© 2012

Последовательность – набор однотипных элементов

данных, для которых определено понятие «следующий элемент последовательности» (это понятие определено для всех элементов, кроме последнего).

Примеры последовательностей:

►

массивы

►

контейнерные классы из библиотеки STL

►

строки (класс string)

►

потоки

Итератор – инструмент для доступа к элементам

последовательности Указатель (имя массива) – частный случай итератора, если в качестве последовательности используется массив Интервал – идущие подряд элементы

последовательности. Интервал задается парой итераторов. Первый итератор связан с первым элементом интервала, второй итератор – с первым элементом, следующим за интервалом. Второй итератор может быть недействительным

**Сортировка массива long M[50]; // заполнение массива sort(M, M+50);**

**Сортировка контейнера vector vector <long> M; // заполнение контейнера sort(M.begin(), M.end());**

**Подсчет количества элементов long \*p = new long [20]; deque <long> d; cout << count(p, p+20, 5) << endl; cout << count(d.begin(), d.end(), 5) << endl;**

Тип итератора Доступ

Итерация Сравнение

Итератор вывода (output iterator)

Только запись

\* ++

Итератор ввода (input iterator)

Только чтение

\*, -> ++ ==, !=

Прямой итератор (forward iterator)

Чтение запись

и

\*, -> ++ ==, !=

Двунаправленный итератор (bidirectional iterator)

Чтение запись

и

\*, -> ++, -- ==, !=

Итератор с произвольным доступом (random-access iterator)

Чтение и запись

++, --, +, -, +=, -= \*, ->, []

==, !=, <, <=, >, >=

Разымено- вание

Тип итератора vector list deque set multiset map multimap

Произвольного доступа

X X

Двунаправленный X X X X X X X

Прямой X X X X X X X

Входной X X X X X X X

Выходной X X X X X X X

Типы итераторов, требующиеся представленным алгоритмам

**Входной Выходной Прямой Двунаправленный Произвольного доступа X**

**Алгоритм for each find count**

x

x

copy replace

unique

**reverse**

sort nth\_element merge accumulate

X

итератор ►

не был инициализирован;

►

итератор указывает на конец последовательности;

►

элемент контейнера, с которым он связан, удален;

►

контейнер, с которым он связан, изменил размеры или уничтожен.

Функциональный объект - класс, в котором определена операция вызова функции. Синонимы: объект-функция, функтор Предикат – функциональный объект (или функция), возвращающий логическое значение. Пример ФО:

**class isbest{ public:**

**int operator() (int a, int b){**

**return (a>b || a%10==0 ? a : b); } };**

**... isbest b; int x, y; cout << b(x, y);**

бинарная функция T3 (T1, T2)

унарная функция T3 (T1)

бинарный предикат bool (T1, T2)

унарный предикат bool (T1)

Здесь T1, T2, T3 – произвольные типы

Задача: подсчитать количество элементов вектора, описанного как

**vector <long> d; больших 5**

Алгоритм count\_if определяет число элементов интервала, для которых верен заданный унарный предикат (функция либо функциональный объект)

**int count\_if (интервал, унарный\_предикат)**

**Первый вариант решения (функция) bool gr5(long a) { return (a>5); } ... cout << count\_if (d.begin(), d.end(), gr5)<< endl;**

Второй вариант решения (функциональный объект) class \_gr5 { public:

**bool operator() (long a) { return (a>5); } }; ... cout << count\_if (d.begin(), d.end(), \_gr5())**

**<< endl;**

Третий вариант решения (с использованием шаблона функции) template <long value> bool gr5(long a) {

**return (a > value); } ... cout << count\_if(d.begin(), d.end(), gr5<10>)**

**<< endl;**

Четвертый вариант решения (функциональный объект, выражение вместо константы) class \_gr { long \_n; public:

**\_gr(long an) { \_n = an; } //конструктор bool operator() (long a) { return (a>\_n); } };**

**...**

**long val = 10; ...**

**cout << count\_if (d.begin(), d.end(), \_gr(val))**

**<< endl;**

Описаны в заголовочном файле <functional>

Арифметические функциональные объекты

Название Вид Результат plus T (T, T) x + y minus T (T, T) x – y multiplies T (T, T) x \* y divides T (T, T) x / y modulus T (T, T) x % y negate T (T) - x

Предикаты стандартной библиотеки

Название Вид Результат equal\_to bool (T, T) x == y not\_equal\_to bool (T, T) x != y greater bool (T, T) x > y less bool (T, T) x < y greater\_equal bool (T, T) x >= y less\_equal bool (T, T) x <= y logical\_and bool (T, T) х && у logical\_or bool (T, T) х || У logical\_not bool (T) ! х

