Стандартная библиотека C++. Standard Template Library (STL)

Состав стандартной библиотеки

Стандартная библиотека C++:

1. Стандартная библиотека C (cmath, cstdlib и др.). При подключении отдельных файлов
2. Библиотека C++.

Стандартная библиотека шаблонов (STL) — подмножество стандартной библиотеки C++. Содержит контейнеры, алгоритмы, итераторы, объекты-функции и др. Иногда термин STL используется вместе (или попеременно) с термином «Стандартная библиотека C++».

Помимо STL, в стандартную библиотеку C++ входят потоки, строки C++, многопоточное программирование (thread, не рассматриваются в этом курсе). Содержимое стандартной библиотеки C++ относится к пространству имен std.

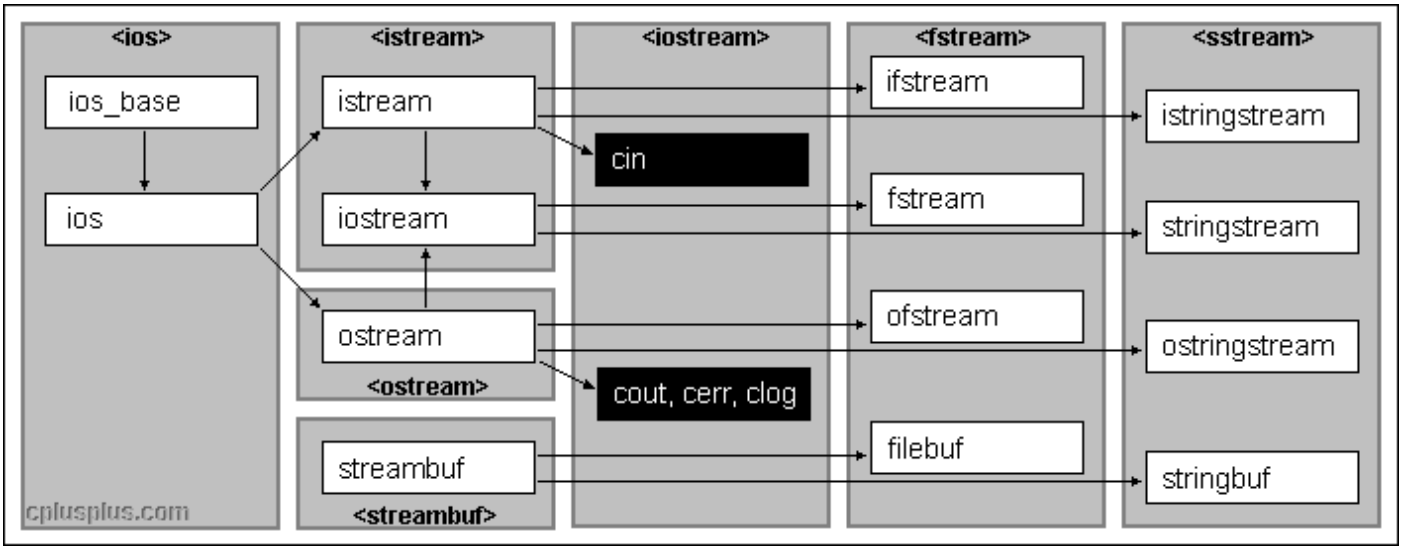
Документация https://cplusplus.com/reference/

Потоки (stream)

Библиотека ввода-вывода — это объектно-ориентированная библиотека, обеспечивающая функционал ввода и вывода с использованием потоков.

Поток — это абстракция, представляющая устройство, на котором выполняются операции ввода и вывода. Потоки, как правило, связаны с физическим источником или местом назначения символов: файл, клавиатура или консоль.

Потоковые классы наследуются от базового класса ios, который является наследником класса ios_base, в котором определены общие операции ввода-вывода.



Для потоков перегружены операции << и >>.

Виды потоков:

- входные
- выходные
- двунаправленные

Манипуляторы

Для всех потоков можно использовать манипуляторы, влияющие на форматирование ввода и вывода. Примеры манипуляторов:

- setw — установка ширины поля вывода.
- endl — отправка в поток символа конца строки '\n'.
- ends — отправка в поток нуль-символа '\0'.
- setprecision — устанавливает формат вывода чисел с плавающей точкой.
- uppercase — для вывода результата в верхнем регистре.

