用

//局部变量p是栈上的数据，test01执行完毕后，释放这个对象

}

int main()

{

test01();

system("pause");

return 0;

}

#### 函数的分类以及调用

#include<iostream>

using namespace std;

//1、构造函数的分类

//按照参数分类：有参和无参构造 无参有称为默认构造函数

//按照类型分类：普通构造和拷贝构造

class person

{

public:

//无参（默认）构造函数

person()

{

cout<<"person的无参构造函数！"<<endl;

}

//有参构造函数

person(int a)

{

age=a;

cout<<"person的有参构造函数！"<<endl;

}

//拷贝构造函数

person(const person &p)//将传入的人身上的所有属性拷贝

{

cout<<"person的拷贝函数的构造！"<<endl;

age=p.age ;//p.age指向的是有参构造函数中的age

}

~person()

{

cout<<"person的析构函数调用"<<endl;

}

int age;

};

//2、函数的调用

void test01()

{

//1、括号法// 建议使用

person p1;//默认构造函数的调用

person p2(10);//有参构造函数

person p3(p2);//拷贝函数的调用

cout<<"p2的年龄为："<<p2.age <<endl;

cout<<"p3的年龄为："<<p3.age <<endl;

//注意事项：调用默认构造函数时，不要加()

//person p1(); 会被编译器认为是函数的声明

//2、显示法

person p4;

person p5=person(10);//有参构造函数

person p6=person(p5);//拷贝函数构造

//匿名对象

person(10);//特点：当执行结束后，系统会立即回收掉匿名对象

person p7=person(10);//该函数执行完后才释放，对比与上面一条代码的区别

cout<<"验证"<<endl;

//注意事项：不要利用拷贝构造函数 初始化匿名对象。如下：

//person(p6);错误代码，p6是一个拷贝构造函数。编译器会认为person(p6)===person p6，当做是person p6的声明

person p8=person(p6);//正确代码

//3、隐式转换法

person p10=20;//相当于person p10=person(20); 有参构造

person p11=p6;//拷贝构造

}

int main()

{

test01();

system("pause");

return 0;

}

#### 拷贝构造函数调用时机

//本节目标：①掌握拷贝构造函数调用的三种时机。②采用合适的方法证明拷贝后的值不在同一地址。

#include<iostream>

using namespace std;

class person

{

public:

person()

{

cout<<"person默认构造函数调用"<<endl;

}

person(int age)

{

Age=age;

cout<<"person有参构造函数调用"<<endl;

}

person(const person &p)

{

Age=p.Age ;

cout<<"person拷贝构造函数调用"<<endl;

}

~person()

{

cout<<"person析构函数调用"<<endl;

}

int Age;

};

//拷贝构造函数调用时机

//1、使用一个已经创建完毕的对象来初始化一个新对象,相当于复制

void test01()

{

person p1(20);

person p2(p1);//用创建好的p1初始化p2

cout<<"p2的年龄为："<<p2.Age <<endl;

}

//2、值传递的方式给函数参数传值

void dowork(person p)

{}

void test02()

{

person p;

dowork(p);

}

//3、值的方式返回局部对象()

person dowork2()

{

person p1;//p1在执行后被释放，这是拷贝出一个新的对象赋予test03函数中的p

cout<<(int)&p1<<endl;

return p1;

}

void test03()

{

person p=dowork2();

cout<<(int)&p<<endl;

}

int main()

{

//test01();

//test02();

test03();

system("pause");

return 0;

}

#### 构造函数调用规则

//本节目标：明白默认构造函数和拷贝构造函数调用的时机。

#include<iostream>

using namespace std;

//创建一个类，C++编译器都会自动添加至少三种函数

//默认构造函数（无参，函数体为空）

//默认析构函数（无参，函数体为空）

//默认拷贝构造函数，对属性进行赋值。

//如果写了有参构造函数，编译器就不会自动提供默认构造函数，但是会提供默认拷贝构造函数。

//如果写了拷贝构造函数，编译器就不会自动提供默认构造函数。

class Person

{

friend void test01();

friend void test02();

public:

Person()

