Задачи к годовому курсу “Язык C++”

Эти задачи давались студентам 1 курса бакалавриата ФИВТ МФТИ в 2016-17 и в 2017-18 учебном году.

Автор текста всех задач, кроме 3, 6, 7 – Мещерин Илья (http://mesyarik.ru, vk.com/mesyarik)

Автор текста задач №3, 6, 7 – Рухович Филипп

*****

ПЕРВЫЙ СЕМЕСТР

Задача №1. Длинная арифметика

В этой задаче **разрешается** подключать <iostream>, <vector> и <string> и только их.

А) (4 балла). Напишите класс BigInteger для работы с длинными целыми числами. Должны поддерживаться операции:

- сложение, вычитание, умножение, деление, остаток по модулю, работающие так же, как и для int; составное присваивание с этими операциями;
- унарный минус, префиксный и постфиксный инкремент и декремент,
- операторы сравнения == != < > <= >=
- вывод в поток и ввод из потока;
- метод toString(), возвращающий строковое представление числа;
- конструирование из int (в том числе неявное преобразование, когда это надо);
- неявное преобразование в bool, когда это надо (должно работать в условных выражениях).

В вашем файле должна отсутствовать функция main(), а сам файл должен называться biginteger.cpp. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Б) (+2 балла) Используя класс BigInteger, напишите класс Rational для работы с рациональными числами сколь угодно высокой точности. Числа Rational должны представляться в виде несократимых обыкновенных дробей, где числитель и знаменатель – сколь угодно длинные целые числа. Должны поддерживаться операции:

- конструктор из BigInteger, из int;
- сложение, вычитание, умножение, деление, составное присваивание с этими операциями; унарный минус;
- операторы сравнения == != < > <= >=
- метод toString(), возвращающий строковое представление числа (вида [минус]числитель/знаменатель), где числитель и знаменатель - взаимно простые числа; если число на самом деле целое, то знаменатель выводить не надо;
- метод asDecimal(size_t precision=0), возвращающий представление числа в виде десятичной дроби с precision знаками после запятой;
- оператор приведения к double.

В вашем файле должна отсутствовать функция main(), а сам файл должен называться rational.cpp. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Задача №2. Перестановки

В этой задаче **нельзя** использовать никакие сущности из STL, кроме потоков ввода-вывода.

А) (2 балла) Напишите класс Permutation для работы с перестановками (в алгебраическом смысле). Перестановка – массив чисел размера n, заполненный числами от 0 до n-1. Должны поддерживаться операции:

- Конструирование из unsigned int (создается тождественная перестановка заданной длины), а также из unsigned int n и массива длины n (создается соответствующая перестановка, предполагаем, что массив корректно заполнен нужными числами);
- Умножение перестановок;
- Инкремент, декремент, а также функции next() и previous(), возвращающие лексикографически следующую и предыдущую перестановку для данной;
- Операторы <, >, ==, !=, <=, >=, сравнивающие перестановки в лексикографическом порядке;
- Обращение по индексу к i-му элементу перестановки, но без возможности его изменения;
- Метод inverse(), возвращающий обратную перестановку для данной;
- void operator() над массивом, который применяет данную перестановку к элементам массива.

В вашем файле должна отсутствовать функция main(), а сам файл должен называться permutation.cpp. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Б) (+1 балл) Допишите в класс перестановок функцию size_t derangementsCount(), возвращающую количество инверсий в перестановке за O(n log n).

Для тестирования данного пункта отправьте свое решение в этот контест (ссылка скоро появится).

В) (+2 балла) Допишите в класс перестановок:

- Функции bool isOdd(), bool isEven(), проверяющие, является перестановка четной или нечетной, работающие за O(n);
- Метод Permutation pow(int degree), возвращающий результат применения данной перестановки degree раз подряд, работающий за O(n).

