1、以下程序的打印输出是（）

A） 3B12　　B） 3B12B　　C）31B2B　　D) 31B2

```
class A
{
    int m_a = 42;
public:
    A() {std::cout << "1"; }
    ~A() { std::cout << "2"; }
};
class B
{
    int m_b = 36;
public:
   B() {std::cout << "3";}
   ~B() { std::cout << "4"; }
   void DoB() { std::cout << "B"; }
};

void *operator new(size_t n, B* buf) { buf->DoB(); return buf; }
void operator delete(void* p, B* buf) { buf->DoB(); }

int main()
{
    B* pb = new B{};
    A* pa = new(pb)A{};
    delete pa;
}
```

godbolt

2、下列模板特化版本语法不正确的是：（　　）

```
template<typename U, typename V>
struct Stra {
};
```

A)

```
template<typename U>
struct Stra<U, char> {
};
```

B)

```
template<typename V>
struct Stra<char, V> {
};
```

C)

```
template<typename U, typename V>
struct Stra<int, double>  {
};
```

godbolt

3、以下代码的期望输出是（　　）

A） 6.3,6.3,6.3　　B）6.3, 6, 6　C）6.3,7,7　　D) 编译不通过

```
void Fun(auto t, auto UVal)
{
    std::cout << std::setprecision(2) << t * UVal << ",";
}

template<auto UVal>
void Fun1(auto t)
{
    std::cout << std::setprecision(2) << t * UVal << ",";
}

int main()
{
    constexpr float a = 3.14f;
    Fun(2, 3.14);
    Fun1<3.14f>(2);
    Fun1<a>(2);
}
```

godbolt

4、以下代码的期望输出是（　　）

A）输出：Test2::Func(int, char) invoked!
B）输出：Test2::Func(U, T) invoked!
C）编译通不过
D）程序会崩溃

```
template <class U, class T>
struct Test2  {
    void Func(U para1, T para2)  {
        std::cout << "Test2::Func(U, T) invoked!" << std::endl;
    }
};

template <>
void Test2<int, char>::Func(int para1, char para2)  {
    std::cout << "Test2::Func(int, char) invoked!" << std::endl;
}

template <class U>
void Test2<U, int>::Func(U para1, int para2)  {
    std::cout << "Test2::Func(U, int) invoked!" << std::endl;
}
```

```
int main()
{
    Test2<int, char> t;
    t.Func(5, 2);
}
```

godbolt

5、已知函数模板 func 的定义，则根据下面函数调用，编译器推导模板参数 T 的类型是：（ ）

A） char　　B）const char[3]　　C）const char*　　D) const char

```
template<typename T>
void func(T t);

const char name[] = "Tom";
func(name);
```

godbolt

6、在 64 位平台上，以下程序的打印输出是：（　　）

A） 88　　B）44　　C) 35　　D) 46

```
#include <iostream>
#include <typeinfo>
#include <string>
#include <iostream>

template<typename T>
void func(T& a, T& b)
{
     std::cout << sizeof(a) << sizeof(b) << std::endl;
}

int main()
{
    func("Tom","Jerry");
}
```

godbolt

7、以下程序的打印输出是：（　　）

A） 2T1 2T2　　B） 2T2 2T2　　C） 2T1 2T1　　D) 2T2 2T1

```
struct T1 {};
struct T2 {};

template <typename T>
struct traits {
    typedef T1 tag;
};

template <>
```

```
struct traits<std::string>  {
    typedef T2 tag;
};

int main()
{
   using namespace std::literals;
   std::cout << typeid(traits<decltype(42)>::tag).name() << std::endl;
   std::cout << typeid(traits<decltype("hello"s)>::tag).name() << std::endl;
}
```

godbolt

8、以下程序的打印输出是：()

A) 122　　B) 222　　C) 121　　D) 编译通不过

```
void bar () { std::cout << "1";}

template<typename T>
struct Base { void bar(){std::cout <<"2";} };

template<typename T>
struct Sub : Base<T> {
    void func1() { bar(); }
    void func2() { this->bar(); }
    void func3() { Sub::bar();};
};

int main()
{
    Sub<int> s;
    s.func1();
    s.func2();
    s.func3();
}
```

godbolt

9、以下程序的打印输出是：()

A) 111　　B) 110　　C) 编译通不过　　D) 可能会崩溃

```
#include <iostream>

template<typename T>
struct ordered
{
    bool operator>(T const& rhs) const
    {
        T const& self = static_cast<T const&>(*this);
        return rhs < self;
    }
};

class MyInt : public ordered<MyInt>
{
```

```
public:
    explicit MyInt(int x) : m_val(x) {}
    bool operator<(const MyInt& rhs) const { return this->m_val < rhs.m_val; }
private:
    int m_val;
};

class YourInt : public ordered<MyInt>
{
public:
    explicit YourInt(int x) : m_val(x) {}
    bool operator<(const YourInt& rhs) const { return this->m_val < rhs.m_val; }
private:
    int m_val;
    int m_array[42];
};

int main()
{
    std::cout << (MyInt(4) < MyInt(7)) << (MyInt(9) > MyInt(3)) << (YourInt(9) >
MyInt(3) ) << std::endl;
}
```

godbolt

10、以下程序的打印输出是：（）

A）01　　B）11　　C）10　　D）00

```
struct input_iterator_tag {};
struct dummy_tag {};

template <typename T> struct traitsA { typedef input_iterator_tag
iterator_category; };
template <typename T> struct traitsB  { typedef dummy_tag dummy_category; };

template <typename T>
struct has_iterator_category
{
private:
    struct two { char a; char b;};
    template <typename U> static two test(...) {};
    template <typename U> static char test(typename U::iterator_category* = 0)
{};
public:
    static const bool value = sizeof(test<T>(0)) == sizeof(char);
};

struct A : public traitsA<A> {};
struct B : public traitsB<B> {};

int main()
{
    std::cout << has_iterator_category<A>::value <<
has_iterator_category<B>::value << std::endl;
}
```

godbolt