

Яцек Галовиц

C++17 STL

СТАНДАРТНАЯ БИБЛИОТЕКА
ШАБЛОНОВ

Packt>

 **НСТЕР®**

Краткое содержание

Предисловие.....	12
Глава 1. Новые возможности C++17	18
Глава 2. Контейнеры STL	43
Глава 3. Итераторы	91
Глава 4. Лямбда-выражения	123
Глава 5. Основы работы с алгоритмами STL	151
Глава 6. Сложные случаи использования алгоритмов STL	188
Глава 7. Строки, классы потоков и регулярные выражения	229
Глава 8. Вспомогательные классы	281
Глава 9. Параллелизм и конкурентность	343
Глава 10. Файловая система	400
Об авторе.....	428
О рецензенте.....	429

Оглавление

Предисловие.....	12
Темы, рассмотренные в этой книге.....	13
Что вам нужно для работы с книгой	14
Для кого предназначено издание	15
Разделы.....	16
Условные обозначения	16
Загрузка примеров кода	17
Глава 1. Новые возможности C++17	18
Введение.....	18
Применяем структурированные привязки (декомпозицию) для распаковки набора возвращаемых значений	19
Ограничиваем область видимости переменных в выражениях if и switch	23
Новые правила инициализатора с фигурными скобками	26
Разрешаем конструктору автоматически выводить полученный тип класса шаблона.....	28
Упрощаем принятие решений во время компиляции с помощью constexpr-if.....	30
Подключаем библиотеки с помощью встраиваемых переменных	33
Реализуем вспомогательные функции с помощью выражений свертки	36
Глава 2. Контейнеры STL	43
Введение.....	43
Используем идиому erase-remove для контейнера std::vector.....	46

Удаляем элементы из неотсортированного объекта класса <code>std::vector</code> за время $O(1)$	50
Получаем доступ к экземплярам класса <code>std::vector</code> быстрым или безопасным способом	53
Сохраняем сортировку экземпляров класса <code>std::vector</code>	55
Вставляем элементы в контейнер <code>std::map</code> эффективно и в соответствии с условиями	57
Исследуем новую семантику подсказок для вставки элементов с помощью метода <code>std::map::insert</code>	61
Эффективно изменяем ключи элементов <code>std::map</code>	64
Применяем контейнер <code>std::unordered_map</code> для пользовательских типов.....	67
Отсеиваем повторяющиеся слова из пользовательского ввода и выводим их на экран в алфавитном порядке с помощью контейнера <code>std::set</code>	70
Реализуем простой ОГН-калькулятор с использованием контейнера <code>std::stack</code>	73
Подсчитываем частоту встречаемости слов с применением контейнера <code>std::map</code>	79
Вспомогательный стилистический редактор для поиска длинных предложений в текстах с помощью <code>std::multimap</code>	82
Реализуем личный список текущих дел с помощью <code>std::priority_queue</code>	87
Глава 3. Итераторы	91
Введение.....	91
Создаем собственный итерабельный диапазон данных.....	95
Обеспечиваем совместимость собственных итераторов с категориями итераторов STL.....	98
Используем оболочки итераторов для заполнения обобщенных структур данных.....	101
Реализуем алгоритмы с помощью итераторов.....	104
Перебор в обратную сторону с применением обратных адаптеров для итераторов	108
Завершение перебора диапазонов данных с использованием ограничителей.....	110
Автоматическая проверка кода итераторов с помощью проверяемых итераторов.....	113
Создаем собственный адаптер для итераторов-упаковщиков.....	117

Глава 4. Лямбда-выражения	123
Введение	123
Динамическое определение функций с помощью лямбда-выражений	125
Добавляем полиморфизм путем оборачивания лямбда-выражений в <code>std::function</code>	129
Создаем функции методом конкатенации	132
Создаем сложные предикаты с помощью логической конъюнкции	136
Вызываем несколько функций с одинаковыми входными данными	138
Реализуем функцию <code>transform_if</code> с применением <code>std::accumulate</code> и лямбда-выражений	141
Генерируем декартово произведение на основе любых входных данных во время компиляции	146
Глава 5. Основы работы с алгоритмами STL	151
Введение	151
Копируем элементы из одних контейнеров в другие	153
Сортируем контейнеры	157
Удаляем конкретные элементы из контейнеров	161
Преобразуем содержимое контейнеров	164
Выполняем поиск элементов в упорядоченных и неупорядоченных векторах	166
Ограничиваем допустимые значения вектора конкретным численным диапазоном с помощью <code>std::clamp</code>	172
Находим шаблоны в строках с помощью функции <code>std::search</code> и выбираем оптимальную реализацию	175
Делаем выборку данных из крупных векторов	179
Выполняем перестановки во входных последовательностях	182
Инструмент для слияния словарей	184
Глава 6. Сложные случаи использования алгоритмов STL	188
Введение	188
Реализуем класс префиксного дерева с использованием алгоритмов STL	189
Создаем генератор поисковых подсказок с помощью префиксных деревьев	194
Реализуем формулу преобразования Фурье с применением численных алгоритмов STL	199

