自学材料

1. 更换EG：char的别名

1.#define \_\_\_(你想要的名字) char(或者其他函数名)

2.typedef char \_\_\_(name you want)

说明：方法2更好用，因为如果用声明的name来定义多个变量的话，方法一就会出错eg.name a,b;(name为float)则编译器编成name \*a,b;

1. 数组模块类（vector,array）
2. vector模块

eg.①vector <int> vi;②int m(0);cin>>m;vector <int> vd(m);

解释：此函数是基于自由存储区的，所以速度比存于栈的慢（eg、array,int a[]）

注意：里面的m是可以让用户输入的！很灵活！

1. array 模块

eg.array <double,4> a={1,2,3,4};

解释：注意格式，特点！它是可以在声明的时候同时初始化的！而vector不可以！

1. 调用函数使用指针（复合函数）,并使用new分配地址

eg.struct A{……};

int main{ A \*p=new A;(注意了，是A不是struct！)

1. 如果将1/0变成真假（false/ture）？（bool类）

输入代码：cout.self(ios\_base::boolalpha);

1. 如何清空缓冲区？
2. cin.ignore(字符数,停止标志（包）);也可只写停止标志。eg.cin.ignore(1024,’\n’)

解释：此函数是从输入流中读取多个字符数（或者读到停止标记）,然后忽略它，不再使用！

拓展：cin.read(str,n);从输入流中提取n个字符储存到str中；

cout.write(buf,n);从buf中输出n个字符

cout.precision(n);小数点后n位（精度）

cout.width(n);  符串宽度为n

m=cin.peek();取缓冲区中（未输出的）第一个字符！但是没什么用。一个for一直读m=cin.get()就可以了

1. getchar();用多次循环可以把缓冲区内的字符全部提取但是麻烦，一次只能提1~2

这个可以用循环弥补：while(getchar(?)!=’/n’)或者while(cin.get()!=’/n’)

1. fflush(stdin); 注意：此函数仅适用于 部分编译器（如 VC6），但是并非所有编译器都要支持这个功能（如 gcc3.2）。这是一个对C标准的扩充。（难用）
2. cin.clear();这个是用来清除错误的。例如：int a;cin>>a;【你输入123a】那么系统会通过I/O（非0）来判断这次读写错误，这样的话，后续的所有cin/cout全部失效甚至程序崩溃。

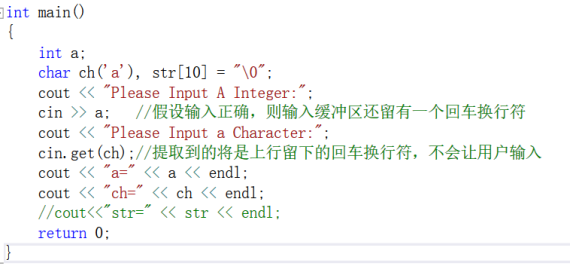
eg.

1. 系统讲解：

谈下提取函数提取不到数据问题及清空输入缓冲区问题分两种情况。

1. 正常输入导致。如果输入正确，每次提取一般输入缓冲区会留下一个回车换行符（cin.getline例外)，如果因它影响下次提取，用cin.get等提取单个字符的函数将其提取掉即可。

例如，如下程序是不会让用户输入字符或字符串的。



E:\安装包\1144986025\Image\Group\P%E`EHVBM1$3Z6U}1FWKQBR.png这两行换成：  
cout << "Please Input a String:";  
cin.get(str, 10);或cin.getline(str, 10);  
同样提取不到字符串  
因为cin.get和cin.getline缺省约定碰到回车换行符表提取结束，而上行留下的就是回车换行符，故str未提取到任何字符就结束，不会让用户输入

解决：将输入缓冲区中剩下的回车换行符提取掉，即清空输入缓冲区

因为输入缓冲区中只有一个字符，故可用：cin.get()即可，加

cin.get((char）ch)

语句之前 如果碰到输入缓冲区中有多个遗留字符需要清空，用循环，如：

while (cin.get() != '\n');

***最后顺带说一下，getline原型：***

getline(char \*str, int len, char end=‘\n’);  
cin.getline如提取了<=len-1个字符就结束，会将end字符亦提取掉

1. 非正常输入导致。原因是用户不按要求输入（如要求用户输入一个整数，用户却输入了多个，多余的将留在输入缓冲区中，导致后续不能让用户输入数据）或用户非法输入（如要求提取一个整数，用户却输入非数字字符等）

***对于2，***要知道C++语言的流类具有发现错误的功能，一旦发现I/O操作出错，会将错误的性质记录在错误状态标志字（理解为一个变量）中，让程序员利用程序进行检测。具体原理后面再讲，大家知道一点：提取出错，错误状态标志字非0。一旦错误状态标志字非0，后续的所有提取将不能进行。