Г) (+2 балла) Используя ранее написанный класс BigInteger, допишите в класс перестановок:

- Конструктор Permutation(unsigned n, BigInteger lexNumber) – создает lexNumber'ю по счету перестановку размера n, если выписать их в алфавитном порядке;
- Функция BigInteger getLexNumber() – для перестановки возвращает ее лексикографический номер от 0 до n!-1;
- Операторы +=(int count), -=(int count), +(int count) и –(int count), соответственно переходящие к перестановке, на count отстоящей от данной в лексикографическом порядке.

Задача 3. Дек

Реализуйте класс deque - динамический массив, поддерживающий следующие операции:

а) push_back;
б) pop_back;
в) push_front;
г) pop_front;
д) operator [] (доступ к i-му элементу в массиве);

Учетное время работы каждой из этих операций должно быть O(1). Количество занимаемой памяти в каждый момент времени должно быть O(текущее количество элементов в векторе). Проведите анализ с помощью метода бухгалтерского учета и с помощью метода потенциалов.

Требования к реализации:

Класс deque должен быть шаблонным. Объем используемой памяти не должен превышать (константа1 * количество элементов + константа2).
Кроме указанных операций, требуется реализовать:

- методы back(), back() const, front(), front() const,
- Оба вида конструкторов: обычный, конструктор копирования.
- operator[], operator[] const,
- методы bool empty() const, size() const,
- константный и неконстантный итераторы произвольного доступа (оба класса необходимо унаследовать его от std::iterator; оба итератора будут похожи, поэтому будет необходимо избавиться от “копипасты”);
- методы begin(), begin() const, cbegin() const, end(), end() const, cend() const, rbegin(), rbegin() const, crbegin() const, rend(), rend() const, crend() const, являющиеся методами итератора произвольного доступа,
- тесты, проверяющие описанную выше функциональность с использованием GoogleTest.
- тесты, проверяющие, что учетное время работы O(1).

Задача 4. Геометрия

Напишите иерархию классов для работы с геометрическими фигурами на плоскости.

- Структура Point - точка на плоскости. Точку можно задать двумя числами типа double. Должны быть открыты поля x и y. Точки можно сравнивать операторами == и !=.
- Класс Line - прямая. Прямую можно задать двумя точками, можно двумя числами (угловой коэффициент и сдвиг), можно точкой и числом (угловой коэффициент). Линии можно сравнивать операторами == и !=.
- Абстрактный класс Shape - фигура.
- Класс Polygon - многоугольник. Многоугольник - частный случай фигуры. У многоугольника можно спросить verticesCount() - количество вершин - и std::vector<Point> getVertices - сами вершины без возможности изменения. Можно спросить isConvex() - выпуклый ли. Можно сконструировать многоугольник из вектора точек-вершин в порядке обхода. Можно сконструировать многоугольник из точек, передаваемых в качестве параметров через запятую (т.е. неуказанное число аргументов). Для простоты будем считать, что многоугольники с самопересечениями никогда не возникают (гарантируется, что в тестах таковые будут отсутствовать).
- Класс Ellipse - эллипс. Эллипс - частный случай фигуры. У эллипса можно спросить std::pair<Point,Point> focuses() - его фокусы; std::pair<Line, Line> directrices() - пару его директрис; double eccentricity() - его эксцентриситет, Point center() - его центр. Эллипс можно сконструировать из двух точек и double (два фокуса и сумма расстояний от эллипса до них);
- Класс Circle - круг. Круг - частный случай эллипса. У круга можно спросить double radius() - радиус. Круг можно задать точкой и числом (центр и радиус).

- Класс Rectangle - прямоугольник. Прямоугольник - частный случай многоугольника. У прямоугольника можно спросить Point center() - его центр; std::pair<Line, Line> diagonals() - пару его диагоналей. Прямоугольник можно сконструировать по двум точкам (его противоположным вершинам) и числу (отношению смежных сторон), причем из двух таких прямоугольников выбирается тот, у которого более короткая сторона расположена по левую сторону от диагонали, если смотреть от первой заданной точки в направлении второй.
- Класс Square - квадрат. Квадрат - частный случай прямоугольника. У квадрата можно спросить Circle circumscribedCircle(), Circle inscribedCircle(). Квадрат можно задать двумя точками - противоположными вершинами.
- Класс Triangle - треугольник. Треугольник - частный случай многоугольника. У треугольника можно спросить Circle circumscribedCircle(), Circle inscribedCircle(), Point centroid() - его центр масс, Point orthocenter() - его ортоцентр, Line EulerLine() - его прямую Эйлера, Circle ninePointsCircle() - его окружность Эйлера.

