Лабораторная работа №9
«Перегрузка операций»

1. Цель работы

Данная работа предназначена для изучения возможностей языка C++, обеспечивающих применение знаков операций к объектам пользовательских типов.

2. Исходные данные

Информация об объявлении и специализации шаблонов дана в лекционном курсе.

3 Задание

Согласно выбранному из таблиц 1–26 описанию требуется составить шаблон класса, перегрузив указанные операции.

	Формулировка задачи	Студент	Группа
1	Bool<T> – булевская формула с операциями И, ИЛИ и НЕ, в которой в качестве имён переменных выступают значения типа T (это могут быть, например, строки). Операции, перегружаемые для Bool<T>: 1. «∗ » и «+» – объединение двух формул с помощью операции И и ИЛИ, соответственно; 2. «!» – формирование новой формулы, являющейся отрицанием текущей формулы; 3. «( )» – вычисление значения формулы по ассоциативному массиву, отображающему имена переменных в булевские значения. Конструктор класса Bool<T> должен принимать имя переменной и формировать формулу, состоящую из обращения к этой единственной переменной. Подразумевается, что более сложные формулы можно собрать с помощью перегруженных операций.		
2	Word<Letter> – «слово», представляющее собой последовательность «букв», представленных объектами некоторого класса Letter. Требования к классу Letter: наличие унарного «−» и операции «==» таких, что для любой «буквы» x справедливо равенство −(−x) == x. Операции, которые должны быть перегружены для Word<Letter>: 1. «+» – конкатенация двух «слов», после которой выполняется «редукция» результирующего «слова», а именно – пары соседних «букв» x и y такие, что x == −y, взаимно уничтожаются до тех пор, пока в результирующем «слове» таких пар не останется; 2. унарный «−» – переворачивание слова с одновременной заменой всех «букв» на обратные им «буквы»; 3. «==», «!=». Класс Word<Letter> должен иметь два конструктора: конструктор без параметров создаёт пустое «слово», и конструктор, имеющий параметр Letter, создаёт «слово», состоящее из единственной «буквы».		

3 SortedSeq<Symbol> – последовательность отсортированных по возрастанию «символов», представленных объектами некоторого класса Symbol. Требования к классу Symbol: наличие операций «==» и «<», а также операции «~», некоторым образом вычисляющей так называемый «обратный символ». Для любого «символа» x должно быть справедливо, что ~(~x) == x. Операции, которые должны быть перегружены для SortedSeq<Symbol>:

1. «+=» – добавление «символов» другой последовательности, сопровождаемое «редукцией», представляющей собой удаление из последовательности всех пар взаимнообратных «символов»;
2. «~» – замена всех «символов» в последовательности на обратные им «символы»;
3. «<», «<=», «>» и «>=» – лексикографическое сравнение последовательностей;
4. «==», «!=».

Класс SortedSeq<Symbol> должен иметь два конструктора: конструктор без параметров создаёт пустую последовательность, и конструктор, имеющий параметр Symbol, создаёт последовательность, состоящую из единственного «символа».

4 PtrStack<T> – стек указателей на структуры типа T. Операции, которые должны быть перегружены для PtrStack<T>:

1. «<<» – добавление указателя на вершину стека (push);
2. «>>» – снятие указателя с вершины стека (pop);
3. empty – проверка на пустоту стека;
4. унарный «* » – возвращает значение, адрес которого лежит на вершине стека;
5. «−>» – осуществляет доступ к полям структуры, адрес которой лежит на вершине стека.

5 Change<T> – идеальный «размен» суммы денег минимальным количеством рублёвых купюр и монет, в котором количества купюр и монет выражаются целочисленным типом T.

Операции:

1. «+» – объединение двух разменов;
2. «−» – пересечение двух разменов;
3. «( )» – получение количества купюр или монет заданного номинала.

