Задание 2 для первого семестра 2 курса

Требуется разработать и реализовать параметризованный класс списочного типа. Класс должен содержать конструктор без параметров, конструктор копирования, другие конструкторы по смыслу решаемых задач.

Набор методов класса должен соответствовать идеологии работы с соответствующей схемой хранения данных, т.е. обеспечивать требуемую дисциплину доступа к данным (возможно, с некоторыми модификациями). В частности, должны быть методы для добавления, извлечения и удаления элементов, методы для прямого доступа к значениям элементов (в соответствии с заданной дисциплиной), методы опроса состояний, если это необходимо. Для контроля за некорректными ситуациями следует использовать механизм исключений (для фатальных отказов) или в некоторых случаях логические возвращаемые значения (для нефатальных).

Для перемещения по спискам и указания текущих позиций нужно реализовать различные итераторы (прямой, обратный, если он возможен, константный и т.п.).

Структура, задающая узел списка, реализуется как внутренний класс этого списка, члены этой структуры могут быть public. Все объекты самого списка помещаются в private секцию, а в public секции описываются только интерфейсные методы, определящие дисциплину работы с элементами данных.

Итераторы реализуются двумя способами (в разных вариантах заданий) — как внутренние классы и как внешние по отношению к основному классу.

Для класса должен быть переопределен оператор вывода <<, который в наглядной форме изображает состояние данного класса.

Для класса должен быть реализован оператор присваивания и также переопределены некоторые операции, например, сложение как объединение множеств элементов.

Как правило, для класса надо будет реализовать процедуры сортировки содержимого и поиска требуемого элемента в соответствии с некоторым внешне задаваемым критерием сравнения. Поиск возвращает итератор на найденный элемент.

После реализации собственного списочного класса, его работу надо сравнить с STL реализациями списочных контейнеров (std::list и/или std::forward_list или других подобных). Это означает, что надо придумать некоторую "задачу" для тестирования своей реализации и также решить эту задачу с использованием классов STL. Интересно сравнить время работы программы в случае вашей реализации и при использовании классов STL.

Подобная тестирующая задача может, например, заключаться в многократном добавлении и удалении элементов в списочную структуру, присваивании списочных структур друг в друга по некоторым правилам, автоматической проверке правильности полученного финального состочния контейнера.

Например, заполнить список последовательными значениями $1, \ldots, n$, проходом по списку удалить все нечетные значения, а четные поделить пополам, выполнить присваивание этого списка другому списку, переставить элементы списка в обратном порядке, проверить, что получившийся список содержит последовательные числа $n/2, \ldots, 1$ а исходный содержит числа $1, \ldots, n/2$. Отладку для малых n можно проводить визуально просматривая распечатку состояния списков. Для больших $n \sim 1000000$ для отладки надо придумать "задачу", в которой ответ проверяется автоматически. Скажем, в описанном выше примере в цикле значения из списка явно сравниваются с известными $1, \ldots, n/2$.

Дополнительные требования к формулировкам и реализациям конкретного задания обсуждаются в рабочем порядке.

Типы объектов

Релизация тестируется на нескольких различных типов хранимых объектов — "простых" и "сложных". Простые объекты — это примитивные числовые типы (int, double, char). Для них можно построить тесты на большое количество элементов с явной проверкой состояния списков после выполнения серии операций.

Более сложные — структурные типы вроде вектора или числовых классов из первого задания. Тут можно ограничиться небольшой размерностью и визуальной проверкой. Т.е. придется определить опреатор вывода и для этих классов.

Наконец, реализация должна работать для “рекурсивного” типа, т.е. стек стеков целых чисел, список списков векторов и т.п. (например, Stack<Stack<int> > и т.п.). Здесь проверяется возможность добавления или извлечения нетривиального класса в подобную списочную структуру.

Варианты структур данных

A. Динамический массив-аллокатор (подробности в лекциях). Реализация на основе списка блоков фиксированного размера. Операции захвата элемента и освобождения элемента.

B1. Однонаправленный список. Внешний итератор.