У любой фигуры можно спросить:

- double perimeter() - периметр;
- double area() - площадь;
- operator==(const Shape& another) - совпадает ли эта фигура с другой;
- isCongruentTo(const Shape& another) - равна ли эта фигура другой в геометрическом смысле;
- isSimilarTo(const Shape& another) - подобна ли эта фигура другой;
- containsPoint(Point point) - находится ли точка внутри фигуры.

С любой фигурой можно сделать:

- rotate(Point center, double angle) - поворот на угол (в градусах, против часовой стрелки) относительно точки;
- reflex(Point center) - симметрию относительно точки;
- reflex(Line axis) - симметрию относительно прямой;
- scale(Point center, double coefficient) - гомотетию с коэффициентом coefficient и центром center.

Задача 5. Matrix

Баллы за эту задачу можно получить только при условии прохождения тестов по задачам BigInteger и Rational.

Написать шаблонный класс Finite<int N>, реализующий концепцию конечного поля из N элементов. Должны поддерживаться арифметические операции, кроме деления, а в случае простого N и деление тоже. Элемент поля Finite<N> должно быть можно сконструировать от int путем взятия остатка от деления на N (в математическом смысле).

После этого, используя ранее написанный класс рациональных чисел, написать класс Matrix с тремя шаблонными параметрами: unsigned M, unsigned N, typename Field=Rational. (По умолчанию берётся поле рациональных чисел, но можно создать матрицу и над конечным полем.)

Матрицы должны поддерживать следующие операции:

- Проверка на равенство: операторы == и !=.
- Конструктор по умолчанию, создающий единичную матрицу. Для неквадратных матриц конструктор по умолчанию не требуется.
- Конструктор, создающий матрицу из vector<vector<T> >. Должно быть можно создать матрицу из vector<vector<int> >.
- Сложение, вычитание, операторы +=, -=. Сложение и вычитание матриц несоответствующих размеров не должно компилироваться.
- Умножение на число.
- Умножение на матрицу и оператор *=, работающие за o(max(M,N)**3), здесь o-малое - не опечатка. Попытка перемножить матрицы несоответствующих размеров должна приводить к ошибке компиляции.
- Метод det(), возвращающий определитель матрицы за O(N**3). Взятие определителя от неквадратной матрицы не должно компилироваться.
- Метод transposed(), возвращающий транспонированную матрицу.
- Метод rank() - вычислить ранг матрицы.
- Метод trace() - вычислить след матрицы.
- Методы inverted() и invert() - вернуть обратную матрицу и обратить данную матрицу.
- Методы getRow(unsigned) и getColumn(unsigned), возвращающие std::vector<Field> из соответствующих значений.
- К матрице должен быть дважды применим оператор [], причём это должно работать как для неконстантных, так и для константных матриц. В первом случае содержимое матрицы должно быть можно таким способом поменять.
- Другие способы изменения содержимого матрицы, кроме описанных выше, должны отсутствовать. Однако не запрещается реализовать дополнительные методы для выполнения каких-либо иных алгебраических операций или для удобства работы, если по названию и сигнатуре этих методов будет без комментариев понятно их действие.
- Квадратные матрицы размера N должно быть можно объявлять всего с одним обязательным шаблонным параметром: SquareMatrix<N>.

Задача 6. BitSet

Напишите класс BitSet для работы с массивом, состоящим из фиксированного количества бит (размер задается как шаблонный аргумент типа size_t) - упрощенный аналог std::bitset из STL.

