1、 这段程序的输出是（B、C、D）

A） hello, what a beautiful world B） 没有输出 C） 程序可能会提前终止 D） 不一定

```
#include <memory>
#include <iostream>
#include <vector>

int main()
{
    auto and y = std::move(std::string("hello, what a beautiful world"));
    std::cout << y << std::endl;
}
```

2、这段代码有什么问题？（B)

A）push 中的 T&& 不是转发引用，所以编译器会报错
B）push 中的 T&& 不是转发引用，所以只能支持对 ravlue 操作
C）push 可能会在 vector 中加入一个悬空引用 dangling reference
D）push 的效率比较低，在可以移动的时候可能退化为拷贝

```
template<typename T>
class Stack {
   std::vector<T> vec;
public:
   void push(T&& t) {
      vec.push_back(std::forward<T>(t));
   }
   // ...
};

    int data;
    std::vector<std::unique_ptr<TreeNode>> children;
};
```

3、下面移动构造和移动赋值的实现（B）

A）有内存泄漏
B）移动构造可以，移动赋值有问题
C）copy-swap 的一种做法，没有问题
D）copy-swap 的一种做法，效率略差

```
template <class T>
class clone_ptr
{
private:
    T* ptr;
public:
    // ...
    clone_ptr(clone_ptr&& p)            // 移动构造函数
        : ptr(p.ptr)    // 拷贝数据的表示
```

```
    {
        p.ptr = 0;      // 把源数据的表示置空
    }
    clone_ptr& operator=(clone_ptr&& p) // 移动赋值
    {
        std::swap(ptr, p.ptr);
        return *this;   // 销毁目标的旧值
    }
};
```

Bjarne
godbolt

4、请指出下列代码的主要错误：（ C ）

A）数据没有初始化　　B）形成递归定义　　C）存在内存泄漏问题

```
struct Child;

struct Parent
{
    int parentData;
    std::shared_ptr<Child> m_child;
};

struct Child
{
    int childData;
    std::shared_ptr<Parent> m_parent;
};

int main()
{
    std::shared_ptr<Parent> parent = std::make_shared<Parent>();
    std::shared_ptr<Child> child = std::make_shared<Child>();
    parent->m_child = child;
    child->m_parent = parent;
}
```

5、已知 Foo 类的实现如下，则下列用法可能存在问题的是：(D)

```
class Foo
{
public:
    std::shared_ptr<Foo> ShareMe() { return std::shared_ptr<Foo>(this); }
};
```

A)

```
std::shared_ptr<Foo> sp1 = std::make_shared<Foo>();
std::shared_ptr<Foo> sp2 = sp1->ShareMe();
```

B)

```
Foo *p1 = new Foo;
std::shared_ptr<Foo> sp2 = p1->ShareMe();
delete p1;
```

C)

```
Foo foo;
std::shared_ptr<Foo> sp = foo.ShareMe();
```

D)　以上全部都存在问题

6、这段程序的打印输出应该是什么？（D）

A）1122
B）1212
C）12
D) 可能会崩溃

```
#include <memory>
#include <iostream>

class A
{
public:
    A() {std::cout <<"1";}
    ~A() {std::cout <<"2";}
    void DoSomething(){};
};

int main (void)
{
   A a;
   A* pa = &a;
   std::shared_ptr<A> spa1 { pa };
   std::shared_ptr<A> spa2 = spa1;
   spa2->DoSomething();
}
```

godbol

7、推荐使用 make_shared() 函数构造 shared_ptr 智能指针的原因是：（　　B C　　）【多选】

A) 不需要使用 new 创建对象，节省内存　　B）合并为一次内存分配动作，更高效

C）使用原位构造，减少开销　　D）不产生异常

8、下列用法正确的是：(　　D　　)

A)

```
void SomeFunc(Widget* w)
{    ...   }

std::shared_ptr<Widget> sp = std::make_shared<Widget>();
Widget* tmp = sp.get();
SomeFunc(tmp);
delete tmp;
```

B)

```
void SomeFunc(Widget* w)
{    ...   }

std::shared_ptr<Widget> sp = std::make_shared<Widget>();
Widget* tmp = sp.get();
SomeFunc(tmp);
std::shared_ptr<Widget> sp2(tmp);
```

C)

```
std::unique_ptr<Widget> up(new Widget);
std::shared_ptr sp(up.get());
```

D)

```
std::unique_ptr<Widget> up(new Widget);
std::shared_ptr sp(std::move(up));
```

9、这段程序的打印输出应该是什么？（A）

A）1:1,42:1,1:1,45:1

B）1:1,1:1,4:1,4:1

C）1:2,42:1,1:2,45:1

D）有未定义行为

```
#include <iostream>
#include <memory>

using std::cout;
using std::endl;

void Func1(std::shared_ptr<int>& shr) { shr.reset(new int(1)); }

void Func2(std::shared_ptr<int>& shr) { *shr += 3; }

int main() {

  auto sp1 = std::make_shared<int>(42);
  auto sp2 = sp1;

  Func1(sp1);
  cout  << *sp1 << ":" << sp1.use_count() << ",";
  cout <<  *sp2 << ":" << sp2.use_count() << ",";
```

```
  Func2(sp2);
  cout  << *sp1 << ":" << sp1.use_count() << ",";
  cout <<  *sp2 << ":" << sp2.use_count() << endl;
}
```

godbolt

10、以下代码的打印输出是（B，D）
A）123
B）1
C）13
D）会崩溃

```
#include <memory>
#include <iostream>

using namespace std;

struct S : enable_shared_from_this<S>
{
  shared_ptr<S> GetThis()
  {
    return shared_from_this();
  }
  S(){ std::cout << "1" << std::endl;}
  ~S(){std::cout << "2" << std::endl;}
  void DoSomething() {std::cout << "3" << std::endl;}
};

int main()
{
   S *p = new S;
   shared_ptr<S> sp2 = p->GetThis();
   sp2->DoSomething();
}
```

godbolt

11、以下程序的打印输出为？（C、D）

A） 13212　B） 12312　C） 1312　　D）程序有问题

```
#include <memory>
#include <iostream>

class A{
public:
   A(){std::cout << "1";}
   ~A(){std::cout << "2";}
   void doSth(void) const { std::cout << "3"; }
};

void *operator new(size_t n, A&& a)
```

```
{
    char* p = static_cast<char *>(::operator new(n));
    a.doSth();
    return p;
}

int main()
{
    A* p0 = new A{};
    A* p1 = new (std::move(*p0))A{};
    delete p1;
}
```

godbolt

12、以下哪些 new 能成功通过编译？（D）
A） 1　B） 12　C） 13　　D）3

```
#include <new>
#include <array>

struct Mem
{
    void* GetMemory() const { return nullptr; }
};

class Base {
public:
 static void* operator new(std::size_t, const Mem& p)
 {
     return p.GetMemory();
 }
};

class Derived : public Base {
};

int main()
{
    Derived d;

    //Derived* p1 = new Derived();                 // 1

    //void* p2 = &d;
    //new (p2) Derived;                            // 2

    Derived* p4 = new (Mem()) Derived;              // 3
}
```

godbolt