第一题

B.运算符重载不可更改运算的优先级。

D.所谓运算符的目，是指在调用时的数据个数。比如a+b的+是双目的，a++的++是单目的。

注意到，运算符的目数和运算符重载的参数列表里的参数个数没有必然关系。这里实际上是全局函数重载和成员函数重载的区别：

比如Object类的对象a与b，如果是全局函数重载，那么函数是这样的：

```
Object operator+(Object a, Object b){....}
```

但是如果是成员函数重载：

```
class Object{
public:.....
    Object operator+(Object b);
    //类内.h里声明
}
//类外，.cpp里定义
Object Object::opprator+(Object b){....}
```

从而可以见得，参数个数和目数没有必然关系。不指明是全局函数重载还是成员函数重载，那么双目运算符的参数可以为2也可以为1。另一方面，单目运算符参数也可以为1或0，甚至2。

为1和0就是在全局函数和成员函数时参数不同，而为2是特指全局函数重载后缀加法。

```
//成员函数重载前缀加法，这就满足了参数为1且是单目运算符
ClassName operator++(int dummy);

//全局函数重载前后缀加法
++a等价于operator++(a)
a++等价于operator++(a,int)
//记住，如果有哑元，则是postfix（后置）,否则，就是prefix（前置）。
```

第二题

> 如果重载自定义类`Test`的输出流运算符，则函数签名必须是`ostream & operator<<(ostream &, const Test &);`

显然是错的，Test不一定需要const，但是ostream流绝对不能有const，因为输出过程是会改变ostream的。以及，三个等号是必须的。函数在类内声明几乎必须写成友元，为了访问private对象。

> 若表达式`a++`中的"++"是作为成员函数重载的运算符,则与`a++`等效的运算符函数调用形式可以为`a.operator++(n)`，其中`n`为任意整数

a=a+b实际上是a=a.operator+(b);而后缀运算符重载a++，编译器会特判为a.operator++(int)，然后调用带有dummy的重载。

```
#include <iostream>
```

```
using namespace std;

class Test {
public:
  int data = 1;
  Test(int d) {data = d;}
  Test operator++ (int) {
    Test test(data);
    ++data;
    return test;
  }
};
int main() {
  Test test(1);
  test++;
  return 0;
}
```

> 运算符重载只能通过类成员函数实现

显然错的。

第三题

> 类的默认构造函数如果没有被用户显式定义，则一定会被编译器隐式合成

还有显式删除默认构造函数，这个操作的意义在于避免危险构造。

有时候，我们也可以显式地声明禁用某些带有风险的构造函数。这种禁用不仅可以禁用编译器合成的默认构造函数，也可以用来禁止一些自动类型转换带来的构造函数调用

```
#include <iostream>
using namespace std;
class A {
private:
    int a = 1;
    double b {2.0};
    char c = 'c';
public:
    A() = default;
    A(int i):a(i) {cout<<i<<endl;}
};
int main(){
    A a('c');
    return 0;
}
输出结果 99
```

这一代码存在风险，本意一定是希望他报错，但是实则不会。从正确性上讲，这样的代码没有问题，char和int可以类型转换，故而将'c'转为了int，调用了参数为int的构造函数。但是从工程的角度讲，这是很危险的行为。因为在开发者看来，用字符初始化应该是未定义的行为。

故而显示地禁用某一构造函数。

```cpp
class A {
private:
        int a = 1;
        double b {2.0};
        char c = 'c';
   public:
        A() = default;//注意default是删除默认构造函数
        A(int i):a(i) {}
        A(char ch) = delete;//delete是删除其他构造函数
};

A a('c'); // 编译错误
```

> 全局对象在`main()`函数调用之后立刻被构造，在`main()`函数执行完之前不会被析构

全局函数在main调用之前就已经构造，在main结束之后才被析构。

> 引用在声明后，可以像指针一样更改指向的对象

引用不能改。

> 可以通过给析构函数传入参数实现在析构时期待的特定行为

析构函数可以在函数体内进行操作，但是不能传参数，也没有返回值。

第四题

```cpp
class B {
public:
   B(int i) {}
};

class A {
private:
   int a = 1;
   B b;   // (2)
public:
   A() = default; // (1)
   A(int i):a(i), b(i) {}
};
```

