报错集：

Linking...

LINK : fatal error LNK1168: cannot open Debug/20001.exe for writing

执行 link.exe 时出错.(连接出错)

解决方法：之前的程序没有完全关闭，关闭程序或关闭工程重启

弹窗提示“\_BLOCK\_TYPE\_IS\_VALID”

解决方法：程序员手动释放了栈区的存储空间，查找代码中delete或free释放的空

间，若不是自己分配的空间，不要主动释放即可

error LINK2001：无法解析的外部符号 xxxxxx

解决办法：删除编译报错部分的临时文件.obj，然后再编译

unhandled exception

解决办法：数组开太小越界 检索了一个容器外的元素 导致内存泄露



C&C++语言基础：

**开码前进行构思，伪代码辅助思考**

**基本代码素养**：表单验证：为空判断

注意基础错误：函数调用不带返回值，输入输出函数修饰符错误 etc

C语言无布尔型数据类型，C++有

**C语言布尔值**：非0都是1

**类型转换**：

double>float>unsigned int>int>short>char

**exit()函数**：立即终止程序，控制权交给操作系统

**C语言生成随机数**

#include <stdlib.h>

#include <time.h>

srand(time(0))

rand() % (n - m + 1) + m;//生成m~n的随机数

**分文件开发**：

X一般是该头文件的大写，用于多文件**防止头文件的重复包含和编译**

#ifndef x //先测试x是否被宏定义过

#define x

程序段1 //如果x没有被宏定义过，定义x，并编译程序段 1

#else

程序段2 //如果x已经定义过了则编译程序段2的语句，“忽视”程序段 1

#endif//终止if

多文件注意：源代码文件&头文件需互相include

头文件.h中：放函数声明， 宏定义&全局变量的声明

不要在头文件中写变量，可在头文件中引用其他头文件（若存在类似返

回值为指针的情况，函数声明需要直接使用其他头文件时）

源代码.c中：放函数定义，引用的头文件&全局变量的定义

**预处理命令：**

C编译系统对程序进行通常的编译前，先对程序内的宏定义，文件

包含，条件编译进行“预处理”

**extern全局标识符：**

一个c文件需要调用另一个c文件里的变量或者函数时使用

可修饰变量or函数

tool.c文件中：int a=5;

tool.h文件中：extern int a;

将变量a声明为全局变量 ，另一个.c文件就能使用此变量a

**高内聚低耦合**：

大幅**提高程序模块的复用性、可移植性**

高内聚：

**一个模块实现内容单一**，尽可能类的每个成员方法只完成一件事

低耦合：

**模块之间相互依赖少**，减少类内部，一个成员方法调用另一个成员方法

降低耦合度的方法：

多用设计模式

少用全局变量

少用类的继承，多用接口隐藏实现的细节

模块的功能化分尽可能的单一，道理也很简单，功能单一的模块供其它

模块调用的机会就少

尽量不用“硬编码”的方式写程序，同时也尽量避免直接用SQL语句操作数

据库

类属性和方法的声明少用public，多用private关键字

增强内聚性的方法：

模块只对外暴露最小限度的接口

删除一个模块，应当只影响有依赖关系的其他模块，而不应该影响其他无关

部分

**const修饰限定符**：

int const\*p=&num;const int \*p=&num; 不能修改指针指向的值

int \*const p=&num; 不能修改指针指向的地址

**常用于引用传递函数形参时对该形参进行修饰防止修改原实参数值**

**函数重载与返回值无关系**

**强制类型转换运算符：**

格式：static\_cast<type>(表达式)

例如：double root=3.14;

int value=static\_cast<int>(root);