6 Element<T> – элемент леса непересекающихся множеств с полезной нагрузкой типа T. Операции, которые должны быть перегружены для Element<T>:

1. унарный «* » – получение значения, лежащего внутри элемента;
2. «==», «!=» – определение, принадлежат ли два объекта Element<T> одному дереву;
3. унарный «!» – возвращает корень дерева;
4. «<<» – объединение деревьев, которым принадлежат два объекта Element<T>.

7 FibNum<T> – целое число типа T, представленное последовательностью нулей и единиц в фибоначчиевой системе счисления. Операции, которые должны быть перегружены для FibNum<T>:

1. префиксный и постфиксный «++» – прибавление единицы;
2. «&» – возвращает наибольшее число, составленное из общих для двух чисел фибоначчиевых слагаемых;
3. «==», «!=», <», «<=», >», «>=»;
4. «T()» – преобразование к типу T.

8 Parcel<T,N> – последовательность терминальных и нетерминальных символов, которая получается в процессе вывода предложения некоторого формального языка в соответствии с правилами контекстно-свободной грамматики этого языка. Терминальные символы обозначаются значениями типа T, а нетерминальные – значениями типа N. Подразумевается, что типы T и N различаются. Операции, которые должны быть перегружены для Parcel<T,N>:

1. «+» – выполняет конкатенацию двух последовательностей с порождением новой последовательности;
2. «+=» – имеет три перегруженные версии:

(a) добавляет другую последовательность в конец текущей;
(b) добавляет терминальный символ в конец текущей последовательности;
(c) добавляет нетерминальный символ в конец текущей последовательности;

3. «( )» – имея два параметра – нетерминал x и последовательность p, порождает на основе текущей последовательности новую последовательность, в которой самое левое вхождение x заменено на p.

Конструктор Parcel<T,N> должен порождать пустую последовательность.

9 SparseMatrix<T, M, N> – разреженная матрица размера M × N с элементами числового типа T. Представление матрицы должно быть оптимизировано таким образом, чтобы нулевые элементы по возможности не хранились. Операции, которые должны быть перегружены для SparseArray<T>:

1. «( )» – получение ссылки на элемент матрицы (принимает в качестве параметров координаты элемента);
2. «+», «∗ » – сложение и умножение матриц, умножение на значение типа T;
3. «==», «!=».

10 Palindrome<T> – палиндром, составленный из значений типа T. Операции:

1. «−» – удаление из палиндрома всех вхождений указанного элемента (например: abcdcba − b = acdca);
2. «+» – добавление к палиндрому нового элемента (новый элемент добавляется слева и справа от середины палиндрома: aa + b = abba, aca + b = abcba);
3. «!» – удаление из палиндрома элементов, расположенных непосредственно слева и справа от середины (например, !abcba = aca);
4. «[ ]» – получение i-го элемента палиндрома.

В классе должно быть два конструктора: один создаёт пустой палиндром, а другой – палиндром длины 1.

11 Seq<T> – последовательность отсортированных по возрастанию значений типа T. (Подразумевается, что для типа T определены операции «<» и «==».) Операции:

1. «+» – слияние двух последовательностей в одну;
2. «∗ » – пересечение двух последовательностей (в результирующей последовательности остаются только элементы, общие для двух пересекаемых последовательностей);
3. «−» – разность последовательностей (результирующая последовательность содержит элементы, присутствующие в первом операнде и отсутствующие во втором);
4. «[ ]» – получение i-го элемента последовательности.

12 Curve<T> – кривая на плоскости, заданная функцией y = f (x), где x и y – числа с плавающей точкой типа T. Конструктор кривой принимает булевское значение и, в зависимости от этого значения, порождает либо кривую y = sin x, либо кривую y = cos x. Операции:

1. «+» – сумма двух кривых: $y = y_1 + y_2 = f_1(x) + f_2(x)$;
2. «−» – разность двух кривых: $y = y_1 - y_2 = f_1(x) - f_2(x)$;
3. «∗ » – умножение кривой на число: $y = ky_1 = kf_1(x)$;
4. «−» – изменение знака (унарный минус): $y = -y_1 = -f_1(x)$;
5. «!» – дифференцирование: $y = dy_1 / dx$;
6. «( )» – вычисление y для указанного x.