> A类的一个对象`a`执行析构时，会先调用类`B`的析构函数来析构a中的数据成员`b`，再调用类`A`的析构函数

回顾组合与继承的知识，析构顺序是和构造顺序相反。而构造顺序是先按照声明次序构造组合的部分，在按照声明次序构造A的部分。故而先构造B b，随后是A的其他部分。

比如在下面的顺序中是构造b，然后c，然后y，然后x。析构则是先析构A，然后C最后B。

```
class A {
private:
   int y = 2;
   int x = 1;
   B b;   // (2)
   C c;
public:
   A() = default; // (1)
   A(int i):a(i), b(i) {}
};
```

> `main`函数中可以使用`A a('c');`来构造类A的对象

没有A(char ch) = delete; 故而先转换了然后调用：A(int i):a(i), b(i) {}

第五题

> 友元类必须在想要访问其私有成员的类内声明并实现

友元必须在类内声明。友元函数在规范意义上不要在类内实现，但是友元类必然不可以在类内实现。

```
class B {};

class A {
    friend B;
};
```

```
class X
{
    friend class Y {}; ☹
}
```

回忆下friend B和friend class B的区别，前者必须B在A之前定义，而后者可以B在A之后定义。

> 被声明为当前类的友元的函数一定是全局函数

并不，其他类的成员函数也能声明为友元，且能够访问当前类的所有数据成员。

> 类的析构函数不能被声明为别的类的友元函数

这是可以的。

> 如果在`A`的类内写有`friend class B`;那么`A`的所有成员函数均能访问`B`的所有成员。

反了。

第六题

> 常量对象既能调用常量成员函数，也能调用非常量成员函数，但在调用时不能修改对象状态

错的！若对象被定义为常量(const ClassName a;)，则它只能调用以const修饰的成员函数，也即是对象中的"数据"不能变。如果调用了非常量的成员函数，那么就有可能改变对象的数据，故而常量对象不可调用非常量成员函数。但是，常量对象可以成为非常量成员函数的参数。

> 非常量对象的常量成员函数能访问不修改对象状态的非常量成员函数

这是错的，实际上不能。而常量对象的常量成员函数更不能。

> 如果在`a.h`中声明并定义全局静态变量`static int v;`，`a.h`被`b.cpp`和`c.cpp`同时包含(即`#include "a.h"`)，则同时编译这三个文件可能会因为多次定义同一个静态变量`v`而导致发生编译失败

由于你加了static，所以作用域不重叠，不会重定义。

> 常量静态数据成员都只能在类外初始化

反例：const static int 和const static enum可以类内初始化。

第七题

> 静态全局对象、常量全局对象都是在进入`main`函数之前构造，执行完`main`函数以后析构

对的！

> 若类A的对象a是类B的静态成员变量，则a在程序执行到它被第一次被访问时初始化

静态成员变量是整个类共享的，先于B被实例化，从而也不一定在程序执行到它被第一次被访问时初始化。

> 函数返回局部对象的引用可能导致运行时错误

> 用new[]构造的对象必须用delete[]释放内存，否则可能会造成内存泄露

都是对的！

第八题

```
class Test{
   const int member1;
   static float member2;
public:
   Test(int mem):member1(mem){}
   int MyMember1() const {return member1;}
   static float Mymember2() {return member2;}
};
float Test::member2 = 0;
```

> member1可以像member2一样在类外初始化

不可以，回忆下什么是类外初始化。类外初始化是指在类外单独定义。但是const数据成员必须就地初始化或者参数列表里构造。

> member1的值在不同的Test对象中可以不同，但不同的Test对象只能访问同一个member2

正确！

> 成员函数MyMember1的函数体内可以增加语句，修改member2的值

常量成员函数仅仅不能改变独属于自己的部分，可以改变静态数据成员，甚至可以改变别的对象的数据成员。

> 定义一个Test类的常量对象，可以调用MyMember1和MyMember2两个成员函数

常量对象只可以调用静态成员函数与常量成员函数。