1.

第一个数据成员放在offset为0的地方，对齐按照对齐系数和自身占用字节数中，二者比较小的那个进行对齐；在数据成员完成各自对齐以后，struct或者union本身也要进行对齐，对齐将按照对齐系数和struct或者union中最大数据成员长度中比较小的那个进行；

### 4. C语言编译过程中，volatile关键字和extern关键字分别在哪个阶段起作用？（填空题）

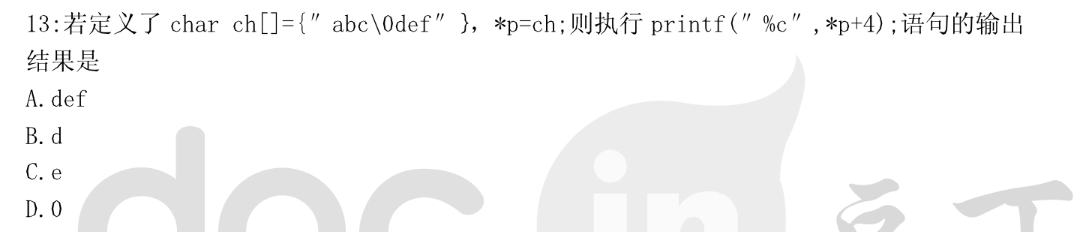
解答：volatile应该是在编译阶段，extern在链接阶段。**volatile关键字的作用是防止变量被编译器优化，而优化是处于编译阶段，所以volatile关键字是在编译阶段起作用。**

7. 指针在任何情况下都可进行>, <, >=, <=, = =运算

指针是地址就和一般数据一样，但地址不可以进行比较运算吧。所以应该是指针变量。

2.是在任何的情况下也应该不对，如果是类型不匹配呢。

8.



C

9. 逗号运算符确保操作数被顺序地处理：先计算左边的操作数，再计算右边的操作数。右操作数的类型和值作为整个表达式的结果。左操作数只是为了副作用需要而被计算，它其值会被丢弃。在计算完左操作数之后，存在一个序列点，例如：

y = ( x = 2.7, sqrt( 2\*x ));

在上述表达式中，在 sqrt（）函数被调用之前，赋值运算会首先发生。整个表达式的值是此函数的返回值。

10. 在c语言中，运算对象必须是整型数的运算符是：

①求余运算符%

②位运算符~、&、|、^、<<、>>。

# 11. (a++) += a

首先运算()里面的，a++，根据C语言的规则，就是先取出a的值再进行自加。  
打个比方a=12，  
分解开来就是(12)+=a,a+=1。  
这样就很明白了，12是常量，不能赋值

13. 写一个宏OFFSET，已知结构体中一个成员，求出该成员在结构体中的内存偏移量，#define OFFSET（type，member） ((size\_t)&(((type\*)0)->member))