Реализация должна основываться на массиве чисел типа unsigned int и хранить по sizeof(unsigned int) * CHAR_BIT бит в каждом числе. (CHAR_BIT можно найти в <climits>)

Должны поддерживаться операции:

- Конструктор без аргументов (должен инициализировать биты нулями).
- Конструктор из unsigned int. Инициализация битов производится с младших битов. Таким образом BitSet<10> test(13) представляется строкой “0000001101”.
- Конструктор копирования.
- Логического и (&), или (|), исключающее или (^).
- Унарный оператор отрицания (~).
- Операции сдвига битов (влево <<, вправо >>). Считаем, что биты обнуляются при выходе за пределы хранимого интервала.
- Оператор присваивания (=) и сокращенные операторы присваивания (<<=, >>=, &=, |=, ^=)
- метод size() - вернуть количество хранимых битов.
- метод count() - количество битов со значением единица.
- метод toString(). Возвращаемая строка должна состоять из нулей и единиц. Наиболее значимые биты идут сначала.
- оператор << для вывода в поток BitSet в виде строки.
- метод bool get(size_t position) - возвращает значение бита в заданной позиции.
- метод set(size_t position, bool value=true) - установить value как значение бита в заданной позиции. Если метод set вызывается без аргументов, то установить значения всех битов как истинное. Нужно уметь поддерживать цепочечные вызовы, например testSet.set(3).set(4).set(10).
- операторы == и !=.

Задача 7. HashMap

Реализуйте шаблонный класс HashMap фиксированного размера, использующий открытую адресацию.

Шаблонные параметры:

0) SIZE - размер хэш-таблицы;
1) Типы Key, Value - ключ и значение;
2) Hash - функтор из Key в unsigned int, хэширующий ключи;
3) Equal - функтор проверки ключей на равенство (с аргументом по умолчанию);
4) GetIndex(keyHash, iterationNumber, tableSize) - функтор, возвращающий индекс, в котором дОлжно искать значение keyHash при заданном tableSize при заданном номере итерации (с аргументом по умолчанию);

Методы:

1) Конструктор, в котором задается размер таблицы; этот размер фиксируется раз и навсегда;
2) Конструктор копирования, оператор =;
3) Метод swap, работающий за O(1);
4) Оператор []; позволяющий получить ссылку на значение по ключу; если такого значения нет, то создать;
5) Метод at (константный и неконстантный), позволяющий получить ссылку на значение по ключу; если такого значения нет, то бросить исключение;
6) Методы empty, size, clear();
7) Метод bool erase(key), удаляющий запись с заданным ключом; если такого ключа нет, то удалять не требуется; метод возвращает true, если элемент был удален, и false иначе.

8) Итераторы, позволяющие выполнять обход таблицы; итераторы должны быть bidirectional; на итераторы налагаются следующие требования:
    a) Методы * и ->, дающие доступ к std::pair<const Key, Value>;
    b) Методы ++ и --, префиксные и постфиксные, выполняющиеся за O(1) в худшем случае;
    c) Должны быть константные и неконстантные итераторы.
9) В классе должны быть определены тип ключа на key_type, хранимого значения как value_type, тип итератора как iterator, тип константного итератора как const_iterator.
10) Методы begin() (константный и неконстантный), cbegin(), end()(константный и неконстантный), cend();
11) Метод find(key), возвращающий итератор на элемент с заданным ключом; если такого элемента нет, необходимо вернуть end().

Запрещается использовать контейнеры и алгоритмы STL (std::pair контейнером не считается).

****

ВТОРОЙ СЕМЕСТР

Задача 8 (версия 2018г). XOR-List

В этой задаче запрещается пользоваться стандартными контейнерами STL, все выделения памяти нужно делать вручную.

Часть 1.

Напишите шаблонный класс StackAllocator<typename T>, интерфейс которого аналогичен интерфейсу std::allocator. Он должен однократно выделять большой блок памяти и многократно выдавать память под объекты последовательно из этого блока. Если блока памяти не хватает, то выделяется новый большой блок, и т.д.. При запросах на деаллокацию объектов нужно ничего не делать. Таким образом, память освобождается только тогда, когда instance аллокатора прекращает использоваться. Это дает большой выигрыш во времени при работе со структурами данных, если заранее известно, что добавлений объекта в структуру будет много, а удалений почти не будет, пока вся структура не будет уничтожена.

