# 牛客网 [c++校招面试题目合集](https://www.nowcoder.com/ta/review-c)

## 分别给出BOOL，int，float，指针变量 与“零值”比较的 if 语句（假设变量名为var）

BOOL型变量：if(!var)

int型变量： if(var==0)

float型变量：

const float EPSINON = 0.00001;

if ((x >= - EPSINON) && (x <= EPSINON)

指针变量：　　if(var==NULL)

## 为什么标准头文件都有类似以下的结构？

#ifndef \_\_INCvxWorksh

#define \_\_INCvxWorksh

#ifdef \_\_cplusplus

extern "C" {

#endif

/\*...\*/

#ifdef \_\_cplusplus

}

#endif

#endif /\* \_\_INCvxWorksh \*/

解释：

作为一种面向对象的语言，C++支持函数重载，而过程式语言C则不支持。函数被C++编译后在symbol库中的名字与C语言的不同。例如，假设某个函数的原型为：   
void foo(int x, int y);   
该函数被C编译器编译后在symbol库中的名字为\_foo，而C++编译器则会产生像\_foo\_int\_int之类的名字。\_foo\_int\_int这样的名字包含了函数名和函数参数数量及类型信息，C++就是靠这种机制来实现函数重载的。   
为了实现C和C++的混合编程，C++提供了C连接交换指定符号extern "C"来解决名字匹配问题，函数声明前加上extern "C"后，则编译器就会按照C语言的方式将该函数编译为\_foo，这样C语言中就可以调用C++的函数了。

## 请说出static和const关键字尽可能多的作用

【解答】   
static关键字至少有下列n个作用：     
（1）函数体内static变量的作用范围为该函数体，不同于auto变量，该变量的内存只被分配一次，因此其值在下次调用时仍维持上次的值；     
（2）在模块内的static全局变量可以被模块内所用函数访问，但不能被模块外其它函数访问；     
（3）在模块内的static函数只可被这一模块内的其它函数调用，这个函数的使用范围被限制在声明它的模块内；     
（4）在类中的static成员变量属于整个类所拥有，对类的所有对象只有一份拷贝；     
（5）在类中的static成员函数属于整个类所拥有，这个函数不接收this指针，因而只能访问类的static成员变量。      
const关键字至少有下列n个作用：     
（1）欲阻止一个变量被改变，可以使用const关键字。在定义该const变量时，通常需要对它进行初始化，因为以后就没有机会再去改变它了；     
（2）对指针来说，可以指定指针本身为const，也可以指定指针所指的数据为const，或二者同时指定为const；     
（3）在一个函数声明中，const可以修饰形参，表明它是一个输入参数，在函数内部不能改变其值；     
（4）对于类的成员函数，若指定其为const类型，则表明其是一个常函数，不能修改类的 成员变量；     
（5）对于类的成员函数，有时候必须指定其返回值为const类型，以使得其返回值不为“左值”。例如：     
const classA operator\*(const classA& a1,const classA& a2);     
operator\*的返回结果必须是一个const对象。如果不是，这样的变态代码也不会编译出错：     
classA a, b, c;     
(a \* b) = c; // 对a\*b的结果赋值     
操作(a \* b) = c显然不符合编程者的初衷，也没有任何意义。

## 说一下static关键字的作用

参考回答:

1. 全局静态变量

在全局变量前加上关键字static，全局变量就定义成一个全局静态变量.

静态存储区，在整个程序运行期间一直存在。

初始化：未经初始化的全局静态变量会被自动初始化为0（自动对象的值是任意的，除非他被显式初始化）；

作用域：全局静态变量在声明他的文件之外是不可见的，准确地说是从定义之处开始，到文件结尾。

2.  局部静态变量

在局部变量之前加上关键字static，局部变量就成为一个局部静态变量。

内存中的位置：静态存储区

初始化：未经初始化的全局静态变量会被自动初始化为0（自动对象的值是任意的，除非他被显式初始化）；

作用域：作用域仍为局部作用域，当定义它的函数或者语句块结束的时候，作用域结束。但是当局部静态变量离开作用域后，并没有销毁，而是仍然驻留在内存当中，只不过我们不能再对它进行访问，直到该函数再次被调用，并且值不变；