{

cout<<"默认构造函数的调用！"<<endl;

}

Person(int age)

{

m\_Age=age;

cout<<"有参构造函数的调用！"<<endl;

}

/\*Person(const int &age)

{

cout<<"拷贝构造函数的调用！"<<endl;

}\*/

~Person()

{

cout<<"析构函数的调用！"<<endl;

}

private:

int m\_Age;

};

void test01()

{

Person p1;

p1.m\_Age =18;

Person p2(p1);//该过程会伴随拷贝构造函数的调用！

cout<<"p2的年龄为："<<p2.m\_Age <<endl;//注：操作此步骤需删除Person类中的拷贝构造函数//

//即使没有创建拷贝构造函数，编译器也会自动将p1的数据自动拷贝到p2上！

}

void test02()

{

Person p3(20);

Person p4(p3);

cout<<"p4的年龄为："<<p4.m\_Age <<endl;

}

void test03()

{

Person p5(10);

}

int main()

{

test01();

test02();

test03();

system("pause");

return 0;

}

//补充案例：

#include<iostream>

using namespace std;

class Person

{

public:

Person()

{}

Person(int age)

{

m\_Age=age;

}

Person(const Person &p)

{

cout<<"傻逼！"<<endl;

}

~Person()

{}

int m\_Age;

};

void test()

{

Person p1(20);

Person p2(p1);

cout<<"p2的年龄："<<p2.m\_Age <<endl;

}

int main()

{

test();

system("pause");

return 0;

}

//如果没有创建Person的拷贝构造函数，编译器还可以将p1的数据拷贝到p2身上。但是如果人为创建了拷贝构造函数，编译器就不会自动拷贝，而是指向人为创造出的拷贝构造函数中的内容。

#### 深拷贝与浅拷贝

//本节目标：明白浅拷贝和深拷贝，学会深拷贝。用恰当的方法验证“先入后出”的原理。

//掌握在析构函数中创建析构代码和在拷贝构造函数中手动开辟堆区存放数据。

//知道深拷贝应用的时机。

#include<iostream>

using namespace std;

//深拷贝与浅拷贝

//浅拷贝：由编译器自动拷贝

//深拷贝：人为在堆区开辟一段空间进行拷贝。传输方重新在堆区开辟一块内存，再传给接收方

class Person

{

friend void test01();

public:

Person()

{

cout<<"Person的默认构造函数调用。"<<endl;

}

Person(int age,int height)

{

m\_Age=age;

//m\_Height=&height; //变量height从有参构造函数传出到test01函数后悔被释放

m\_Height=new int (height);//将height变量储存到堆区，不会被释放。

cout<<"Person的有参构造函数调用。"<<endl;

}

Person(const Person &p)//自己解决浅拷贝带来的问题。

{

m\_Age=p.m\_Age ;

//在拷贝的过程中由于p1和p2的m\_Height是一样的，而p1的地址在p1运行后就被释放掉。所以p2不能重复释放p1上的那段地址。

//解决方法：将p1的m\_Height的值放入另一个堆区，用p2的m\_Height去接收p1的m\_Height的地址。

m\_Height=new int(\*p.m\_Height );

//此处new int 而不是new Person是因p.m\_Height是int数据类型的。

cout<<"Person的拷贝构造函数调用。"<<endl;

}

~Person()

{

if(m\_Height!=NULL)//注意：堆区开辟的数据在使用完后最好释放掉

{

delete m\_Height;

m\_Height=NULL;

}

cout<<"析构函数的调用。"<<endl;

}

private:

int m\_Age;

int\* m\_Height;

};

void test01()

{

Person p1(18,187);

cout<<"p1的年龄为："<<p1.m\_Age <<" p1的身高为："<<\*p1.m\_Height <<endl;//p1.m\_Height访问的就是有参构造函数中的变量height。

Person p2(p1);

cout<<"p2的年龄为："<<p2.m\_Age <<" p2的身高为："<<\*p2.m\_Height <<endl;

//注意：编译器先释放p2，再释放p1.“先进后出”。

}

int main()

{

test01();

system("pause");

return 0;

