

1. mutable 成员一般在什么情况下定义？mutable 成员可以同时定义为 static、const、volatile 或它们的组合吗？说明理由。

### 参考答案

mutable 用于修饰类的非静态数据成员。它使得 const 对象的 mutable 数据成员可以被修改。mutable 不能与 const、static 连用，但可以与 volatile 连用。

2. 类的实例成员指针和静态成员指针有什么不同？在使用时为什么实例成员指针需要绑定对象（引用、对象指针）？而静态成员指针不需要绑定对象（引用、对象指针）？

### 参考答案

实例成员指针：是一个偏移量，不能移动和参与运算，需要结合对象或对象指针来使用。实例成员指针指向的是实例成员（数据成员和函数成员），而实例成员需要依附对象（this 指针）才能访问，所以在使用时为什么实例成员指针需要绑定对象（引用、对象指针）。

静态成员指针：实际上是普通的指针，存放的是成员的物理地址，不需要结合对象使用。普通的指针不依赖任何对象（没有 this 指针），所以在使用时静态成员指针不需要绑定对象（引用、对象指针）。

3. 分析如下定义是否正确，并指出错误原因。

```
struct A {
    char *a, b, *geta( );
    char A::*p;
    char *A::*q( );
    char *(A::*r)( );
};
int main(void) {
    A a;
    a.p = &A::a;
    a.p = &A::b;
    a.q = &A::geta;
    a.r = a.geta;
    a.r = &a.geta;
    a.r = &A::geta;
}
```

### 参考答案

```
void main(void) {
    A a;
    a.p = &A::a;    //错： 不能将&A::a的类型char *A::*转换为a.p的类型char A::*
    a.p = &A::b;    //对
    a.q = &A::geta; //错： a.q是1个函数， 不是函数指针
}
```

```

a.r = a.geta;    //错： a.r是函数成员指针， a.geta既不是取函数geta的地址，也不是调用geta函数
a.r = &a.geta;   //错： a.r是函数成员指针， &a.geta不是函数成员指针
a.r = &A::geta;  //对
}

```

4. 分析如下定义是否正确，并指出错误原因。

```

struct A {
    static int x = 1;
    static const int y = 2;
    static const volatile int z = 3;
    static volatile int w = 4;
    static const float u = 1.0f;
};

```

如果A中数据成员设置缺省值是错误的，则删除缺省值（即删除“=...”），那么下面的语句是否正确？

```

static int A::x = 11;
int A::y = 22;
int volatile A::z = 33;
int volatile A::w = 44;
const float A::u = 55.0f;

```

### 参考答案

```

struct A {
    static int x = 1;           //错： static，只有const 整型变量（不能带volatile）才能设置缺省值
    static const char y = 'a'; //对
    static const volatile int z = 3; //错： static，只有const 整型变量（不能带volatile）才能设置缺省值
    static volatile int w = 4;     //错： static，只有const 整型变量（不能带volatile）才能设置缺省值
    static const float u = 1.0f;   //错： static，只有const 整型变量（不能带volatile）才能设置缺省值
};
static int A::x = 11;           //错： 不能写static（只能 int A::x = 11）
char A::y = 22;                 //错： y已经有缺省值（表示已经初始化）
int volatile A::z = 33;         //错： 应当为 int const volatile A::z = 33
int volatile A::w = 44;         //对
const float A::u = 55.0f;       //对

```

5. 分析如下定义是否正确，并指出错误原因。

```

class A {
    static int *j, A::*a, i[5];
public:
    int x;
    static int &k, *n;
};
int y = 0;
int A::i[5] = {1, 2, 3};
int *A::j = &y;
int A::*j = &A::x;

```

```
int A::*A::a = &A::x;
int &A::k = y;
int *A::n = &y;
```

### 参考答案

全部都是对的，没有错误。

6. 分析如下定义是否正确，并指出错误原因。

```
class A {
    int a;
    static friend int f();
    friend int g();
public:
    friend int A();
    A(int x): a(x) { };
} a(5);

int f() { return a.a; }
int g() { return a.a; }
```

### 参考答案

语句 `friend int A()` 错误：构造函数不能有返回值，其次 `friend` 只能修饰不属于类的函数。

7. 分析错误原因。

```
struct A {
    int x = 1;
    volatile mutable int y = 2;
    const mutable int z = 3;
    static mutable int q = 4;
    extern int p = 5;
    static int t;
    int f() { return ++x; }
    int g() const { return ++y; }
    int h() const { return ++x; }
    mutable int e() { return x; }
} a;
extern int p1 = 0;
a.f();

int main() {
    const A b;
    a.t = -1;
    a.x = 10;
    a.y = 11;
    a.f();
    b.x = 20;
    b.y = 21;
```