**内联函数inline**：

函数本身功能简单，代码很短，使用频率很高，程序频繁调用该函数所花费的时

间多，使得程序执行效率降低

inline直接将函数的代码嵌入到程序中

内联函数一般适合于不存在while和switch等复杂的结构且只有1~5条语句的

小函数上

**typedef和define的区别:**

功能不同：

**typedef：**用来定义类型的别名，起到类型易于记忆的功能。另一

个功能是定义机器无关的类型

**define:** 不只是可以为类型取别名，还可以定义常量、变量、编译

开关等

原理不同：

**typedef:** 关键字，在编译时处理，有类型检查功能；

**define：**是C语言中定义的语法，是预处理指令，在预处理时进行

简单而机械的字符串替换，不作正确性检查

**域作用符::的作用：**

标识作用域级别；标识成员属于哪个类；限定成员的作用范围；指出作用域的范围

**vc6只支持C++98标准，不支持C++11及其以后内容**

**debug断点别打在跳过不执行的语句上**

输入输出&运算符：

**getch()函数**:

<conio.h>

无输入缓冲区，输入直接读取

**putch()函数:**

<conio.h>

没回显

**getchar()函数：**

<stdio.h>

有缓冲区，输入完后按回车键读取

**putchar()函数:**

<stdio.h>

有回显，回车结尾

**“\b \b”:**

退一格，后向前空格覆盖之前的内容，后再退一格

**i++ & ++i** ：

++i(前++)更省内存，因为不会有中间变量

**==比较运算符:**

0==num比num==0写法好在哪：

若比较运算符错写成赋值运算符，此处会报错

控制语句：

for语句：循环次数已确定

while语句：循环次数根据循环体的执行情况确定

do\_while语句：循环次数根据循环体的执行情况确定&循环体至少执行一次

switch语句表达式&常量表达式：

Switch表达式的值可以包含任何变量名，只能为整数

Case后的值必须为整数常量

条件(三目)运算符：

表达式1?表达式2:表达式3

优先级>赋值运算，<关系&算术运算符

结合方式：从右向左

数组&字符串：

**注意数组下标产生的逻辑问题**

**memcpy函数：**

被包含在头文件string.h内，常用于数组初始化

参数（目标内存块首地址，原内存块首地址，要复制的字节数）

**自定义函数：**

多次执行，功能单一，目标明确的独立代码块

有利于：编写、维护、调试，符合高类聚低耦合标准

**多维数组作为函参时，只可省略一维大小**

**字符数组**：

0：空操作符 ‘0’：0字符 ‘\0’：字符串结束标志

**strlen函数:**

此方法得到的字符串长度不包括结束符

**sprintf函数：**

用于字符串拼接

**sscanf函数：**

用于将字符串转化为对应数据类型

**中文：**

一个中文字符占用2B，用两个char数据类型接收

ch>=0xA1 && ch<=0xFE

一个中文字符删除也要删两次”\b \b”\*2

**sstream库**：

主要用来进行数据类型转换

stringstream

substr(int,int)：获取子字符串(求从指定位置开始，并具有指定的长度)

c\_str()：把string对象转换为字符数组指针char\*

str()：将stringstream类型转换为string类型

**字符数组&字符指针的区别：**

字符数组：存放多个字符

字符指针：存放字符串首地址

字符数组赋值：只能单个单个字符分别赋值

字符指针赋值：可直接整体赋值

字符数组内存：在编译时分配具体内存

字符指针内存：把已定义的字符数组的地址给它赋值时

字符数组地址：不能通过赋值改变

字符指针地址：能通过赋值改变

**string类数据.c\_str():**

将string字符串数据转化成char类数组

**字符数组&整型数据类型转换：**

<stdlib.h>

atoi()

itoa()：char\* itoa(int I,char \*s,int radix)

将字符串转换成对应进制的整型数据

sprint()

**检索字符串中的子串：**

strstr(char\*,const char\*)