1) Критерий сортировки Для ассоциативных контейнеров критерий – по возрастанию

less<> (меньше), следующие записи эквивалентны: set <int> coll;

**set <int, less<int> > coll;**

По убыванию greater<> :

**set <int, greater<int> > coll;**

2) Изменение знака на противоположный

**template <class X> class PrintElement {**

**public:**

**void operator() (X elem) const {**

**cout << elem << ' '; } };**

**Продолжение void main() {**

**list<int> coll; // Заполняем элементы коллекции for (int i = 1; i < 5; i++) {**

**coll.push\_back(i); coll.push\_front(-i); } for\_each(coll.begin(), coll.end(),**

**PrintElement<int>()); cout << "\n";**

**// Изменяем знак элементов коллекции**

**transform(coll.begin(), coll.end(),**

**coll.begin(), negate<int>());**

**// Выводим элементы коллекции for\_each(coll.begin(), coll.end(),**

**PrintElement<int>()); cout << "\n"; }**

*Ответ*

*-4 -3 -2 -1 1 2 3 4 4 3 2 1 -1 -2 -3 -4*

Адаптером функции называют функцию, которая получает в качестве аргумента функцию и конструирует из нее другую функцию. На месте функции может быть также функциональный объект.

Стандартная библиотека содержит описание нескольких типов адаптеров:

•

связыватели для использования функционального объекта с двумя аргументами как объекта с одним аргументом;

•

отрицатели для инверсии значения предиката.

Название Вид Результат

not1 bool (bool(T)) отрицание унарного

предиката

not2 bool (bool (T, T)) отрицание бинарного предиката bind2nd bool(bool (T t1, T t2), const T t3) преобразует

бинарный предикат в унарный,

bind1st bool(bool (T t1, T t2), const T t3)

подставляя значение t3 вместо t2 и t1 соответственно

Для инверсии less<int>() нужно записать

**not2(less<int>()) Эквивалентно greater\_equal <int>()**

**Пятый вариант решения рассмотренной ранее задачи: long k; ... cout << count\_if (d.begin(), d.end(), bind2nd(greater <long> (), 5)) << endl;**

Вычислить количество элементов целочисленного массива, меньших 40:

**#include <iostream> #include <functional> #include <algorithm> using namespace std; void main() {**

**int m[8] = {45, 65, 36, 25, 674, 2, 13, 35}; cout << count\_if(m, m + 8, bind2nd(less<int>(), 40)); } Функция bind2nd превращает условие сравнения х < у в условие х < 40.**

немодифицирующие ►

алгоритмы;

►

модифицирующие алгоритмы;

►

алгоритмы удаления;

►

перестановочные алгоритмы;

►

алгоритмы сортировки;

►

алгоритмы упорядоченных интервалов;

►

численные алгоритмы.

Некоторые алгоритмы могут принадлежать сразу нескольким категориям.

Немодифицирующие алгоритмы

►

не изменяется ни последовательность, ни ее элементы.

Модифицирующие алгоритмы

►

структура последовательности не изменяется, изменяются только ее элементы;

►

после «удаления» некоторых элементов освободившееся место заполняется мусором.

Общее

►

передается интервал последовательности, заданный парой итераторов;

►

контроля за выходом за границы последовательности нет;

►

алгоритмы работают в режиме замены, а не в режиме вставки. Специальные итераторы вставки inserter и back\_inserter переводят алгоритм в режим вставки.

Для повышения мощности и гибкости некоторые алгоритмы позволяют передавать пользовательские объект-функции или предикаты:

►

унарный предикат может передаваться алгоритму поиска в качестве критерия поиска. Унарный предикат проверяет, соответствует ли элемент заданному критерию;

►

бинарный предикат может передаваться алгоритму сортировки в качестве критерия сортировки. Бинарный предикат сравнивает два элемента;

►

унарный предикат может использоваться как критерий, определяющий, к каким элементам должна применяться операция;

►

предикаты используются для модификации операций в численных алгоритмах.

суффикс ►

\_if используется при наличии двух похожих форм алгоритма с одинаковым количеством параметров; первой форме передается значение, а второй — функция или объект функции. В этом случае версия без суффикса \_if используется при передаче значения, а версия с суффиксом \_if — при передаче функции или объекта функции.

►

суффикс \_сору означает, что алгоритм не только обрабатывает элементы, но и копирует их в приемный интервал. Например, алгоритм reverse() переставляет элементы интервала в обратном порядке, a reverse\_copy() копирует элементы в другой интервал в обратном порядке.

Спасибо за внимание

29