来源：2022年8月在东主楼九区某机房捡到。

OOP2022期末考试题

选择题

请回答下列单项选择题 （每道题有且仅有1个答案）

1. 以下说法不正确的是

A. g++中-c选项要求只编译不链接

B. 进行宏定义 `#define M(y) y*y+3*y`, 则`M(1+1)`的值是7

C. 在一个编译单元内、全局整型变量x的定义`int x;`只能出现一次、但声明`extern int x;`可以出现多次

D. 将main函数写成`int main(int argc, char** argv) {...}`形式后、`argv[0]`是指向程序的第一个命令行参数的指针

2. 关于下列代码的描述不正确的是

```
#include <iostream>
using namespace std;

class A {
public:
    const char* s;
    A(const char* str) : s(str) {
        cout << s << " A constructing" << endl;
    }
    ~A() {
        cout << s << " A destructing" << endl;
    }
};

class B {
public:
    static A a1;
    const A a2;
    B(const char* str) : a2(str) { cout << str << " B constructing" << endl;
}
    ~B() { cout << "B destructing" << endl; }
};
```

```
void foo() {
    static A obj("static obj");
}

A B::a1("static B::a1");  // (1)

int main() {
    // (2)
    cout << "main starts" << endl;
    static B b("static b");
    for (int i = 0; i < 4; i++) {
        foo();
    }
    cout << B::a1.s << endl;  // (3)
    cout << "main ends" << endl;
    return 0;
}
```

A. (1)处的代码可以被挪动至(2)处、编译仍能成功且输出结果不变

B. 将(3)处的代码修改为：`cout << b.a1.s << endl;`、编译仍能成功且输出结果不变

C. 在(2)后、一共调用了2次类A的构造函数

D. 在(3)后、一共有3个类A的对象执行析构

3. 关于下列代码的描述不正确的是（编译时添加选项--std=c++11 -fno-elide-constructors）

```
#include <iostream>
using namespace std;

class MyInt{
public:
    MyInt(int i = 0) : val(i)  // (a)
    {    }
    MyInt(const MyInt& src) : val(src.val)  // (b)
    {    }
    MyInt(MyInt&& src) : val(src.val)  // (c)
    {    }
    operator int() const  // (d)
    {
        return val;
    }
    MyInt& operator=(MyInt&& rhs)  // (e)
    {
        val = rhs.val;
        return *this;
    }
    MyInt& operator=(const MyInt& rhs)  // (f)
    {
        val = rhs.val;
```

```
        return *this;
    }
    int get_value() const
    {
        return val;
    }
private:
    int val;
};

void print(MyInt obj)
{
    cout << obj.get_value() << endl;
}

void print(int value)
{
    cout << value << endl;
}

int main()
{
    MyInt a;
    a = -3.8;  // (1)
    auto b = a + 1;  // (2)
    auto c = a;  // (3)
    print(a); print(b); print(c);  // (4)
    return 0;
}
```

A. (1)处会先将double转换为int，再调用(a)构造对象，最后调用(e)进行移动赋值

B. (2)处会先调用(d)将变量a转换为int，执行加法，然后调用(a)构造对象，最后调用(c)进行移动构造

C. (3)处会调用(b)进行拷贝构造

D. (4)处三个语句的输出是 -3\n-2\n-3\n，其中\n代表换行符

4. 关于下面程序，说法正确的是

```
#include <iostream>
using namespace std;
class A {
public:
    A() { fun(); }
    virtual void fun() {}
    ~A() {}
};

class B : A {
public:
    B() { fun(); }
```

```
    void fun() {}
    ~B() {}
};

class C : public B {
public:
    C() { fun(); }
    virtual void fun() {}
    ~C() {}
};

int main() {
    B *pb = new C();
    pb->fun();
    delete pb;
    cout << "All done!" << endl;
    return 0;
}
```

