Стандартная библиотека

В любой программе, кроме операторов языка, используются средства библиотек, включаемых в среду программирования. Различные среды предоставляют в распоряжение программиста разные наборы средств, облегчающих создание программ, – например, компиляторы Microsoft Visual C++ и Qt C++ содержат библиотеки классов для написания приложений Windows.

Часть библиотек стандартизована, то есть должна поставляться с любым современным компилятором языка C++.

Стандартную библиотеку C++ можно условно разделить на две части:

- **К первой части относятся функции, макросы, типы и константы, унаследованные из библиотеки C;**
- **Ко второй – классы и другие средства C++, она содержит классы, шаблоны и другие средства для ввода, вывода, хранения и обработки данных как стандартных типов, так и типов, определенных пользователем.**

Классы стандартной библиотеки можно разделить на группы в соответствии с их назначением:

- **Потоковые классы предназначены для управления потоками данных между оперативной памятью и внешними устройствами (например, дисками и консолью), а также в пределах оперативной памяти,**
- **Строковый класс предназначен для удобной и защищенной от ошибок работы с символьными строками,**
- **Контейнерные классы реализуют наиболее распространенные структуры для хранения данных – например, списки, вектора и множества. В библиотеку входят также алгоритмы, использующие эти контейнеры,**
- **Итераторы предназначены для унифицированного доступа к компонентам контейнерных и других классов,**
- **Математические классы поддерживают эффективную обработку массивов с плавающей точкой и работу с комплексными числами,**
- **Диагностические классы обеспечивают динамическую идентификацию типов и объектно-ориентированную обработку ошибок,**
- **Остальные классы обеспечивают динамическое распределение памяти, адаптацию к локальным особенностям, обработку функциональных объектов и т.д.**

Стандартная библиотека

Для использования средств стандартной библиотеки в программу требуется включить с помощью директивы #include соответствующие заголовочные файлы.

Например, потоки описаны в <iostream>, а списки – в <list>.

Компоненты заголовочных файлов без расширения .h определены в пространстве имен std, а одноименные файлы с расширением .h – в глобальном пространстве имен.

Имена заголовочных файлов C++ для функций библиотеки C, определенные в пространстве имен std, начинаются с буквы c, например, <cstdio>, <cstring>, <ctime>. Для каждого заголовочного файла вида <cX> существует файл <X.h>, определяющий те же имена в глобальном пространстве имен.

Наиболее внимательно, во избежание путаницы, надо работать с именами заголовков строк: string – заголовок для работы со строковыми классами C++, cstring – версия заголовка функций C в пространстве имен std, string.h – то же самое в глобальном пространстве имен, cstring.h – старый заголовок для работы со строковыми классами C++ в глобальном пространстве имен в одной из оболочек.

Часть стандартной библиотеки, в которую входят контейнерные классы, алгоритмы и итераторы, называют Стандартной библиотекой шаблонов (STL – Standard Template Library).

Рассмотрим инструментальные средства C++ используемые для хранения компонентов в библиотеках.

Контейнеры

В реальных задачах обычно требуется обрабатывать группы данных довольно большого объема. Поэтому в любом языке программирования, в том числе и в C++, существуют средства объединения данных в группы — обычно это массивы. Но массивы являются слишком простыми, а потому очень ненадежными конструкциями и одних массивов как средств объединения однородных данных явно недостаточно.

В процессе развития языков программирования, и C++ в частности, выработана более общая конструкцию объединения однородных данных в группу – контейнер.

Контейнеры (containers) – это объекты, хранящие внутри себя другие объекты.

Контейнер – это специализированная коллекция, являющаяся "логическим" хранилищем одного или нескольких компонентов.

Контейнеры управляют взаимодействием компонентов друг с другом и с внешней средой приложения. С помощью контейнеров можно отслеживать компоненты по принципу "первым поступил – первым обслужен" (first-in, first-out, FIFO) и обращаться к компонентам по индексу.

Контейнеры также предоставляют средства для удаления компонентов, которые больше не нужны.

Контейнер – это не визуальное или физическое, а логическое хранилище компонентов.

Контейнер инкапсулирует один или несколько компонентов и предоставляет оболочку, через которую могут взаимодействовать клиенты.

Контейнеры

По способу доступа к компонентам контейнеры могут быть:

Прямого доступа – обеспечивают доступ по номеру (по индексу) компонента, аналогично обращению к компонентам массива – v[7].

Последовательного доступа – обеспечивают хранение конечного количества однотипных величин в виде непрерывной последовательности.

Ассоциативного доступа – похожи на контейнеры прямого доступа, но обеспечивают быстрый доступ к данным по ключу.