13 Set<Letter> – множество «букв», представленных объектами некоторого класса Letter. (Требования к классу Letter: наличие операции «==», а также операции «!», некоторым образом вычисляющей так называемую «обратную букву». Для любой «буквы» x должно быть справедливо, что !(! x) == x.) Операции:

1. «&=» – добавление «букв» другого множества, сопровождаемое «редукцией», представляющей собой удаление из множества всех пар

взаимнообратных «букв»;
2. «&=» – добавление в множество отдельной «буквы», также сопровождаемое «редукцией»;
3. «!» – замена всех «букв» в множестве на обратные им «буквы»;
4. «==», «!=».

Класс Set<Letter> должен иметь конструктор без параметров, который создаёт пустое множество.

14 Series<T> – положительный числовой ряд

$$\sum_{i=i0}^{\infty} a_i,$$

где $a_i \geq 0$ – числа типа T, и $i_0 \geq 0$. Конструктор ряда принимает в качестве параметра i_0 и указатель на функцию, вычисляющую i-тый член ряда. Операции:

1. «∗ » – умножение на число;
2. «+» – сумма двух рядов;
3. «( )» – получение i-го члена ряда (если $i < i_0$, операция возвращает 0).

15 Shape<T> – множество точек в пространстве T × T, задающих некоторую геометрическую фигуру. Операции, перегружаемые для Shape<T>:

1. «+» и «−» – объединение и разность двух множеств;
2. «( )» – проверка принадлежности точки множеству.

У класса Shape<T> должно быть два конструктора:

1. первый конструктор принимает координаты нижней левой и верхней правой вершин прямоугольника, каждая сторона которого параллельна одной из осей координат, и порождает множество точек, принадлежащих этому прямоугольнику;
2. аналогично, второй конструктор порождает множество точек круга по координатам центра и радиусу.

16 Matrix<T, N> – антидиагональная матрица размера N × N с элементами типа T. (Все

элементы антидиагональной матрицы, кроме лежащих на диагонали, идущей от нижнего левого угла до верхнего правого угла, равны нулю. Матрица должна быть представлена только числами, лежащими на диагонали.) Операции:

1. «+» – сумма двух матриц;
2. «∗ » – произведение двух матриц;
3. «[ ]» – получение значения элемента, расположенного на i-той строке в j-том столбце.

17 Divs<T, N> – последовательность степеней простых делителей, на которые раскладывается некоторое натуральное число типа T, простые делители которого не превышают N. Операции, которые должны быть перегружены для Divs<T, N>:

1. «∗ =» – домножение на число, представленное другой последовательностью;
2. «∗ » – умножение на число, представленное другой последовательностью;
3. «&» – наибольший общий делитель числа, представленного текущей последовательностью, и числа, представленного другой последовательностью;
4. «==», «!=», «<», «<=», >», «>=»;
5. «T()» – преобразование к типу T.

18 Curve<T> – кривая на плоскости, заданная функцией y = f (x), где x и y – числа с плавающей точкой типа T. Конструктор кривой порождает кривую y = exp (x). Операции:

1. «+» – сумма двух кривых: $y = y_1 + y_2 = f_1 (x) + f_2 (x)$;
2. «−» – разность двух кривых: $y = y_1 − y_2 = f_1 (x) − f_2 (x)$;
3. «∗ » – умножение кривой на число: $y = ky_1 = kf_1 (x)$;
4. «−» – изменение знака (унарный минус): $y = −y_1 = −f_1 (x)$;
5. «!» – дифференцирование: $y = dy_1 / dx$; (TODO: ничего не делает!)
6. «( )» – вычисление y для указанного x.

19 MergingMap<K,V> – ассоциативный массив,

отображающий ключи типа K в значения типа V. Требование к классу V:

1. наличие конструктора, принимающего в качестве параметра целое число и в случае, если это число равно 0, порождающего некоторое значение, играющее для типа V роль «нуля»;
2. наличие бинарной операции «+», позволяющей каким-то образом получать «сумму» двух значений (подразумевается, что эта операция является ассоциативной, и вышеупомянутый «ноль» является относительно неё нейтральным элементом).

