Программирование на языке C++

Шаблов Анатолий
anatoliishablov@gmail.com

ИТМО, весенний семестр 2025

Коллекции

Коллекции

Коллекция – набор объектов одного типа (элементов), допускающий как операции со всей коллекцией в целом, так и с отдельными элементами. Другое название – контейнер.

Экземпляр коллекции хранит элементы внутри себя, копирование или перемещение экземпляра коллекции приводит к копированию или перемещению всех её элементов.

Некоторые свойства коллекций

Общие:

- Хранение множества элементов одного типа.
- Последовательный доступ ко всем элементам.
- Операции над элементами: немодифицирующий доступ (чтение), модифицирующий доступ (запись), удаление и добавление.

Различающиеся:

- Эффективность реализации тех или иных операций.
- Доступ к элементам в определенном порядке.
- Некоторые специфические операции.

Понятие вычислительной сложности

- Вычислительные затраты на обработку.
- Затраты памяти на представление или обработку.

Стандартные коллекции: простые последовательности

- std::array – массив элементов со статическим размером.
- std::vector – массив элементов с динамическим размером.
- std::deque – двусторонняя очередь.
- std::forward_list – односвязный список.
- std::list – двусвязный список.

Стандартные коллекции: упорядоченные множества

- std::set – упорядоченное множество.
- std::map – упорядоченный ассоциативный массив.
- std::multiset – упорядоченное мультимножество.
- std::multimap – упорядоченный ассоциативный массив с повторением ключей.

Стандартные коллекции: неупорядоченные множества

- std::unordered_set – неупорядоченное множество.
- std::unordered_map – неупорядоченный ассоциативный массив.
- std::unordered_multiset – неупорядоченное мультимножество.
- std::unordered_multimap – неупорядоченный ассоциативный массив с повторением ключей.

Адаптеры коллекций

- std::stack – LIFO.
- std::queue – FIFO.
- std::priority_queue – приоритетная очередь, быстрый доступ к наибольшему элементу.

Внутреннее представление

- Непрерывная область памяти – std::array, std::vector.
- Набор независимых подобластей – std::deque.
- Каждый элемент в отдельном узле списка – std::forward_list, std::list.
- Дерево – std::set, std::map, std::multiset, std::multimap.
- Хеш-массив – std::unordered_set, std::unordered_map, std::unordered_multiset, std::unordered_multimap.

Сложность некоторых операций

Итераторы

Итераторы

Итератор – абстракция “указателя” на элемент некоторой последовательности.

Тип итератора может быть связан с конкретным типом коллекции/контейнера, но бывают и иные итераторы. Реализация скрывает в себе подробности доступа к указываемому элементу коллекции и связь с “соседними” элементами.

Каждая коллекция задает некоторый порядок на множестве элементов, если представить это как массив в памяти, итераторы имитируют указатели на элементы этого массива.

Итераторы и указатели: сходство

- Разыменование для доступа к элементу (операторы * и ->).
- Равенство/неравенство.
- Инкремент для смещения к следующему элементу.
- Итератор может указывать на “после последнего элемента” коллекции.
- Некоторые итераторы можно декрементировать для смещения к предыдущему элементу.
- Над некоторыми итераторами возможна арифметика, как с указателями.
- Некоторые итераторы поддерживают доступ по индексу (аналогично указателям).
- Некоторые итераторы можно сравнивать

Итераторы и указатели: различия

Указатель удовлетворяет “концепции итератора”, но итераторы коллекций от указателей отличаются.

- Является сложным типом (классом).
- Эффективность операций зависит от конкретной реализации.
- Всегда связан с конкретным объектом коллекции или иной сущности.
- Некоторые действия над объектом коллекции могут инвалидировать итератор.
- Для итераторов не работает приведение типов, возможное для указателей.

Концепция: итератор

- Объект можно копировать
- Определен оператор *
- Определен оператор префиксного инкремента (условие – итератор должен быть указывать на элемент коллекции; результатом будет либо итератор, указывающий на следующий элемент, либо на после последнего)

Концепция: итератор на чтение (input)

- Удовлетворяет требованиям итератора.
- Объекты можно сравнивать на равенство/неравенство.
- Определен оператор ->.
- Определен оператор постфиксного инкремента.
- Не гарантирована валидность копий итератора после его инкремента Таким образом, не гарантирована возможность многократного прохода по элементам, только однократного.