Для работы с некоторыми манипуляторами обязательно подключать `#include <iomanip>`.

Пример работы с манипуляторами 1_3_IOmanip.cpp

Стандартные потоки

Предназначены для обеспечения ввода и вывода из стандартных источников. Для работы со стандартными потоками нужно подключить `#include <iostream>`.

Предопределенные стандартные потоки:

- cin — стандартный входной поток.
- cout — стандартный выходной поток.
- cerr — стандартный выходной поток ошибок.
- clog — стандартный выходной поток логов.

Стандартные потоки по умолчанию связаны с клавиатурой (входной поток) и консолью (выходные потоки), но их можно переопределить, например, связав с файлом.

Файловые потоки

Файловые потоки — это объекты C++ для работы с файлами. Как только файловый поток используется для открытия файла, любая операция ввода или вывода, выполняемая с этим потоком, физически отражается в файле. Для работы с файловыми потоками нужно подключить `#include <fstream>`.

Для создания файловых потоков нужно объявлять переменные соответствующих классов и связывать их с конкретными файлами. По умолчанию файлы открываются как текстовые, возможности можно менять с помощью флагов. Флаги открытия файлов:

- app — открытие на дозапись. Индикатор устанавливается в конец потока перед каждой операцией вывода.
- ate — индикатор устанавливается в конец потока при открытии файла.
- binary — открытие файла в двоичном формате.
- in — разрешает ввод данных в файл, всегда установлен для ifstream.
- out — разрешает вывод данных из файла, всегда установлен для ofstream.
- trunc — разрешает перезапись данных в файле.

Флаги можно комбинировать с помощью побитового ИЛИ (|).

Подробнее про режимы открытия https://cplusplus.com/reference/ios/ios_base/openmode/

Пример 5_5_Files.cpp

Строковые потоки

Предназначены для удобных операций со строками. Для работы со строковыми потоками нужно подключить `#include <sstream>`.

Пример 7_Classes.cpp

Стандартная библиотека шаблонов (Standard Template Library, STL)

Контейнеры

Контейнеры предназначены для хранения больших объемов одинаковых данных. В C++ контейнеры реализованы в виде шаблонов классов. Контейнер управляет пространством хранения своих элементов и предоставляет функции-члены для доступа к ним либо напрямую, либо через итераторы.

Контейнеры предоставляют удобную реализацию структур, часто используемых в программировании:

- массивы (array);
- динамические массивы (vector);
- очереди (queue);
- стеки (stack);
- кучи (priority_queue);
- связанные списки (list);
- деревья (set);
- ассоциативные массивы (map).

Основное достоинство контейнеров в их повышенной надежности и переносимости.

Существуют два типа контейнеров:

- Последовательные
- Ассоциативные

Многие контейнеры имеют несколько общих функций-членов и обладают общими функциональными возможностями. Решение о том, какой тип контейнера использовать для конкретной задачи, как правило, зависит не только от функциональности, но и от эффективности его функциональных элементов. Это особенно актуально для последовательных контейнеров, которые часто являются компромиссами между сложностью вставки/удаления элементов и скорости доступа к ним.

Container	Insert Head	Insert Tail	**Insert**	Remove Head	Remove Tail	**Remove**	Index Search	Find
vector	n/a	O(1)	O(n)	O(1)	O(1)	O(n)	O(1)	O(log n)
list	O(1)	O(1)	O(1)	O(1)	O(1)	O(1)	n/a	O(n)
deque	O(1)	O(1)	n/a	O(1)	O(1)	O(n)	n/a	n/a
queue	n/a	O(1)	n/a	O(1)	n/a	n/a	O(1)	O(log n)
stack	O(1)	n/a	n/a	O(1)	n/a	n/a	n/a	n/a
map	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)	O(log n)
multimap	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)*	O(log n)
set	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)	O(log n)
multiset	n/a	n/a	O(log n)	n/a	n/a	O(log n)	O(1)*	O(log n)

Последовательные контейнеры

В последовательных контейнерах элементы расположены последовательно и упорядочены, у каждого элемента имеется номер, по которому к нему можно обращаться.