Класс StackAllocator должен быть STL-совместимым, то есть позволять использование в качестве аллокатора для стандартных контейнеров. Должны быть реализованы методы:

- Конструктор по умолчанию, конструктор копирования, деструктор;
- Методы allocate, deallocate, а также rebind.

Проверьте себя: напишите тестирующую функцию, которая создает std::list и выполняет над ним последовательность случайных добавлений/удалений элементов. Если вы все сделали правильно, то std::list<int, StackAllocator<int>>, скорее всего, будет работать быстрее, чем std::list<int, std::allocator<int>>.

Часть 2.

Напишите шаблонный класс XorList - реализацию структуры данных, которая умеет все то же, что и обычный list, с той же асимптотикой, но экономнее расходует память: помимо полезной информации, каждая нода требует лишь sizeof(Node*) дополнительной памяти, а не 2*sizeof(Node*), как в обычном list'е. Общее описание этой структуры данных можно найти в Википедии.

У класса должно быть два шаблонных параметра: тип данных, хранящийся в списке, и тип используемого аллокатора. Должно быть реализовано следующее (полужирным шрифтом выделены пункты, необходимые в первую очередь - при проверке им будет уделяться особое внимание):

- Конструкторы:
  explicit XorList(const Allocator& alloc = Allocator());
  XorList(size_type count, const T& value = T(), const Allocator& alloc = Allocator());
- Конструктор копирования, **конструктор перемещения**, деструктор, копирующий и **перемещающий** операторы присваивания;
- Метод size(), работающий за O(1);
- Методы push_back, push_front, pop_back, pop_front, **корректно работающие как с lvalue, так и с rvalue-ссылками**;
- Двунаправленные итераторы (константные итераторы писать необязательно);
- **Правильное выделение и освобождение памяти через аллокатор (должен поддерживаться как std::allocator, так и StackAllocator);**
- Методы insert_before(iterator, const T&), **insert_before(iterator, T&&)**, insert_after(iterator, const T&), **insert_after(iterator, T&&)**, erase(iterator) - для удаления и добавления одиночных элементов в список.

Проверьте себя еще раз: выполните последовательность случайных добавлений-удалений элементов в XorList<int, StackAllocator<int>>. Работает ли это быстрее, чем для XorList<int, std::allocator<int>>?

Задача 9 (была в 2017г). Умные указатели.

Хотим предложить написать умные указатели. Это лучшее средство от памяти и ее утечек. Там, где есть умные указатели, никогда не утечет память. Очень хорошо помогает для динамической памяти. Цена указателя – 4 балла. На 12 баллов – 3 штуки.

Напишите класс UniquePtr – упрощенный аналог std::unique_ptr.
Должны быть реализованы:

- Конструктор из c-style указателя, конструктор перемещения, перемещающий оператор присваивания, деструктор.
- Оператор *, оператор ->, метод get().
- Методы release, reset, swap.

Напишите класс SharedPtr – упрощенный аналог std::shared_ptr.
Должны быть реализованы:

- Конструктор из c-style указателя, конструктор копирования, конструктор перемещения, копирующий и перемещающий операторы присваивания, деструктор.
- Конструктор из WeakPtr – если реализован WeakPtr, см. далее.
- Оператор *, оператор ->, метод get().
- Метод use_count, возвращающий число SharedPtr'ов, указывающих на этот же объект.
- Методы reset и swap.

Напишите класс WeakPtr – упрощенный аналог std::weak_ptr. В нем реализуйте:

- Конструктор из SharedPtr, конструктор копирования, конструктор перемещения, копирующий и перемещающий операторы присваивания, оператор присваивания SharedPtr, деструктор.
- Метод expired() и метод use_count, возвращающий число SharedPtr'ов, указывающих на этот же объект.
- Метод lock(), возвращающий SharedPtr.
- Методы reset и swap.

Задача 10 (была оба года одинаковая). Tuple

Напишите шаблонный класс с переменным количеством аргументов - Tuple (кортеж), обобщение класса std::pair, простенький аналог std::tuple из C++11.