}

#### 初始化列表

//本节目标：学会初始化成员属性的三种方法。

#include<iostream>

using namespace std;

//初始化列表

class person

{

public:

//传统初始化操作

//person(int a,int b,int c)

//{

// A=a;

// B=b;

// C=c;

//}

int A;

int B;

int C;

//利用初始化列表给成员属性赋初值。

/\*person():A(10),B(20),C(30)

{}\*/

//另外一种写法：

/\*person(int a,int b,int c):A(a),B(b),C(c)

{}\*/

};

void test01()

{

/\*person p(12,43,23);

cout<<"A="<<p.A <<endl;

cout<<"B="<<p.B <<endl;

cout<<"C="<<p.C <<endl;\*/

}

int main()

{

test01();

system("pause");

return 0;

}

#### 类对象作为类成员

//本节目标：在一个类属性中创建另一个类对象(类似于类镶嵌类)。学会对类属性中的类对象赋值。

//在类属性中创建类成员的方法：就像创建普通变量那样，语法格式为： 类名 对象（变量）名称

//给类属性中的类成员赋值的注意事项：形参列表不能是变量，只能是数据。

#include<iostream>

using namespace std;

#include<string>

//类对象作为类成员

//创建 学生 类

class Student

{

friend class Teacher;

private:

string s\_Name;

int s\_Age;

int s\_Score;

};

//创建 老师 类

class Teacher

{

public:

Teacher(string name,int t\_age,string s\_name,int s\_age,int s\_score)

{

t\_Name=name;

t\_Age=t\_age;

s.s\_Age =s\_age;

s.s\_Name =s\_name;

s.s\_Score =s\_score;

}

private:

string t\_Name;

int t\_Age;

Student s;

};

void test01()

{

Student s;

Teacher t("老李老师",39,"李文",19,100);

}

int main()

{

test01();

system("pause");

return 0;

}

#### 静态成员变量

//本节目标：学会创建静态成员变量。掌握访问静态成员变量的方法。知道静态成员变量的访问权限。

#include<iostream>

using namespace std;

class person

{

public:

static int m\_A;//静态成员变量

//静态成员变量的特点：

//1、在编译阶段分配内存。

//2、类内声明，类外初始化

//3、所有对象共享同一份数据

private:

static int m\_B;//静态成员变量也是有访问权限的

};

int person::m\_A=10;//类外声明

int person::m\_B=20;

void test01()

{

//1、通过对象访问静态成员

person p1;

p1.m\_A =100;

cout<<"p1.m\_A="<<p1.m\_A<<endl;

person p2;

//p2.m\_B =200;//

//2、通过类名

person::m\_A =1;

cout<<"m\_A="<<person::m\_A<<endl;

//p2.m\_B =300;静态成员变量也是有访问权限的

}

int main()

{

test01();

system("pause");

return 0;

}

#### 静态成员函数

#include<iostream>

using namespace std;

//本节目标：静态成员函数的两种访问方法：通过类名和通过对象

//静态成员函数只能访问静态成员属性，但非静态成员函数可以访问静态或非静态属性。

//静态成员函数

//所有对象共享同一个函数

//静态成员函数只能访问静态成员变量

class person

{

public:

//静态成员函数

static void func()

{

m\_A=100;//静态成员函数可以访问静态成员变量

//m\_B=200;//静态成员函数不可以访问非静态成员变量

cout<<"static void func的调用！"<<endl;

}

static int m\_A;//静态成员变量

int m\_B;

private:

static void func2()

{

cout<<"static void func2的调用！"<<endl;

}

};

void test01()

{

//1、通过对象访问

person p;

p.func ();

//2、通过类名访问

person::func ();

//person::func2 ();//无访问权限

}

int main()

{

test01();

system("pause");

return 0;

}

#### 成员变量和成员函数分开存储

//本节目标：知道类中的不同变量和函数的储存位置，学会用适当的方法验证。知道每一个非静态函数只有一个函数实例，也就是多个同类型的对象会共用一块代码。这一块代码通过下一节的this指针可以知道。

#include<iostream>

using namespace std;