Отметим, что примитивные числовые типы языка C++ удовлетворяют требованиям к классу V и могут быть использованы для проверки работоспособности класса MergingMap<K,V>. Операции, которые должны быть перегружены для MergingMap<K,V>:

1. «[ ]» – возвращает ссылку на значение, связанное с указанным ключом (в случае отсутствия в ассоциативном массиве словарной пары с указанным ключом такая пара автоматически добавляется в массив, причём её значением становится «ноль»);
2. «+» – объединение двух ассоциативных массивов A и B, результатом которого является ассоциативный массив, содержащий такие словарные пары <k, v>, что k является ключом хотя бы в одном из объединяемых массивов, а v = A[k] + B[k].
3. 3. «==», «!=».

Конструктор класса MergingMap<K,V> должен принимать в качестве параметра целое число и создать пустой ассоциативный массив. Параметр конструктора может либо игнорироваться, либо восприниматься как прогнозируемый размер ассоциативного массива для более эффективного выделения памяти. Работоспособность шаблона MergingMap следует проверить для случаев MergingMap<string, int> и MergingMap<string, MergingMap<string, int>>.

20 NumSet<T> – множество чисел, имеющих тип T. Конструктор класса NumSet<T> принимает в качестве параметров границы интервала [a, b] и формирует множество, содержащее числа, принадлежащие этому интервалу. Подразумевается, что более сложные множества могут быть сконструированы с помощью перегруженных операций. Операции, перегружаемые для NumSet<T>:

1. «∗ » и «+» – пересечение и объединение двух множеств, соответственно;
2. «( )» – проверка принадлежности значения типа T множеству.

21 Program<Statement, Env> – последовательность «команд», представленных объектами некоторого класса Statement, которые можно выполнять в окружении, заданном некоторым классом Env. Подразумевается, что окружение содержит данные, необходимые для работы «команд». Требование к классу Statement: наличие операции «()», которая принимает в качестве параметра ссылку на объект класса Env, выполняет «команду» и возвращает номер команды, на которую должно быть передано управление, или −1, если данная команда завершает закодированную последовательностью программу. Операции, которые должны быть перегружены для Program<Statement, Env>:

1. «<<» и «>>» – добавление новой команды в конец или в начало последовательности, соответственно (эти операции возвращают ссылку на текущую последовательность);
2. «( )» – выполнение последовательности команд до тех пор, пока некоторая команда не возвратит −1 (принимает в качестве параметра ссылку на объект класса Env);
3. «+» – конкатенация двух последовательностей;
4. «==», «!=».

Класс Program<Statement, Env> должен иметь конструктор без параметров, который создаёт пустую последовательность.

22 IntSet<T> – множество целых чисел типа T,

конструируемое на основе предиката, определяющего принадлежность числа множеству. Конструктор множества принимает в качестве параметра указатель на функцию, принимающую число и возвращающую true или false в зависимости от того, принадлежит число множеству или не принадлежит. Операции:

1. «+» – объединение двух множеств;
2. «* » – пересечение двух множеств;
3. «!» – дополнение множества;
4. «* » – умножение всех элементов множества на целое число;
5. «/» – целочисленное деление всех элементов множества на целое число;
6. «( )» – проверка принадлежности числа множеству.

23 LinearProgram<Statement, Env> – последовательность «команд», представленных объектами некоторого класса Statement, которые можно выполнять в окружении, заданном некоторым классом Env. Подразумевается, что окружение содержит данные, необходимые для работы «команд». Требование к классу Statement: наличие операции «()», которая принимает в качестве параметра ссылку на объект класса Env, выполняет «команду» и возвращает булевское значение, сообщающее об успешности выполнения команды. Операции, которые должны быть перегружены для LinearProgram<Statement, Env>:

1. «+=» – добавление новой команды в конец последовательности;
2. «( )» – выполнение последовательности команд до первой команды, выполненной неуспешно, или до конца (принимает в качестве параметра ссылку на объект класса Env, возвращает bool);
3. «* » – конкатенация двух последовательностей;
4. «==», «!=».

