1、以下程序的输出是（12）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. class Base
2. {
3. public:
4. Base(**int** j) : i(j) {  }
5. virtual ~Base() {  }
6. void func1()
7. {
8. i \*= 10;
9. func2();
10. }
11. **int** getValue()
12. {
13. return i;
14. }
15. protected:
16. virtual void func2()
17. {
18. i++;
19. }
20. protected:
21. **int** i;
22. };
24. class Child : public Base
25. {
26. public:
27. Child(**int** j) : Base(j) {  }
28. void func1()
29. {
30. i \*= 100;
31. func2();
32. }
33. protected:
34. void func2()
35. {
36. i += 2;
37. }
38. };
40. **int** main(void)
41. {
42. Base \*pb = new Child(1);
43. pb->func1();
44. cout<<pb->getValue()<<endl;
45. delete pb;
47. return 0;
48. }

class Base

{

public:

Base(int j) : i(j) { }

virtual ~Base() { }

void func1()

{

i \*= 10;

func2();

}

int getValue()

{

return i;

}

protected:

virtual void func2()

{

i++;

}

protected:

int i;

};

class Child : public Base

{

public:

Child(int j) : Base(j) { }

void func1()

{

i \*= 100;

func2();

}

protected:

void func2()

{

i += 2;

}

};

int main(void)

{

Base \*pb = new Child(1);

pb->func1();

cout<<pb->getValue()<<endl;

delete pb;

return 0;

}

2、请问程序的输出结果是（30）  
#define DOUBLE(x) x+x // x\*2  
int i = DOUBLE(5)\*5;  
cout<<i<<endl;  
3、写出一下程序的输出（死循环）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. **int** main(void)
2. {
3. **char** num;
4. for(num = 0; num < 255; )
5. num += num;
6. printf("%d\n",num);
7. return 0;
8. }

int main(void)

{

char num;

for(num = 0; num < 255; )

num += num;

printf("%d\n",num);

return 0;

}

4、程序出错在什么阶段？（）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. **int** main(void)
2. {
3. http://www.sogou.com
4. cout<<"welcome to sogou"<<endl;
6. return 0;
7. }

int main(void)

{

http://www.sogou.com

cout<<"welcome to sogou"<<endl;

return 0;

}

A、编译阶段出错     B、运行阶段出错     C、编译和运行都出错    D、程序运行正常  
因为http://www.sogou.com中//后面是注释，前面的http:是标签（类似于goto语句中的标签）。（这个题目碉堡了）  
  
5、下面程序执行结果为【说明：X86\_64环境】（D）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. **int** main(void)
2. {
3. **int** a[4][4] = {
4. {1,2,3,4},
5. {50,60,70,80},
6. {900,1000,1100,1200},
7. {13000,14000,15000,16000} };
8. **int** (\*p1)[4] = a;
9. **int** (\*p2)[4] = &a[0];
10. **int** \*p3 = &a[0][0];
11. printf("%d %d %d %d\n",\*(\*(a+1)-1),\*(\*(p1+3)-2)+1,\*(\*(p2-1)+16)+2,\*(p3+sizeof(p1)-3));
13. return 0;
14. }

int main(void)

{

int a[4][4] = {

{1,2,3,4},

{50,60,70,80},

{900,1000,1100,1200},

{13000,14000,15000,16000} };

int (\*p1)[4] = a;

int (\*p2)[4] = &a[0];

int \*p3 = &a[0][0];

printf("%d %d %d %d\n",\*(\*(a+1)-1),\*(\*(p1+3)-2)+1,\*(\*(p2-1)+16)+2,\*(p3+sizeof(p1)-3));

return 0;

}

A、16000 1101  13002 2  
B、4  2  3  60  
C、16000  2  3  2  
D、4  1101  13002  60  
p1为指向一维数组的指针，所以a + 1指向{50，60,70,80}这一维的地址。减一则为4的地址；同理第二个输出1101。同理，由于数组的列是4，所以\*(p2 - 1) + 16就相当于\*(p2) + 12，所以第三个输出13002。  
第四个由于p1是指针，所以sizeof(p1)为8（68位的系统），所以第四个输出60。  
  