//成员变量 和 成员函数 是分开存储的

class person{};

class student

{

public:

int m\_A;//非静态成员变量 属于类对象

static int m\_B;//静态成员变量 不属于类对象

void func(){}//非静态成员函数 不属于类对象

static void func2(){}//静态成员函数 不属于类对象

};

int student::m\_B =10;

void test01()

{

person p;

//空对象占用的内存空间为：1字节

//C++编译器会给每个空对象也分配一个字节空间，是为了区分空对象所占内存的位置

//每个空对象也应该有一个独一无二的内存地址

cout<<"size of p="<<sizeof(p)<<endl;

}

void test02()

{

student s;

cout<<"size of s="<<sizeof(s)<<endl;

}

int main()

{

test01();

test02();

system("pause");

return 0;

}

#### this指针的用途

//本节目标：掌握this指针的用法及原理。知道返回值和返回引用的区别：值返回会调用拷贝构造函数，返回引用则不会。

#include<iostream>

using namespace std;

#include<string>

//this指针

class Person

{

public:

Person (int age)

{

this->m\_Age=age;

}

Person& PersonAdd (Person &p)

{

this->m\_Age +=p.m\_Age;

cout<<"this指针的地址："<<(int)this<<endl;

return \*this;

//如果返回值是值而不是指针，那么在返回值的过程中会调用拷贝构造函数，将会创建一个新的\*this，而不是原先的\*this

//为了保证对某个数据持续更改，因此返回引用。（可以认为返回引用就是返回变量本身）

}

int m\_Age;

};

void tets01()

{

Person p1(18);

cout<<"p1的年龄为："<<p1.m\_Age <<endl;

}

void test02()

{

Person p2(20);

Person p3(30);

p3.PersonAdd (p2).PersonAdd (p2).PersonAdd (p2);

cout<<"this指针的地址："<<(int)&p3<<endl;

}

int main()

{

test02();

system("pause");

return 0;

}

#### 空指针访问成员函数

//知道空指针访问成员函数的注意事项。

#include<iostream>

using namespace std;

//空指针是可以访问成员的！

class Person

{

public:

Person():m\_Age(10)

{}

void showname()

{

cout<<"this is person class"<<endl;

}

void showage()//错误函数

{

cout<<"age="<<m\_Age<<endl;//编译器默认m\_Age是this->m\_Age;

//报错原因是此时的this指针指向的是NULL

}

//预防措施如下

int showage1()

{

if(this==NULL)

{

return 0;

}

cout<<"age="<<m\_Age<<endl;//编译器默认m\_age是this->m\_age;

}

int m\_Age;

};

void test01()

{

Person\* p=NULL;

p->showname ();

p->showage1 ();

}

int main()

{

test01();

system("pause");

return 0;

}

#### const修饰成员函数

//本节目标：掌握常函数（仅限成员函数）和常对象的表示方法。知道常函数和常对象的特点。知道mutable的作用

#include<iostream>

using namespace std;

//常函数

class person

{

public:

//this指针的本质：是指针常量，指针指向的对象是不可以修改的

//在成员函数后面加const，修饰的是this指向的值，让指针指向的值也不可以修改。

void show() const //注意const写在后面

{

//m\_A=100;//如果该成员函数不是常函数，该操作正确。因为this是指针常量，虽然不能修改指针的方向，但是可以修改指针指向的内容

//this=NULL; //this指针不可以修改指针的指向，因为this指向的是对象p

this->m\_B =100;

}

void func(){}

int m\_A;

mutable int m\_B;//特殊声明，即使在常函数中，也可以修改这个值,在前面加mutable

};

void test01()

{

person p;

p.show ();

}

//常对象

void test02()

{

const person p;//在对象前面加const，变为常对象

//p.m\_A =100;

p.m\_B =200;//m\_B是特殊值，在常对象下也可以修改

//常对象只能调用常函数

p.show ();

//p.func ();常对象 不可以调用普通成员函数，因为普通成员函数可以修改属性，而常对象不能修改属性。

}

int main()

{

test01();

test02();

system("pause");

return 0;

}