Класс должен обладать следующим набором методов:

- Конструктор по умолчанию, который инициализирует все элементы кортежа значениями по умолчанию;
- Конструктор из набора аргументов, являющихся const lvalue-ссылками;
- Конструктор из набора аргументов, являющихся универсальными ссылками;
- Конструкторы копирования и перемещения, операторы присваивания (copy и move), деструктор;
- Метод swap, меняющий местами значения двух кортежей при условии, что у них одинаковые наборы шаблонных аргументов.

Помимо этого, нужно реализовать функции, не являющиеся членами класса:

- Функция makeTuple, создающая новый Tuple с нужными типами по данному набору аргументов-объектов.
- Функция get с шаблонным параметром size_t i, которая принимает tuple и возвращает ссылку на i-й элемент кортежа. Причем ссылка должна быть того же вида, что была и ссылка на принятый tuple (lvalue, const lvalue или rvalue).
- Функция get с шаблонным параметром T, которая принимает tuple и возвращает ссылку на тот элемент кортежа, который имеет тип T. Причем ссылка должна быть того же вида, что была и ссылка на принятый tuple (lvalue, const lvalue или rvalue). Если в кортеже несколько элементов типа T, никаких требований на поведение функции не налагается (можно выдавать ошибку компиляции, можно не выдавать и делать что угодно на этапе выполнения).
- Функция tupleCat, которая возвращает кортеж, являющийся конкатенацией нескольких кортежей, переданных в качестве аргументов.
- Операторы сравнения для Tuple, сравнивающие кортежи лексикографически.

Задача 11 (версия 2017г). Effective Modern Dynamic Programming

У маленького мальчика Илюши было k клетчатых досочек размеров M_i на N_i, где M_i <= 6, N_i <= 100. Дома по вечерам, когда все взрослые смотрели программу “Время”, он любил делать замощения этих досочек доминошками размера 1x2. Однажды ему пришел в голову вопрос: а сколькими различными способами он может замостить каждую из этих досочек по модулю 1000000007?

Уже тогда ощущая себя могучим программистом, он с азартом бросился к компьютеру и начал писать программу, которая бы дала ответ на столь животрепещущий вопрос. Он написал следующее:

```
const int MODULUS = 1000000007;

#include “ahalaimahalai”
#include <iostream>

int main() {
        std::cout << AhalaiMahalai<M_1, N_1>::value << ‘\n’;
        std::cout << AhalaiMahalai<M_2, N_2>::value << ‘\n’;
        ...
        std::cout << AhalaiMahalai<M_k, N_k>::value << ‘\n’;
}
```

То есть на самом деле внутри функции main() он написал k строк кода - по одной строке для каждой пары M_i, N_i, где вместо M_i и N_i подставил конкретные числа (нам они, увы, неизвестны). В остальном его программа выглядела именно так, как написано выше.

Увидев это, к Илюше подошел дедушка и сказал: “Глупый ты, глупый мальчик Илюша! Ничего ты такой программой посчитать не сможешь! В хороших, годных программах обязательно есть слово define и слово goto. А в твоей программе нет даже ветвлений (if, switch, тернарный оператор), даже циклов (for, while, do). Так что ничего у тебя не получится, если ты, конечно, сам не посчитаешь ответы для всех M_i, N_i каким-то другим способом и не запишешь эти готовые ответы в программу. Эх, Илюша, не умеешь ты программировать”.

Илюша страшно обиделся и сказал: “Нет, дедушка! Это ты ничего не понимаешь! *Эффективные и современные* программы на C++ обходятся без всего того, что ты назвал! И я тебе это докажу!”

Но доказать так и не сумел. И к кому бы вы думали он обратился за помощью? Конечно, к Вам.

Напишите код файла ahalaimahalai так, чтобы Илюшина программа выводила желаемое, но при этом файл бы не содержал ничего того, что перечислил дедушка. Более формально:

- Запрещены ключевые слова if, for, while, do, switch, goto;
- Запрещено использование тернарного оператора;
- Запрещена директива препроцессора define;
- Запрещено использовать любые литералы, кроме 0 и 1, а размер файла с кодом должен не превышать 5 КБ (иначе дедушка скажет “ну, это ты сам посчитал, а не твоя программа, - это каждый может”).