检查子串2是否在字符串1中

**一般字符数组不作为函数返回值，将字符数组做为参数传入函数**

函数：

**函数形参只能为变量**

自定义函数注释格式：

说明函数名称+函数作用+函数参数+函数返回

**函数堆栈**：注意内存开销，及时结束函数体并返回值

**数据传递：**

**参数传递**

**返回值传递**

**全局变量传递**

**参数传递：**

**值传递：**简单的把实参的值传递给形参，函数中对函参的修改不会影响传值的变量

**地址传递：**把实参的地址传递给形参，函数中对函参的修改会直接影响该函参地址

对应的变量

**引用传递：**

**使用指针非常高效，因为它只需传递地址大小的数据。因此尽量使用地址传递**

**在c++中使用引用传参是一种非常高效、简洁且不容易出错的方法**

**宏定义函数和内联函数的区别：**

**宏函数：**在预编译的时候把所有的宏名用宏体来替换，简单的字符串替换，预处理

器用复制宏代码的方式代替函数的调用，省去了函数压栈退栈过程，提高了效率；

没有类型检查的，无论对错都是直接替换；

**内联函数：**本质上是一个函数，内联函数一般用于函数体的代码比较简单的函数，

不能包含复杂的控制语句；在编译时进行代码插入，编译器会在每处调用内联函数

的地方直接把内联函数的内容展开，省去函数的调用的开销，提高效率；

会进行类型的检查，内联函数满足函数的性质，比如有返回值、参数列表；

如果内联函数函数体过于复杂，编译器将自动把内联函数当成普通函数来执行

结构体：

**结构体赋值**：对一个已经定义好的结构变量，**不可以**采用整体赋值的方式

**结构体变量大小计算**：

各成员数据向占用最大内存成员数据看齐

int id; 4 4+4 8

double price; 8 8 8 8+8+24

char name[20]; 2\*8+4+4 24

内存&指针：

所有指针定义时先给指针赋初值（避免野指针）：

NULL（定义在“stdio.h”中） 0x00000000

**无类型指针**：void\*p

什么类型的地址都能接收

**指针变量所占内存**：与基础数据类型无关，只与操作系统有关

指针可指向字符串常量

**堆区（heap）**：程序员分配释放（malloc、free）

**栈区（stack）**：编译器自动分配释放，存放函数的参数值，局部变量的值

**分配动态内存**:

记得最后释放申请的动态内存

(int\*)强制类型转化

可分配连续动态内存

int \*x;

x=(int\*)malloc(sizeof(int));

if(x==NULL)

{

printf("申请动态内存失败\n");

exit(0);

}

...

free(x);

**new和malloc的区别：**

new/delete为关键词；malloc/free为库函数

new在调用时，先为对象分配内存，再调用对象构造函数；

malloc不会；

new调用时不需要明确给出分配的内存空间大小；

malloc需要；

new分配内存空间成功返回对象指针；

malloc分配内存空间成功返回无符号void\*类型指针

**sizeof和strlen的区别：**

sizeof是运算符；strlen是库函数

sizeof用来获取某个数据类型所占用空间的字节数；strlen用来计算指定的字符串

s 的长度，不包括结束字符"\0"

sizeof()在编译时计算好了；strlen()在运行时计算

sizeof()的参数类型多样化（数组，指针，对象，函数都可以）；strlen()的参数必须

是字符型指针（传入数组时自动退化为指针）

**内存泄露：**

1.数组越界访问or容器越界访问

2.局部变量销毁导致函数返回失效，今后多次经过同一函数，该函数内尽量避免使

用局部变量，将变量声明为类属性

**指针和引用区别**

指针：保存另一个变量内存地址的变量

引用：另一个变量的别名，一旦被初始化就不能被改变

一个具有自动间接性的常量指针，相当于编译器帮助实现了自动间接性取值，

即：编译器帮助加上了 \*

引用内部实现为指针

**引用&**：

只能引用数组元素，不能引用数组

不能定义引用的引用，所以当函数的参数为引用时，引用不能作实参

链表基础&应用：

**链表初始化**

创建头指针

给头指针动态分配内存空间

头指针指向NULL

**循环向后遍历直到找到最后一个结点**

while(p->next!=NULL)

{

p=p->next;

}

**循环向后遍历所有结点**

while(p!=NULL)

{

…

p=p->next;

}

**or**

While(p->next!=NULL)

{

p=p->next;

…

}

文件：

**fopen**

打开方式：r+：读写模式打开对应文件 w+：读写模式新建并打开对应文件

不推荐写绝对路径：无法保证所有用户都有对应磁盘

推荐写相对路径：相对于工程文件所在目录下

./ 指当前文件夹 ../ 指当前文件夹的父级

文件夹

不能创建文件夹，只能创建文件

**fwrite & fread**

fread：接收缓冲区的首地址，一个单元所在字节数，读出几个单元，操作的文

件指针

成功时返回读出的单元数，否则返回0

**fflush(fp)**：

刷新文件缓冲区，将缓冲区内内容写入文件

同fclose一起使用

**文件打开函数自定义封装**（若文件打开失败则直接创建同名新文件）

fp=fopen(filename,”r+”);

if(fp==NULL)

{

fp=fopen(filename,”w+”);

}

return fp;

**fseek(fp,0,SEEK\_END)**

偏移到文件尾部

**rewind(fp);**

文件指示器回到开头

回到开头才能重新读内容

**C++:<fstream>文件流库**

\*.open 打开文件对象方式 ios::in/out/app...读出/写入/追加...