Концепция: итератор на запись (output)

- Удовлетворяет требованиям итератора.
- Определен оператор постфиксного инкремента.
- Валидно выражение *i = x где i – объект итератора, а x некоторое значение; после выполнения этого выражение не гарантируется разыменуемость итератора или валидность его предыдущих копий.
- Не гарантирована валидность копий итератора после его инкремента.
- Операцию разыменования допустимо использовать только для присвоения значения Таким образом, не гарантирована возможность многократного прохода по элементам, только однократного.

Концепция: мутабельный итератор (input and output)

Если итератор удовлетворяет обоим концепциям – input и output, то его называют mutable, это значит, что указываемые значения можно читать, а можно присваивать им другие значения.

Концепция: итератор непрерывной области (contiguous)

- Удовлетворяет требованиям итератора.
- Элементы коллекции, с которой связан итератор, размещены в непрерывной области в памяти и в том порядке, в котором осуществляется обход с помощью итераторов.
- Валидна операция *(a + n) и эквивалентна *(std::addressof(*a)+ n)

Концепция: итератор прямого обхода (forward)

- Удовлетворяет требованиям итератора на чтение.
- a == b ⇔ &(*a)== &(*b) (либо оба не разыменуются).
- Инкремент итератора не меняет валидность его копий.
- Инкремент итератора не меняет результат разыменования его копий.
- Равенство итераторов гарантирует равенство результатов их инкремента.
- Если итератор является мутабельным, то присвоение через итератор не меняет его валидность или валидность его копий Таким образом, гарантирована возможность многократного прохода по элементам.

Концепция: итератор двустороннего обхода (bidirectional)

- Удовлетворяет требованиям итератора прямого обхода.
- Определен оператор декремента (префиксного и постфиксного).
- Декремент не меняет валидность копий итератора.
- Результат декремента итератора, указывающего на первый элемент коллекции не определен.

Концепция: итератор произвольного доступа (random access)

- Удовлетворяет требованиям двунаправленного итератора.
- Поддерживает арифметику, аналогичную арифметике указателей.
- Определен оператор доступа по индексу, подобно как для указателей.
- Определены операторы сравнения <, >, <=, >=.

Примеры концепций

- Указатель на элемент массива – удовлетворяет требованиям итератора непрерывной* области.
- Итераторы коллекций array, vector, deque – непрерывная* область.
- Итераторы list, set, map – двунаправленный.
- Итераторы forward_list, unordered_set, unordered_map – прямого обхода.
- back_insert_iterator – итератор на запись, добавляет элементы в конец коллекции.

Некоторые операции над итераторами

```
#include <iterator>
void std::advance(It & it, Distance distance);
Distance std::distance(It from, It to);
It std::next(It it, Distance distance);
It std::prev(It it, Distance distance);
```

Обход коллекций

- std::begin()
- std::end()
- std::cbegin()
- std::cend()

- std::rbegin()
- std::rend()
- std::crbegin()
- std::crend()

Некоторые адаптеры итераторов

```
RIt std::make_reverse_iterator(It it);
It std::front_inserter(Container & c);
It std::back_inserter(Container & c);
It std::inserter(Container & c, It it);
```

Некоторые алгоритмы

Алгоритмы стандартной библиотеки

Алгоритм – некоторая функция, принимающая пару итераторов, задающих начало и конец последовательности и, возможно, другие аргументы. Оперирует элементами последовательности.

```
std::vector v { 4, 5, 3, 1, 2 };
std::sort(v.begin(), v.end() - 1);
std::copy(v.begin(), v.end(),
          std::ostream_iterator(std::cout, ", "));
```

Типы алгоритмов

- Не модифицирующие (any_of, for_each, count, find).*
- Разбивающие (partition).
- Сортирующие (sort, nth_element).
- Общие модифицирующие (copy, transform, remove, unique).
- Операции над сортированными последовательностями (binary_search, merge, set_intersection).
- И другие...