6、在32位操作系统gcc编译器环境下，下面的程序的运行结果是（A）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. class A
2. {
3. public:
4. **int** b;
5. **char** c;
6. virtual void print()
7. {
8. cout<<"this is father's function!"<<endl;
9. }
10. };
12. class B : A
13. {
14. public:
15. virtual void print()
16. {
17. cout<<"this is children's function!"<<endl;
18. }
19. };
20. **int** main(void)
21. {
22. cout<<sizeof(A)<<"  "<<sizeof(A)<<endl;
24. return 0;
25. }

class A

{

public:

int b;

char c;

virtual void print()

{

cout<<"this is father's function!"<<endl;

}

};

class B : A

{

public:

virtual void print()

{

cout<<"this is children's function!"<<endl;

}

};

int main(void)

{

cout<<sizeof(A)<<" "<<sizeof(A)<<endl;

return 0;

}

A、12  12  
B、8  8  
C、9  9  
D、12  16  
7、以下哪些做法是不正确或者应该极力避免的：【多选】（ACD）  
A、构造函数声明为虚函数  
B、派生关系中的基类析构函数声明为虚函数  
C、构造函数调用虚函数  
D、析构函数调用虚函数  
8、关于C++标准模板库，下列说法错误的有哪些：【多选】（AD）  
A、std::auto\_ptr<Class A>类型的对象，可以放到std::vector<std::auto\_ptr<Class A>>容器中  
B、std::shared\_ptr<Class A>类型的对象，可以放到std::vector<std::shared\_ptr<Class A>>容器中  
C、对于复杂类型T的对象tObj，++tObj和tObj++的执行效率相比，前者更高  
D、采用new操作符创建对象时，如果没有足够内存空间而导致创建失败，则new操作符会返回NULL  
A中auto是给别人东西而自己没有了。所以不符合vector的要求。而B可以。C不解释。new在失败后抛出标准异常std::bad\_alloc而不是返回NULL。  
  
9、有如下几个类和函数定义，选项中描述正确的是：【多选】（B）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. class A
2. {
3. public:
4. virtual void foo()   {  }
5. };
7. class B
8. {
9. public:
10. virtual void foo()   {  }
11. };
13. class C : public A , public B
14. {
15. public:
16. virtual void foo()   {  }
17. };
19. void bar1(A \*pa)
20. {
21. B \*pc = dynamic\_cast<B\*>(pa);
22. }
24. void bar2(A \*pa)
25. {
26. B \*pc = static\_cast<B\*>(pa);
27. }
29. void bar3()
30. {
31. C c;
32. A \*pa = &c;
33. B \*pb = static\_cast<B\*>(static\_cast<C\*>(pa));
34. }

class A

{

public:

virtual void foo() { }

};

class B

{

public:

virtual void foo() { }

};

class C : public A , public B

{

public:

virtual void foo() { }

};

void bar1(A \*pa)

{

B \*pc = dynamic\_cast<B\*>(pa);

}

void bar2(A \*pa)

{

B \*pc = static\_cast<B\*>(pa);

}

void bar3()

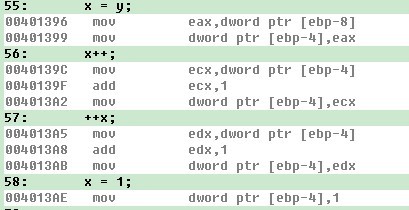
{

C c;

A \*pa = &c;

B \*pb = static\_cast<B\*>(static\_cast<C\*>(pa));

}

A、bar1无法通过编译  
B、bar2无法通过编译  
C、bar3无法通过编译  
D、bar1可以正常运行，但是采用了错误的cast方法  
选B。dynamic\_cast是在运行时遍历继承树，所以，在编译时不会报错。但是因为A和B没啥关系，所以运行时报错(所以A和D都是错误的)。static\_cast：编译器隐式执行的任何类型转换都可由它显示完成。其中对于：（1）基本类型。如可以将int转换为double(编译器会执行隐式转换)，但是不能将int\*用它转换到double\*（没有此隐式转换）。（2）对于用户自定义类型，如果两个类无关，则会出错（所以B正确），如果存在继承关系，则可以在基类和派生类之间进行任何转型，在编译期间不会出错。所以bar3可以通过编译（C选项是错误的)。  
  