A. A::fun 与 C::fun 是虚函数，而 B::fun 不是虚函数

B. 程序执行过程中 C::fun 一共被调用了四次

C. 程序能够正常编译、运行、输出 All done! 并正常退出

D. 执行 delete pb 时会调用 C 的析构函数

5. 以下关于多态与模板的说法正确的是

A. 函数模板如果将声明写到头文件中，将定义写到源文件中，编译链接过程中未必会出错

B. 如果把重写覆盖实现成了重写隐藏，加 override 关键字可以让它变回重写覆盖

C. 抽象类的派生类需要实现抽象类的全部纯虚函数，否则可能会在运行过程中因调用纯虚函数导致程序崩溃

D. 将某类的指针使用 dynamic_cast 转换为另一个类的指针时，如果转换失败会抛出 bad_cast 异常

6. 下列关于程序性能的说法正确的是

A. 如果调用了函数模板，则对应函数需要在程序运行时实例化，这样会带来额外的时间成本

B. std::vector 的 push_back 方法每次操作的复杂度均为 O(1)

C. 如果使用类的指针调用它的虚函数，则需要在程序运行时查询实际的函数

D. 设计模式的应用主要是为了提高程序运行效率，不能提高开发效率

7. 下列关于 STL 的说法正确的是

A. std::tuple 可以像数组一样用 [] 访问元素，如果越界则会在编译时直接报错

B. 使用流式文件输出写入文本文件时，每次完成 << 运算符的操作后便会完成一次文件写入，不存在缓冲机制

C. 对元素类型为 T 的数组使用 std::sort 时，如果想通过类来实现一个函数对象作为比较函数传入 std::sort，则对应的类应当实现了方法 bool operator(const T&a, const T&b)

D. 即便使用了智能指针也可能会出现内存泄漏的情况

8. 下列关于设计模式的说法中错误的是

A. 逻辑复杂而结构稳定的场景下使用模板方法比策略模式更合适

B. 在使用迭代器模式时，外部算法使用迭代器，一般需要针对不同的底层数据结构进行不同的实现

C. 适配器模式可以用于在接入第三方组件时完成接口对接

D. 代理模式常在被代理对象不存在时主动创建被代理对象，而装饰器模式通常不会创建被装饰对象

提交格式

注意：考试期间选择题的提交不反馈得分，合法的提交答案都会显示50分

请你提交一个文本文件，第i行是A\B\C\D之一，代表第i题的答案。

若你不想提交第i题，请将第i行留空。

你提交的文本文件至少要有8行，且前8行必须由A\B\C\D组成，否则将被认为是无效提交。

以下给出一个合法的提交答案的例子：

```
A
B
C
D
A
C
A
A
```

评测器对第一行的解析结果如下：

```
Valid answer: ['A']
```

动物模拟

题目描述

本题需要用类模拟鸟类（Bird）和鱼类（Fish）两种动物，实现歌唱（sing）和游泳（swim）两个方法。两种动物都有红色（Red）和蓝色（Blue）两种。鸟类会歌唱但不会游泳，鱼类会游泳但不会歌唱。对某种动物调用方法时，需要输出对应的动物、方法以及颜色信息。例如，对红色的鸟类调用 sing 方法时，输出 `Red bird is singing.`，对任何颜色的鸟类调用 swim 方法时，输出`Bird can not swim.`。同理，对蓝色的鱼类调用 swim 方法时，输出 `Blue fish is swimming.`，对任何颜色的鱼类调用 swim 方法时，输出`Fish can not sing.`。当动物类被析构时，也需要输出相关信息，例如蓝色的鸟类被析构，输出 `Blue bird is gone.`。

要求：

- 使用抽象类`Animal`表示动物，`Bird`和`Fish`两个类继承自`Animal`。你需要实现虚函数以便通过指向`Animal`的指针访问`Bird`和`Fish`的实例。
- 根据题目提供的 main.cpp 文件（从这里下载）进行数据的输入输出。你需要实现 `Animal.h`、`Bird.h`和`Fish.h`三个文件。

输入说明

输入第一行时是 n（$5 \leq n \leq 10000$），代表模拟的次数。

接下来n行，每行有两个0或1的整数`op`和`color`，`op`为0代表构造鸟类，`op`为1代表构造鱼类。`color`为0代表构造的动物为红色，`color`为1代表构造的动物为蓝色。

输出说明

输出若干行，为每次模拟得到的信息。

输入样例

```
8
0 0
1 0
0 0
1 0
0 1
1 1
0 1
1 1
```

输出样例

```
Red bird is singing.
Bird can not swim.
Red bird is gone.
Fish can not sing.
Red fish is swimming.
Red fish is gone.
Red bird is singing.
Bird can not swim.
Red bird is gone.
Fish can not sing.
Red fish is swimming.
```

```
Red fish is gone.
Blue bird is singing.
Bird can not swim.
Blue bird is gone.
Fish can not sing.
Blue fish is swimming.
Blue fish is gone.
Blue bird is singing.
Bird can not swim.
Blue bird is gone.
Fish can not sing.
Blue fish is swimming.
Blue fish is gone.
```