**//const 和 mutable 相互矛盾（不能连用）**

**//mutable 只能修饰实例数据成员（不能与 static 连用）**

**//类的成员不能用 extern 存储模型修饰**

**/\*this 为 const，不能修改 x**

**//mutable 只能修饰实例数据成员**

**//不能调用程序（只能声明变量和函数、或定义变量）**

**//a.t 不存在（没有创建）**

**//b 是 const，不能修改 b.x**

```

        b.f(); //不能调用 b.f(), 因为 b 是 const、而 A::f()中的*this 是非 const
    }

```

8. 一个程序由 2 个模块 1.cpp 和 2.cpp 组成。分析下面的程序，指出错误原因，并指出 m( ) 和 main( ) 中每条正确的语句所访问的内容。

#### //1.CPP

```

namespace A {
    int y = 2;
    int g() { return -2; }
    extern int f();
}
using namespace A;
int x = 11;
int y = 22;
int f() { return 1; }
int g() { return 2; }
namespace A {
    int z = 3;
    int f(int x) { return x; }
    extern int f();
}
void m() {
    h(); //error, 访问不到
    f(1); //正确: A::f(int)
    f(); //error, 全局 f()? A::f()?
    g(); //error, 全局 g()? A::g()?
    ::f(); //正确: 全局 f()
    A::g(); //正确: A::g()
    cout << y; //error, 全局 y? A::y ?
    cout << z; //正确: A::z
}

```

#### //2.CPP

```

namespace A {
    int x = -1;
    int y = -2; //error, 重复定义
    int f() { return 1; }
    int h() { return 0; }
}
using A::h;
using A::x;
using A::t; //error, t 应当在该 using 语句之前在 A 中定义
using A::g; //error, 不能直接引用 1.cpp 中的 A::g
float x = 0; //error, 重定义
namespace A {
    int h(int x) { return x; }
}

```

```

        int t;
    }
int main( ) {
    h(1);          //error, 访问不到 A::h(int)
    return z;      //error, 访问不到 A::z
}

```

9. 完成下面堆栈类 STACK 和 REVERSE 类的函数成员定义。

```

class STACK {
    const int max;      //栈能存放的最大元素个数
    int top;            //栈顶元素位置
    char *stk;
public:
    STACK(int max);
    ~STACK( );
    int push(char v);    //将v压栈，成功时返回1，否则返回0
    int pop(char &v);    //弹出栈顶元素，成功时返回1，否则返回0
};

class REVERSE: STACK {
public:
    REVERSE(const char *str); //将字符串的每个字符压栈
    ~REVERSE( );            //按逆序打印字符串
};

void main(void) {
    REVERSE a("abcdefg");
}

```

### 参考答案

```

STACK::STACK(int max): top(0), max( (stk=new char[max])? max : 0) { }

STACK::~STACK( ) { if(stk) { delete stk; stk = 0; *(int *)&max =0; } }

int STACK::push(char v) {
    if (top >= max) return 0;
    stk[top++] = v;
    return 1;
}

int STACK::pop(char &v) {
    if (top <= 1) return 0;
    v = stk[--top] = v;
    return 1;
}

REVERSE ::REVERSE(const char *str): STACK(strlen(str)) {
    for(int s = 0; s < strlen(str); s++) {

```

```

        push(str[s]);
    }
}
REVERSE :: ~REVERSE() {
    char c;
    while( pop(c) ) printf("%c", c);
}

```

10. 找出下面的错误语句，说明错误原因。然后，删除错误的语句，指出类 A、B、C 可访问的成员及其访问权限。

```

class A {
    int a1;
protected:
    int a2;
public:
    int a3;
    ~A() { };
};
class B: protected A {
    int b1;
protected:
    int b2;
public:
    A::a1;
    A::a2;
    int b3;
    ~B() { };
};
struct C: private B {
    int c1;
protected:
    int c2;
    B::A::a2;
    A::a3;
public:
    using B::b2;
    int c3;
    int a3;
    ~C() { };
};
int main() {
    C c;
    cout << c.b2;
    cout << c.B::b2;
}

```

**参考答案**

### 错误的语句:

```
class B: A::a1;           //error, 无法访问 A::a1
class C: int a3;          //error, 与 “A::a3;” 冲突
main(): cout << c.B::b2;  //B 是 private, 不能访问
```

### 类成员访问权限:

#### **class A**

```
private:  a1
protected: a2
public:   a3, ~A()
```

#### **class B**

```
private:  b1
protected: b2; A::(a3, ~A())
public:   b3, ~B(); A::a2
```

#### **class C**

```
private:  B::(b3, ~B()); A::~~A())
protected: c2; B::A::(a2, a3)
public:   c1, c3, ~C(); B::b2
```