10、在Intel CPU上，以下多线程对int型变量x的操作，哪几个不是原子操作，假定变量的地址都是对齐的。【多选】（ABC）  
A、x = y   B、x++     C、++x     D、x = 1  
看下在VC++6.0下的汇编命令即可：从图可以看出本题只有D选项才是原子操作。  
  
11、一般情况下，下面哪些操作会执行失败？【多选】（BCD）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. class A
2. {
3. public:
4. string a;
5. void f1()
6. {
7. printf("Hello World");
8. }
9. void f2()
10. {
11. a = "Hello World";
12. printf("%s",a.c\_str());
13. }
14. virtual void f3()
15. {
16. printf("Hello World");
17. }
18. virtual void f4()
19. {
20. a = "Hello World";
21. printf("%s",a.c\_str());
22. }
23. };

class A

{

public:

string a;

void f1()

{

printf("Hello World");

}

void f2()

{

a = "Hello World";

printf("%s",a.c\_str());

}

virtual void f3()

{

printf("Hello World");

}

virtual void f4()

{

a = "Hello World";

printf("%s",a.c\_str());

}

};

A、A \*aptr = NULL;  aptr->f1();  
B、A \*aptr = NULL;  aptr->f2();  
C、A \*aptr = NULL;  aptr->f3();  
D、A \*aptr = NULL;  aptr->f4();  
至于A为什么正确，因为A没有使用任何成员变量，而成员函数是不属于对象的，所以A正确。其实，A\* aptr = NULL;aptr->f5();也是正确的，因为静态成员也是不属于任何对象的。至于BCD，在B中使用了成员变量，而成员变量只能存在于对象，C有虚表指针，所以也只存在于对象中。D就更是一样了。但是，如果在Class A中没有写public，那么就全都是private，以至于所有的选项都将会失败。  
  
12、C++下，下面哪些template实例化使用，会引起编译错误？【多选】（CEF）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. template<class Type> class stack;
2. void fi(stack<**char**>);      //A
3. class Ex
4. {
5. stack<**double**> &rs;    //B
6. stack<**int**> si;        //C
7. };
9. **int** main(void)
10. {
11. stack<**char**> \*sc;      //D
12. fi(\*sc);              //E
13. **int** i = sizeof(stack<string>);    //F
15. return 0;
16. }

template<class Type> class stack;

void fi(stack<char>); //A

class Ex

{

stack<double> &rs; //B

stack<int> si; //C

};

int main(void)

{

stack<char> \*sc; //D

fi(\*sc); //E

int i = sizeof(stack<string>); //F

return 0;

}

选C E F;  请注意stack和fi都只是声明不是定义。我还以为在此处申明后，会在其他地方定义呢，坑爹啊。  
由于stack只是声明，所以C是错误的，stack不能定义对象。E也是一样，stack只是申明，所以不能执行拷贝构造函数，至于F，由于stack只是声明，不知道stack的大小，所以错误。如果stack定义了，将全是正确的。  
13、以下哪个说法正确（）

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. **int** func()
2. {
3. **char** b[2]={0};
4. strcpy(b,"aaa");
5. }

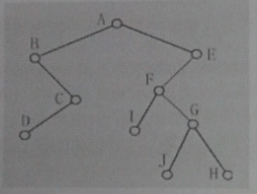
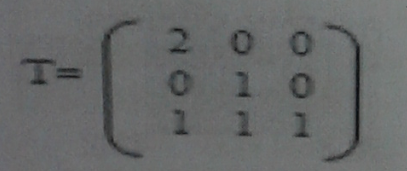
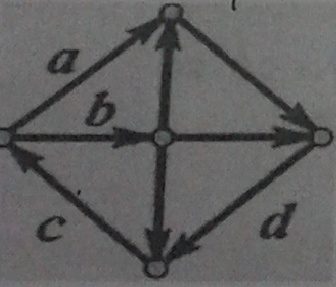
int func()