Каждый контейнер характеризуется именем и типом входящих в него компонентов.

Имя контейнера – это имя переменной в программе, которое подчиняется правилам видимости C++. Как объект, контейнер имеет временем жизни, оно не зависит от времени жизни его компонентов.

Тип контейнера складывается из типа самого контейнера и типа входящих в него компонентов. Тип контейнера – это не тип его компонентов. Тип контейнера определяет способ доступа к компонентам.

Размер контейнера может быть либо определен при объявлении, либо не задан. В первом случае получаем контейнер фиксированной длины. Во втором случае имеем контейнер переменного размера, количество компонентов которого изменяется во время работы программы.

Контейнеры

Операции с контейнером

Все операции с контейнером можно разделить на несколько групп:

- операции с контейнером как объектом;
- операции доступа к компонентам, включая операцию замены значений компонентов;
- операции добавления и удаления отдельных компонентов или групп компонентов;
- операции поиска компонентов и групп компонентов;
- прочие (специальные) операции, зависящие от вида контейнера.

Одна из основных операций с контейнером как объектом – объединение двух контейнеров с получением нового контейнера, она может быть реализована в различных вариантах:

- простое сцепление двух контейнеров, в новый контейнер попадают все компоненты и первого, и второго контейнеров; операция не коммутативна;
- объединение упорядоченных контейнеров, называемое слиянием, в новый контейнер попадают все компоненты первого и второго контейнеров; объединенный контейнер упорядочен; операция коммутативна;
- объединение двух контейнеров как объединение множеств, в новый контейнер попадают только те компоненты, которые есть хотя бы в одном контейнере; операция коммутативна;
- объединение двух контейнеров как пересечение множеств, в новый контейнер попадают только те компоненты, которые есть в обоих контейнерах; операция коммутативна.

Контейнеры
Операции с контейнером

Одной из операций с контейнером является извлечение из него части компонентов и создание из них нового контейнера. Часто эту операцию выполняет конструктор, а требуемая часть контейнера задается двумя итераторами.

Операции доступа к компонентам контейнера рассмотрены ранее.

Операции добавления и удаления компонентов работают только для контейнера переменного размера. Очевидно, что эти операции с компонентами можно выполнять разными способами:

- **добавлять и удалять компоненты в начале контейнера;**
- **то же самое делать в "хвосте" контейнера;**
- **вставлять компоненты перед текущим компонентом или после него, удалять текущий компонент;**
- **делать вставки в соответствии с некоторым порядком сортировки компонентов контейнера, в этом случае обязательно выполняется операция поиска;**
- **удалять компонент, содержимое которого равно заданному, в этом случае "за кадром" тоже работает операция поиска.**

Первые три операции обычно применяются к последовательным контейнерам.

Если же контейнер ассоциативный, то он упорядочен по полю доступа, поэтому операции вставки и удаления всегда выполняются в последних вариантах.

Но и последовательный контейнер может быть отсортирован, поэтому операция вставки тоже может вставлять "по порядку".

Контейнеры-множества. Множество – это контейнер, в котором каждый компонент единственный. Помимо операций объединения и пересечения, для контейнеров-множеств реализуется операция вычитания множеств: в контейнер-результат попадают только те компоненты первого контейнера, которых нет во втором; операция не коммутативна. Очень часто с множествами выполняется операция проверки включения, которая фактически является операцией поиска (как отдельного компонента, так и подмножества компонентов).

Тип компонентов оказывает существенное влияние на то, какие операции могут выполняться с контейнером. Например, для строк операция сортировки обычно не нужна, а для числовых контейнеров или для списка счетов в банке такая операция может быть не только полезной, но и необходимой.

Контейнеры

Реализация контейнеров

Контейнеры, как правило, реализуются с помощью указателей и динамической памяти. Использование указателей и динамических переменных в классах в сочетании с перегрузкой операций представляет собой удивительно мощный механизм создания новых типов данных – контейнеры стандартной библиотеки тому наглядный пример.

