STL – Standard Template Library

Структура библиотеки

Стандартная библиотека шаблонов содержит пять основных видов компонентов:

- алгоритм (***algorithm***) – определяет вычислительную процедуру;
- контейнер (***container***) – управляет набором объектов в памяти;
- итератор (***iterator***) – обеспечивает для алгоритма средство доступа к содержимому контейнера;
- функциональный объект (***function object***) – инкапсулирует функцию в объекте для использования другими компонентами;
- адаптер (***adaptor***) – адаптирует компонент для обеспечения различного интерфейса.

Алгоритмы

Все алгоритмы отделены от деталей реализации структур данных и используют в качестве параметров типы итераторов.

Для некоторых алгоритмов предусмотрены их копирующие версии. Так, для алгоритма ***algorithm*** его копирующая версия называется ***algorithm_copy***. Копирующие версии алгоритмов, которые берут предикаты, называются ***algorithm_copy_if***.

Контейнеры

Контейнеры – это объекты, которые содержат другие объекты. Контейнеры управляют размещением в памяти и освобождением этих объектов с помощью конструкторов, деструкторов, операций вставки и удаления.

Функция-член ***size*()** возвращает число элементов в контейнере. Её семантика определяется правилами конструкторов, вставок и удалений.

Функция-член ***begin*()** возвращает итератор, ссылающийся на первый элемент в контейнере.

Функция-член ***end*()** возвращает итератор, ссылающийся на элемент, который следует за последним элементом в контейнере.

Если тип итератора контейнера принадлежит к категории двунаправленных итераторов или итераторов произвольного доступа, то контейнер называется ***reversible*** (обратимым).

Последовательности (Sequences)

Последовательность – это вид контейнера, который организует конечное множество объектов одного и того же типа в строгом линейном порядке. Библиотека обеспечивает три основных вида последовательных контейнеров: ***vector*** (вектор), ***list*** (список) и ***deque*** (двусторонняя очередь).

Библиотека также предоставляет контейнерные адаптеры, которые созданы из основных видов последовательностей, такие как ***stack*** (стек), ***queue*** (очередь) и ***priority_queue*** (очередь с приоритетом).

Вектор (Vector)

Стандартный контейнер ***vector*** – вид последовательности, которая поддерживает итераторы произвольного доступа. Кроме того, он поддерживает операции вставки и удаления в конце с постоянным (амортизированным) временем; вставка и удаление в середине занимают линейное время. Управление памятью обрабатывается автоматически.

Список (List)

Стандартный контейнер ***list*** – вид последовательности, которая поддерживает двунаправленные итераторы и операции вставки и удаления с постоянным временем в любом месте последовательности, с управлением памятью, обрабатываемым автоматически. В отличие от векторов и двусторонних очередей, быстрый произвольный доступ к элементам списка не поддерживается, но многим алгоритмам, во всяком случае, только и нужен последовательный доступ.

Двусторонняя очередь (Deque)

Стандартный контейнер ***deque*** – вид последовательности, которая, подобно вектору, поддерживает итераторы произвольного доступа. Кроме того, она поддерживает операции вставки и удаления в начале или в конце за постоянное время; вставка и удаление в середине занимают линейное время. Как с векторами, управление памятью обрабатывается автоматически.

Ассоциативные контейнеры (Associative containers)

Ассоциативные контейнеры – это обобщение понятия ассоциативного массива. Ассоциативный массив, часто называемый ***отображением*** (***map***), а иногда ***словарем*** (***dictionary***), содержит пары значений. Зная одно значение,

называемое ***ключом*** (***key***), можно получить доступ к другому, называемому ***отображённым значением*** (***mapped value***).

Ассоциативные контейнеры обеспечивают быстрый поиск данных, основанных на ключах. Библиотека предоставляет четыре основных вида ассоциативных контейнеров: ***map*** (словарь), ***multimap*** (словарь с дубликатами), ***set*** (множество) и ***multiset*** (множество с дубликатами). Множества можно рассматривать как вырожденные ассоциативные массивы, в которых ключу не соответствует никакого значения, т.е. здесь отслеживаются только ключи.

Ассоциативные контейнеры берут в качестве параметров ключ ***Key*** и упорядочивающее отношение ***Compare***, которое вызывает полное упорядочение по элементам ***Key***. Кроме того, ***map*** и ***multimap*** ассоциируют произвольный тип ***T*** с ***Key***. Объект типа ***Compare*** называется ***сравнивающим объектом*** (***comparison object***) контейнера.