{

char b[2]={0};

strcpy(b,"aaa");

}

A、Debug版崩溃，Release版正常  
B、Debug版正常，Release版崩溃  
C、Debug版崩溃，Release版崩溃  
D、Debug版正常，Release版正常  
选A。因为在Debug中有ASSERT断言保护，所以要崩溃，而在Release中就会删掉ASSERT,所以会出现正常运行。但是不推荐如此做，因为这样会覆盖不属于自己的内存，这是搭上了程序崩溃的列车。  
  
数据结构类  
37、每份考卷都有一个8位二进制序列号，当且仅当一个序列号含有偶数个1时，它才是有效的。例如：00000000 01010011 都是有效的序列号，而11111110不是，那么有效的序列号共有（128）个。  
38、对初始状态为递增序列的数组按递增顺序排序，最省时间的是插入排序算法，最费时间的算法（B）  
A、堆排序  B、快速排序   C、插入排序   D、归并排序  
39、下图为一个二叉树，请选出以下不是遍历二叉树产生的顺序序列的选项【多选】（BD）  
  
A、ABCDEFIGJH  
B、BDCAIJGHFE  
C、BDCAIFJGHE  
D、DCBJHGIFEA  
40、在有序双向链表中定位删除一个元素的平均时间复杂度为（）  
A、O（1）   B、O（N）   C、O(logN)      D、O(N\*logN)   
41、将10阶对称矩阵压缩存储到一维数组A中，则数组A的长度最少为（）  
A、100     B、40     C、55      D、80  
42、将数组a[]作为循环队列SQ的存储空间，f为队头指示，r为队尾指示，则执行出队操作的语句为（B）  
A、f = f+1     B、f = (f+1)%m      C、r = (r+1)%m     D、f = (f+1)%(m+1)  
43、以下哪种操作最适合先进行排序处理？  
A、找最大、最小值   B、计算算出平均值   C、找中间值  D、找出现次数最多的值  
44、设有一个二维数组A[m][n]，假设A[0][0]存放位置在644（10），A[2][2]存放位置在676（10），每个元元素占一个空间，问A[3][3]存放在什么位置？（C）脚注（10）表示用10进制表示  
A、688     B、678     C、692     D、696  
45、使用下列二维图形变换矩阵A=T\*a，将产生的变换结果为（D）  
  
A、图形放大2倍  
B、图形放大2倍，同时沿X、Y坐标轴方向各移动一个单位  
C、沿X坐标轴方向各移动2个单位  
D、沿X坐标轴放大2倍，同时沿X、Y坐标轴方向各移动一个单位  
46、体育课的铃声响了，同学们都陆续地奔向操场，按老师的要求从高到矮站成一排。每个同学按顺序来到操场时，都从排尾走向排头，找到第一个比自己高的同学，并站到他的后面，这种站队的方法类似于（）算法。  
A、快速排序      B、插入排序      C、冒泡排序    D、归并排序  
47、处理a.html文件时，以下哪行伪代码可能导致内存越界或者抛出异常（B）  
         int totalBlank = 0;  
         int blankNum = 0;  
         int taglen = page.taglst.size();  
A       for(int i = 1; i < taglen-1; ++i)  
        {  
                 //check blank  
B             while(page.taglst[i] == "<br>" && i < taglen)  
               {  
C                       ++totalBlank;  
D                       ++i;  
               }  
E             if(totalBlank > 10)  
F                      blankNum += totalBlank;  
G             totalBlank = 0;  
        }  
注意：以下代码中taglen是html文件中存在元素的个数，a.html中taglen的值是15，page.taglst[i]取的是a.html中的元素，例如page.taglst[1]的值是<html>  
a.html的文件如下：  
<html>  
< title>test</title>  
< body>  
< div>aaaaaaa</div>  
< /body>  
< /html>  
< br>  
<br>  
< br>  
< br>  
< br>  
48、对一个有向图而言，如果每个节点都存在到达其他任何节点的路径，那么就称它是强连通的。例如，右图就是一个强连通图，事实上，在删掉哪几条边后，它依然是强连通的。（A）  
  