При реализации контейнеров надо определить:

1. **Способ выделения памяти**
   - **Выделение памяти операцией new[] обеспечивает выделение непрерывной области памяти. Количество компонентов обычно задается выражением, вычисляемым во время работы программы. Такая форма практически всегда используется для реализации динамических массивов.**
   - **Второй способ распределения памяти – выделение одиночного компонента операцией new, для этого обычно требуется реализация контейнеров с переменным количеством компонентов. При этом не только память для компонента выделяется динамически, но и в состав самого компонента входят один или несколько (чаще всего два) указателей для связи компонентов друг с другом.**
2. **Способ освобождения памяти**

   Выделением памяти, как правило, занимается конструктор контейнера. Возвращение памяти обычно возлагается на деструктор. Операции возврата памяти являются парными для операций выделения памяти: если память выделялась операцией new, то возвращать память нужно операцией delete; если же память выделялась массивом (операцией new[]), то и возвращать ее нужно соответствующей операцией delete[].
3. **Способ копирования и присваивания**
   - **Для динамических классов, в которых используются указатели, стандартный копирующий конструктор (по умолчанию поэлементное копирование полей класса - поверхностное копирование - shallow copying) не подходит, требуется реализация глубокого копирования (deep copying), иначе могут возникнуть висячие и потерянные ссылки.**
   - **Для операций присваивания также требуется глубокое копирование.**

Итераторы

Для работы с контейнерами используется ряд инструментов. Один из них: Итераторы (iterator) – это объекты, напоминающие указатели. Они позволяют перемещаться по содержимому контейнера так, как указатель перемещается по элементам массива. При помощи итераторов можно просматривать контейнеры, не заботясь о фактических типах данных, используемых для доступа к компонентам.

Существует пять видов итераторов:

Итератор	Вид доступа
Итератор произвольного доступа	Хранит и извлекает значения. Обеспечивает произвольный доступ к компонентам.
Двунаправленный итератор	Хранит и извлекает значения. Перемещается вперёд и назад.
Прямой итератор	Хранит и извлекает значения. Перемещается только вперёд.
Итератор ввода	Извлекает, но не хранит компоненты. Перемещается только вперед.
Итератор вывода	Хранит, но не извлекает значения. Перемещается только вперед.

Итератор – это обобщение понятия указателя для работы с различными структурами данных стандартным способом. Для того, чтобы можно было реализовать алгоритмы, корректно и эффективно работающие с данными различной структуры, стандарт определяет не только интерфейс, но и требования ко времени доступа с помощью итераторов.

Итераторы действуют как указатели. Их можно увеличивать, уменьшать и разыменовывать, применяя оператор "*".

Поскольку итератор является обобщением понятия "указатель", семантика у них одинаковая, и все функции, принимающие в качестве параметра итераторы, могут также работать и с обычными указателями.

Алгоритмы

В ряду инструментов работы с контейнерами важное место занимают:

Алгоритмы (algorithms) – они выполняют операции над содержимым контейнеров. Существуют алгоритмы для инициализации, сортировки, поиска или замены содержимого контейнеров. Ряд алгоритмы предназначены для работы с последовательностью (sequence), которая представляет собой линейный список компонентов внутри контейнера.

Каждый алгоритм реализован в виде шаблона или набора шаблонов функции, поэтому может работать с различными видами последовательностей и данными разнообразных типов.

Все алгоритмы можно разделить на несколько категорий:

- ✔ **немодифицирующие операции с последовательностями;**
- ✔ **модифицирующие операции с последовательностями;**
- ✔ **алгоритмы, связанные с сортировкой;**
- ✔ **алгоритмы работы с множествами и пирамидами;**
- ✔ **Кроме того, библиотека содержит обобщенные численные алгоритмы.**

В качестве параметров алгоритму передаются итераторы, определяющие начало и конец обрабатываемой последовательности. Вид итераторов определяет типы контейнеров, для которых может использоваться данный алгоритм.

Например, алгоритм сортировки (sort) требует для своей работы итераторы произвольного доступа, поэтому он не будет работать с контейнером list.

Алгоритмы не выполняют проверку выхода за пределы последовательности.

Другие инструменты библиотек

Кроме контейнеров, алгоритмов и итераторов в библиотеках предусмотрены другие стандартные инструменты.

Наиболее важными среди них являются распределители (allocators), предикаты (predicates), функции сравнения (comparison function) и функторы (function objects). (Иногда функторы называют объектами-функциями).

Каждый итератор имеет распределитель, подобранный именно для него. Распределители управляют выделением памяти для контейнера. Распределитель, предусмотренный по умолчанию, является объектом класса allocator.

Некоторые алгоритмы и контейнеры используют специальный тип функции, называемый предикатом. Существуют два вида предикатов: унарные и бинарные. Эти функции возвращают логические значения true или false. Условия, от которых зависят логические значения предикатов, программист может формулировать самостоятельно. В дальнейшем унарные предикаты мы будем относить к типу UnPred, а бинарные – к типу BinPred Аргументы бинарных предикатов всегда перечисляются по порядку: first, second. Аргументами как унарных, так и бинарных предикатов являются объекты, хранящиеся в контейнере.

Некоторые алгоритмы и классы используют специальную разновидность бинарных предикатов, предназначенных для сравнения двух компонентов, – функции сравнения. Они возвращают значение true, если первый аргумент меньше второго.

