Лабораторная работа №4 по курсу «Проектирование программного обеспечения интеллектуальных систем» (2 курс 1 семестр)

Тема: Обобщенное программирование. Библиотека STL.
Цель: Получить навыки обобщенного программирования с использованием шаблонов.

Содержание

1 Задание

В соответствии с вариантом нужно реализовать шаблонную функцию(функции) для сортировки. Необходимо, чтобы разработанная функция(функции) позволяла сортировать массивы и векторы(`std::vector<>`) любых объектов (как встроенных типов, так и пользовательских), продемонстрировать это путём создания собственного класса, массив объектов которого необходимо отсортировать. Для этого можно изучить реализацию функций сортировки, из стандартной библиотеки, позволяющих сортировать массивы и векторы.

Варианты заданий:

1. Bubble sort. Patience sorting.
2. Cocktail sort. Strand sort.
3. Comb sort. Tournament sort.
4. Gnome sort. Pigeonhole sort.

5. Selection sort. Bucket sort.
6. Insertion sort. Counting sort.
7. Shell sort. LSD Radix sort.
8. Binary tree sort. MSD Radix sort.
9. Library sort. Spreadsort.
10. Merge sort. Bozo sort.
11. In-place merge sort. Bogosort.
12. Heapsort. Stooge sort
13. Smoothsort. Simple pancake sort.
14. Quicksort. Sorting networks
15. Introsort. Tacosort.

2 Задание (усложненное)

В этой лабораторной работе студенту необходимо реализовать шаблон STL-контейнера в соответствие с выбранным вариантом. Реализованныйный шаблон контейнера должен соответствовать следующим требованиям:

- иметь как минимум один шаблонный аргумент, который задает тип элементов, для хранения которых будет использоваться специализация контейнера
- предлагать `typedef`'ы `value_type`, `reference`, `const_reference`, `pointer` и др. (полный перечень определяется вариантом задания и разработчиком)
- конструктор по умолчанию (создает пустой контейнер)
- конструктор копирования
- деструктор
- проверка на пустой контейнер (метод `empty`)
- очистка контейнера (метод `clear`)
- перегруженный оператор присваивания `=`
- перегруженные операторы сравнения: `==`, `!=`, `>`, `<`, `>=`, `<=`
- методы для доступа к элементам контейнера (перечень зависит от варианта задания)
- методы для добавления элементов в контейнер (перечень зависит от варианта задания)
- методы для удаления элементов из контейнера (перечень зависит от варианта задания)
- предлагать классы итераторов для перебора элементов и методы для их создания (тип итераторов и перечень методов зависит от варианта задания)

- перегруженный оператор вывода <<, который использует итераторы контейнера и обобщенный алгоритм `std::for_each` для вывода элементов в поток
- при возникновении ошибочной ситуации должно выбрасываться исключение

Механизм тестирования на усмотрение разработчика: меню или юнит-тесты.

При защите лабораторной работы студент должен уметь использовать шаблоны C++, обладать знаниями о структуре STL, свойствах STL-контейнеров, типах STL-итераторов, использовании STL-алгоритмов и функторов.

Для получения теоритических сведений студенту необходимо изучить главу 14 «Шаблоныk, главу 20 «Структуры данныхk, главу 22 «Библиотека стандартных шаблоновk из книги Х. Дейтела и П. Дейтела «Как программировать на C++k 2008 года издания. Для получения более обширной практической информации об использовании шаблонов и STL будет изучение семинаров 3 и 6 из книги Павловская Т. А., Щупак Ю. А. «C++. Объектно-ориентированное программирование: Практикумk.

Дополнительная информация:

- Более подробно про итераторы и STL (на русском языке)
- Документация по SGI STL
- Что такое классы свойств (traits)?

Усложнение

В качестве второго аргумента шаблона контейнера добавить класс свойств типа элемента контейнера.

2.1 Варианты заданий

2.1.1 Неориентированный граф (Матрица смежности)

При выборе этого варианта необходимо реализовать шаблонный класс контейнера для представления неориентированного графа с использованием матрицы смежности. Класс не должен раскрывать способ представления графа, а должен предлагать typedef'ы методы и итераторы для работы. У шаблона должен быть один аргумент, который определяет тип, значения которого будут храниться в вершине графа.

Реализованный шаблон класс представления неориентированного графа должен соответствовать следующим требованиям (общие требования см. в общей части задания):

- проверка присутствия вершины в графе
- проверка присутствия ребра между вершинами в графе
- получение количества вершин
- получение количества ребер
- вычисление степени вершины
- вычисление степени ребра
- добавление вершины
- добавление ребра

- удаление вершины
- удаление ребра
- двунаправленный итератор для перебора вершин
- двунаправленный итератор для перебора ребер (совет: обратите внимание на класс `std::pair`)
- двунаправленный итератор для перебора ребер, инцидентных вершине
- двунаправленный итератор для перебора вершин, смежных вершине
- удаление вершины по итератору на вершину
- удаление ребра по итератору на ребро
- обратные (reverse) модификации для всех итераторов
- константные модификации для всех итераторов

Необходимость создания шаблонных классов-оберток для представления вершин и ребер на усмотрение разработчика.

Дополнительная литература:

- Способы представления графов

2.1.2 Ориентированный граф (Матрица смежности)

Основные требования смотреть в 2.1.1.

2.1.3 Неориентированный граф (Матрица инцидентности)

Основные требования смотреть в 2.1.1.

2.1.4 Ориентированный граф (Матрица инцидентности)

Основные требования смотреть в 2.1.1.

2.1.5 Неориентированный граф (Список смежности)

Основные требования смотреть в 2.1.1.

2.1.6 Ориентированный граф (Список смежности)

Основные требования смотреть в 2.1.1.

2.1.7 Неориентированный граф (Упорядоченные списки ребер)

Основные требования смотреть в 2.1.1.

2.1.8 Ориентированный граф (Упорядоченные списки ребер)

Основные требования смотреть в 2.1.1.

2.1.9 Неориентированный граф (Ортогональный список смежности)

Основные требования смотреть в 2.1.1.

2.1.10 Ориентированный граф (Ортогональный список смежности)

Основные требования смотреть в 2.1.1.

2.1.11 Неориентированный граф (Структура Вирта)

Основные требования смотреть в 2.1.1.

Для перебора смежных вершин реализовать последовательный (однонаправленный) итератор.

2.1.12 Ориентированный граф (Структура Вирта)

Основные требования смотреть в 2.1.1. Требования смотреть в 1-ом варианте и 11-ом вариантах.

2.1.13 Неориентированный граф (Модифицированная структура Вирта)

Основные требования смотреть в 2.1.1.

2.1.14 Ориентированный граф (Модифицированная структура Вирта)

Основные требования смотреть в 2.1.1.