A、a       B、b        C、c             D、d  
  
100、一种计算机，其有如下原子功能：  
1、赋值   a=b  
2、+1操作，++a; a+1;  
3、循环，但是只支持按次数的循环   for（变量名）{/\*循环里面对变量的修改不影响循环次数\*/}  
4、只能处理0和正整数  
5、函数调用    fun(参数列表)  
请用伪代码的形式分别在这个计算机上编程实现变量的加法、减法、乘法。  
  
fun\_add(a , b)  
{  
}  
fun\_multi(a , b)  
{  
}  
fun\_minus(a , b)  
{  
}  
问题的关键在于如何实现自减一操作。  
本来让-1自增n次即可实现n的自减的，但系统偏偏又不支持负数。

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. fun\_add(a , b)
2. {
3. result = a;
4. for(b)
5. ++result;
6. return result;
7. }
9. fun\_muti(a , b)
10. {
11. result = 0;
12. for(b)
13. result = fun\_add(result , a);
14. return result;
15. }
17. dec(**int** n)
18. {
19. temp = 0;
20. result = 0;
21. for(n)
22. {
23. result = temp;   //result永远比temp少1,巧妙地减少了一次自增
24. ++temp;
25. }
26. return result;
27. }
28. /\*
29. 上面的dec这段函数代码执行后，result的值将变为n-1。注意到这段代码在自增时是如何巧妙地延迟了一步的。
30. 现在，我们相当于有了自减一的函数dec。实现a-b只需要令a自减b次即可
31. \*/
32. fun\_minus(a , b)
33. {
34. result = a;
35. for(b)
36. result = dec(result);
37. }

fun\_add(a , b)

{

result = a;

for(b)

++result;

return result;

}

fun\_muti(a , b)

{

result = 0;

for(b)

result = fun\_add(result , a);

return result;

}

dec(int n)

{

temp = 0;

result = 0;

for(n)

{

result = temp; //result永远比temp少1,巧妙地减少了一次自增

++temp;

}

return result;

}

/\*

上面的dec这段函数代码执行后，result的值将变为n-1。注意到这段代码在自增时是如何巧妙地延迟了一步的。

现在，我们相当于有了自减一的函数dec。实现a-b只需要令a自减b次即可

\*/

fun\_minus(a , b)

{

result = a;

for(b)

result = dec(result);

}

101、实现一个队链表排序的算法，C/C++可以使用std::list<int>，Java使用LinkedList<Integer>  
要求先描述算法，然后再实现，算法效率尽可能高效。  
主要考察链表的归并排序。  
要点：需要使用快、慢指针的方法，找到链表的的中间节点，然后进行二路归并排序