3. 静态函数

在函数返回类型前加static，函数就定义为静态函数。函数的定义和声明在默认情况下都是extern的，但静态函数只是在声明他的文件当中可见，不能被其他文件所用。

函数的实现使用static修饰，那么这个函数只可在本cpp内使用，不会同其他cpp中的同名函数引起冲突；

warning：不要再头文件中声明static的全局函数，不要在cpp内声明非static的全局函数，如果你要在多个cpp中复用该函数，就把它的声明提到头文件里去，否则cpp内部声明需加上static修饰；

4. 类的静态成员

在类中，静态成员可以实现多个对象之间的数据共享，并且使用静态数据成员还不会破坏隐藏的原则，即保证了安全性。因此，静态成员是类的所有对象中共享的成员，而不是某个对象的成员。对多个对象来说，静态数据成员只存储一处，供所有对象共用

5. 类的静态函数

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>);

## 说一下C++和C的区别

面向对象与面向过程的却别说明：https://zhuanlan.zhihu.com/p/75265007

设计思想上：

C++是面向对象的语言，而C是面向过程的结构化编程语言

语法上：

C++具有封装、继承和多态三种特性

C++相比C，增加多许多类型安全的功能，比如强制类型转换、

C++支持范式编程，比如模板类、函数模板等

## 说一说c++中四种cast转换

参考回答:

C++中四种类型转换是：static\_cast, dynamic\_cast, const\_cast, reinterpret\_cast

1、const\_cast

用于将const变量转为非const

2、static\_cast

用于各种隐式转换，比如非const转const，void\*转指针等, static\_cast能用于多态向上转化，如果向下转能成功但是不安全，结果未知；

3、dynamic\_cast

用于动态类型转换。只能用于含有虚函数的类，用于类层次间的向上和向下转化。只能转指针或引用。向下转化时，如果是非法的对于指针返回NULL，对于引用抛异常。要深入了解内部转换的原理。

向上转换：指的是子类向基类的转换

向下转换：指的是基类向子类的转换

它通过判断在执行到该语句的时候变量的运行时类型和要转换的类型是否相同来判断是否能够进行向下转换。

4、reinterpret\_cast

几乎什么都可以转，比如将int转指针，可能会出问题，尽量少用；

5、为什么不使用C的强制转换？

C的强制转换表面上看起来功能强大什么都能转，但是转化不够明确，不能进行错误检查，容易出错。

## 请说一下C/C++ 中指针和引用的区别？

参考回答:

1.指针有自己的一块空间，而引用只是一个别名；

2.使用sizeof看一个指针的大小是4，而引用则是被引用对象的大小；

3.指针可以被初始化为NULL，而引用必须被初始化且必须是一个已有对象 的引用；

4.作为参数传递时，指针需要被解引用才可以对对象进行操作，而直接对引 用的修改都会改变引用所指向的对象；

5.可以有const指针，但是没有const引用；

6.指针在使用中可以指向其它对象，但是引用只能是一个对象的引用，不能 被改变；

7.指针可以有多级指针（\*\*p），而引用至于一级；

8.指针和引用使用++运算符的意义不一样；

9.如果返回动态内存分配的对象或者内存，必须使用指针，引用可能引起内存泄露。

## 8. new和malloc的区别

1、new分配内存按照数据类型进行分配，malloc分配内存按照指定的大小分配；

2、new返回的是指定对象的指针，而malloc返回的是void\*，因此malloc的返回值一般都需要进行类型转化。

3、new不仅分配一段内存，而且会调用构造函数，malloc不会。

4、new分配的内存要用delete销毁，malloc要用free来销毁；delete销毁的时候会调用对象的析构函数，而free则不会。

5、new是一个操作符可以重载，malloc是一个库函数。

6、malloc分配的内存不够的时候，可以用realloc扩容。扩容的原理？new没用这样操作。

7、new如果分配失败了会抛出bad\_malloc的异常，而malloc失败了会返回NULL。

8、申请数组时： new[]一次分配所有内存，多次调用构造函数，搭配使用delete[]，delete[]多次调用析构函数，销毁数组中的每个对象。而malloc则只能sizeof(int) \* n。

## 9. 如何判断内存泄漏？

为了判断内存是否泄露，我们一方面可以使用linux环境下的内存泄漏检查工具Valgrind,另一方面我们在写代码时可以添加内存申请和释放的统计功能，统计当前申请和释放的内存是否一致，以此来判断内存是否泄露。

## 10. 什么时候会发生段错误

访问了不属于进程地址空间的内存,不存在的内存

内存越界，数组越界，试图修改字符串常量的内容 等

## 11. 什么是memory leak，也就是内存泄漏

内存泄漏的分类：

1. 堆内存泄漏 （Heap leak）。对内存指的是程序运行中根据需要分配通过malloc,realloc new等从堆中分配的一块内存，再是完成后必须通过调用对应的 free或者delete 删掉。如果程序的设计的错误导致这部分内存没有被释放，那么此后这块内存将不会被使用，就会产生Heap Leak.