14. 数组名和指针区别

**因为数组名只是一个符号，不是一个变量，因此不能作为一个左值，因此不能被修改**

指针是一个变量，有自己对应的存储空间，而数组名仅仅是一个符号，不是变量，因而没有自己对应的存储空间

15. 二叉树什么时候会退化

比如在二叉搜索树里面，如果一直从小到大插入元素，就退化成一个链表了

16. 不能用 sizeof（）函数，如何判断操作系统是16位，还是32位

  16位系统中，int变量的范围-32768到+32767,32767+1变为-32768。可以利用这个特性来判断。

19. C++如何实现多态？虚表指针是什么时候被初始化的？实例化一个对象需要那几个阶段？（）

构造函数中进行虚表指针的初始化，

#### 20. 内联函数与宏定义区别

（1）内联函数在编译时展开，宏在预编译时展开；

（2）内联函数直接嵌入到目标代码中，宏是简单的做文本替换；

（3）内联函数有类型检测、语法判断等功能，而宏没有；

（4）inline函数是函数，宏不是；

（5）宏定义时要注意书写（参数要括起来）否则容易出现歧义，内联函数不会产生歧义；

21. static变量存放在哪里？什么时候分配的内存

分配内存的时刻么，似乎没什么明确的约束，但总之能确定是在初始化前。一般实现会生成预留存储的代码，在程序开始的时候就分配内存，只是没有被这些变量占用。

静态局部变量初始化是在第一次调用函数的时候初始化

22. new和malloc的区别

1、属性

new/delete是C++关键字，需要编译器支持。malloc/free是库函数，需要头文件支持。

2、参数

使用new操作符申请内存分配时无须指定内存块的大小，编译器会根据类型信息自行计算。而malloc则需要显式地指出所需内存的尺寸。

3、返回类型

new操作符内存分配成功时，返回的是对象类型的指针，类型严格与对象匹配，无须进行类型转换，故new是符合类型安全性的操作符。而malloc内存分配成功则是返回void \* ，需要通过强制类型转换将void\*指针转换成我们需要的类型。

4、分配失败

new内存分配失败时，会抛出异常。malloc分配内存失败时返回NULL。

5、自定义类型

new会先调用new函数，申请足够的内存（通常底层使用malloc实现）。然后调用该类型的构造函数，初始化成员变量，最后返回自定义类型指针。delete先调用析构函数，然后调用delete函数释放内存（通常底层使用free实现）。

malloc/free是库函数，只能动态的申请和释放内存，无法强制要求其做自定义类型对象构造和析构工作。

6、重载

C++允许重载new/delete操作符，而malloc不允许重载。

7、内存区域

new操作符从自由存储区（free store）上为对象动态分配内存空间，而malloc函数从堆上动态分配内存。

23. 引用和指针的区别

1、指针是一个实体，需要分配内存空间。引用只是变量的别名，不需要分配内存空间。

2、引用在定义的时候必须进行初始化，并且不能够改变。指针在定义的时候不一定要初始化，并且指向的空间可变。（注：不能有引用的值不能为NULL）

3、有多级指针，但是没有多级引用，只能有一级引用。

4、指针和引用的自增运算结果不一样（指针是指向下一个空间，引用时引用的变量值加1

5、sizeof 引用得到的是所指向的变量（对象）的大小，而sizeof 指针得到的是指针本身的大小。

6、引用访问一个变量是直接访问，而指针访问一个变量是间接访问。

26. const常量和#define的区别（编译阶段、安全性、内存占用等）

用#define max 100 ; 定义的常量是没有类型的（不进行类型安全检查，可能会产生意想不到的错误），所给出的是一个立即数，编译器只是把所定义的常量值与所定义的常量的名字联系起来，define所定义的宏变量在预处理阶段的时候进行替换，在程序中使用到该常量的地方都要进行拷贝替换；

用const int max = 255 ; 定义的常量有类型（编译时会进行类型检查）名字，存放在内存的静态区域中，在编译时确定其值。在程序运行过程中const变量只有一个拷贝，而#define所定义的宏变量却有多个拷贝，所以宏定义在程序运行过程中所消耗的内存要比const变量的大得多

32.不用sizeof

#include <stdio.h>

#define MySizeof(Value) (char \*)(&value+1)-(char\*)&value

int main()

{

int i ;

double \*q;

printf("%d\r\n",MySizeof(i));