Кроме заголовков, характерных для разных шаблонных классов, в библиотеках используются заголовки <utility> и <functional>.

Шаблоны, определенные в заголовке <functional>, позволяют создавать объекты, в которых определена операторная функция operator(). Такие объекты называются функторами. Их часто применяют, в частности в шаблонных алгоритмах, вместо указателей на функции. Использование функторов вместо указателей на функции повышает эффективность кода.

Библиотека STL

Структура библиотеки

Библиотека 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.

Такое разделение позволяет уменьшить количество компонентов.

Например, вместо написания функции поиска элемента для каждого вида контейнера можно создать единственную версию, которая работает с каждым из них, пока удовлетворяется основной набор требований.

Следующее описание разъясняет структуру библиотеки:

Если программные компоненты сведены в таблицу как трёхмерный массив, где одно измерение представляет различные типы данных (например, int, double), второе измерение представляет различные контейнеры (например, вектор, связный список, файл), а третье измерение представляет различные алгоритмы с контейнерами (например, поиск, сортировка, перемещение по кругу). Если i, j и k - размеры измерений, тогда должно быть разработано i*j*k различных версий кода. При использовании же шаблонных функций, которые берут параметрами типы данных, нужно только j*k версий. Далее, если заставить алгоритмы работать с различными контейнерами, то нужно просто j+k версий.

Это значительно упрощает разработку программ, а также позволяет очень гибким способом использовать компоненты в библиотеке вместе с определяемыми пользователем компонентами. Пользователь может легко определить специализированный контейнерный класс и использовать для него библиотечную функцию сортировки. Для сортировки пользователь может выбрать какую-то другую функцию сравнения либо через обычный указатель на сравнивающую функцию, либо через функциональный объект (объект, для которого определён operator()), который сравнивает. Если пользователю необходимо выполнить передвижение через контейнер в обратном направлении, то используется адаптер reverse_iterator.

Библиотека STL

В основе STL (Standard Template Library) – стандартной библиотеки шаблонов C++ лежит технология контейнеров.

В STL различают:

- **Контейнеры последовательного доступа, которые обеспечивают и прямой, и последовательный варианты доступа. К ним относятся векторы (vector), двусторонние очереди (deque) и списки (list), а также так называемые адаптеры, то есть варианты, контейнеров – стеки (stack), очереди (queue) и очереди с приоритетами (priority_queue).**
- **Ассоциативные контейнеры: словари (tap), словари с дубликатами (multimap), множества (set), множества с дубликатами (multiset) и битовые множества (bitset).**

Библиотека STL

Библиотека расширяет основные средства C++ последовательным способом, так что программисту на C/C++ легко начать пользоваться библиотекой.
Например, библиотека содержит шаблонную функцию merge (слияние).
Когда пользователю нужно два массива *a* и *b* объединить в *c*, то это может быть выполнено так:

```
int a[1000];
int b[2000];
int c[3000];
 ...
merge (a, a+1000, b, b+2000, c);
```

Когда пользователь хочет объединить вектор и список (оба - шаблонные классы в библиотеке) и поместить результат в заново распределённую неинициализированную память, то это может быть выполнено так:

```
vector<Employee> a;
list<Employee> b;
 ...
Employee* c = allocate(a.size() + b.size(), (Employee*) 0);
merge(a.begin(), a.end(), b.begin(), b.end(),
                raw_storage_iterator <Employee*, Employee> (c));
```

где begin() и end() - функции-члены контейнеров, которые возвращают правильные типы итераторов или указателеподобных объектов, позволяющие merge выполнить задание, а raw_storage_iterator - адаптер, который позволяет алгоритмам помещать результаты непосредственно в неинициализированную память, вызывая соответствующий конструктор копирования.

Библиотека STL

Более сложный пример – фильтрующая программа, которая берёт файл и беспорядочно перетасовывает его строки:

```
main(int argc, char**)
 {
   if(argc != 1) throw("usage: shuffle\n");
   vector<string>  v;
   copy(istream_iterator<string> (cin),istream_iterator<string>(),
                                  inserter(v, v.end()));
  random_shuffle(v.begin(), v.end());
  copy(v.begin(), v.end(), ostream_iterator<string>(cout));
 }
```

В этом примере copy перемещает строки из стандартного ввода в вектор, но так как вектор предварительно не размещён в памяти, используется итератор вставки, чтобы вставить в вектор строки одну за другой. (Эта методика позволяет всем функциям копирования работать в обычном режиме замены также, как в режиме вставки.) Потом random_shuffle перетасовывает вектор, а другой вызов copy копирует его в поток cout.