Стандартные библиотеки: контейнеры и итераторы

13 июня 2018 г.

Стандартная библиотека - коллекция классов и функций написанных на базовом языке C++.

* Заголовочные файлы стандартной библиотеки C++ не имеют расширения «.h».

В стандартную библиотеку входит стандартная библиотека шаблонов (STL) - набор шаблонных классов и функций общего назначения. Ядро STL составляют контейнеры, алгоритмы и итераторы.
Т.к. STL состоит из шаблонов, ее алгоритмы применимы к данным практически любого типа.

Алгоритмы обрабатывают содержимое контейнеров.

Итераторы подобны указателям. Они позволяют циклически опрашивать содержимое контейнера, практически так же, как это делается с помощью указателя. Итераторы объявляются с помощью ключевого слова iterator. Существует пять типов итераторов:

*В общем случае итератор, имеющий бОльшие возможности доступа, можно использовать вместо итератора с меньшими возможностями.

Итераторы	**Описание итератора**
Произвольного доступа (RandIter)	Сохраняют и считывают значения; позволяют организовать произвольный доступ к элементам контейнера
Двунапраленные (BiIter)	Сохраняют и считывают значения; обеспечивают инкрементно-декрементное перемещение
Однонаправленные (ForIter)	Сохраняют и считывают значения; обеспечивают только инкрементное перемещение
Входные (InIter)	Считывают, но не записывают значения; обеспечивают только инкрементное перемещение
Выходные (OutIter)	Записывают, но не считывают значения; обеспечивают только инкрементное перемещение

*Работа с итераторами такая же, как и с указателями.

*Также есть реверсивные итераторы, позволяющие перемещаться по последовательности в обратном порядке.

Так в STL очень важны распределители памяти (управляет выделением памяти для контейнера), предикаты (специальные функции, возвращающие значения true/false) и функции сравнения (сравнивают два элемента последовательности)

Контейнеры делятся на последовательные (контейнеры последовательности), ассоциативные и контейнеры-адаптеры.

Контейнеры последовательности

Последовательные контейнеры поддерживают указанный пользователем порядок вставляемых элементов.
Контейнер vector ведет себя как массив, но может автоматически увеличиваться по мере необходимости.Он поддерживает прямой доступ и связанное хранение и имеет очень гибкую длину. По этим и многим другим причинам контейнер vector является наиболее предпочтительным последовательным контейнером для большинства областей применения. Если вы сомневаетесь в выборе вида последовательного контейнера, начните с использования вектора.

Классическая спецификация:

template <class T, class Allocator = allocator<T> > class vector
где T - тип сохраняемых данных, a Allocator- распределитель памяти.

Есть конструкторы для
- пустого вектора;
- вектора с num элементов и val значениями;
- создания такого же вектора, как и вектор v;
- вектора элементами в диапазоне итераторов start и end.

* Чтобы объекты наших классов были элементами вектора, надо определять конструктор по умолчанию и операторы == и <.

Объявление:

vector <int> v; vector <char> cv(5); vector <char> cv(5,'x'); vector <int> iv2(iv)

* Для вектора определены: ==, <, <=, !=, >, >=, [ ]

Самые важные функции для векторов:

- size () - возвращает размер вектора

- begin() и end() - возвращение итератора в начало и конец
- push_back() - вставляет элемент в конец вектора
- insert () - вставляет элемент в какое-то место вектора
- erase () - удаляет указанные элементы

* Доступ к элементам вектора ведется через индексы или итераторы

Контейнер array обладает некоторыми преимуществами контейнера vector, однако его длина не обладает такой гибкостью.

Контейнер deque (двусторонняя очередь) обеспечивает быструю вставку и удаление в начале и в конце контейнера. Он, как и контейнер vector, обладает преимуществами прямого доступа и гибкой длины, но не обеспечивает связанное хранение. Дополнительные сведения см. в разделе класс deque.Контейнер deque (двусторонняя очередь) обеспечивает быструю вставку и удаление в начале и в конце контейнера. Он, как и контейнер vector, обладает преимуществами прямого доступа и гибкой длины, но не обеспечивает связанное хранение.

Контейнер list — это двунаправленный список, который обеспечивает двунаправленный доступ, быструю вставку и удаления в любом месте контейнера, но не поддерживает прямой доступ к элементам контейнера. Доступ к его элементам последовательный, либо с начала в конец, либо с конца в начало.

Есть конструкторы для создания: - пустого списка;
- списка с num элементов val значений;
- такого же списка, как и ob;
- списка, содержащего элементы в диапазоне от итераторов start и end;

* Что объект нашего класса был значением списка нужно определить конструктор по умолчанию и оператор «".

* Список можно сортировать с помощью функции sort ();
* Списки можно объединять. Получим один общий список и один пустой.

