C++程序设计作业 1

姓名：_____ 学号：_____ 课程编号：_____

一、选择题

1. （不定项选择题）大型项目开发中，约定合理的编码规范有助于减少编码错误，提高协作效率。以下关于 C++的编码习惯描述中，不推荐的有（　　）

A. 使用宏*#define SQUARE(X) X*X* 替换简单平方函数，从而减少过多的函数调用开销。

B. 对于 namespace，应尽量多使用 using 指示语句如 *using namespace std;*来避免单行代码长度过长。

C. 合理的注释可大幅增加代码的可读性，常见的注释场景包含：接口函数注释、临时/需优化逻辑 To-do 注释、复杂逻辑说明注释等。

D. 项目开发场景中，文件过多会导致编译复杂耗时，因此应该尽量将业务逻辑集中在少量文件、少数大函数中。

E. 全局变量具有多文件可见、读写效率高等优点，应当尽可能多使用无命名空间限定的全局变量进行数据共享。

2. （不定项选择题）C++社区中有一条经典的编码建议：“尽可能使用 const”。下述关于 const 关键字的说法或使用方式正确的有（　　）

A. 令函数返回常量值，可以减少因使用错误造成的意外，如 *const int func()*函数在判断条件 *if (func() == val)*误输入为 *if (func() = val)*时会在编译阶段报错，避免发生预期外的效果。

B. 对于语句 *const char* p1 = str;*和 *char const* p2 = str;*(str 为已定义的 char 数组类型变量)，两者表示的含义相同，均表示指向 C 风格字符串的常指针（指针 const，所指内容无限定）。

C. 类的 const 成员函数不能修改类对象的任何成员变量（包含普通成员变量及静态成员变量），除非成员变量是 mutable 变量。

D. 对于自定义类型，在函数传参中，可以使用引用避免参数拷贝，此时增加 const 限定，则可进一步避免函数内部修改此引用参数值。

E. 使用 const 变量替换 C 语言风格的宏常量，可以让编译器支持该常量的静态类型检查。

3. （不定项选择题）static 关键字在 C++语法中有许多使用场景，下述关于 static 和全局变量说法错误的有（　　）

A. 类内由 static 修饰的成员变量称为静态成员变量，静态成员变量只可通过类访问，不能通过对象访问。

B. 全局变量无论是否添加 static 修饰符，生命周期都和程序的生命周期相同，且所有文件均可见。

C. 内置类型全局变量无显式初始化情况下，会被程序自动初始化为 0。

D. 类内由 static 修饰的成员函数称为静态成员函数，静态成员函数与普通成员函数的主要区别是静态成员函数不包含指向对象本身的 this 指针，所以无法在静态成员函数中访问非静态成员变量。

E. 若一般函数中包含 static 修饰的静态变量，则此函数不应作为内联函数使用。

4. （不定项选择题）下列关于 C++函数重载的说法正确的有（　　）

A. C++支持函数重载，而 C 语言不支持，所以在 C++中使用 C 语言函数时，需要添加 *extern "C"*声明。

B. 对于普通函数，函数名相同，参数列表（参数个数、顺序）及返回值任意一处存在区别均可形成函数重载

C. 函数名与参数列表相同的成员函数，可通过 const 限定符进行重载

D. 类的析构函数可以支持函数重载，所以一个类内可定义多种同函数名的析构函数

E. *void func(int)*与 *void func(const int)*无法重载，类似的，*void func(int*)*与 *void func(const int*)*也无法重载

5. （不定项选择题）初始化与赋值是一对容易混淆的概念，关于它们，下述说法正确的有（　　）

A. 初始化不是赋值，初始化的含义是创建变量时赋予其一个初始值，而赋值的含义是把对象的当前值擦除，以一个新值来替代。

B. 在类的构造函数大括号内部使用=运算符修改成员变量，均为赋值操作。

C. const 常量和引用必须进行显式初始化，且之后不能再进行赋值。

D. 应确定对象被使用前已先被初始化，以避免预期外行为，例如内置类型局部变量在不显式初始化情况下，不被初始化，直接使用可能导致结果异常。

E. C++11 支持类内初始化，在类内部定义成员变量 *const int val_ = 3;*类内初始值，则此类生成的所有对象包含的 val_值均为 3。

二、实践题

6. C++11 新增了 auto 关键字的用法，使用 auto 关键字可自动推断变量的类型。运行参考程序，回答如下问题：

（1）定义变量 val1 ~ val10 的语句部分存在错误，列出这些存在问题的语句；

（2）注释（1）中有问题的语句后，尝试分析得出其他正常初始化变量的类型（提示：可使用 *typeid(val4).name()*命令输出类型参考信息）；

（3）运行程序，回答在三种范围 for 循环中，哪些发生了额外的拷贝操作。

参考程序：

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

```
class A {
public:
    A() { std::cout << "construct" << std::endl; }
    ~A() { std::cout << "deconstruct" << std::endl; }
};

int main() {
    auto val1;                  ••••••••••••••••••••••••••••••••••• ①
    auto val2 = 1;              ••••••••••••••••••••••••••••••••••• ②
    auto val3 = 2L;             ••••••••••••••••••••••••••••••••••• ③
    auto val4 = 0.0 + val2;     ••••••••••••••••••••••••••••••••••• ④
    auto val5 = new auto(1.0f); ••••••••••••••••••••••••••••••••••• ⑤
    auto val6 = new auto{1, 2}; ••••••••••••••••••••••••••••••••••• ⑥
    auto val7 = 0, *val8 = &val7; ••••••••••••••••••••••••••••••••• ⑦
    auto val9 = 0, val10 = 3.14; ••••••••••••••••••••••••••••••••• ⑧

    std::vector<A> vec(3);
    for (auto val : vec) {}     ••••••••••••••••••••••••••••••••••• ⑨
    for (auto& val : vec) {}    ••••••••••••••••••••••••••••••••••• ⑩
    for (const auto& val : vec) {} ••••••••••••••••••••••••••••••• ⑪

    return 0;
}
```

7. 在 C++中，一般配合使用 *new([])/delete([])*进行堆上自由内存的申请及释放。实际上，还存在一种名为 placement new 的特殊内存分配方式，运行参考程序，回答以下问题：

（1）buffer 数组、p1 所指对象、p2 所指对象使用的是哪里的内存？（堆/栈/静态区/…）

（2）能否像释放 p1 一样释放 p2？尝试解释原因。

（3）为何要循环调用 p2 各元素的析构函数？尝试解释原因。

参考程序

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

struct A {
    A() : a_(3), b_("b") {}
```

```
    int a_;
    std::string b_;
};

int main() {
    const int size = 3;
    const int bufferSize = 128;
    char buffer[bufferSize] = {0};

    A* p1 = new A[size];
    A* p2 = new (buffer) A[size];

    std::cout << "p1 is " << p1 << std::endl;
    std::cout << "p2 is " << p2 << std::endl;
    std::cout << "buffer is " << &buffer << std::endl;

    delete[] p1;
    for (size_t i = 0; i < size; ++i) {
        p2[i].~A();
    }

    return 0;
}
```

三、编程题

8. 利用类的封装特性，实现一个最多只能产生 2 个对象的类型。

(可不考虑继承关系及多线程影响。提示：除直接构造类型对象外，拷贝行为也可能产生新的对象。可使用 C++11 语法= delete 禁止拷贝构造和拷贝赋值函数)