ФПМИ МФТИ, весна 2020
ООП 1 курс, основной поток. Задания по C++

Задача 1. Длинная арифметика

Дедлайн: 23.03.2020 09:00 мск

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

Часть 1

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. Умножение должно работать **за о-малое от n^2.**
- унарный минус, префиксный и постфиксный инкремент и декремент. Префиксный инкремент и декремент должны работать за O(1) в среднем.
- 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);
- преобразование в bool, когда это надо (должно работать в условных выражениях).
- опционально - литеральный суффикс bi для написания литералов типа BigInteger, см. справку здесь https://en.cppreference.com/w/cpp/language/user_literal

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

Часть 2

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 строковое 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.h. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Задача 2. 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.

В вашем файле должна отсутствовать функция main(), а сам файл должен называться geometry.h. В качестве компилятора необходимо выбирать Make. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Задача 3. List

В этой задаче вам предлагается реализовать полноценный STL-подобный контейнер (упрощенный относительно настоящего STL, разумеется, но приближенный к нему). Напишите шаблонный класс List<T> - двусвязный список. Нужно правильно поддержать move-семантику, а также итераторы. (Для знатоков: аллокаторы поддерживать не надо.) Должно быть реализовано следующее:

- Конструкторы: List() - пустой лист; List(size_t count, const T& value = T()) - лист из count элементов, изначально равных value каждый.
- Конструктор копирования, конструктор перемещения, деструктор, копирующий и перемещающий операторы присваивания;
- Метод size(), работающий за O(1);
- Методы front() и back(), позволяющие получить ссылку на начальный и конечный элемент листа соответственно;
- Метод clear(), который опустошает лист, и метод empty(), проверяющий, является ли лист пустым;

- Двунаправленные итераторы, как константные, так и неконстантные, а также reverse-итераторы. При этом нужно не допустить копипасты всего кода итераторов; о том, как этого добиться, см. например здесь https://stackoverflow.com/questions/2150192/how-to-avoid-code-duplication-implementing-const-and-non-const-iterators Reverse-итераторы можно поддержать с помощью std::reverse_iterator.
- Методы begin(), end(), cbegin() и cend(), реализованные так, чтобы корректно работал код for (const auto& value : lst) для обхода вашего контейнера.
- Методы insert(const_iterator it, const T&) и insert(const_iterator it, T&&) для вставки элементов в контейнер по итератору, а также insert(const_iterator it, InputIter frist, InputIter last) для вставки диапазона.
- Метод erase по итератору, а также erase от диапазона итераторов.
- Методы push_back, pop_back, push_front и pop_front. Разумеется, методы push_back и pop_back должны корректно обрабатывать как rvalue, так и lvalue.
- Методы emplace, emplace_front и emplace_back, позволяющие сконструировать и вставить по итератору элемент типа T из переданных аргументов, не вызывая для T ни конструктор копирования, ни move-конструктор.
- Метод reverse, разворачивающий лист в обратном порядке. Время работы O(n).
- Метод unique, который удаляет из контейнера все последовательные дубликаты, то есть все такие элементы x, что x равен предыдущему элементу листа. (См. документацию std::unique для лучшего понимания.) Метод sort, так уж и быть, реализовывать не надо.

Вставка в любое место и удаление из любого места одного элемента должны работать за гарантированное O(1) относительно количества элементов. Итераторы на имеющиеся в листе элементы должны не инвалидироваться после любой вставки и после удаления, если это удаление не затрагивает элемент под данным итератором. Для создания листа из элементов типа T тип T не обязан иметь ни конструктор по умолчанию, ни копирующий конструктор. Наличия лишь конструктора перемещения у T должно быть достаточно, чтобы использовать этот тип в контейнере.

В вашем файле должна отсутствовать функция main(), а сам файл должен называться list.h. Никакими стандартными контейнерами в своем коде пользоваться нельзя. Ваш код будет вставлен посредством #include в программу, содержащую тесты.

Задача 4. Tuple

В этой задаче вам предлагается освоить продвинутое использование шаблонов, в том числе шаблоны с переменным количеством параметров и некоторые compile-time вычисляемые фичи на шаблонах.

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.

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