后者将导致系统本次I/0错误状态标志字值非零，前者一般不会，

但影响后续提取并可能在其后造成错误状态标志字非零。由于属于

非正常输入，故两者经常同时出现，故可合并解决

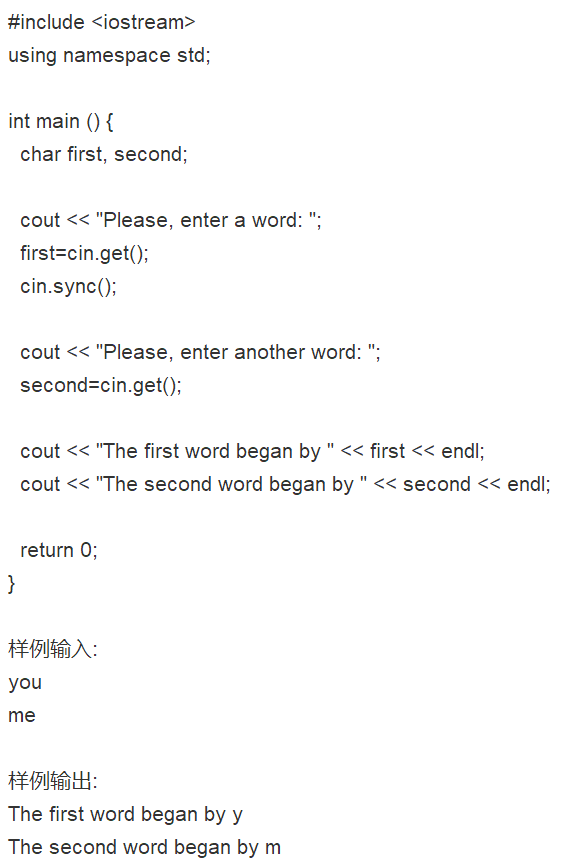
解决办法：1.调用cin.clear(0)设置错误状态标志字值为0，使I/o操作能正

常进行；

2.清空输入缓冲区。（当然，还可用cin.rdstate函数读取错误性

质，此不说）

1. cin.sync();用来清除未提取的字符；



1. cout的注意点：

eg.cout<<(x<y);注意必须要有（），因为有优先级；其次，此处输出的要么是1要么是0

如果此处要转化成bool类型，则输入:cout.setf(ios\_base::boolalpha)

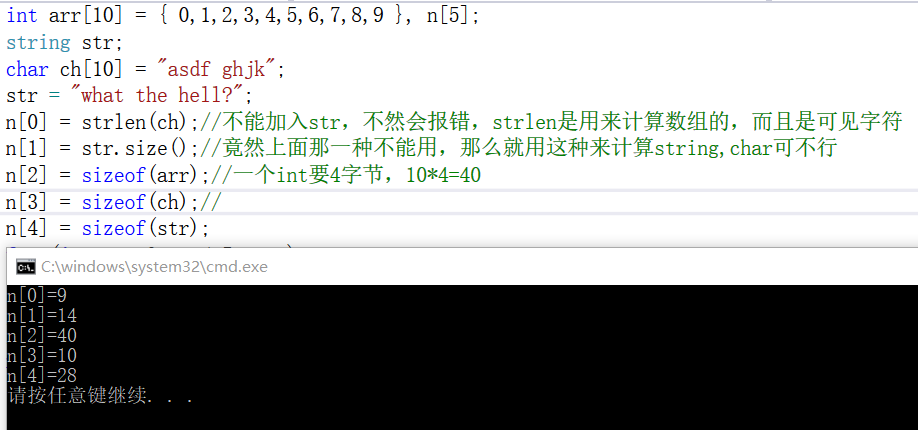
说起优先级：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 优先级 | 运算符 | 名称或含义 | 使用形式 | 结合方向 | 说明 |
| 1 | [] | 数组下标 | 数组名[常量表达式] | 左到右 |  |
| () | 圆括号 | （表达式）/函数名(形参表) |  |
| . | 成员选择（对象） | 对象.成员名 |  |
| -> | 成员选择（指针） | 对象指针->成员名 |  |
| 2***注意\*和++/--是一个优先级的，所以\*--p/--\*p/\*p—是不一样的*** | - | 负号运算符 | -表达式 | 右到左 | 单目运算符 |
| (类型) | 强制类型转换 | (数据类型)表达式 |  |
| ++ | 自增运算符 | ++变量名/变量名++ | 单目运算符 |
| -- | 自减运算符 | --变量名/变量名-- | 单目运算符 |
| \* | 取值运算符 | \*指针变量 | 单目运算符 |
| & | 取地址运算符 | &变量名 | 单目运算符 |
| ! | 逻辑非运算符 | !表达式 | 单目运算符 |
| ~ | 按位取反运算符 | ~表达式 | 单目运算符 |
| sizeof | 长度运算符 | sizeof(表达式) |  |
| 3 | / | 除 | 表达式/表达式 | 左到右 | 双目运算符 |
| \* | 乘 | 表达式\*表达式 | 双目运算符 |
| % | 余数（取模） | 整型表达式/整型表达式 | 双目运算符 |
| 4 | + | 加 | 表达式+表达式 | 左到右 | 双目运算符 |
| - | 减 | 表达式-表达式 | 双目运算符 |
| 5 | << | 左移 | 变量<<表达式 | 左到右 | 双目运算符 |
| >> | 右移 | 变量>>表达式 | 双目运算符 |
| 6 | > | 大于 | 表达式>表达式 | 左到右 | 双目运算符 |
| >= | 大于等于 | 表达式>=表达式 | 双目运算符 |
| < | 小于 | 表达式<表达式 | 双目运算符 |
| <= | 小于等于 | 表达式<=表达式 | 双目运算符 |
| 7 | == | 等于 | 表达式==表达式 | 左到右 | 双目运算符 |
| != | 不等于 | 表达式!= 表达式 | 双目运算符 |
| 8 | & | 按位与 | 表达式&表达式 | 左到右 | 双目运算符 |
| 9 | ^ | 按位异或 | 表达式^表达式 | 左到右 | 双目运算符 |
| 10 | | | 按位或 | 表达式|表达式 | 左到右 | 双目运算符 |
| 11 | && | 逻辑与 | 表达式&&表达式 | 左到右 | 双目运算符 |
| 12 | || | 逻辑或 | 表达式||表达式 | 左到右 | 双目运算符 |
| 13 | ?: | 条件运算符 | 表达式1? 表达式2: 表达式3 | 右到左 | 三目运算符 |
| 14 | = | 赋值运算符 | 变量=表达式 | 右到左 |  |
| /= | 除后赋值 | 变量/=表达式 |  |
| \*= | 乘后赋值 | 变量\*=表达式 |  |
| %= | 取模后赋值 | 变量%=表达式 |  |
| += | 加后赋值 | 变量+=表达式 |  |
| -= | 减后赋值 | 变量-=表达式 |  |
| <<= | 左移后赋值 | 变量<<=表达式 |  |
| >>= | 右移后赋值 | 变量>>=表达式 |  |
| &= | 按位与后赋值 | 变量&=表达式 |  |
| ^= | 按位异或后赋值 | 变量^=表达式 |  |
| |= | 按位或后赋值 | 变量|=表达式 |  |
| 15 | , | 逗号运算符 | 表达式,表达式,… | 左到右 | 从左向右顺序运算 |