Класс LinearProgram<Statement, Env> должен иметь конструктор без параметров, который создаёт пустую последовательность.

24 LazyMatrix<T> – матрица с элементами типа T неопределённого размера, растущая по мере надобности. Требование к типу T: наличие

конструктора по умолчанию. Операции, которые должны быть перегружены для LazyMatrix<T>:

1. «( )» – получение ссылки элемент с индексами i и j (размер матрицы должен быть автоматически увеличен, если i или j выходят за её пределы);
2. m и n – возвращают количество строк и столбцов, соответственно;
3. «!» – транспонирование матрицы (возвращает новую матрицу).
4. «==», «!=».

25 PtrQueue<T> – очередь указателей на структуры типа T, реализованная через кольцевой буфер. Операции, которые должны быть перегружены для PtrQueue<T>:

1. «<<» – добавление указателя на в очередь (enqueue);
2. «!» – вытаскивание указателя из очереди (dequeue);
3. empty – проверка на пустоту очереди;
4. унарный «* » – возвращает значение, адрес которого лежит в начале очереди (туда указывает head);
5. «−>» – осуществляет доступ к полям структуры, адрес которой лежит в начале очереди.

26 LazyArray<T> – массив с элементами типа T неопределённого размера, растущий по мере надобности. Должен быть реализован через класс vector. Требование к типу T: наличие конструктора по умолчанию. Операции, которые должны быть перегружены для LazyArray<T>:

1. «[ ]» – получение ссылки на i-тый элемент массива (размер массива должен быть автоматически увеличен, если i выходит за пределы массива);
2. «( )» – формирование подмассива, содержащего элементы с индексами из указанного диапазона (принимает в качестве параметров границы диапазона, возвращает новый LazyArray<T>).
3. «==», «!=».

27 SparseArray<T> – разреженный массив, отображающий неотрицательные целые числа в значения типа T. Массив должен быть

реализован через хэш-таблицу. Требование к типу T: наличие конструктора по умолчанию (т. к. разреженный массив должен уметь создавать значения типа T). Операции, которые должны быть перегружены для SparseArray<T>:

1. «[ ]» – получение ссылки на i-тый элемент массива;
2. «( )» – формирование подмассива, содержащего элементы с индексами из указанного диапазона (принимает в качестве параметров границы диапазона, возвращает новый SparseArray<T>).
3. «==», «!=».

28 Function<A, R, F> – «обёртка» вокруг функции (или объекта класса с перегруженной операцией «()») типа F, принимающей параметр типа A и возвращающей значение типа R. Операции, перегружаемые для Function<A, R, F>:

1. «( )» – вызов функции (принимает значение типа A и возвращает значение типа R);
2. «∗ » – композиция двух функций.

Конструктор класса Function<A, R, F> должен принимать параметр типа F. Указание: для решения задачи можно составить шаблон класса AbstractFunction<A, R> и сделать класс Function<A, R, F> наследником AbstractFunction<A, R>. Тогда композицию функций $f : A \to B$ и $g : B \to R$ можно будет представить объектом класса Composition<A, B, R>, также являющегося наследником AbstractFunction<A, R>.

29 Ring<T> – кольцевой двунаправленный список с ограничителем, в узлах которого хранятся значения типа T. Операции, которые должны быть перегружены для Ring<T>:

1. «+» – конкатенация двух списков;
2. «<<» и «>>» – добавление нового элемента в конец или начало списка, соответственно;
3. «∗ » – поиск элемента, содержащего указанное значение (возвращает bool);
4. «/=» – удаление элемента, содержащего указанное значение;
5. «==», «!=».

Класс Ring<T> должен иметь конструктор без параметров, который создаёт пустой список.