2. 系统资源泄露（Resource Leak）。主要指程序使用系统分配的资源比如 Bitmap,handle ,SOCKET等没有使用相应的函数释放掉，导致系统资源的浪费，严重可导致系统效能降低，系统运行不稳定。

3. 没有将基类的析构函数定义为虚函数。当基类指针指向子类对象时，如果基类的析构函数不是virtual，那么子类的析构函数将不会被调用，子类的资源没有正确是释放，因此造成内存泄露。

## 12. 如何采用单线程的方式处理高并发

在单线程模型中，可以采用I/O复用来提高单线程处理多个请求的能力，

然后再采用事件驱动模型，基于异步回调来处理事件来

## 13. C++11中的可变参数模板、右值引用和lambda这几个新特性

<https://www.nowcoder.com/ta/review-c/review?page=56>

## 14. Linux虚拟地址空间

为了防止不同进程同一时刻在物理内存中运行而对物理内存的争夺和践踏，采用了虚拟内存。

虚拟内存技术使得不同进程在运行过程中，它所看到的是自己独自占有了当前系统的4G内存。所有进程共享同一物理内存，每个进程只把自己目前需要的虚拟内存空间映射并存储到物理内存上。 事实上，在每个进程创建加载时，内核只是为进程“创建”了虚拟内存的布局，具体就是初始化进程控制表中内存相关的链表，实际上并不立即就把虚拟内存对应位置的程序数据和代码（比如.text .data段）拷贝到物理内存中，只是建立好虚拟内存和磁盘文件之间的映射就好（叫做存储器映射），等到运行到对应的程序时，才会通过缺页异常，来拷贝数据。还有进程运行过程中，要动态分配内存，比如malloc时，也只是分配了虚拟内存，即为这块虚拟内存对应的页表项做相应设置，当进程真正访问到此数据时，才引发缺页异常。

请求分页系统、请求分段系统和请求段页式系统都是针对虚拟内存的，通过请求实现内存与外存的信息置换。

虚拟内存的好处：

1.扩大地址空间；

2.内存保护：每个进程运行在各自的虚拟内存地址空间，互相不能干扰对方。虚存还对特定的内存地址提供写保护，可以防止代码或数据被恶意篡改。

3.公平内存分配。采用了虚存之后，每个进程都相当于有同样大小的虚存空间。

4.当进程通信时，可采用虚存共享的方式实现。

5.当不同的进程使用同样的代码时，比如库文件中的代码，物理内存中可以只存储一份这样的代码，不同的进程只需要把自己的虚拟内存映射过去就可以了，节省内存

6.虚拟内存很适合在多道程序设计系统中使用，许多程序的片段同时保存在内存中。当一个程序等待它的一部分读入内存时，可以把CPU交给另一个进程使用。在内存中可以保留多个进程，系统并发度提高

7.在程序需要分配连续的内存空间的时候，只需要在虚拟内存空间分配连续空间，而不需要实际物理内存的连续空间，可以利用碎片

虚拟内存的代价：

1.虚存的管理需要建立很多数据结构，这些数据结构要占用额外的内存

2.虚拟地址到物理地址的转换，增加了指令的执行时间。

3.页面的换入换出需要磁盘I/O，这是很耗时的

4.如果一页中只有一部分数据，会浪费内存。

# 牛客C/C++错题

## 1．虚函数使用

https://www.nowcoder.com/test/question/done?tid=39952465&qid=112248#summary

因为虚函数本质上是通过类对象的虚表进行访问，而且类的成员函数除了虚函数，其他都不存储在类当中，因此类对象不存在的情况下，无法使用虚函数，其他函数都可以正常访问（前提是这些函数都没有存取类对象的成员变量）