1. 模块化函数：

eg.｛int a=0,b=0;cout<<a<<b<<endl;//allow｝cout<<b<<a<<endl;//not allow。因为a,b在模块里面，外面无效（注意，for也是这样）

1. 计算数组长度



1. str???的使用：里面不能用string（因为他已经够完善，直接+-就可以了）

strcmp（str1,str2）:用于比较str1,str2两字符数组或者字符串常量，元素是否相同，如果相同则return 0；也可用”????”来代替str1/2。

他接受字符串地址（指针，数组名，字符串常量）。

strcat(str1,str2):将str2接到str1后面

strcpy(str1,str2)：把str1中的值赋给str2

顺便提一下cin的多种用法：

string类：(cin>>str).get();

getline(cin,str);

char类：ch=cin.get/getline(); cin.get(line)(ch,n\_elem)

关于限制cout的长度及输出精度：

cout.setf(你想显示的格式,格式的类型);

#pragma warning(disable:4996)

void main()

{

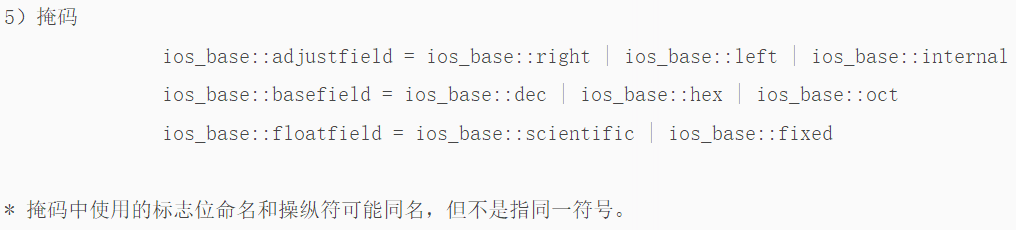
char str1[4] = "abc";

char str2[4] = "abb";

cout << "比对结果："<<strcmp(str1, str2) << endl;

}

//只有相同的时候才返回0，其他时候都是第一个在前就返回1，第二个在前就返回-1



（***cout.setf(ios::left,ios::adjustfield); cout.setf(ios::showpoint,ios::showpint);***）

cout.setbase(n);设置输出的进制（只能是8、10、16其中之一）

cout.precision(n);设置输出的精度（小数点后几位）

cout.setw(n); 设域宽为n个字符   
这个控制符的意思是保证输出宽度为n。如：   
cout<<setw(3)<<1<<setw(3)<<10<<setw(3)<<100; 输出结果为   
1 10100 （默认是右对齐）当输出长度大于3时(<<1000)，setw(3)不起作用。

参考资料：<http://blog.sina.com.cn/s/blog_69996ae60100kpdl.html>

关于new的使用：最重要的是细心！不越界（可以预先加大分配的内存）、记得删内存（加delete）



参考：<http://blog.csdn.net/wu_lai_314/article/details/8241697>

“CRT detected that the application wrote to memory after end of heap buffer”