\*.seekg 光标定位 ios::end/beg 开头/末尾

\*.tellg 计算文件大小

\*.peek 向后遍历数据

\*.getline 从文件中读出一行数据（字符串常配合sscanf获得想要

数据）

\*.close 关闭文件对象

面向对象方法(Object Oriented Programming)：

**C++: 面向对象的程序设计 OOP(Object Oriented Programming)**

**C： 传统的面向过程的程序设计 POP(Process Oriented Programming)**

**面向对象的程序设计**：

对象=（算法+数据结构）

程序=对象\*n

类&对象：

**Class类声明：莫要忘记最后加引号**

**带参构造函数：**每次创建新的类对象时运行一次，用于初始化新的类对象

**默认构造函数**：没有参数or所有的参数都有默认值的构造函数

默认构造函数必写，带参构造函数可写

**拷贝构造函数**：如果类中没有说明复制构造函数，则系统自动生成一个缺省复制构造

函数，作为该类的公有成员

使用一个类对象去初始化另一个类对象

**浅复制**：将对象数据成员的值进行简单的复制

默认拷贝构造函数就为浅复制

**深复制**：将对象数据成员的值进行复制，而且对指针型数据成员生成新空间后复制对

应值

拷贝构造函数为深复制，指针赋值需要深复制

**explicit关键字**：

只提供给类的构造函数使用的关键字

构造函数只有一个参数很危险：隐式转换，会把类类型任意转化成其他类型

单个参数的构造函数前加explicit即可

**类对象大小**

类对象内所有数据成员大小之和

类函数为所有类对象共享，不同对象响应相同的消息时，调用的是同一个函数，

使用this指针进行调用

**this指针**：

每个对象都有一个指向自身的this指针（对象的内存地址）

访问对象自身（\*this）

**类的静态成员&静态成员函数static**：

静态数据成员不能由构造函数初始化，只能在类定义外使用::运算符来完成

静态成员函数不能声明为const

静态成员函数与类的对象无关，没有this指针

即使不存在类的对象，静态成员函数也存在，并且可以调用

**const数据成员&成员函数：**

const数据成员的初始化：在构造函数使用初始化列表

const成员函数不会修改调用它的对象

const成员函数只能访问数据成员的值，不能修改它

void 函数名() const

**存取权限的补充 mutable**:

用于定义更改频繁的数据成员. 数据成员即使在const对象或在const

成员函数中都可以被修改

**触发拷贝构造函数的三种情况**：

类对象直接赋值

类对象作为函参传参时

类对象作为函数返回值

**C++类&对应.cpp文件名要一致**

**C++类必须写足够多公有接口**

get & set 囊括所有数据成员

继承：

**继承的意义**：

便于管理系统中的对象

系统扩充比较容易- 利用程序代码的再用性

加快程序发展的速度

减少程序的错误

**类的复合&继承：**

有（has\_a）——类的复合 （人和心脏的关系）

是（is\_a）——类的继承

**父类私有成员**：

能被派生类继承

不能被派生类的成员函数直接访问

可通过定义于基类的公有和受保护成员函数访问

**子类可重写父类方法**

**父类私有成员**：

能被派生类继承

不能被派生类的成员函数直接访问

可通过定义于基类的公有和受保护成员函数访问

多态&虚函数：

**类指针的使用**：

基类指针操作派生类对象（安全）

派生类指针操作基类对象（危险）

**声明类对象指针时不会调用构造&析构函数，不是真正构造了新的类对象**

基类对象→派生类对象的转换 ：

强制类型转换：static\_cast运算符（仅限派生类以public方式继承基类）

向下转型不安全,没有自动转换的机制

**基类对象空间可能没派生类对象空间大，强转可能出问题**

**多态性**：

**程序根据具体不同的情况执行不同的操作**

有助于更好地对程序进行**抽象**

控制模块能专注于一般性问题的处理

具体的操作交给具体的对象去做

有助于**提高程序的可扩展性**

把控制模块与被操作的对象分开

添加已定义类的新对象，并能管理该对象

添加新类（已有类的派生类）的新对象，并能管理该对象

**虚函数**：

**用关键字virtual声明的函数，用于实现动态联编，以此实现多态性**

只有通过基类指针或引用调用虚函数才能引发**动态绑定**(用基类指针指向派生类指

针)