Контейнер forward_list — однонаправленный список. Это версия контейнера list только с доступом в прямом направлении.

Ассоциативные контейнеры

В ассоциативных контейнерах элементы вставляются в предварительно определенном порядке — например, с сортировкой по возрастанию. Также доступны неупорядоченные ассоциативные контейнеры. Ассоциативные контейнеры можно объединить в два подмножества: сопоставления (set) и наборы (map)

Контейнер map, который иногда называют словарем, состоит из пар "ключ-значение". Ключ используется для упорядочивания последовательности, а значение связано с ключом. Например, map может содержать клю-

чи, представляющие каждое уникальное ключевое слово в тексте, и соответствующие значения, которые обозначают количество повторений каждого слова в тексте. map — это неупорядоченная версия unordered_map.

set — это контейнер уникальных элементов, упорядоченных по возрастанию. Каждое его значение также является и ключом. set — это неупорядоченная версия unordered_set.

Контейнеры map и set разрешают вставку только одного экземпляра ключа или элемента. Если необходимо включить несколько экземпляров элемента, следует использовать контейнер multimap или multiset. Неупорядоченные версии этих контейнеров — unordered_multimap и unordered_multiset.

Упорядоченные контейнеры map и set поддерживают двунаправленные итераторы, а их неупорядоченный аналоги — итераторы с перебором в прямом направлении.

Есть конструкторы для создания: - пустого отображения;
- отображения, содержащего те же элементы, что и m;
- отображения с элементами в диапазоне итераторов start и end;

Чтобы объект нашего класса мог быть использован в качестве ключа, нужно определить конструктор по умолчанию и оператор «". Все основные операторы сравнения для класса map определены.

* Создать пару ключ можно через конструктор или функцией make_pair ();

Разнородный поиск в ассоциативных контейнерах

Упорядоченные ассоциативные контейнеры (сопоставление, мультиотображение, набор и мультинабор) теперь поддерживают разнородный поиск. Это означает, что вам больше не нужно передавать объект точно такого же типа как ключ или элемент в функциях-членах, таких как find() и lower_bound(). Вы можете передать объект любого типа, для которого определен перегруженный operator<, позволяющий выполнять сравнение с типом ключа.

Разнородный поиск включается дополнительно, когда указывается средство сравнения "ромбовидный функтор"std::less<> или std::greater<> при объявлении переменной контейнера, как показано ниже:

```
std::set<BigObject, std::less<» myNewSet;
```

В следующем примере показано, как можно перегрузить operator<, чтобы дать возможность пользователям std::set выполнять поиск, просто передав небольшую строку, которую можно сравнивать с членом BigObject::id каждого объекта.

```
include <set>
include <string>
include <iostream>
include <functional>
class BigObject

public:
string id;
explicit BigObject(const string
s):id(s)
bool operator< (const BigObject
other) const

return this->id < other.id;


// Other members....

inline bool operator<(const string
otherId, const BigObject
obj)

return otherId < obj.id;


inline bool operator<(const BigObject
obj, const string
otherId)

return obj.id < otherId;
```

Контейнеры-адаптеры

Контейнер-адаптер — это разновидность последовательного или ассоциативного контейнера, который ограничивает интерфейс для простоты и ясности. Контейнеры-адаптеры не поддерживают итераторы.

Контейнер queue соответствует семантике FIFO (первым поступил — первым обслужен). Первый элемент передается— то есть, помещается в очередь — должен быть первым извлекается— то есть, удаленных из очереди.

Контейнер priority_queue упорядочен таким образом, что первым в очереди всегда оказывается элемент с наибольшим значением.

Контейнер stack соответствует семантике LIFO (последним поступил — первым обслужен). Последний элемент, отправленный в стек, становится первым извлекаемым элементом.

Поскольку контейнеры-адаптеры не поддерживают итераторы, их нельзя использовать в алгоритмах STL.

Требования для элементов контейнеров

Как правило, элементы, вставленные в контейнер STL, могут быть практически любого типа объекта, если их можно копировать. Элементы, доступные только для перемещения — например, объекты vector<unique_ptr<T»>, создаваемые с помощью unique_ptr<>, — также можно использовать, если вы не вызываете функции-члены, которые пытаются скопировать их.

Деструктору не разрешено вызывать исключение.

Для упорядоченных ассоциативных контейнеров — ранее описанных в этом разделе — необходимо определить открытый оператор сравнения. (По умолчанию это оператор operator<, однако поддерживаются даже типы, которые не работают с operator<.)

Для некоторых операций в контейнерах может также потребоваться открытый конструктор по умолчанию и открытый оператор равенства. Например, неупорядоченным ассоциативным контейнерам требуется поддержка сравнения на равенство и хэширования.

Доступ к элементам контейнера

Доступ к элементам контейнеров осуществляется с помощью итераторов.