1. 关于预处理：<http://www.cnblogs.com/liangliangh/p/3585326.html>

以及调用头文件的相关表述

C++/C程序的头文件以“.h”为后缀。以下是假设名称为 graphics.h的头文件：  
　　#ifndef GRAPHICS\_H (作用：防止graphics.h被重复引用)  
　　#define GRAPHICS\_H  
　　#include.... (作用：引用标准库的头文件)  
　　...  
　　#include... (作用：引用非标准库的头文件)  
　　...  
　　void Function1(...); (作用：全局函数声明)  
　　...  
　　class Box (作用：类结构声明)  
　　{  
　　...  
　　};  
　　#endif

注意：1.在外部使用extern访问其他文件中的数据时需要include，而且数据会同时在两边变化！

2.防止头文件名被重复利用：

1. 关于静态储存的相关事项：
2. 因为是公有的，所以可以节省内存空间（对于多对象来说，静态数组只存储一处），它还可以更新值
3. 静态数据成员使用方法：

成员在定义或说明的时候前面家关键字static。//静态变量的定义

静态成员初始化与一般的有所不同：格式：（必初始化）

***<数据类型><类型名>::<静态数据成员名>=<值>***

1. 初始化在类体外进行，而前面不加static。因为这是初始化，不是定义。

**初始化**时不应该加成员的访问权限符eg.***private,public***

初始化要使用::（作用域符号）此，静态数据成员是类的成员而不是对象的成员

1. 应用静态数据成员时：***<类名>::<静态成员名>***

EG：

class StaticTest

{

public:

StaticTest(int a, int b, int c);

void GetNumber();

void GetSum();

static void f1(StaticTest &s);

private:

int A, B, C;

static int Sum;

};---------------------------------------

#include "StaticTest.h"

#include <iostream>

using namespace std;

int StaticTest::Sum = 0;//静态成员在此初始化（不能在头文件中进行）

StaticTest::StaticTest(int a, int b, int c)

{

A = a;

B = b;

C = c;

Sum += A + B + C;

}

void StaticTest::GetNumber()

{

cout << "Number = " << endl;

}

void StaticTest::GetSum()

{

cout << "Sum = " << Sum <<endl;

}

void StaticTest::f1(StaticTest &s)

{ cout << s.A << endl;//静态方法不能直接调用一般成员，可以通过对象引用实现调用

cout << Sum <<endl;

}

#include "StaticTest.h"

#include <stdlib.h>

int main(void)

{

StaticTest M(3, 7, 10), N(14, 9, 11);

M.GetNumber();

N.GetSum();

M.GetNumber();

N.GetSum();

StaticTest::f1(M);

system("pause");

return 0;

}

·函数说明：从输出结果可以看到Sum的值对M对象和对N对象都是相等的。这是因为在初始化M对象时，将M对象的三个int型数据成员的值求和后赋给了Sum，于是Sum保存了该值。在初始化N对象时，对将N对象的三个int型数据成员的值求和后又加到Sum已有的值上，于是Sum将保存另后的值。所以，不论是通过对象M还是通过对象N来引用的值都是一样的，即为54，s.A=3。

静态成员函数

　　静态成员函数和静态数据成员一样，它们都属于类的静态成员，它们都不是对象成员。因此，对静态成员的引用不需要用对象名。

　　在静态成员函数的实现中不能直接引用类中说明的非静态成员，可以引用类中说明的静态成员（这点非常重要）。如果静态成员函数中要引用非静态成员时，可通过对象来引用。从中可看出，调用静态成员函数使用如下格式：***<类名>::<静态成员函数名>(<参数表>)***;

注意：在外部原文件中使用static是代表不让其他原文件使用。而且：

#include"stdafx.h"

#include<iostream>

using namespace std;

int sum(int x);

void main()

{

cout<< "#1 " << sum(5) << endl;// 所谓的静态就是如此，sum调用时（5）

cout << "#2 " << sum(5) << endl;//一直在使用已经被改变过的ssum（10）

}

int sum(int x)

{

static int ssum(0);

if (x == 0)

{

return ssum;

}

else

{

ssum =sum(x - 1)+1;

}

}

1. 有关返回引用类型的几点说明：
2. 返回值的时候是先创立一个返回值类型的临时变量用于储存返回值，然后模块释放空间
3. 因为他的返回值类型为引用！所以可以作为左值，也可以作为右值！可以更该值
4. 因为要释放模块的空间，所以凡是在模块里面定义的变量都不能进行返回！因为读取不了他的地址。
5. 如何使用数组表示（指向）多个函数？

***return\_typename***

***(\*name[funtion number])(typename ,typename,…)={funtion name1,funtion name2,…}***

eg:double (\*pf)[3](double,double)={sub,add,mean};

附加：如和在函数声明的时候指向函数？

eg:*double cal(double x,double y,double (\*pf)****(double,double))****;*

1. 关于输出格式的补充

C++输出格式控制

setprecision(n)可控制输出流显示浮点数的数字个数。

*C++默认的流输出数值有效位是6*，所以不管数据是多少，都只输出六位。

如果setprecision(n)与setiosflags(ios::fixed)或者setiosflags(ios\_base::fixed)合用，可以控制小数点右边的数字个数。setiosflags(ios::fixed)是用定点方式表示实数。