Замечание. Когда вы будете тестировать свою программу, скорее всего, вам понадобится при компиляции добавить параметр -ftemplate-depth=20000.

Задача 11 (версия 2018г). Шаблонное мышление

У маленького мальчика Илюши, помимо замощения клетчатых досочек доминошками во время просмотра взрослыми программы “Время”, было и еще одно увлечение. Когда все уходили на работу и они с дедушкой оставались дома одни, они любили играть в следующую игру. Вначале дедушка доставал из мешка некоторое количество конфет и раскладывал их по нескольким кучкам. После этого ходы делались по очереди. За ход можно было съесть любое количество конфет из любой кучки, а проигрывал тот, кому не оставалось возможности сделать ход. Кто будет ходить первым, выбирал Илюша. Да вот беда: умный дедушка всегда обыгрывал маленького мальчика. Конечно же, Илюше это надоело и он, воодушевленный своими предыдущими успехами в программировании, решил создать программу для определения того, как играть в данную игру оптимально.

Он сел за компьютер и написал такой код:

```
#include “ahalaimahalai”
#include <iostream>

int main() {
  constexpr int who = AhalaiMahalai<N_1, N_2, …, N_k>::who;
  constexpr int whence = AhalaiMahalai<N_1, N_2, …, N_k>::whence;
  constexpr int how = AhalaiMahalai<N_1, N_2, …, N_k>::how;
  std::cout << who << ‘ ‘ << whence << ‘ ‘ << how;
}
```

(Здесь вместо N_1, …, N_k были подставлены конкретные целые числа, равные изначальному количеству конфет соответственно в 1-й, …, k-й кучках. Нам они, увы, неизвестны; известно лишь, что каждое из них не превосходило 1000000, а число k не превосходило 10.)

Ответ на Илюшину задачу должен был состоять из трех чисел. Первое число - who - это номер игрока, обладающего выигрышной стратегией (1 или 2). Если who==1, то в переменных whence и how должен быть записан его первый выигрышный ход: whence - номер кучки, из которой нужно брать конфеты (в 1-индексации), how - сколько конфет нужно из нее взять. Если же who==2, то в переменных whence и how должны быть записаны нули.

Наблюдательный дедушка, увидев это, подошел к Илюше и грустно вздохнул: “Э-хе-хе-хе-хе… Разве ж какая-то маленькая программа вроде этой может выиграть в такой сложной математической игре? Шаблонно ты мыслишь, Илюша, шаблонно. С таким *шаблонным мышлением* не стать тебе математиком. Да и вообще, я в своей жизни повидал много сложных программ, а в твоей нет ни слова define, ни слова goto, ни даже ветвлений (if, switch, тернарный оператор), даже циклов (for, while, do). Эх, Илюша, никудышный ты программист, а математик - и подавно”.

Илюша страшно обиделся и сказал: “Нет, дедушка! Сам ты ничего не понимаешь! Времена уже другие, и никакая это не сложная математическая задача! В современном мире даже *шаблонного мышления* вполне достаточно, чтобы тебя обыграть! И я тебе это докажу!”

Но доказать так и не сумел. И к кому бы вы думали он обратился за помощью? Конечно, к Вам.

Напишите код файла ahalaimahalai так, чтобы Илюшина программа выводила желаемое, но при этом файл бы не содержал ничего того, что перечислил дедушка. Более формально:

- Запрещены ключевые слова if, for, while, do, switch, goto, а также функции setjmp и longjmp;
- Запрещено использование тернарного оператора;
- Запрещена директива препроцессора define (кроме случаев использования её для избежания двойного включения хэдера), а также все другие директивы препроцессора кроме #include и #pragma once;
- Запрещено использовать любые литералы, кроме 0 и 1, а размер файла с кодом должен не превышать 5 КБ (иначе дедушка скажет “ну, это ты сам посчитал, а не твоя программа, - это каждый может”).