在派生类中重新定义虚函数时，**必须保证该函数的值和参数与基类中的说明完全**

**一致**，否则就属于重载

若在派生类中没有重新定义虚函数，则该类的对象将使用其基类中的虚函数代码

虚函数**必须是类的一个成员函数**，不能是友元，

虚函数**可以是另一个类的友元**

虚函数**不得是一个静态成员**

**虚析构函数**：

如果要操作具有继承关系的类的动态对象，最好使用虚析构函数。特

别是在析构函数需要完成一些有意义的操作——比如释放内存时

**重载**：

**在同一作用域内定义多个同名函数**

要求这些函数具有参数的类型、类型个数or顺序不相同，返回值类型不影响

**作用域相同**

**函数名相同**

**参数不同**

**virtual关键字可有可无**

**返回值可以不同**

**重定义：**

**派生类重新定义父类成员函数**

发生在基类和派生类中，只要函数名一致就会重定义

**作用域不同**

**函数名相同**

**返回值可以不同**

**参数不同时，无论有无virtual，基类的函数将被隐藏**

**参数相同时，基类函数没有virtual，基类的函数被隐藏**

**重写：**

**派生类重新定义基类纯虚函数**

针对多态性，利用同一接口实现不同的实例

**作用域不同**

**函数名相同**

**参数相同**

**基类函数必须有virtual关键字，不能有static**

**返回值相同**

**重写函数的访问修饰符可以不同（public，private，protected）**

友元&运算符重载：

**友元**：

在于提高程序的运行效率

破坏面向对象的封装性，少用

**加法友元运算符重载**：

不仅得在类内声明友元，还得在类声明后声明

**前后置一元运算符重载**：

++a 前置++

complex &operator++();

a++ 后置++

complex operator++(int);

int：函数重载区别于operator++()

模板：

**模板：**

当处理的数据类型随实际情况变化时，可以将数据类型作为可变的部分从程序中

抽取出来

函数模板规定了对数据的处理流程

某些数据类型（模板的参数）要等到模板实例化时再确定具体的类型

**函数模板的实例化由编译器来完成**

**创建类模板的实例**：

类名 <类型实参表> 对象名称;

**宏替换**：

也可实现参数化多态

#define max(a,b) ((a) > (b) ? (a) : (b))

问题 避开类型检查--不会进行类型检查

**泛型编程**：

将程序写得尽可能**通用**

将算法从特定的数据结构中抽象出来，成为通用的

**C++的模板为泛型程序设计奠定了关键的基础**

**类模板定义&实现都放在头文件中**

STL：

Container(容器)+Adapter(适配器)+Algorithm(算法)+Iterator(迭代器+Function

object(函数对象) +Allocator(分配器)

**向量（vector）**:

可变大小数组，连续存储，内存连续，支持随机访问

**双端队列（deque）**：

≈向量，队头队尾出队入队，双端队列收尾插入删除效率高

**链表（list）**：

可变大小，连续存储，内存不连续，不支持随机访问

**list迭代器遍历**:

链表：

it = 链表名.begin(); it != 链表名字.end(); it++

不能为 it <链表名字.end()

因为链表内存空间不连续

**关联容器(map)：字典键值对**

检索速度快

\*.insert 加入键值对

**多半常用：容器名[key]=value 增加键值对**

it->first 迭代器指向对应键

it->second 迭代器指向对应值

常用迭代器获取对应键值对元素

\*.erase 删除指定键值对 根据键删除

\*.find 根据指定键检索值

**sort排序算法:**

sort(指定排序段);

可使用自定义函数进行排序段

**带<>运算符的都得在命名空间中操作**

**迭代器辅助函数：**

advance(it,n) 将迭代器位置前进or后退n个位置

**注意<>运算符，防止双>>引起歧义**