如果与setiosnags(ios::scientific)合用，可以控制指数表示法的小数位数。setiosflags(ios::scientific)是用指数方式表示实数。（科学计数法）

I/O流常用控制符：

使用控制符时，在程序开头加投文件*#include <iomanip>* C++有两种方法控制格式输出：1、用格式控制符；2、用流对象的成员函数 格式控制符：

dec 设置基数为10

hex 设置基数为16

oct 设置基数为8

setfill(c) 设置填充字符c

setprecision(n) 设置显示小数精度为n位

setw(n) 设置域宽为n个字符

setiosflags(ios::fixed) 固定的浮点显示

setiosflags(ios::scientific) 指数表示

setiosflags(ios::left) 左对齐

setiosflags(ios::right) 右对齐

setiosflags(ios::skipws) 忽略前导空白

setiosflags(ios::uppercase) 16进制数大写输出

setiosflags(ios::lowercase) 16进制小写输出

成员函数：

flags(10) 设置基数为10

flags(16) 设置基数为16

flags(8) 设置基数为8

flags(c) 设置填充字符c

precision(n) 设置显示小数精度为n位

width(n) 设置域宽为n个字符

在新版本的c++中头文件已经用iomanip取代了iomanip.h。

　　以下是一些常用的函数:

　　dec 置基数为10 相当于"%d"

　　hex 置基数为16 相当于"%X"

　　oct 置基数为8 相当于"%o"

　　setfill(c) 设填充字符为c

　　setprecision(n) 设显示小数精度为n位

　　setw(n) 设域宽为n个字符

　　setiosflags(ios::fixed) 固定的浮点显示

　　setiosflags(ios::scientific) 指数表示

　　setiosflags(ios::left) 左对齐

　　setiosflags(ios::right) 右对齐

　　setiosflags(ios::skipws 忽略前导空白

　　setiosflags(ios::uppercase) 16进制数大写输出

　　setiosflags(ios::lowercase) 16进制小写输出

　　setiosflags(ios::showpoint) 强制显示小数点

　　setiosflags(ios::showpos) 强制显示符号

上面的内容我自己测试一些代码，不全，但是这个格式太多了，以后用到再来看看。

View Code

可以不使用#include<iomanip>的

cout.precision()设置小数点后精确度，

cout.width()设置宽度，

cout.setf()设置显示格式，比如

cout.setf(ios::left)左对齐

cout.setf(ios::showpoint)不管是否有小数位，显示小数点

cout.fill();不足宽度则填充，如cout.fill('0');

如这次周赛1002，如果使用COUT在输出前要这样设置一下。

cout.precision(6);

cout.width(8);

cout.setf(ios::left);

cout.setf(ios::showpoint);

cout.fill('0');

仅仅cout.precision(6)和cout.setf(ios::showpoint)时，不知何原因如果为0只显示到小数点后5位，所以为了在最后加个0,要加上其它3项补充（部分是iomanip里的）：

long flags( ) const 返回当前的格式标志。

long flays(long newflag) 设置格式标志为newflag，返回旧的格式标志。

long setf(long bits) 设置指定的格式标志位，返回旧的格式标志。

long setf(long bits,long field)将field指定的格式标志位置为bits，返回旧的格式标志

long unsetf(long bits) 清除bits指定的格式标志位，返回旧的格式标志。

long fill(char c) 设置填充字符，缺省条件下是空格。

char fill( ) 返回当前填充字符。

int precision(int val) 设置精确度为val，控制输出浮点数的有效位，返回旧值。

int precision( ) 返回旧的精确度值。

int width(int val) 设置显示数据的宽度(域宽),返回旧的域宽。

int width( )只返回当前域宽，缺省宽度为0。这时插入操作能按表示数据的最小宽度显示

数据

dec 十进制的输入输出

hex 十六进制的输入输出

oct 八进制的输入输出

例如用cout<<hex<<i<<endl; 即可以让变量i以16进制的格式输出。

ws 提取空白字符

flush 刷新流

resetiosflags(long) 请除特定的格式标志位

setiosflags(long) 设置特定的格式标志位

setfill(char) 设置填充字符

setprecision(int) 设置输出浮点数的精确度

setw(int) 设置域宽格式变量

一：标准输入函数cin 不知道说它是个函数对还是不对，它是代表标准的输入设备--键盘。他是属于流的，他的用法和流的用法是一样的。也就是：cin>>变量;

小小的说明一下，输入多个变量可以写在一行，如:cin>>x>>y>>z; 这样写不是不允许，而是不好看，如果是不同的变量类型，那就更是没头没脑了。除了你，人家是不知道该输入什么的，所以，一般在输入语句的前面，我们一般都 要做一个提示，请输入×××，让人家心里有个底，知道这个变量是做什么的。 另外，这个函数是不用带地址符号"&"的，也不用写明变量类型，千万不要跟scanf混淆。当然他就也不检查变量输入是否合法。如：

int i; cout<<"please input a number:" cin>>i; cout<<"i="<<i<<endl;