提交格式

你需要提交Animal.h、Bird.h和Fish.h三个文件。我们会将你提交的文件和我们预先设置好的main.cpp、makefile一起编译运行。

评分标准

考试100%为OJ评分。

小小军的作业

题目背景

小军是一名优秀的大学生，他的弟弟小小军是一名优秀的小学生。最近，小小军在学习四则运算，为了帮小小军检查作业，小军要写一个程序来计算一些表达式的值，请你帮帮他。

题目描述

小小军的作业总共包含三种类型的表达式：

- A 类表达式：$x \times x + y \times y + z \times z$；
- B 类表达式：$x \times y + y \div z - x \times z$；
- C 类表达式：$y \times 3 \times z - x \times z \div (y - 2) - x \times y \div (z + 1)$；

共有 N 道题目，每道题目会给定一个表达式类型和一组 x, y, z 的值，请你求出对应表达式的值。

你需要支持整数、分数两种数据类型：在使用整数类型计算时，上述除号（$\div$）视作整除运算（即 c++ 中 int 类型的 / 运算）；在使用分数类型计算时，你需要输出最简分数。

要求

小军已经完成了主程序 main.cpp（从这里下载），你需要在不改动这些文件的情况下，提供其他文件（包括calculate.h、Fraction.h、Fraction.cpp），来完整实现题目的要求。

输入输出格式

输入

第一行一个整数 N，表示题目的数量。

接下来 N 行，每行描述一个表达式：

- 以 A、B、C 的其中之一开头，表示表达式的类型；
- 紧跟着三个整数 x, y, z；
- 最后为 int、fraction 的其中之一，表示数据类型（前者表示整数类型，后者表示分数类型）；
- 以上各项内容以单个空格分隔。

输入输出样例

输出 N 行，依次为每道题目的答案。对于要求使用分数类型的题目，输出结果应当使用 / 直接分割分子与分母（如 1/2、-3/5 等），且需要化简为最简形式（如 2/4 应化为 1/2、1/1 应化为 1）。

输入

```
4
A 1 2 3 int
B -4 5 -2 int
C 98 99 100 fraction
B 1 1 1 fraction
```

输出

```
14
-30
289040006/9797
1
```

数据规模

对于 50% 的测试点，保证仅包含 A 类表达式，且数据类型一定为 int（整数类型）。

对于 80% 的测试点，保证仅包含 A、B 类表达式。

对于所有测试点，保证 $N \leq 1,000$，且 x, y, z 的绝对值不超过 100。

对于所有测试点，保证所有表达式在计算过程中不会出现除数为零的情况。

提交格式

你需要提交calculate.h、Fraction.h、Fraction.cpp三个文件。我们会将你提交的文件和我们预先设置好的main.cpp、makefile一起编译运行。

评分标准

考试100%为OJ评分。

提示

在最简分数的化简时，需要将分子、分母同时除以它们的最大公约数。正整数的最大公约数求解可以参考如下代码（该 gcd 函数的返回值即为 a 与 b 的最大公约数）：

```
int gcd(int a, int b){
    return b ? gcd(b, a % b) : a;
}
```

当然，你也可以通过其他方法自己实现最简分数的化简功能。

如果程序运行中出现 int 型整数溢出的情况，可以考虑使用 long long 类型。

小明的PVector管理

题目描述

注意：请合理安排考试时间，可以选择实现部分任务以获得部分分数

小明想最近了解到了可持久化数据结构（persistent data structure）的概念，因此想仿照着设计一个可持久化的vector容器，称为PVector。

Subtask1

简单来说，可持久化数据结构支持在修改之后仍然保留原来的历史版本。例如，任意修改PVector的操作将返回一个全新的PVector对象，而原有PVector不变。

```
// Subtask 1
PVector<int> a1;                    // Create an empty PVector, a1 = []
cout << a1 << endl;                 // Output: []
auto a2 = a1.push_back(1);          // a2 = [1]
cout << a2 << endl;                 // Output: [1]
int index = 0, value = 2;
auto a3 = a2.set(index, value);     // modify an element, a3 = [2]
cout << a3 << endl;                 // Output: [2]
auto a4 = a2.push_back(3);          // a4 = [1, 3]
cout << a4 << endl;                 // Output: [1, 3]
cout << a4[1] << endl;              // Output: 3
auto a5 = a4.push_back(5);          // a5 = [1, 3, 5]
cout << a5 << endl;                 // Output: [1, 3, 5]
```