Итератор — это объект, который может выполнять итерацию элементов в контейнере STL и предоставлять доступ к отдельным элементам. Все контейнеры STL предоставляют итераторы, чтобы алгоритмы могли получить доступ к своим элементам стандартным способом, независимо от типа контейнера, в котором сохранены элементы.

Вы можете использовать итераторы явно, с помощью члена и глобальных функций, таких как begin() и end(), а также операторов ++ и – для перемещения вперед или назад. Вы можете также использовать итераторы неявно, с циклом range-for или (для некоторых типов итераторов) подстрочным оператором [].

В STL началом последовательности или диапазона является первый элемент. Конец последовательности или диапазона всегда определяется как элемент, следующий за последним элементом. Глобальные функции begin и end возвращают итераторы в указанный контейнер. Типичный цикл явных итераторов, включающий все элементы, выглядит следующим образом:

```
vector<int> vec 0,1,2,3,4 ;
for (auto it = begin(vec); it != end(vec); it++)

// Access element using dereference operator
cout « *it « ;
```

Того же можно достичь более простым способом, с помощью цикла range-for:

```
for (auto num : vec)

// no deference operator
cout « num « ;
```

Существует пять категорий итераторов:

- **Итератор вывода** - Итератор вывода X может выполнить итерацию последовательности с помощью оператора ++ и один раз записать элемент с помощью оператора *.

- **Итератор ввода** - Итератор ввода X может выполнить итерацию последовательности с помощью оператора ++ и прочитать элемент любое количество раз с помощью оператора *. Вы можете сравнить итераторы ввода с помощью операторов ++ и !=. После выполнения приращения любой копии итератора ввода ни одну из других копий нельзя будет безопасно сравнивать, разыменовывать и выполнять приращение.

- **Однонаправленный** - Однонаправленный итератор X может выполнять итерацию последовательности с помощью оператора ++ и прочитать любой элемент или записать неконстантные элементы любое количество раз с помощью оператора *. Вы можете получить доступ к членам элементов с помощью оператора -> и сравнить однонаправленные итераторы с помощью операторов == и !=. Вы можете сделать несколько копий однонаправленного итератора, каждая из

которых может быть разыменована и для нее может быть выполнено независимое приращение. Однонаправленный итератор, который инициализируется без ссылки на какой-либо контейнер, вызывается в пустом однонаправленном итераторе. Пустые однонаправленные итераторы всегда равны.

- **Двунаправленный** - Двунаправленный итератор X может использоваться вместо прямого итератора. Вы можете также выполнить уменьшение двунаправленного итератора, как в –X, X– или (V = *X–). Получить доступ к членам элементов и сравнить двунаправленные итераторы можно так же, как и однонаправленные итераторы.

- **Произвольный доступ** - Итератор произвольного доступа X может использоваться вместо двунаправленного итератора. С итератором произвольного доступа можно использовать подстрочный оператор [] для доступа к элементам. Вы можете использовать операторы +, -, += и -= для перемещения указанного количества элементов вперед или назад, а также для вычисления расстояния между итераторами. Вы можете сравнить двунаправленные итераторы с помощью ==, !=, <, >, <= и >=.

Все итераторы можно назначать и копировать. Они считаются простыми объектами и поэтому часто передаются и возвращаются по значению, а не по ссылке. Обратите внимание, что ни одна из операций, описанных выше, не может создавать исключения при выполнении с допустимым итератором.

Иерархия категорий итераторов может быть представлена в виде трех последовательностей. Для доступа в режиме только для записи в последовательность можно использовать любой из следующих итераторов.

---

output iterator
-> forward iterator
-> bidirectional iterator
-> random-access iterator

Стрелка вправо означает "могут быть заменены". Любой алгоритм, использующий итератор вывода, должен хорошо работать с прямым итератором, например, но не наоборот.

Для доступа в режиме только для чтения в последовательность можно использовать любой из следующих итераторов.

---

input iterator
-> forward iterator
-> bidirectional iterator
-> random-access iterator

Итератор ввода является самым слабым по всем категориям в этом смысле.

Наконец, для доступа в режиме чтения и записи в последовательность можно использовать любой из следующих итераторов.

---

forward iterator
-> bidirectional iterator
-> random-access iterator

Указатель на объект всегда можно использовать как итератор произвольного доступа, поэтому он может относиться к любой категории итераторов, если он поддерживает необходимый уровень доступа для чтения и записи в последовательность, которую он обозначает.

Итератор Iterator, не являющийся указателем на объект, должен также определять типы элементов, необходимые для специализации iterator_traits<Iterator>. Обратите внимание, что эти требования могут быть выполнены путем наследования Iterator от общего базового класса iterator.

Важно знать, каковы возможности и ограничения каждой категории итераторов, чтобы понимать, как итераторы используются контейнерами и алгоритмами в STL.