总结：访问虚函数，必须有对象的存在。原因：对象中存放虚函数表指针

## 2.一个const对象只能访问const成员函数

<https://www.nowcoder.com/test/question/done?tid=39952465&qid=56839#summary>

C++ Primer 中说了，在一个对象调用其成员函数时，它隐含的一个形参 this 指针。

例如，我们定义了一个函数 CTest::ttt(); 实际上在编译器中该函数的定义就是 CTest::ttt(CTest \*const this) ，该 this 指针所指向的内容可以改变，但是该 this 指针不可以被改变。当我们用 CTest 的一个对象 test1 调用 ttt 函数时即 test1.ttt() 时，编译器就会将它解释为 ttt(&test1) ，所以我们在 ttt 中使用 this 指针就可以改变对象 test1 的成员变量了。

但是，当我们的对象是 const 对象时，即 const CTest ttt; 这时候表示什么意思呢， ttt 的内容是不可以改边的，当我们把 &ttt 作为一个参数传到形参 this 时，矛盾出现了： ttt 是一个常量，其成员不可以被改变； this 指针的成员变量是可以改变的。如果我能正确的将 ttt 的地址传给 this ，那么 ttt 这个常量的值不是可以在 this 中被改变了吗，所以，编译器是不允许这种情况出现的，就提示错误了。故， const 对象不能访问非 const 成员函数。

同理，当我们的成员函数是 const 成员函数时，例： CTest::ttt() const, 在编译器解释时会将该函数解释为 CTest::ttt(const CTest \* const this),this 指针及其所指向的内容都不可以被修改，前面提到的矛盾也就不存在了，所以 const 对象可以访问 const 成员变量。

## 3. 在C++中，引用和指针的区别是

引用总是指向一个对象,指针可能不指向对象

引用不能用const修饰，而指针可以

引用创建时必须初始化，而指针则可以在任何时候被初始化

## 4.memmove接口

https://www.nowcoder.com/test/question/done?tid=39991607&qid=14696#summary

原型：void \*memmove( void\* dest, const void\* src, size\_t count );

功能：由src所指内存区域复制count个字节到dest所指内存区域。

如果目标区域和源区域有重叠的话，memmove能够保证源串在被覆盖之前将重叠区域的字节拷贝到目标区域中。但复制后src内容会被更改。

## 5. 关于子类型的描述中，（C ）是错误的？

https://www.nowcoder.com/test/question/done?tid=39991607&qid=25525#summary

在公有继承下,派生类是基类的子类型

子类型关系是不可逆的

子类型就是指派生类是基类的子类型

一种类型当它至少提供了另一种类型的行为,则这种类型是另一种类型的子类型

子类型必须是子类继承了父类的所有可继承特性，也即公有继承，才能说是子类型，否则就只是单纯的子类

## 6. 结构化程序设计的思想包括：自顶向下、逐步求精、模块化、限制使用 goto 语句

## 7.四种类型转换

dynamic\_cast :

继承体系安全向下转型或跨系转型；找出某对象占用内存的起始点

static\_cast:

同旧式C转型，如int 到double

const\_cast:

常用于去除某个对象的常量性

reinterpret\_cast

不具备移植性，常见用途是转化函数指针类型

## 8. 为避免运算过程中出现整型溢出可以考虑的办法有（）

将运算结果和可以表示的最大整数进行比较

检测符号位的变化

将计算结果减去加数看是否与另一加数相等

比较参数的长度

选项1 ： 运算结果进行比较的话，已经溢出导致结果变化，去比较也为时已晚。 所以不正确

选项2 ： 检测符号为变化可以防止符号溢出，正确。

选项3 ： a+b = c   c - a != b 则c溢出 正确

选项4 ： 参数长度    0000000000000000000000000 这个长度算溢出么 不正确

## 9. 观察下面一段代码

class ClassA

{

public:

virtual ~ ClassA(){};

virtual void FunctionA(){};

};

class ClassB

{

public:

virtual void FunctionB(){};

};

class ClassC : public ClassA,public ClassB

{

public:

};

ClassC aObject;