进一步，小明希望这些操作的内存与时间开销不要太大，**即对于每一个push_back或set操作，应该只记录修改，不能将整个PVector都复制一遍。**

为了测试他的代码，他使用自定义类型Point构造了PVector<Point>。每个Point包含x, y两个坐标，并且重载了输出流运算符（具体可以下载代码查看）。同时，对于Point类型，小明会检查所有Point对象的构造、析构次数。你需要满足：

- Point对象的构造次数不应该超过给定的参考值，在subtask1中是push_back和set的操作数量的4倍。

- Point对象的析构次数应该与构造次数相等，避免内存泄露。

当然，若你满足不了上述条件，我们也会有部分分，具体可以查看题目最后的评分标准。

Subtask2

在Subtask1的基础上，小明还想增加一个撤销功能。对于一个PVector，可以通过undo函数获得上一次修改前的版本。注意undo可以多次使用。如果已经是最初的，操作无效并输出cannot undo。（操作无效时请返回当前对象，不做任何操作。）

特别的，undo操作不应该消耗构造次数，即要求：

- Point对象的构造次数不应该超过给定的参考值，在subtask2中是push_back和set的操作数量的4倍。

（如果你对以下代码的输出有疑惑，我们提供一张图片展现了测试代码中各个对象的关系，请在最后的链接中下载）

```
// Subtask 2
// Codes after Subtask 1
auto b4 = a5.undo();                    // b4 = [1, 3]
auto b2 = b4.undo();                    // b3 = [1]
auto b1 = b2.undo();                    // b1 = []
b1.undo();                              // Output: cannot undo
auto b6 = b2.push_back(0);              // b6 = [1, 0]
```

Subtask3

最后，小明想添加一个终极功能：合并两个PVector的修改。如果对于同一个PVector pv_origin，做出了不同的修改后分别得到了pv_a和pv_b。那么使用pv_merge = pv_a.update(pv_b)将得到综合两个修改的结果。具体来说：

- 从pv_origin到pv_b的所有操作将插入到pv_a的修改之后。这次update被认为是一次操作，即通过pv_merge.undo()可以回到pv_a的状态。特殊地，允许pv_origin是pv_a或者pv_b本身。
- 如果pv_a和pv_b不是从同一个PVector修改而来，那么操作无效，应输出cannot update: no origin found。（操作无效时请返回当前对象，不做任何操作。）
- 如果pv_a和pv_b的修改有冲突：即都在队尾插入了元素，或者都修改了同一个元素：那么操作无效，应输出cannot update: conflicts found。（操作无效时请返回当前对象，不做任何操作。）

注意，在判断修改是否有冲突时，可能有两种需要考虑的情况：

- 被合并的对象可能会经历过undo操作，此时被撤销的操作不应考虑在冲突里。举例来说：
  - 从pv_origin做出了修改A、B得到了pv_a。（其中A、B可能是push_back, set, update操作）。
    - 从pv_origin做出了修改D、E、F，再经历一次undo得到了pv_b。（其中D、E、F可能是push_back, set, update操作）。
  - pv_a.update(pv_b) 将在pv_a上进行D、E操作，而不是D、E、F、undo操作，因此不用考虑 操作F 与 操作A、B 是否有冲突。
- 如果对同一个对象执行了两次*字面上一样*的操作，我们仍然认为他们经历了不同的修改，在合并时会产生冲突。举例来说：

- 从pv_origin做出了push_back(1)得到了pv_a。
- 再次从pv_origin做出了push_back(1)得到了pv_b。
- pv_a.update(pv_b)将产生冲突，因为我们认为两次push_back(1)都增加了序列长度，因此产生了冲突。

特别的，update操作最多消耗的构造次数应该和update涉及操作数量有关，即要求：

- Point对象的构造次数不应该超过给定的参考值，在subtask3中是(push_back和set的操作数量 + update所涉及操作数量)*4。
- 其中，update所涉及的操作数量是指，从pv_origin到pv_b、pv_a之间的所有操作数量之和（操作中push_back,set,update都只计一次）。