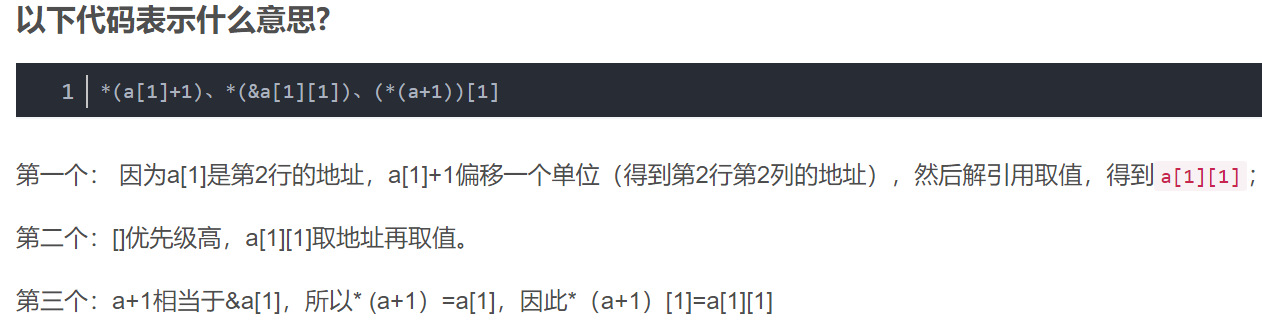
printf("%d\r\n",MySizeof(f));

printf("%d\r\n",MySizeof(a));

printf("%d\r\n",MySizeof(q));

return 0;

}

36.

38. class A

{};

class B :virtual public A{};

printf("A.sizeof[%d]B.sizeof[%d]",sizeof(A),sizeof(B));

A是空类，其大小为1；B不是空类，其大小为4.因为含有指向虚基类的指针。

39. 前序：1 2 3 4 5 6 7 8

中序：3 2 5 4 1 7 6 8

后序：？

40. int array[5]={7,8,3,2,6};

int \*p=(int\*)(array+1);

int \*m=(int\*)(&array+1);

printf("%d %d %d\n",\*(array+1),\*(p+1),\*(m+1));

8 ，3 ，-222225；

41. 介绍一下申请内存；  
2、追问内存溢出和内存泄漏（因为我把内存分区也说了一下）  
3、追问malloc、vmalloc、kmalloc区别（其实他还想继续追问内核态和应用态，可能时间不够）；

43. int i=3;  
int j;

j = sizeof(++i+ ++i);

printf(“i=%d j=%d”, i ,j);

sizeof运算符是以字节为单位返回操作数的大小,在编译时进行，不会对操作数进行运算,所以 i 的值是不变的就是3

46.  已知中序遍历的序列为abcdef，高度最小的不可能的二叉树的前序遍历是（C）

A、dbacfe B、cbaedf C、cabefd D、dbacef

47. 下列有关动态链接库说法错误的是（C）

A、动态链接库可以静态加载，也可以调用LoadLibrary函数动态加载；

B、使用GetModuleFileName，并传入该动态链接库的句柄，可以得到动态链接库文件的全路径；

C、动态链接库的入口点函数是DllMain，它只会在首次加载该动态链接库时执行一次;

D、同一个动态链接库在内存中只会存在一份；

解析：

1、The DllMain function is an optional method of entry into a dynamic-link library (DLL)。（简要翻译：对于动态链接库，DllMain是一个可选的入口函数。）这句话很重要，很多初学者可能都认为一个动态链接库肯定要有DllMain函数。其实不然，像很多仅仅包含资源信息的DLL是没有DllMain函数的。

2、系统是在什么时候调用DllMain函数的呢？静态链接时，或动态链接时调用LoadLibrary和FreeLibrary都会调用DllMain函数。DllMain的第二个参数fdwReason指明了系统调 Dll的原因，它可能是:：DLL\_PROCESS\_ATTACH、DLL\_PROCESS\_DETACHDLL\_THREAD\_ATTAC、

DLL\_THREAD\_DETACH。

以下从这四种情况来分析系统何时调用了DllMain。

3、DllMain函数是DLL模块的默认入口点。当Windows加载DLL模块时调用这一函数。系统首先调用全局对象的构造函数，然后调用全局函数DLLMain。DLLMain函数不仅在将DLL链接加载到进程时被调用，在DLL模块与进程分离时（以及其它时候）也被调用。