如果你输入的是一个字符如'a'那么他也不检查，但你输出的结果不是正确的，这要是手工进行检查。当然他也跟scanf一样，如果在循环内部输入不合法的变量值，那么也将陷入死循环。如下：

View Code

如上一个程序，如果你输入的不合法，那就将陷入死循环。解决的办法有个一，把cin>>i;语句移到判断循环的语句中，那么，你输入的如果是不合法的变量，他将跳出循环。 cin是用空格来分隔输入的。请看看如下的例子：

/\*一个空格分隔使输入的变量达不到希望的值\*/ #include <iostream.h> main() { char str[20]; cout<<"please input a string:"; cin>>str; /\*你试着输入"hello word"\*/ cout<<endl<<"str="<<str; }

看得到是什么结果呢？得到的仅仅是str=hello，为什么呢？因为cin是以空格为分隔的，当你输入一个空格时，那他就认为后面的输入不属于这里了， 认为应该给后面的变量了。另外，当你输入的字符串大于分配的空间时，还会出现溢出现象。当然，还有整行输入的函数，包括空格也一起输入了，以后也会学到。

二、标准输出函数cout 说cout是函数，也跟cin一样，不知道对不对。他代表的是标准输出设备--显示器。其实前面已经用过很多次这个函数了。我们就通过一个例子来进行格式化的输出就是了，大家就体会体会这个例子就行了，比printf灵活了很多。 首先，我们可以按16进制，8进制和10进制来显示我们的数据，如下：

View Code

我们用cout.setf()设置输出的格式，用cout.unsetf()取消格式。可以看出10进制在输出的时候不管有没有设置基指示

符ios:: showbase，都没用，8进制再输出的时候在前面加0，而16进制是在前面加0X。而对于数值中字母大写输出，只对16进制

有用，以后我们就应该看情 况使用了。当然，我们前面已经说了，还有一种方法也可以实现格式化输出，那就是使用操纵算子，如下，

View Code

我们用以上的程序也可以输出同样的结果，可见他的灵活。我们现在输出下列一段文字：

第一章

1.1 什么是C语言...........................1

1.11 C语言的历史..........................58

第二章

方法很多种啦，我们可以这样写：

View Code

我们多次设置了宽度，为的是使我们的间距能一致，也使用了对齐方式，为的是使我们的数据能对齐显示，看起来美观

。我们还使用了填充方式。我们下面用操纵算子来实现也是可以的。

View Code

我们输出了同样的效果，不过依我的性格，我更喜欢用操纵算子来进行格式化输出。最后我们看看浮点数的格式输出，

如下例：

View Code

同样，我们也一样能用操纵算子实现同样的功能：

View Code

在c/c++系统中除了标准的输入输出外，还提供了更多的输入函数。

这写函数主要有

getch(),getche(),getchar(),cin.get(),putch(),putchar(),cout.put(),gets(),cin.getline(),puts()。

另外 还有些为了让缓冲区不影响程序的正确操作的缓冲去的操作，如：cin.putback(),fflush(stdin),cout.flush().我们 做一下简单的说明。

1、getch()和getche(),非缓冲式输入，从键盘读入一个字符。getch()读入字符不显示。有conio.h支持。

2、cin.get(),getchar()，缓冲式输入，从键盘读入一个字符，并显示。getchar()由stdio.h支持，cin.get()由iostream.h支持。

3、putch()和putchar()，非缓冲式输出，输出一个字符到显示器。putch()由conio.h支持，putchar()由stdio.h支持。

4、cout.put()，缓冲式输出，输出一个字符到显示器。由iostream.h支持。

5、gets()和cin.geline()，缓冲式输入，读入一字符串（包括空格，不包括最后的回车），gets()由stdio.h支持，cin.getline()由iostream.h支持。

6、puts()，非缓冲输出，输出一个字符串，由stdio.h支持。

7、cin.putback()，把一个字符送回输入缓冲区。

8、fflush(stdin)，清除输入缓冲区操作。无法清除cin.get()等带来的操作。

9、cout.flush()，清楚输出缓冲区。 在这里我们稍微说一下输入/输出缓冲区，这是为了减少程序访问io带来中断而设的一段空间。当程序满足某个刷新条件时，那就将清理缓冲区。具体条件为：

1、输入缓冲区 a，程序要求输入时，按下了回车键。 b，遇到程序结束。 c，遇到下一条输入语句。 d，遇到清除缓冲区操作 e，缓冲区溢出 2、输出缓冲区 a，输出缓冲区溢出 b，遇到下一条输入语句 c，使用刷新缓冲区迫使清除

例2 用控制符控制输出格式

View Code

注意：当对一个变量的输出使用了cout.setf(ios::showpoint)时，他接下来的全部输出都是showpoint（这个目标的一生~）

***附：了解一级指针的引用和二级指针在链表初始化中的应用：***

[***https://www.cnblogs.com/D-DZDD/p/7245890.html***](https://www.cnblogs.com/D-DZDD/p/7245890.html)