30 Palindrome<T> – палиндром, составленный из значений типа T. Операции:

1. «+» – добавление к палиндрому нового элемента (новый элемент добавляется в начало палиндрома и в его конец: bb + a = abba);
2. «!» – удаление из палиндрома крайних элементов (например, !abba = bb);
3. «/» – удаление из левого операнда элементов, принадлежащих правому операнду (например: abcdcba/bdb = acca);
4. «[ ]» – получение ссылки на i-ый элемент палиндрома (учесть возможность изменения значения элемента через эту ссылку).

В классе должно быть два конструктора: один создаёт пустой палиндром, а другой – палиндром, состоящий из единственного элемента.

31 Seq<T> – последовательность значений типа T. (Подразумевается, что для типа T определена операция «==».) Операции:

1. «+» – конкатенация двух последовательностей;
2. «/» – деление последовательности на k равных по длине частей (в результате получается вектор последовательностей, длины которых различаются не более, чем на 1);
3. «!» – удаление из последовательности дублирующихся значений (в результате формируется новая последовательность);
4. «[ ]» – получение i-го элемента последовательности.

32 F<N, T> – арифметическая формула с операциями сложения, вычитания, умножения и деления, в которой в качестве имён переменных выступают значения типа N (это могут быть, например, строки), а в качестве констант – значения числового типа T. Операции, перегружаемые для F<N, T>:

1. «+», «−», «∗ » и «/» – объединение двух

формул с помощью соответствующих операций;
2. унарный «−» – формирование новой формулы, являющейся отрицанием текущей формулы;
3. «( )» – формирование новой формулы, которое заключается в выполнении двух действий:
   (a) подстановка в текущую формулу значений переменных, взятых из ассоциативного массива, отображающего имена переменных в значения типа T;
   (b) упрощение формулы, достигаемое за счёт вычисления значений подвыражений, не содержащих имена переменных;
4. «<<» – вывод формулы в поток вывода.

У класса F<N, T> должно быть два конструктора: первый принимает имя переменной и формирует формулу, состоящую из обращения к этой единственной переменной; а второй принимает константу и формирует формулу, состоящую из этой константы. Подразумевается, что более сложные формулы можно собрать с помощью перегруженных операций.

33 Relation<T> – отношение на множестве значений типа T. Операции, которые должны быть перегружены для Relation<T>:

1. «+» – объединение двух отношений;
2. «* » – пересечение двух отношений;
3. «~» – транзитивное замыкание отношения;
4. «( )» – проверка принадлежности указанной пары значений отношению (имеет два параметра типа T, возвращает bool);
5. «==», «!=».

34 NNF<T> – булевская формула с операциями И, ИЛИ и НЕ, в которой в качестве имён переменных выступают значения типа T (это могут быть, например, строки). Формула должна находиться в так называемой нормальной форме отрицания (Negation Normal Form), в которой операция НЕ применяется только к значениям переменных. Операции, перегружаемые для NNF<T>:

1. «∗ » и «+» – объединение двух формул с помощью операции И и ИЛИ, соответственно;
2. «!» – формирование новой формулы, являющейся отрицанием текущей формулы (здесь надо не забыть о нормальной форме отрицания!);
3. «( )» – вычисление значения формулы по ассоциативному массиву, отображающему имена переменных в булевские значения;
4. «<<» – вывод формулы в поток вывода.

Конструктор класса NNF<T> должен принимать имя переменной и формировать формулу, состоящую из обращения к этой единственной переменной. Подразумевается, что более сложные формулы можно собрать с помощью перегруженных операций.

35 Curve<T> – кривая на плоскости, заданная функцией $\boldsymbol{r} = f(t)$, где − $\boldsymbol{r}$ – радиус-вектор с координатами типа T, а t – параметр, также задаваемый числом типа T. Конструктор траектории принимает в качестве параметра указатель на функцию f. Операции:

1. «+» – сумма двух кривых: $\boldsymbol{r} = \boldsymbol{r_1} + \boldsymbol{r_2} = f_1(t) + f_2(t)$;
2. «∗ » – умножение кривой на число: $\boldsymbol{r} = k\boldsymbol{r_1} = kf_1(t)$;
3. «( )» – вычисление радиус-вектора для указанного t.