（如果你对以下代码的输出有疑惑，我们提供一张图片展现了测试代码中各个对象的关系，请在最后的链接中下载。）

```
// Subtask 3
// Codes after Subtask 2
auto c7 = a3.update(a5);          // c7 = [2, 3, 5]
auto c3 = c7.undo();              // c3 = [2]
auto c8 = a5.update(a3);          // c8 = [2, 3, 5]
auto c5 = c8.undo();              // c5 = [1, 3, 5]
auto c9 = a1.update(c8);          // c9 = [2, 3, 5]
auto c10 = c8.update(a1);         // c10 = [2, 3, 5]
auto c11 = c10.undo();            // c11 = [2, 3, 5]
auto c12 = a5.update(a3);         // c12 = [2, 3, 5]

PVector<int> other;
a3.update(other);                   // Output: cannot update: no origin found
a4.update(b6);                      // Output: cannot update: conflicts found
c10.update(b6);                     // Output: cannot update: conflicts found
a2.update(c9);                      // Output: cannot update: conflicts found
c12.update(c8);                     // Output: cannot update: conflicts found
```

提示

- 小明已经实现了部分代码，可以在下方链接中下载，你可以基于小明的代码修改，也可以完全自己来实现。他的实现思路如下：
  - 小明完全不会使用vector容器，而是使用装饰器模式来避免复制原有对象。具体来说，每层装饰器只记录修改的部分，通过重写覆盖T get(int index)函数来修改元素访问时返回的内容。
  - 由于PVector是一个对象，因此无法直接使用虚函数。他在PVector内部维护一个指针，指向真正的容器对象Data。
    - 传参和返回值中使用const T&有助于减少构造函数调用次数(但注意不能返回局部变量的引用）。
  - 小明目前还没考虑内存泄露问题，但预计可以使用智能指针解决。
- 为了简单考虑，我们对题目做出以下限制：
  - 你只用考虑PVector<int>和PVector<Point>。
  - 所有操作保证不会越界。
- 本题的样例测试代码在main_int.cpp、main_point.cpp中。我们提供main_int.cpp、main_point.cpp、point.h和Makefile。

- 特殊地，Makefile支持测试部分subtask。如果你只想测试subtask1，可以使用make subtask1命令。
- 为了辅助你的理解，我们提供一张图片展现了测试代码中各个对象的关系，请在以下链接中下载。
- 链接中我们提供了完整的样例输出。
- 文件下载地址：下载链接。

提交格式

你只需提交pvector.h。我们会将你提交的文件和我们预先设置好的main_int.cpp、main_point.cpp、point.h、Makefile一起编译运行。

评分标准

注：在2022年6月的考试中，本题以50分记。

我们共有3个subtask：

- SUBTASK1（25分）：你需要实现push_back, set, 流输出, 下标访问操作。保证所有PVector长度不超过100，操作不超过100。
- SUBTASK2（25分）：在SUBTASK1基础上，你需要实现undo操作。保证所有PVector长度不超过100，操作数量不超过100。
- SUBTASK3（50分）：在SUBTASK2基础上，你需要实现update操作。保证所有PVector长度不超过100，操作数量不超过100。

每个subtask中会有两个样例测试点，即下发的main_int.cpp, main_point.cpp。另外也有2个隐藏测试点，会相应的更改样例代码以及point.h代码进行测试。一般来说，如果正确实现了题目要求，设计符合复杂度要求，并能通过样例测试点（而不是通过某种方法直接输出标准答案），也应该能够通过隐藏测试点。（若你认为存在问题，请及时联系监考老师。）每个subtask分数分为4挡：

- 能通过main_int.cpp以及只包含int的隐藏测试点，获得该subtask 25%分数。
- 满足以上条件，并能通过main_point.cpp以及包含Point的隐藏测试点，获得该subtask 50%分数。
- 满足以上条件，并且构造数量符合要求，即不超过(push_back和set的操作数量 + update所涉及操作数量) * 4，获得该subtask 75%分数。
- 满足以上条件，并且没有内存泄露，获得该subtask 100%分数。

注意你不用同时通过3个子任务再提交，我们会将每一个子任务的代码拆开，分别编译。

考试100%为OJ评分。

时间限制：2s
内存限制：256M

对于OJ反馈的解释

该题共有3个subtask，每个subtask会有4个测试点。从上至下分别是：

- main_int样例。
- 只包含int的隐藏测试点。
- main_point样例。
- 包含Point的隐藏测试点。

其中：

- 前2个测试点均正确时将获得25% subtask得分。

- 后2个测试点最低得分为25时（代表构造函数次数超过限制），获得50% subtask得分。
- 后2个测试点最低得分为50时（代表出现内存泄露），获得75% subtask得分。
- 后2个测试点均正确时，获得100% subtask得分。

Subtask中的每个测试点得分之和不等于该subtask总分属于正常现象。