Контрольные вопросы. Задание 16.

6 марта 2023 г.

1 Как организован механизм генерации случайных чисел в библиотеке random?

Библиотека random включает две основные сущности: uniform random bit generators (URBGs – равномерные генераторы случайных битов) и random number distributions (генераторы случайных чисел). Первая из них включает в себя механизмы случайных чисел, которые представляют собой генераторы псевдослучайных чисел, генерирующие целочисленные последовательности с равномерным распределением, а также генераторы истинных случайных чисел, если они доступны. Вторая сущность преобразуют выходные данные первой в различные статистические распределения.

Более подробно, URBG – это функциональный объект, возвращающий целочисленные значения без знака таким образом, что каждое значение в диапазоне возможных результатов имеет (в идеале) одинаковую вероятность возврата. Генераторы унифицированных случайных битов не предназначены для использования в качестве генераторов случайных чисел: они используются в качестве источника случайных битов (генерируются массово, для эффективности). Любой равномерный генератор случайных битов может быть подключен к любому распределению случайных чисел, чтобы получить случайное число (формально, случайную переменную). Все URBG соответствуют требованиям UniformRandomBitGenerator.

Механизмы генерации случайных чисел (random number engines) генерируют псевдослучайные числа, используя начальные данные в качестве источника энтропии. Выбор механизма для использования включает в себя ряд компромиссов: линейный конгруэнтный движок (linear_congruential_engine) умеренно быстр и требует очень мало памяти для состояния. Генераторы Фибоначчи с запаздыванием (subtract_with_carry_engine) работают очень быстро даже на процессорах без расширенных наборов арифметических инструкций за счет большей памяти состояния и иногда менее желательных спектральных характеристик. Вихрь Мерсенна (mersenne_twister_engine) медленнее и требует большего хранения состояния, но при правильных параметрах имеет самую длинную неповторяющуюся последовательность с наиболее желательными спектральными характеристиками (для данного определения желаемого). Несколько популярных алгоритмов генерации предопределены. Кроме того, можно использовать адаптеры механизмов случайных чисел, генерирующие псевдослучайные числа, используя другой механизм случайных чисел в качестве источника энтропии.

Зачастую, используемые начальные данные делают процесс генерации случайных чисел детерминированным (что нарушает случайность), поэтому существует недетерминированный универсальный генератор случайных битов std::random_device. Правда, разрешено реализовывать std::random_device с использованием механизма псевдослучайных чисел, если не поддерживается генерация недетерминированных случайных чисел.

Наконец, распределения случайных чисел, удовлетворяющие требованиям RandomNumberDistributio выполняют постобработку выходных данных URBG таким образом, что итоговые данные распределяются в соответствии с определенной статистической функцией плотности вероятности.

Типичная схема генерации выглядит следующим образом. Сначала используется std::random_device, который генерирует зерно (seed). Затем одному из движков генерации случайных чисел передаётся это зерно, и полученный генератор далее может быть использован для получения чисел с некоторым распределением.

2 Чем отличаются функциональные объекты от функций и лямбда-выражений?

Функциональные объекты, или функторы, – это классы с переопределённым оператором operator(). Поэтому их синтаксис напоминает таковой у функций, но последние не являются классами, чем и обусловлены отличия:

- Можно создавать различные экземпляры функциональных объектов, т.к. функторы – это классы. Причём каждый класс имеет свой тип в то время, как у функций тип Return_Type (Types...).
- Функциональный объект может иметь состояние (благодаря данным-членам), у функции же можно использовать static переменную, но этого делать не рекомендуется
- Функциональные объекты работают быстрее функций, переданных через указатель. Собственно говоря, из-за возможности компилятора оптимизировать код часто функторы или лямбда-функции оказывается использовать лучше, чем указатели на функции (например, при передаче функтора в функцию), т.к. указатели используют память более сложным образом.

Лямбда-выражения, являющиеся локальными функцими, которые можно создавать прямо внутри какого-либо выражения, сочетают в себе преимущества указателей на функции и функциональных объектов, благодаря чему их удобно создавать и использовать на месте (в том числе используя захват). Как и функциональные объекты, лямбда-выражения позволяют хранить состояния, но их компактный синтаксис в отличие от функциональных объектов не требует объявления класса.

3 Какими наборами возможностей обладают итераторы различных категорий?

Все категории итераторов могут быть инкрементированы на единицу, а также поддерживают инициализацию копированием.

Рассмотрим более подробно возможности итераторов различных категорий:

- Input iterator поддерживает сравнение на равенство/неравенство (использование operator==, operator!=) и может быть разыменован только как rvalue (т.е. только для получения значения, т.к. связан с потоком ввода данных).
- Output iterator может быть разыменован как lvalue для использования слева от знака присваивания.
- Forward iterator обладает теми же возможножностями что Input и Output iterators, а также может быть скопирован и использован для повторного обхода.
- Bidirectional iterator обладает теми же возможножностями что Input, Output и Forward iterators, а также может быть уменьшен на единицу.

- Random access iterator поддерживает:
  - все возможности итераторов остальных категорий;
  - арифметические операции + и - (арифметика указателей);
  - сравнения между указателями (operator>, operator<, operator>=, operator<=);
  - операции увеличения и уменьшения на (operator+=, operator-=);
  - разыменование по индексу ([]).

Существуют также insert iterators/inserters, позволяющие алгоритмам стандартной библиотеки работать со вставками в начало/конец.

4 Какая классификация предлагается для алгоритмов стандартной библиотеки?

Классификация алгоритмов стандартной библиотеки:

- немодифицирующие;
- модифицирующие;
- удаления;
- перестановки;
- сортировки;
- для упорядочивания диапазонов;
- численные.

Один из важнейших алгоритмов std::for_each относится к двум классам алгоритмов STL: он является и модифицирующим, и немодифицирующим.

5 Почему алгоритмы стандартной библиотеки предпочтительнее собственных?

Алгоритмы стандартной библиотеки предпочтительнее собственных, поскольку не будет необходимости писать собственный код, который будет выглядеть более громоздко, т.е. использование алгоритмов стандартной библиотеки проще, короче и быстрее.

Литература.

[1] Конспект семинара. Макаров И.С.

[2] https://en.cppreference.com/w/cpp/numeric/random