Лабораторная работа №11
«Перегрузка операций»

Скоробогатов С.Ю.

23 мая 2016 г.

1 Цель работы

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

2 Исходные данные

...

3 Задание

3.1 Word<Letter>

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».

3.2 Seq<Element>

Seq<Element> – последовательность элементов, представленных объектами некоторого класса Element.

Требования к классу Element: наличие операции «!», некоторым образом вычисляющей «обратный» элемент, и операции «==» таких, что для любого элемента x справедливо равенство !(! x) == x.

Операции, которые должны быть перегружены для Seq<Element>:

1. «*» – конкатенация двух последовательностей, после которой выполняется «редукция» результирующей последовательности, а именно – пары соседних элементов x и y такие, что x == !y, взаимно уничтожаются до тех пор, пока в результирующей последовательности таких пар не останется;

2. «!» – переворачивание последовательности с одновременной заменой всех элементов на обратные им элементы;

3. «<<» и «>>» – добавление элемента в конец и в начало последовательности, соответственно, после которого выполняется «редукция»;

4. «==», «!=».

Класс Seq<Element> должен иметь конструктор без параметров, который создаёт пустую последовательность.

3.3 Word<Symbol>

Word<Symbol> – «слово», представляющее собой последовательность «символов», представленных объектами некоторого класса Symbol.

Требование к классу Symbol: наличие операции «==».

Операции, которые должны быть перегружены для Word<Symbol>:

1. «&» – конкатенация двух «слов», после которой выполняется «редукция» результирующего «слова», а именно – пары соседних «символов» x и y такие, что x == y, взаимно уничтожаются до тех пор, пока в результирующем «слове» таких пар не останется;

2. унарный «−» – переворачивание слова;

3. «+» – добавление «символа» в конец или в начало «слова», после которого выполняется «редукция» (если левый операнд – «символ», то он добавляется в начало второго операнда, который должен быть «словом»; если же левый операнд – «слово», то «символ» добавляется в его конец);

4. «==», «!=».

Класс Word<Symbol> должен иметь конструктор без параметров, который создаёт пустое «слово».

3.4 SortedSeq<Symbol>

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».

3.5 Set<Letter>

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, которые должны быть перегружены для Set<Letter>:

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.

3.6 LinearProgram<Statement, Env>

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.

3.7 Program<Statement, Env>

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.

3.8 Divs<T, N>

Divs<T, N> – последовательность степеней простых делителей, на которые раскладывается некоторое натуральное число типа T, не превышающее 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.

3.9 FibNum<T>

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.

3.10 Ring<T>

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.

3.11 Relation<T>

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. «==», «!=».

3.12 SparseArray<T>

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. «==», «!=».

3.13 SparseMatrix<T, M, N>

SparseMatrix<T, M, N> – разреженная матрица размера $M \times N$ 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. «==», «!=».

3.14 SparseSet<A, B>

SparseSet<A, B> – разреженное множество с целочисленными элементами, принадлежащими диапазону от A до B, реализованное через два массива sparse и dense.

Операции, которые должны быть перегружены для SparseSet<A, B>:

1. «+», «*» – объединение и пересечение двух множеств;
2. «<<» – добавление числа в множество (возвращает ссылку на текущее множество);
3. «>>» – удаление числа из множества (возвращает ссылку на текущее множество);
4. «()» – проверка принадлежности числа множеству (принимает число в качестве параметра, возвращает **bool**);
5. «==», «!=».

3.15 LazyArray<T>

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. «==», «!=».

3.16 PtrStack<T>

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.

3.17 Polyline<P>

Polyline<P> – ломаная линия, состоящая из точек типа P.
Требование к типу P: наличие метода dist, вычисляющего расстояние до другой точки.
Операции, которые должны быть перегружены для Polyline<P>:

1. «<<» и «>>» – добавление точки в конец или в начало ломаной, соответственно (операции возвращают ссылку на текущую ломаную);

2. « [] » – возвращает ссылку на i-тую точку ломаной;

3. count – возвращает количество точек ломаной;

4. «==», «!=», <», «<=», >», «>=» – ломаные сравниваются по длине.

3.18 (*) Function<A, R, F>

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, которые должны быть перегружены 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>.

3.19 MergingMap<K,V>

MergingMap<K,V> – ассоциативный массив, отображающий ключи типа K в начения 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 $\langle k, v \rangle$, что k является ключом хотя бы в одном из объединяемых массивов, а $v = A[k] + B[k]$.

3. «==», «!=».

Конструктор класса MergingMap<K,V> должен принимать в качестве параметра целое число и создаать 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**>>.

3.20 Parcel<T,N>

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.