49. 编译程序不仅包含词法分析、语法分析、中间代码生成、目标代码生成，还包括（）  
正确答案: A B D

A 代码优化

B 表格管理

C 源代码优化

D 出错处理

50. **栈解旋**：当发生异常时，**从进入try块后，到异常被抛掷前**，这期间在栈上的构造的所有对象都会被自动析构。析构的顺序与构造的顺序相反，这一过程被称为栈的解旋。  
**A项多了堆对象。**

**51.** 由于寄存器变量储存在寄存器而非内存中，所以无法获取寄存器变量的地址。”

52. C++的空类有哪些成员函数

 缺省构造函数。

 缺省拷贝构造函数。

 缺省析构函数。

 缺省赋值运算符。

 缺省取址运算符。

 缺省取址运算符 const。

注意：有些书上只是简单的介绍了前四个函数。没有提及后面这两个函数。但后面这两个函数也是空类的默认函数。另外需要注意的是，只有当实际使用这些函数的时候，编译器才会去定义它们。

53. 流操作符重载为什么返回引用

在程序中，流操作符>>和<<经常连续使用。因此这两个操作符的返回值应该是一个仍旧支持这两个操作符的流引用。其他的数据类型都无法做到这一点。

注意：除了在赋值操作符和流操作符之外的其他的一些操作符中，如+、-、\*、/等却千万不能返回引用。因为这四个操作符的对象都是右值，因此，它们必须构造一个对象作为返回值。

54全局变量可不可以定义在可被多个.C 文件包含的   
头文件中？为什么？   
【标准答案】可以，在不同的C 文件中以static形式来声   
明同名全局变量。可以在不同的C文件中声明同名的全   
局变量，前提是其中只能有一个C文件中对此变量赋初   
值，此时连接不会出错。

57**一语句实现*x*是否为*2*的若干次幂的判断**

*int i = 512;*

*cout << boolalpha << ((i & (i - 1)) ? false : true) << endl;*

58*c*和*c++*中*struct*的主要区别是*c*中的*struct*不可以含有成员函数，而*c++*中的*struct*可以。

59 C++重载的时候可以使用函数返回值吗，C++调用一个函数是可以忽略其返回值的，这种情况下编译器就无法根据返回值类型来确定调用哪一个函数。

     所以，重载不能用返回值类型来区别。

60

map和unordered\_map的底层实现？目前map支持map[10] = 20这种操作，其底层怎么实现的？和insert有什么区别？

【】构造一个临时对象，调用赋值操作符赋值，析构，。Insert直接使用值构造一个。插入新值用insert，更新已有用【】

64 sizeof（“\0”）=2；

68 在main函数执行前执行的函数写法

alias: 设置函数别名。

aligned: 设置函数对齐方式。

always\_inline/gnu\_inline:

函数是否是内联函数。

constructor/destructor:

主函数执行之前、之后执行的函数。

format:

指定变参函数的格式输入字符串所在函数位置以及对应格式输出的位置。

noreturn:

指定这个函数没有返回值。

请注意，这里的没有返回值，并不是返回值是void。而是像\_exit/exit/abord那样

执行完函数之后进程就结束的函数。

weak：指定函数属性为弱属性，而不是全局属性，一旦全局函数名称和指定的函数名称

命名有冲突，使用全局函数名称。

void before() \_\_attribute\_\_((constructor));

void after() \_\_attribute\_\_((destructor));

void before() {

printf("this is function %s\n",\_\_func\_\_);

return;

}

void after(){

printf("this is function %s\n",\_\_func\_\_);

return;

}

70

class Base

{

public:

int m\_a, m\_b;

Base(int a = 3, int b = 5):m\_a(a), m\_b(b){}

int func\_a(){return m\_a - m\_b;}

virtual int func\_b(){return m\_a + m\_b;}

};

class Derived:public Base

{

public:

Derived(int a = 4, int b = 7):Base(a, b){}

virtual int func\_a(){return m\_b + m\_a;}

int func\_b(){return m\_b - m\_a;}

};

int main()

{

Base \*aa, \*bb;

aa = new Base(4, 7);

bb = new Derived(3, 5);

cout<<aa->func\_a()<<' '<<aa->func\_b()<<' '<<bb->func\_a()<<' '<<bb->func\_b()<<endl;

delete aa;delete bb;

return 0;

}

请问其运行结束时的打印信息是

答：-3 11 -2 2基类没有定义虚函数，用基类指针调用则调用基类的函数

72虚函数表中有没有名字？

一个对象必须包含一些有助于它在运行时选择正确函数的信息, 常见的做法是编译器将虚函数的名字转换成函数指针表中对应的索引值,这张表就是所谓虚函数表,每个含有虚函数的类都有它自己的虚函数表用于辨识虚函数.即使调用函数不清楚对象的大小和数据布局, 虚函数表中的函数也能确保对象被正确使用. 调用函数的实现只需要知道对象中虚函数表的指针的位置和每个虚函数的索引,就可以了, 这种虚函数机制的效率非常接近普通函数调用(相差不超过25%),空间开销包含两部分, 如果类中有虚函数,则每个对象需要一个额外指针,另外需要一个虚函数表.

73拷贝初始化和直接初始化，初始化和赋值的区别?

ClassTest ct1("ab"); 这条语句属于直接初始化，它不需要调用复制构造函数，直接调用构造函数ClassTest(const

char \*pc)，所以当复制构造函数变为私有时，它还是能直接执行的。

ClassTest ct2 = "ab"; 这条语句为复制初始化，它首先调用构造函数 ClassTest(const char\* pc) 函数创建一个临时对象，然后调用复制构造函数，把这个临时对象作为参数，构造对象ct2；所以当复制构造函数变为私有时，该语句不能编译通过。

ClassTest ct3 = ct1;这条语句为复制初始化，因为 ct1 本来已经存在，所以不需要调用相关的构造函数，而直接调用复制构造函数，把它值复制给对象 ct3；所以当复制构造函数变为私有时，该语句不能编译通过。

ClassTest ct4（ct1）;这条语句为直接初始化，因为 ct1 本来已经存在，直接调用复制构造函数，生成对象 ct3 的副本对象 ct4。所以当复制构造函数变为私有时，该语句不能编译通过。

要点就是拷贝初始化和直接初始化调用的构造函数是不一样的，但是当类进行复制时，类会自动生成一个临时的对象，然后再进行拷贝初始化。

75hash查询效率低下的原因，如何优化

在发生hash冲突时，一个bucket中会存储多个元素，这些hash冲突的元素以链表形式存储在一个bucket中：此时hash表性能会下降，根据hash算法确定bucket位置后，还要遍历链表，找到指定的元素。

load factor较小时，添加元素时很容易找到空的bucket，hash冲突少（因为可用的空bucket很多），存储性能较高；已装元素的bucket少，很容易从中找到指定的元素，查找性能较高；但遍历集合（hash表）时，要过滤掉大量的空bucket，很花时间，所以遍历时比较慢。

* load factor　　负载因子，等于size/capacity，即已装元素的bucket数占总bucket数的比例。0表示空的hash表，0.5表示半满的hash表。

当load factor达到设置的负载极限时，会发生rehashing（重哈希/再散列），hash表会自动成倍地增加容量（capacity），将原有的元素都移到新的hash表中（会重新分配存储位置），而此时原有的元素是极多的，这会增加很大的开销。

负载极限设置较高时，节省内存（空桶较少），但添加、查找元素效率较低，时间开销会增大；负载极限较低时，添加、查找元素效率较高，但会增加内存开销。默认为0.75，是时间、空间的折中，我们可根据需要自行设置。