Примеры последовательных контейнеров:

- Массив (array) — это последовательный контейнер фиксированного размера. Содержат определенное количество элементов, упорядоченных в строгой линейной последовательности. Поддерживает произвольный доступ к любому элементу в контейнере. Добавлять или удалять элементы из контейнера нельзя https://cplusplus.com/reference/array/array/ .
- Вектор (vector) — это последовательный контейнер переменного размера. Поддерживает произвольный доступ к любому элементу в контейнере. Обеспечивает добавление и удаление элементов из любого места контейнера www.cplusplus.com/reference/vector/vector/

- Список (list) — двухсвязный список. Поддерживает только последовательный двухнаправленный доступ к элементам. Обеспечивает удаление и добавление элементов в начале и в конце контейнера www.cplusplus.com/reference/list/list/
- Дек (deque) — двусторонняя очередь. Поддерживает произвольный доступ к любому элементу в контейнере. Обеспечивает удаление и добавление элементов в начале и в конце контейнера www.cplusplus.com/reference/deque/deque/

Также существуют так называемые адаптеры контейнеров (container adaptor). Технически они не являются контейнерами, а инкапсулируют один из вышеописанных контейнеров (например, вектор) и позволяют работать с этими контейнерами определенным образом. Это следующие типы

- std::stack<>: представляет структуру данных «стек».
- std::queue<>: представляет структуру данных «очередь».
- std::priority_queue<>: также представляет очередь, но при этому ее элементы имеют приоритеты.

Пример работы с вектором 9_1_Vector.cpp

Ассоциативные контейнеры

Ассоциативные контейнеры представляют такие контейнеры, где с каждым элементом ассоциирован некоторый ключ, и этот ключ применяется для доступа к элементу в контейнере.

Примеры ассоциативных контейнеров:

- Словарь (map) — контейнер, где хранятся пары ключ-значение, где и ключ, и значение могут быть любых типов, однако ключ должен быть уникальным www.cplusplus.com/reference/map/
- Множество (set) — позволяет хранить только уникальные значения, не имеющие дубликатов www.cplusplus.com/reference/set/set/

Поскольку доступ к элементам ассоциативных контейнеров по индексу невозможен, для работы с ними используются итераторы.

Итераторы

Итераторы обеспечивают доступ к элементам контейнера. В C++ итераторы реализуют общий интерфейс для различных типов контейнеров, что позволяет использовать единой подход для обращения к элементам разных типов контейнеров.

Стоит отметить, что итераторы имеют только контейнеры, адаптеры контейнеров — типы std::stack, std::queue и std::priority_queue итераторов не имеют.

Для каждого типа контейнеров необходимо объявлять свой итератор. Например, для контейнера `vector<int>` итератор будет объявляться как `vector<int>::iterator`.

Возможности итераторов зависят от контейнера, с которым они используются. Например, итераторы begin() и end() позволяют получить доступ к первому и последнему элементам контейнера. Операция `++` доступна для всех итераторов, однако операция `--` не будет работать для однонаправленных контейнеров.

Подробнее про итераторы www.cplusplus.com/reference/iterator/

Пример работы со словарем 9_2_Map.cpp

Алгоритмы

Еще одна часть STL, содержит готовые реализации стандартных алгоритмов. Состав www.cplusplus.com/reference/algorithm/

Для выполнения заданий потребуются:

1. generate (синтаксис и пример www.cplusplus.com/reference/algorithm/generate/). Заполняет существующий контейнер значениями. Принимает три параметра:
    1. Прямой итератор начала диапазона (контейнер.begin()).
    2. Прямой итератор конца диапазона (контейнер.end()).
    3. Функция или функтор (см.стр.122 Лаптев и др.), возвращающая значение нужного типа.
2. for_each (синтаксис и пример www.cplusplus.com/reference/algorithm/for_each/). Это не оператор цикла! Применяет функцию к каждому элементу контейнера. Принимает три параметра:
    1. Прямой итератор начала диапазона (контейнер.begin()).
    2. Прямой итератор конца диапазона (контейнер.end()).
    3. Функция или функтор (см.стр.122 Лаптев и др.), выполняющая действие.