**[cpp]** [view plaincopyprint?](http://blog.csdn.net/hackbuteer1/article/details/8016173)

1. typedef struct LNode
2. {
3. **int** data;
4. struct LNode \*next;
5. }LNode , \*LinkList;
7. // 对两个有序的链表进行递归的归并
8. LinkList MergeList\_recursive(LinkList head1 , LinkList head2)
9. {
10. LinkList result;
11. if(head1 == NULL)
12. return head2;
13. if(head2 == NULL)
14. return head1;
15. if(head1->data < head2->data)
16. {
17. result = head1;
18. result->next = MergeList\_recursive(head1->next , head2);
19. }
20. else
21. {
22. result = head2;
23. result->next = MergeList\_recursive(head1 , head2->next);
24. }
25. return result;
26. }
28. // 对两个有序的链表进行非递归的归并
29. LinkList MergeList(LinkList head1 , LinkList head2)
30. {
31. LinkList head , result = NULL;
32. if(head1 == NULL)
33. return head2;
34. if(head2 == NULL)
35. return head1;
36. while(head1 && head2)
37. {
38. if(head1->data < head2->data)
39. {
40. if(result == NULL)
41. {
42. head = result = head1;
43. head1 = head1->next;
44. }
45. else
46. {
47. result->next = head1;
48. result = head1;
49. head1 = head1->next;
50. }
51. }
52. else
53. {
54. if(result == NULL)
55. {
56. head = result = head2;
57. head2 = head2->next;
58. }
59. else
60. {
61. result->next = head2;
62. result = head2;
63. head2 = head2->next;
64. }
65. }
66. }
67. if(head1)
68. result->next = head1;
69. if(head2)
70. result->next = head2;
71. return head;
72. }
74. // 归并排序，参数为要排序的链表的头结点，函数返回值为排序后的链表的头结点
75. LinkList MergeSort(LinkList head)
76. {
77. if(head == NULL)
78. return NULL;
79. LinkList r\_head , slow , fast;
80. r\_head = slow = fast = head;
82. // 找链表中间节点的两种方法
83. /\*
84. while(fast->next != NULL)
85. {
86. if(fast->next->next != NULL)
87. {
88. slow = slow->next;
89. fast = fast->next->next;
90. }
91. else
92. fast = fast->next;
93. }\*/
95. while(fast->next != NULL && fast->next->next != NULL)
96. {
97. slow = slow->next;
98. fast = fast->next->next;
99. }
101. if(slow->next == NULL)    // 链表中只有一个节点
102. return r\_head;
103. fast = slow->next;
104. slow->next = NULL;
105. slow = head;
107. // 函数MergeList是对两个有序链表进行归并，返回值是归并后的链表的头结点
108. //r\_head = MergeList\_recursive(MergeSort(slow) , MergeSort(fast));
109. r\_head = MergeList(MergeSort(slow) , MergeSort(fast));
110. return r\_head;
111. }

typedef struct LNode

{

int data;

struct LNode \*next;

}LNode , \*LinkList;

// 对两个有序的链表进行递归的归并

LinkList MergeList\_recursive(LinkList head1 , LinkList head2)

{

LinkList result;

if(head1 == NULL)

return head2;

if(head2 == NULL)

return head1;

if(head1->data < head2->data)

{

result = head1;

result->next = MergeList\_recursive(head1->next , head2);

}

else

{

result = head2;

result->next = MergeList\_recursive(head1 , head2->next);

}

return result;

}

// 对两个有序的链表进行非递归的归并

LinkList MergeList(LinkList head1 , LinkList head2)

{

LinkList head , result = NULL;

if(head1 == NULL)

return head2;

if(head2 == NULL)

return head1;

while(head1 && head2)

{

if(head1->data < head2->data)

{

if(result == NULL)

{

head = result = head1;

head1 = head1->next;

}

else

{

result->next = head1;

result = head1;

head1 = head1->next;

}

}

else

{

if(result == NULL)

{

head = result = head2;

head2 = head2->next;

}

else

{

result->next = head2;

result = head2;

head2 = head2->next;

}

}

}

if(head1)

result->next = head1;

if(head2)

result->next = head2;

return head;

}

// 归并排序，参数为要排序的链表的头结点，函数返回值为排序后的链表的头结点

LinkList MergeSort(LinkList head)

{

if(head == NULL)

return NULL;

LinkList r\_head , slow , fast;

r\_head = slow = fast = head;

// 找链表中间节点的两种方法

/\*

while(fast->next != NULL)

{

if(fast->next->next != NULL)

{

slow = slow->next;

fast = fast->next->next;

}

else

fast = fast->next;

}\*/

while(fast->next != NULL && fast->next->next != NULL)

{

slow = slow->next;

fast = fast->next->next;

}

if(slow->next == NULL) // 链表中只有一个节点

return r\_head;

fast = slow->next;

slow->next = NULL;

slow = head;

// 函数MergeList是对两个有序链表进行归并，返回值是归并后的链表的头结点

//r\_head = MergeList\_recursive(MergeSort(slow) , MergeSort(fast));

r\_head = MergeList(MergeSort(slow) , MergeSort(fast));

return r\_head;

}