Словарь (Map)

Стандартный контейнер ***map*** – ассоциативный контейнер, который поддерживает уникальные ключи (не содержит ключи с одинаковыми значениями) и обеспечивает быстрый поиск значений другого типа ***T***, связанных с ключами.

Словарь с дубликатами (Multimap)

Стандартный контейнер ***multimap*** – ассоциативный контейнер, который поддерживает равные ключи (возможно, содержит множественные копии того же самого значения ключа) и обеспечивает быстрый поиск значений другого типа ***T***, связанных с ключами.

Множество (Set)

Стандартный контейнер ***set*** – ассоциативный контейнер, который поддерживает уникальные ключи (не содержит ключи с одинаковыми значениями) и обеспечивает быстрый поиск ключей.

Множество с дубликатами (Multiset)

Стандартный контейнер ***multiset*** – ассоциативный контейнер, который поддерживает равные ключи (возможно, содержит множественные копии того же самого значения ключа) и обеспечивает быстрый поиск ключей.

Итераторы

Итераторы – это обобщение указателей, которые позволяют программисту работать с различными структурами данных (контейнерами) единообразным способом. Итераторы – это объекты, которые имеют операторную функцию ***operator****, возвращающую значение некоторого класса или встроенного типа ***T***, называемого ***значимым типом*** (***value type***) итератора. Для каждого типа итератора, для которого определено равенство, имеется соответствующий знаковый целочисленный тип, называемый ***типом расстояния*** (***distance type***) итератора.

Так как итераторы – обобщение указателей, их семантика – обобщение семантики указателей в C++. Это гарантирует, что каждая функция-шаблон, которая использует итераторы, работает и с обычными указателями. Есть пять категорий итераторов в зависимости от операций, определённых для них: ***ввода*** (***input iterators***), ***вывода*** (***output iterators***), ***последовательные*** или ***однонаправленные*** (***forward iterators***), ***двунаправленные*** (***bidirectional iterators***) и ***произвольного доступа*** (***random access iterators***).

Последовательные итераторы удовлетворяют всем требованиям итераторов ввода и вывода и могут использоваться всякий раз, когда определяется тот или другой вид.

Двунаправленные итераторы удовлетворяют всем требованиям последовательных итераторов и могут использоваться всякий раз, когда определяется последовательный итератор.

Итераторы произвольного доступа удовлетворяют всем требованиям двунаправленных итераторов и могут использоваться всякий раз, когда определяется двунаправленный итератор.

Имеется дополнительный атрибут, который могли бы иметь последовательные, двунаправленные и произвольного доступа итераторы, то есть они могут быть ***модифицируемыми*** (***mutable***) или ***постоянными*** (***constant***) в зависимости от того, ведёт ли себя результат ***operator**** как ссылка или как ссылка на константу. Постоянные итераторы не удовлетворяют требованиям итераторов вывода.

Точно так же, как обычный указатель на массив гарантирует, что имеется значение указателя, указывающего за последний элемент массива, так и для любого типа итератора имеется значение итератора, который указывает за последний элемент соответствующего контейнера. Эти значения называются ***законечными*** (***past-the-end***) значениями.

Значения итератора, для которых ***operator**** определён, называются ***разыменовываемыми*** (***dereferenceable***). Библиотека никогда не допускает, что законечные значения являются разыменовываемыми.

Итераторы могут также иметь ***исключительные*** (***singular***) значения, которые не связаны ни с каким контейнером. Например, после объявления неинициализированного указателя (например, ***int* p;***), всегда должно предполагаться, что ***p*** имеет исключительное значение указателя. Разыменовываемые и законечные значения всегда являются неисключительными.

Чтобы шаблоны алгоритмов могли работать непосредственно с потоками ввода-вывода, предусмотрены соответствующие классы-шаблоны, подобные итераторам.

Итератор входного потока ***istream_iterator<T>*** читает (используя ***operator>>***) последовательные элементы из входного потока, для которого он был создан. После своего создания итератор каждый раз при использовании ++ читает и сохраняет значение ***T***. Если достигнут конец потока, итератор становится равным значению ***end-of-stream*** (***конец-потока***). Конструктор без параметров ***istream_iterator()*** всегда создает итераторный объект конца потокового ввода, являющийся единственым законным итератором, который следует использовать для конечного условия. Результат ***operator**** для конца потока не определён, а для любого другого значения итератора возвращается ***const T&***.

```
template <class T, class Distance = ptrdiff_t>
class istream_iterator : public input_iterator<T, Distance>
{
  friend bool operator==(const istream_iterator<T,
                         Distance>& x,
                         const istream_iterator<T, Distance>& y);
  public:
  istream_iterator();
```

```
  istream_iterator(istream& s);
  istream_iterator(const istream_iterator<T, Distance>& x);
  ~istream_iterator();
  const T& operator*() const;
  istream_iterator<T, Distance>& operator++();
  istream_iterator<T, Distance> operator++(int);
};
```