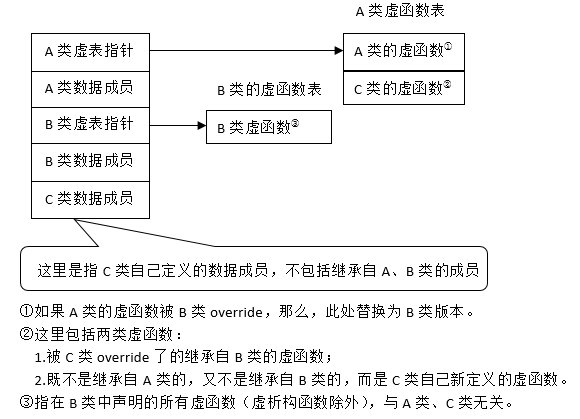
ClassA\* pA=&aObject;

ClassB\* pB=&aObject;

ClassC\* pC=&aObject;

关于pA,pB,pC的取值,下面的描述中正确的是:

pA和pB不相同



## 10. 关于C++中的友元函数说法正确的是

https://www.nowcoder.com/test/question/done?tid=40115036&qid=313999#summary

友元函数需要通过对象或指针调用

友元函数是不能被继承的

友元函数没有this指针

友元函数破环了继承性机制

## 11．下列描述中，（ ）是错误的。

正确答案: C   你的答案: B (错误)

内联函数的主要解决程序的运行效率问题。

内联函数的定义或声明必须出现在内联函数第一次被调用之前。

内联函数中可以包括各种语句

对内联不可以进行异常接口声明

使用内联函数的时候要注意：

1. 递归函数不能定义为内联函数
2. 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，否则编译系统将该函数视为普通函数。
3. 内联函数只能先定义后使用，否则编译系统也会把它认为是普通函数。
4. 对内联函数不能进行异常的接口声明。

## 12.程序输出的是什么

void main (void) {

    char a[] = “SF-TECH” ;

    a++;

    printf (“%s”, a);

}

在C++中，虽然数组名是一个指向数组首地址的指针，但是，数组名是一个常量指针，常量是不能放在=的左边的。数组和指针的区别在于，指针是变量，是用来存储指向数据的地址的变量，而数组名是常量。所以不能对数组名直接进行++操作。可以重新定义指针，指向数组首地址，对其进行++操作。

#include <stdio.h>

void main (void) {

char \*ptr;

char a[] = "SF-TECH";

ptr = a;

ptr++;

// a++;

printf ("%s\n", ptr);

}

## 13．下述有关c++的虚类和java接口的描述，说法错误的是？

正确答案: C D   你的答案: A B (错误)

c++虚类相当与java里面的抽象类

c++中没有接口的概念，与之对应的是纯虚类，对应的是java的接口

纯虚函数和虚函数的区别在于前者只包含定义，而后者还可以包含函数体。

一个抽象类和接口中的方法必须是抽象方法

## 14. 下列代码中的错误是（）

(1)   public class Test

(2)   {

(3)       public static void main(String [] args)

(4)       {

(5)           int i;

(6)           i+=1;

(7)       }

(8)     }

非法的表达式 i+=1

找不到符号i

类不应为public

尚未初始化变量i

**在方法内定义的变量在使用之前必须初始化，否则报错。**

## 15. 有以下代码程序的输出结果是?

#include <stdio.h>

#include <stdlib.h>

void fun ( double \*pl,double \*p2,double \*s)

{

s ＝ ( double\*) calloc ( 1,sizeof(double));

\*s ＝ \*pl + \*(p2+1);

｝

main( )

{

double a [2] ＝ ｛1.1,2.2｝,b [2] ＝ ｛10.0,20.0｝, \*s ＝ a;

fun (a,b,s);

printf ( "%5.2f\n",\* s) ;

｝

Print: 1.10

如果想输出21.10

void function(double \*pl, double \*p2, double \*s)

{

double\* p = (double\*)calloc(1, sizeof(double));

\*p = \*pl + \*(p2 + 1);

\*s = \*p;

}

int main()

{

double a[2] = { 1.1, 2.2 }, b[2] = {10.0, 20.0}, \*s = a;

function(a, b, s);

printf("%5.2f\n", \*s);

system("pause");

return 0;

}

## 16. 运行时的输出结果是（）

#include<iostream>

using namespace std;

class MyClass

{

public:

MyClass(int i = 0)

{

cout << i;

}

MyClass(const MyClass &x)

{

cout << 2;

}

MyClass &operator=(const MyClass &x)

{

cout << 3;

return \*this;