77

 已知中序遍历的序列为abcdef，高度最小的不可能的[二叉树](https://so.csdn.net/so/search?q=%E4%BA%8C%E5%8F%89%E6%A0%91" \t "_blank)的前序遍历是（C）

A、dbacfe B、cbaedf C、cabefd D、dbacef

79

class A

{

virtual void g()

{

cout << "A::g" << endl;

}

private:

virtual void f()

{

cout << "A::f" << endl;

}

};

class B : public A

{

void g()

{

cout << "B::g" << endl;

}

virtual void h()

{

cout << "B::h" << endl;

}

};

typedef void (\*Fun)(void);

int main()

{

B b;

Fun pFun;

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

pFun = (Fun) \* ((int\*)\*(int\*)(&b) + i);

pFun();

}

}输出结果：  
B::g  
A::f  
B::h  
注意：本题主要考察了面试者对虚函数的理解程度。一个对虚函数不了解的人很难正确的做出本题。  
在学习面向对象的多态性时一定要深刻理解虚函数表的工作原理。

80

计算机内部如何存储负数和浮点数？

负数比较容易，就是通过一个标志位和补码来表示。

拓展问题：

什么是补码？

负数补码为反码加1

正数补码为原码

负数为什么用补码？

统一加减法，正负零问题

对于浮点类型的数据采用单精度类型（float）和双精度类型(double)来存储，float数据占用32bit,double数据占用64bit,我们在声明一个变量float f= 2.25f的时候，是如何分配内存的呢？如果胡乱分配，那世界岂不是乱套了么，其实不论是float还是double在存储方式上都是遵从IEEE的规范的，float遵从的是IEEE R32.24 ,而double 遵从的是R64.53。更多可以参考浮点数表示。