注：一级指针上面的值可以看作是值，可以通过赋值来进行对地址上面的值进行修改，但是，如果要修改（更改指向）值下面的指针就要求指针的地址了，然后通过他的地址进行修改（\*/\*\*/&\*只是修改的方式，值，指针，引用）

得出结论

1，初始化链表头部指针需要用二级指针或者一级指针的引用。

2，销毁链表需要用到二级指针或者一级指针的引用。

3，插入、删除、遍历、清空结点用一级指针即可。

分析：

1，只要是修改头指针则必须传递头指针的地址，否则传递头指针值即可（即头指针本身）。这与普通变量类似，当需要修改普通变量的值，需传递其地址，否则传递普通变量的值即可（即这个变量的拷贝）。使用二级指针，很方便就修改了传入的结点***一级指针的值***。 如果用一级指针，则***只***能通过指针*修改****指针所指内容***，却无法修改指针的值，也就是指针所指的内存块。所以创建链表和销毁链表需要二级指针或者一级指针引用。

2，不需要修改头指针的地方用一级指针就可以了，比如插入，删除，遍历，清空结点。假如头指针是L，则对L->next 及之后的结点指针只需要传递一级指针。

3，比如一个结点p，在函数里要修改p的指向就要用二级指针，如果只是修改p的next指向则用一级指针就可以了

函数中传递指针，在函数中改变指针的值，就是在改变实参中的数据信息。但是这里改变指针的值实际是指改变指针指向地址的值，因为传递指针就是把指针指向变量的地址传递过来，而不是像值传递一样只是传进来一个实参副本。所以当我们改变指针的值时，实参也改变了。

二分法：示例：

#include "stdafx.h"

using namespace std;

const int N = 10;

int main()

{

int arr[N], head, tail, mid, x, i, temp(0);

cout << "赋值！" << endl;

for (i = 0; i < N; i++)

{

arr[i] = i;

}

cout << "赋值完毕！" << "\n输入查找的数：";

cin >> x;

head = 0;

tail = N - 1;

mid =(int)(N - 1) / 2;

for (; x != arr[mid];)

{

if (x > arr[mid])

{

head = mid;

}

else {

tail = mid;

}

*mid = (int)(head + tail) / 2;*

}

cout << "所处位置：" << mid << endl;

return 0;

}

（主要思想：缩小区间！）

涉及知识点：强制缩窄变换：在前面加上(typename)；

16、名称空间的使用：

一般情况下，都会把类（class）和结构放在名称空间里面【注意：声明和定义必须在同一个命名空间里】，然后尽量少在头文件中使用using，倒入名称时，首选将其作用域设置为局部而不是全局首选作用域解释符或者using声明法。

eg:

object.h

#pragma once

namespace pers

{

struct ID

{

double id;

char name[20];

};

void where(double);

}

#include"stdafx.h"

#include<iostream>

#include"object.h";

using namespace std;

int main()

{

using namespace pers;//1

using pers::ID;//2

using pers::where;//2

return 0;

}

如果是int，你给他输入3.4，然后下面再次要你输入时，他就会这样：

int main()

{

int a;

cin >> a;

cin.clear();

***int/double*** b;

cin >> b;

cout << a << " " << b << endl;

return 0;

}

版本1：

3.6

3 -858993460

请按任意键继续. . .

*实际int只提取3，留下.6，再让后面的提取*

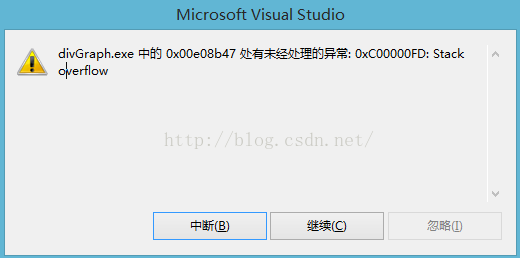
版本2：

3.6

3 0.6

请按任意键继续. . .

17.问题描述：c++程序运行时候出现：divGraph.exe 中的 0x00e08b47 处有未经处理的异常: 0xC00000FD: Stack overflow报错。



分析：Windows系统默认分页大小为1M 1024 \*1024，当申请的空间大于1M时就会出现如下问题：

         在Debug下调试程序，出现如下错误：test dword ptr [eax],eax ; probe page.   
         Stack overflow。

解决方案：1.减小申请空间的大小。（一般是数组空间）  
                  2.增大堆栈空间的大小。 方法为：项目->属性->链接器->系统->堆栈保留大小

参考：<http://www.educity.cn/wenda/365407.html>

18. 堆（Stack）已损坏大多是由于越界操作引起的（stack overflow）

19. 通用保护错误（GPF），表明程序试图访问禁止他访问的内存单元，是一种糟糕的信号。

20.初始化和赋值：

折腾我挺长一段时间,基本挺明白了，先来个区别说明：赋值操作是在两个已经存在的对象间进行的，而初始化是要创建一个新的对象，并且其初值来源于另一个已存在的对象。编译器会区别这两种情况，赋值的时候调用重载的赋值运算符，初始化的时候调用拷贝构造函数。如果类中没有拷贝构造函数，则编译器会提供一个默认的。这个默认的拷贝构造函数只是简单地复制类中的每个成员。 下面看例子。

c++中初始化和赋值操作差别是很大的。

　　对于基本数据类型差别不大：

　　比如：

　　int a = 12; // initialization, copy 0X000C to a

　　a = 12; // assignment, copy 0X000C to a

　　但是对用户自定义的数据类型比如String 初始化和赋值就差别很大：

class String {

　　 public:

　　 String( const char \*init ); // intentionally not explicit!