}

~MyClass()

{

cout << 4;

}

};

int main()

{

MyClass obj1(1), obj2(2);

MyClass obj3 = obj1;

return 0;

}

结果：122444

解析：

MyClass obj3 = obj1;

obj3还不存在，所以调用拷贝构造函数输出2，

如果obj3存在，obj3=obj，则调用复制运算符重载函数，输出3

## 17. 在重载某运算符时，若运算符函数的形参表中没有参数，则可能的情况有( )

ABC

该运算符是一个单目运算符。

该运算符函数有一个隐含的参数this。

该运算符函数是类的成员函数。

该运算符函数是类的友元函数。

语法规定 前置单目运算符重载为成员函数时没有形参,后置单目运算符重载为成员函数时需要一个int型形参..有可能所以A对

友元函数的参数：

因为友元函数没有this指针，则参数要有三种情况：

1、 要访问非static成员时，需要对象做参数；--常用(友元函数常含有参数)

2、 要访问static成员或全局变量时，则不需要对象做参数

3、 如果做参数的对象是全局对象，则不需要对象做参数，所以D错

## 18.以下说法正确的是

正确答案: A B D 你的答案: A D (错误)

gcc编译.c文件，\_\_cplusplus没有定义，编译器按照c编译代码

gcc编译.cpp文件，\_\_cplusplus有定义，编译器按照c++编译代码

g++编译.c文件, \_\_cplusplus没有定义，编译器按照c编译代码

g++编译.cpp文件， \_\_cplusplus有定义，编译器按照c++编译代码

## 19. 看以下代码：

class A

{

public:

~A();

};

A::~A()

{

printf("delete A ");

}

class B : public A

{

public:

~B();

};

B::~B()

{

printf("delete B ");

}

请问执行以下代码输出什么

A \*pa = new B();

delete pa;

经测试，有两种情况，

1）A 基类析构函数未加virtual，

执行“A \*pa = newB();delete pa; ”，输出“delete A”；

执行“B \*pa = newB();delete pa; ”，输出“delete B delete A”；

2)A 基类析构函数加virtual修饰，

执行“A \*pa = newB();delete pa; ”，输出“delete B delete A”；

执行“B \*pa = newB();delete pa; ”，输出“delete B delete A”；

## 20．若已定义的函数有返回值，则以下关于该函数调用的叙述中错误的是（）。

正确答案: D   你的答案: C (错误)

函数调用可以作为独立的语句存在

函数调用可以作为一个函数的实参

函数调用可以出现在表达式中

函数调用可以作为一个函数的形参

解释：

返回值存在寄存器中，没有地址，不能作为形参，但可以作为实参。

函数调用参数的数据传递是单向传递，即实参传给形参，不能由形参传给实参。所以函数调用不可以作为另一个函数的形参！

## 21. 以下函数中，和其他函数不属于一类的是\_\_\_\_\_。

https://www.nowcoder.com/test/question/done?tid=40248494&qid=25450#summary

正确答案: D   你的答案: A (错误)

fread

gets

getchar

pread

getline

scanf

pread是系统调用，其他事IO函数

## 22. 友元运算符 a++ 被C++编译器解释为:

正确答案: B   你的答案: D (错误)

operator++(a)

a.operator++(0)

operator++(a,0)

a.operator++()

解释

a+b operator+(a,b) a.operator+(b)  
-a operator-(a) a.operator-()  
a++ operator++(a,0) a.operator++(0)  
–a operator–(a) a.operator–();

# 牛客面经

## 面经一

### 堆和栈的区别，怎样解决栈溢出问题

区别

上一篇准备中堆和栈的区别

栈溢出的几种情况

一、局部数组过大。当函数内部的数组过大时，有可能导致堆栈溢出。

二、递归调用层次太多。递归函数在运行时会执行压栈操作，当压栈次数太多时，也会导致堆栈溢出。

三、指针或数组越界。这种情况最常见，例如进行字符串拷贝，或处理用户输入等等。

解决这类问题的办法有两个，

一是增大栈空间,二是改用动态分配，使用堆（heap）而不是栈（stack）。

1. 同步和异步的区别

同步，可以理解为在执行完一个函数或方法之后，一直等待系统返回值或消息，这时程序是出于阻塞的，只有接收到返回的值或消息后才往下执行其他的命令。

异步，执行完函数或方法后，不必阻塞性地等待返回值或消息，只需要向系统委托一个异步过程，那么当系统接收到返回值或消息时，系统会自动触发委托的异步过程，从而完成一个完整的流程。