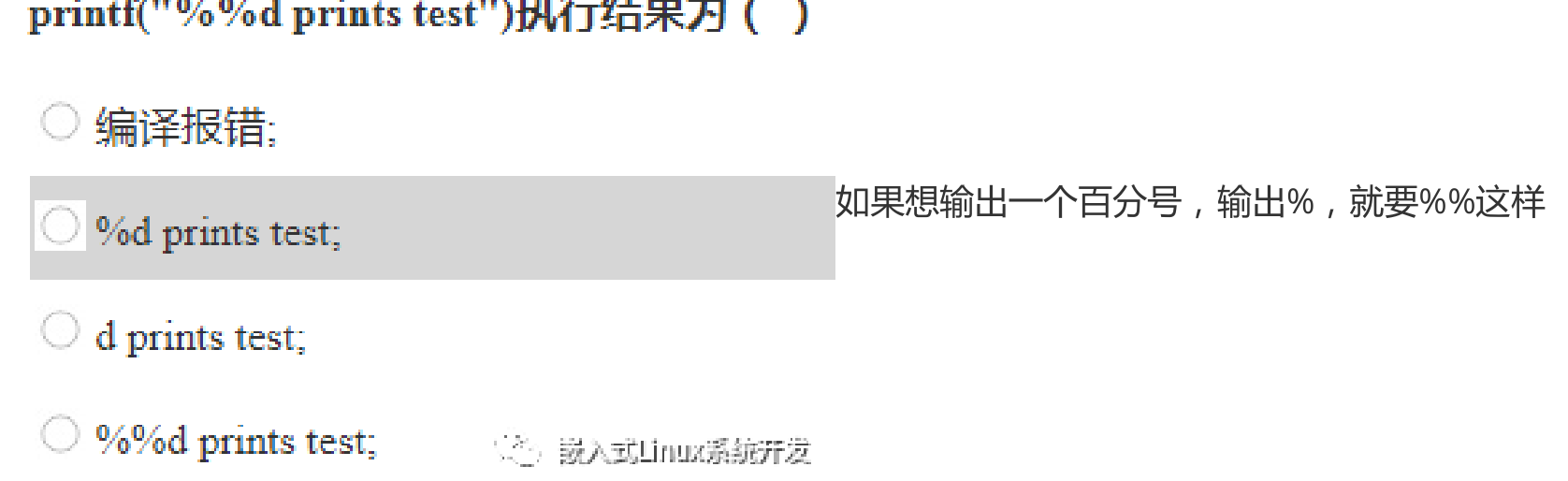
无论是单精度还是双精度在存储中都分为三个部分：

1). 符号位(Sign) : 0代表正，1代表为负

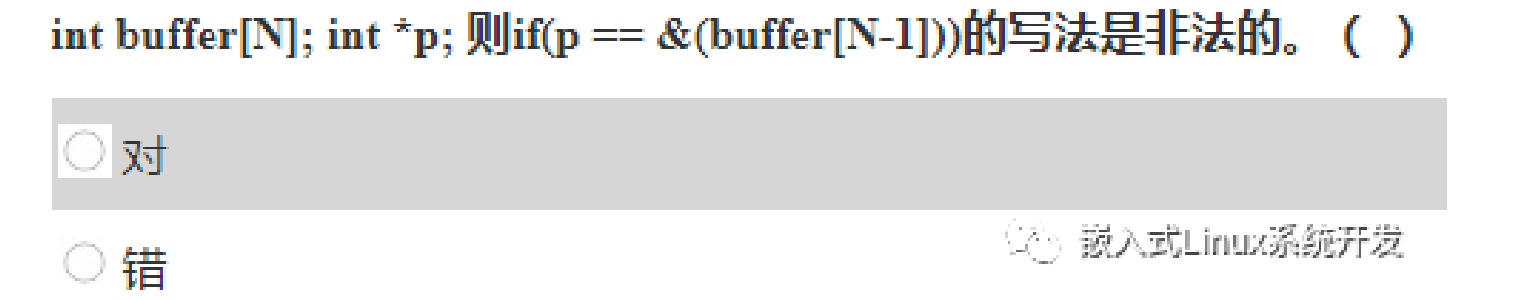
2). 指数位（Exponent）:用于存储科学计数法中的指数数据，并且采用移位存储

3). 尾数部分（Mantissa）：尾数部分

81



82



83 -Werror，它要求GCC将所有的警告当成错误进行处理

84，在数据安全的情况下大类型的数据向小类型的数据转换一定要显示的强制类型转换。

85提高c++性能，你用过哪些方式去提升（构造、析构、返回值优化、临时对象（使用operator=()消除临时对象）、内联（内联技巧、条件内联、递归内联、静态局部变量内联）、内存池、使用函数对象不使用函数指针、编码（编译器优化、预先计算）、设计（延迟计算、高效数据结构）、系统体系结构（寄存器、缓存、上下文切换））。

86对于浮点类型的数据采用单精度类型（float）和双精度类型(double)来存储，float数据占用32bit,double数据占用64bit,我们在声明一个变量float f= 2.25f的时候，是如何分配内存的呢？如果胡乱分配，那世界岂不是乱套了么，其实不论是float还是double在存储方式上都是遵从IEEE的规范的，float遵从的是IEEE R32.24 ,而double 遵从的是R64.53。更多可以参考浮点数表示。

无论是单精度还是双精度在存储中都分为三个部分：

1). 符号位(Sign) : 0代表正，1代表为负

2). 指数位（Exponent）:用于存储科学计数法中的指数数据，并且采用移位存储

3). 尾数部分（Mantissa）：尾数部分

。

Staticcast转换时执行必要的检查（指针越界，类型检查）其操作数相对是安全的（将int转double会补足比特位，），reinterpretcast,从比特模型进行转换（比较底层，移植性差）机器相关性。

char \*p="abcd";

printf("%s",p);

p[0]='a';

printf("%s",p);

回答：第三行代码错误，首先p是char类型的指针，不能这么给p进行赋值，要么将第一行改个一维数组，要么使用strcpy，memcpy等方法。