　　 ~String();

　　 String( const String &that );

　　 String &operator =( const String &that );

　　 String &operator =( const char \*str );

　　 void swap( String &that );

　　 friend const String // concatenate

　　 operator +( const String &, const String & );

　　 friend bool operator <( const String &, const String & );

　　 //...

　　 private:

　　 String( const char \*, const char \* ); // computational

　　 char \*s\_;

　　};

　　初始化的构造过程比较简单：先分配一个足够大的空间然后填充上数据：

　　String::String( const char \*init ) {

　　 if( !init ) init = "";

　　 s\_ = new char[ strlen(init)+1 ];

　　 strcpy( s\_, init );

　　}

　　析构过程更简单：

String::~String() { delete [] s\_; }

　　但是如果赋值操作就复杂多了：

　　String &String::operator =( const char \*str ) {

　　 if( !str ) str = "";

　　 char \*tmp = strcpy( new char[ strlen(str)+1 ], str ); // 多了中间变量

　　 delete [] s\_;　// 多了删除s\_;

　　 s\_ = tmp; 　// 多一个赋值操作！现在是指向字符的指针，如果是个大对象，效率的差别可想而知．

　　 return \*this;

　　}

建议在条件允许的情况下最好在初始化的时候就赋值，而尽量避免用=号赋值了，比如用成员初始化列表来初始化成员数据，不在构造函数里用赋值操作给成员数据．

17.0，NULL，nullptr都可以用来表示空的意思；但是，最好用nullptr来表示空指针。

18. 0x003E66EB 处有未经处理的异常(在 Project1.exe 中): 0xC0000005: 读取位置 0xCDCDCDCD 时发生访问冲突。 出现了

一般是使用了已经被释放的空间（或者没有进行分配？）

19. **char \*p[] 是数组**，**元素类型为字符指针**。作变量定义时需要有元素数量，或者有初始化表达式以确定元素数量。

char \*\*p 是指针，其指向的一个字符指针。通常用于遍历像上面这样的数组。

char (\*p)[] 是指针，其指向的是名义上的数组（因为没有具体数组元素数目）。作变量定义时需要明确其所指的数组的元素数目，这对像之后的 p++ 这类操作的结果有决定性影响。通常用于遍历二维数组。

以上这三个，作函数参数声明时，方括号内可为空。

若为空，则以上三个语义完全相同，C编译器无法对它们进行区别，对应的实参类型都必须是一个指针变量的地址。C99规定，作参数声明时，“指向T的指针”和“T的数组”没有区别。

若方括号内指定了具体元素数目，则效果等于变量定义时的效果。再加一个char p[][8]，做函数参数声明时 等价于 char (\*p)[8]。

20.纯虚函数：

纯虚函数是一种特殊的虚函数，它的一般格式如下：

class <类名>

{

virtual <类型><函数名>(<参数表>)=0;

…

};

在许多情况下，在基类中不能对虚函数给出有意义的实现，而把它声明为纯虚函数，它的**实现留给该基类的派生类去做**。这就是纯虚函数的作用。

纯虚函数可以让类先具有一个操作名称，而没有操作内容，让派生类在继承时再去具体地给出定义。凡是含有纯虚函数的类叫做抽象类。这种类不能声明对象，只是作为基类为派生类服务。除非在派生类中完全实现基类中所有的的纯虚函数，否则，派生类也变成了抽象类，不能实例化对象。

一般而言纯虚函数的函数体是缺省的，但是*也可以给出纯虚函数的函数体（此时纯虚函数变为虚函数）*，这一点经常被人们忽视，调用纯虚函数的方法为***baseclass::virtual function***

21.静态数据成员**不能**在头文件中初始化，因为可能造成多次声明

静态成员函数只能用静态成员

22.如果你NEW了2个成员，你用delete a,b;只能释放a。

23.建议如果使用eg构造函数等有转换功能的函数，就应该在前面声明explicit（禁止隐式调用）//小知识：bool类型，如果没有初始化可能会自动给true(最常)或者false（主要看编译器）

24.exit(0)和exit(1)有什么不一样？

exit是一个函数，进程退出时会有一个值，exit函数的参数就是指明进程退出的返回值，操作系统根据这个值来判断是否是正常退出。你也可以通过GetExitCodeProcess来获取这个值（windows下），一般情况下退出值是0表示正常(exit(0))，其它情况都是不正常的。

25.检查cin错误所引发的异常，使用cin.fail()，返回runtime\_error即可