Итератор выходного потока ***ostream_iterator<T>*** записывает (используя ***operator<<***) последовательные элементы в выходной поток, из которого он был создан. Если он был создан с параметром конструктора ***char****, эта строка, называемая ***строкой разделителя*** (***delimiter string***), записывается в поток после того, как записывается каждое ***T***.

```
template <class T>
class ostream_iterator : public output_iterator
{
  public:
  ostream_iterator(ostream& s);
  ostream_iterator(ostream& s, const char* delimiter);
  ostream_iterator(const ostream_iterator<T>& x);
  ~ostream_iterator();
  ostream_iterator<T>& operator=(const T& value);
  ostream_iterator<T>& operator*();
  ostream_iterator<T>& operator++();
  ostream_iterator<T>& operator++(int);
};
```

Функциональные объекты

Функциональные объекты – это объекты, для которых определена операторная функция ***operator()***. Там, где ожидается передача указателя на функцию алгоритмическому шаблону, интерфейс установлен на приём объекта с определённым ***operator()***. Это не только заставляет алгоритмические шаблоны работать с указателями на функции, но также позволяет им работать с произвольными функциональными объектами. Например, если необходимо поэлементно сложить два вектора ***a*** и ***b***, содержащие ***double***, и поместить результат в ***a***, это можно сделать так:

```
transform(a.begin(), a.end(), b.begin(), a.begin(), plus<double>());
```

Адаптеры

Адаптеры – классы-шаблоны, которые обеспечивают отображения интерфейса. Например, ***insert_iterator*** обеспечивает контейнер интерфейсом итератора вывода.

Адаптеры контейнеров (Container adaptors)

Часто бывает полезно обеспечить ограниченные интерфейсы контейнеров. Библиотека предоставляет ***stack***, ***queue*** и ***priority_queue*** через адаптеры, которые могут работать с различными типами последовательностей.

Стек (Stack)

Любая последовательность, поддерживающая операции ***back*()**, ***push_back*()** и ***pop_back*()**, может использоваться для модификации ***stack***. В частности, могут использоваться ***vector***, ***list*** и ***deque***.

Например, ***stack*< *vector*<*int*> >** – целочисленный стек, созданный из ***vector***, а ***stack*< *deque*<*char*> >** – символьный стек, созданный из ***deque***.

Очередь (Queue)

Любая последовательность, поддерживающая операции ***front*()**, ***push_back*()** и ***pop_front*()**, может использоваться для модификации ***queue***. В частности, могут использоваться ***list*** и ***deque***.

Очередь с приоритетами (Priority queue)

Любая последовательность, с итератором произвольного доступа и поддерживающая операции ***front*()**, ***push_back*()** и ***pop_front*()**, может использоваться для модификации ***priority_queue***. В частности, могут использоваться ***vector*** и ***deque***.

Адаптеры итераторов (Iterator adaptors)

Обратные итераторы (Reverse iterators)

Двунаправленные итераторы и итераторы произвольного доступа имеют соответствующие адаптеры обратных итераторов, которые выполняют итерации через структуру данных в противоположном направлении. Они имеют те же самые сигнатуры, что и соответствующие итераторы. Фундаментальное соотношение между обратным итератором и его соответствующим итератором ***i*** установлено тождеством ***&*(reverse_iterator(i)) = &*(i - 1)***.

Итераторы вставки (Insert iterators)

Чтобы было возможно иметь дело с вставкой таким же образом, как с записью в массив, в библиотеке обеспечивается специальный вид адаптеров итераторов, называемых ***итераторами вставки*** (***insert iterators***).

Итератор вставки создается из контейнера и, возможно, одного из его итераторов, указывающих, где вставка происходит, если это ни в начале, ни в конце контейнера. Итераторы вставки удовлетворяют требованиям итераторов вывода. Он подобен курсору, указывающему в контейнере, где происходит вставка.

Итератор ***back_insert_iterator*** вставляет элементы в конце контейнера, ***front_insert_iterator*** вставляет элементы в начале контейнера, а ***insert_iterator*** вставляет элементы, куда итератор указывает в контейнере.

Адаптеры функций (Function adaptors)

Адаптеры функций работают только с классами функциональных объектов с определенными типами параметров и типом результата. В библиотеке адаптеры представлены отрицателями (***Negators***) и привязками (***Binders***).