同步实时处理结果（如打电话）

异步分时处理结果（如发短信）

# 涉及的概念

## 1.抽象类

1、一个子类只能继承一个抽象类（虚类），但能实现多个接口；

2、一个抽象类可以有构造方法，接口没有构造方法；

3、一个抽象类中的方法不一定是抽象方法，即其中的方法可以有实现（有方法体），接口中的方法都是抽象方法，不能有方法体，只有声明；

4、一个抽象类可以是public、private、protected、default,

接口只有public;

5、一个抽象类中的方法可以是public、private、protected、default，

接口中的方法只能是public和default

## 2.接口

* **满足下面条件的C++类则称为接口**

1. 类中没有定义任何的成员变量
2. 所有的成员函数都是公有的
3. 所有的成员函数都是纯虚函数
4. 接口是一种特殊的抽象类

## 3.struct和class的区别

struct 能包含成员函数 能继承 能实现多态 和class的区别在于默认访问控制类型不同 struct是public的，class是private 就算把程序中的class全换成struct，程序依然能正常运行。 还有另一个很少涉及的区别是:class 可以用于定义模板参数，但是struct不行。

## 4. 什么是gcc / g++

首先说明：gcc 和 GCC 是两个不同的东西

GCC:GNU Compiler Collection(GUN 编译器集合)，它可以编译C、C++、JAV、Fortran、Pascal、Object-C、Ada等语言。

gcc是GCC中的GUN C Compiler（C 编译器）

g++是GCC中的GUN C++ Compiler（C++编译器）

一个有趣的事实就是，就本质而言，gcc和g++并不是编译器，也不是编译器的集合，它们只是一种驱动器，根据参数中要编译的文件的类型，调用对应的GUN编译器而已，比如，用gcc编译一个c文件的话，会有以下几个步骤：

Step1：Call a preprocessor, like cpp.

Step2：Call an actual compiler, like cc or cc1.

Step3：Call an assembler, like as.

Step4：Call a linker, like ld

由于编译器是可以更换的，所以gcc不仅仅可以编译C文件

所以，更准确的说法是：gcc调用了C compiler，而g++调用了C++ compiler

gcc和g++的主要区别

1. 对于 \*.c和\*.cpp文件，gcc分别当做c和cpp文件编译（c和cpp的语法强度是不一样的）

2. 对于 \*.c和\*.cpp文件，g++则统一当做cpp文件编译

3. 使用g++编译文件时，**g++会自动链接标准库STL，而gcc不会自动链接STL**

4. gcc在编译C文件时，可使用的预定义宏是比较少的

5. gcc在编译cpp文件时/g++在编译c文件和cpp文件时（这时候gcc和g++调用的都是cpp文件的编译器），会加入一些额外的宏，这些宏如下：

#define \_\_GXX\_WEAK\_\_ 1  
#define \_\_cplusplus 1  
#define \_\_DEPRECATED 1  
#define \_\_GNUG\_\_ 4  
#define \_\_EXCEPTIONS 1  
#define \_\_private\_extern\_\_ extern

6.        在用gcc编译c++文件时，为了能够使用STL，需要加参数 –lstdc++ ，但这并不代表 gcc –lstdc++ 和 g++等价，它们的区别不仅仅是这个主要参数

-g - turn on debugging (so GDB gives morefriendly output)

-Wall - turns on most warnings

-O or -O2 - turn on optimizations

-o - name of the output file

-c - output an object file (.o)

-I - specify an includedirectory

-L - specify a libdirectory

-l - link with librarylib.a

使用示例：g++ -ohelloworld -I/homes/me/randomplace/include helloworld.C

## 公有继承，私有继承，保护继承

1.公有继承时基类中各成员属性保持不变，基类中private成员被隐藏。派生类的成员只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象只能访问基类中的public成员。

2.私有继承时基类中各成员属性均变为private，并且基类中private成员被隐藏。派生类的成员也只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象不能访问基类中的任何的成员。

3.保护继承时基类中各成员属性均变为protected，并且基类中private成员被隐藏。派生类的成员只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象不能访问基类中的任何的成员。