Определяем ошибку суммы двух векторов.....	207
Реализуем отрисовщик множества Мандельброта в ASCII.....	210
Создаем собственный алгоритм split.....	215
Создаем полезные алгоритмы на основе стандартных алгоритмов gather.....	219
Удаляем лишние пробелы между словами.....	223
Компрессия и декомпрессия строк.....	225
Глава 7. Строки, классы потоков и регулярные выражения	229
Введение.....	229
Создание, конкатенация и преобразование строк.....	231
Удаляем пробелы из начала и конца строк.....	234
Преимущества использования std::string без затрат на создание объектов std::string.....	236
Считываем значения из пользовательского ввода.....	240
Подсчитываем все слова в файле.....	243
Форматируем ваши выходные данные с помощью манипуляторов потока ввода-вывода.....	245
Инициализируем сложные объекты из файла вывода.....	251
Заполняем контейнеры с применением итераторов std::istream.....	254
Выводим любые данные на экран с помощью итераторов std::ostream.....	258
Перенаправляем выходные данные в файл для конкретных разделов кода.....	262
Создаем пользовательские строковые классы путем наследования std::char_traits.....	266
Токенизация входных данных с помощью библиотеки для работы с регулярными выражениями.....	271
Удобный и красивый динамический вывод чисел на экран в зависимости от контекста.....	275
Перехватываем читабельные исключения для ошибок потока std::iostream.....	277
Глава 8. Вспомогательные классы	281
Введение.....	281
Преобразуем единицы измерения времени с помощью std::ratio.....	282
Выполняем преобразование между абсолютными и относительными значениями с использованием std::chrono.....	287
Безопасно извещаем о сбое с помощью std::optional.....	290

Применяем функции для кортежей.....	293
Быстрое создание структур данных с помощью <code>std::tuple</code>	296
Замена <code>void*</code> с использованием <code>std::any</code> для повышения безопасности типов.....	303
Хранение разных типов с применением <code>std::variant</code>	306
Автоматическое управление ресурсами с помощью <code>std::unique_ptr</code>	311
Автоматическое управление разделяемой памятью кучи с использованием <code>std::shared_ptr</code>	314
Работаем со слабыми указателями на разделяемые объекты	320
Упрощаем управление ресурсами устаревших API с применением умных указателей.....	324
Открываем доступ к разным переменным — членам одного объекта	327
Генерируем случайные числа и выбираем правильный генератор случайных чисел	330
Генерируем случайные числа и создаем конкретные распределения с помощью STL	335
Глава 9. Параллелизм и конкурентность	343
Введение.....	343
Автоматическое распараллеливание кода, использующего стандартные алгоритмы	344
Приостанавливаем программу на конкретный промежуток времени	350
Запускаем и приостанавливаем потоки.....	352
Выполняем устойчивую к исключениям общую блокировку с помощью <code>std::unique_lock</code> и <code>std::shared_lock</code>	356
Избегаем взаимных блокировок с применением <code>std::scoped_lock</code>	363
Синхронизация конкурентного использования <code>std::cout</code>	366
Безопасно откладываем инициализацию с помощью <code>std::call_once</code>	370
Отправляем выполнение задач в фоновый режим с применением <code>std::async</code>	372
Реализуем идиому «производитель/потребитель» с использованием <code>std::condition_variable</code>	377
Реализуем идиому «несколько производителей/потребителей» с помощью <code>std::condition_variable</code>	381
Распараллеливание отрисовщика множества Мандельброта в ASCII с применением <code>std::async</code>	387
Небольшая автоматическая библиотека для распараллеливания с использованием <code>std::future</code>	391

Глава 10. Файловая система	400
Введение.....	400
Реализуем нормализатор пути файла	401
Получаем канонические пути к файлам из относительных путей.....	404
Составляем список всех файлов в каталоге.....	407
Инструмент текстового поиска в стиле gper.....	412
Инструмент для автоматического переименования файлов	415
Создаем индикатор эксплуатации диска	418
Подбиваем статистику о типах файлов	420
Инструмент для уменьшения размера папки путем замены дубликатов символьными ссылками	423
Об авторе	428
О рецензенте.....	429