B2. Однонаправленный список. Внутренний итератор.

C1. Двунаправленный список. Внешний итератор.

C2. Двунаправленный список. Внутренний итератор.

D1. Кольцевой однонаправленный список. Внешний итератор.

D2. Кольцевой однонаправленный список. Внутренний итератор.

E1. Кольцевой двунаправленный список. Внешний итератор.

E2. Кольцевой двунаправленный список. Внутренний итератор.

F1. Очередь на базе списка с константным двунаправленным итератором просмотра очереди (внешний). Доступ, добавление, удаление по правилам очереди. Двунаправленный итератор для доступа на чтение зачений по всем элементам.

F2. Очередь на базе списка с константным двунаправленным итератором просмотра очереди (внутренний).

G1. Дек на базе кольцевого списка. Итератор перемещения “окна” (внешний). Итератор показывает на два соседних элемента в кольце. Эти элементы и есть head и tail. Перемещение итератора — переход к соседней паре.

G2. Дек на базе кольцевого списка. Итератор перемещения “окна” (внутренний). Итератор показывает на два соседних элемента в кольце. Эти элементы и есть head и tail. Перемещение итератора —- переход к соседней паре.

График выполнения задания

Работа над заданием предполагает еженедельный контроль по следущему графику (естественно, его можно выполнять с опережением)

неделя 1: разработка и согласование описания (интерфейса) класса. Нужно представить описание класса с указанием прототипов требуемых методов (функций) и внутренних объектов класса.

неделя 2: реализация некоторых методов и первичный тест на простых типах данных, проверка работы механизма исключений Нужно представить тест с проверкой работы отдельных функций класса: по принципу “операция – распечатка состояния”. В частности, должна быть готова функция распечатки (оператор «). Проверка на разных простых типах. Проверка на некорректных обращениях.

неделя 3. реализация более полного набора методов, тест совместной работы этих методов, тест на сложных типах данных. Тест с “нагрузкой” - выполняются массовые операции с автоматической проверкой результата. Например, добавили 1000000 последовательных целых чисел. Прошлись итератором и удалили четные. Прошлись итератором и проверили, что осталисть только нечетные и при этом ни одно нечетное число не пропало. Другие подобные тесты, смысл которых в многократном выполнении операций в середине и по краям списка. Присваивание списков друг другу. Некоторые ошибки (по памяти) могут не проявиться в однократной операции, а массовая проверка их обнаружит. Тест на “рекурсивных” данных типа список списков векторов :))) Т.е. элементами списка являются тоже списками. Тестирование правильности копирования и доступа. При доступе к элементу мы также получаем возможность работать с этим элементом как с контейнером. Как будет работать процедура распечатки?

неделя 4: Сравнение с STL библиотекой. Получить из данного начального состояния списка другое состояние, определяемое некоторыми правилами, с помошью ваших реализаций и с помощью классов библиотеки STL. Это означает, что вы строите класс с точна таким же интерфейсом, как ваш исходный “список”, но вся его реализация в основном состоит в прямой переадресации действий на методы STL классов. Например, в STL есть класс list — список с некоторым набором операций. Вот вы и берете этот list за основу и ваши операции перенаправляете на операции класса list. В честности, сортировку вашего списка можно сравнить с применением алгоритма sort для list. Более подробно все детали задачи обсуждается индивидуально в рабочем порядке.

Таким образом, отчетность по этому заданию будет содержать 4 контрольные точки, соответственно данным этапам. В абсолютных датах окончательный срок сдачи всего задания — начало ноября.

В реальности реализация всех заявленных возможностей и требований может вылиться в весьма большой объем работы. Поэтому не стоит пытаться сделать их все сразу. В первую очередь следует реализовывать то, без чего нельзя обойтись в случае каждого конкретного теста. Далее, расширяя логику и задачи тестирования, можно дописывать недостающие фрагменты кода. Поэтому после понимания постановки задачи и разработки базового интерфейса вашего класса надо сразу формулировать задачи тестов и под эти задачи начинать реализовывать методы вашего класса и сам тестовый код.