Лекция 7.
Библиотеки STL

3 семестр

Лектор: ст.пр. Бельченко Ф.М.

Библиотека STL

Механизм шаблонов встроен в компилятор C++, чтобы дать возможность программистам делать свой код короче за счет обобщенного программирования. Естественно, существуют и стандартные библиотеки, реализующие этот механизм. STL является самой эффективной библиотекой C++ на сегодняшний день.

Содержимое STL

Контейнер (англ. *container*) — хранение набора объектов в памяти.

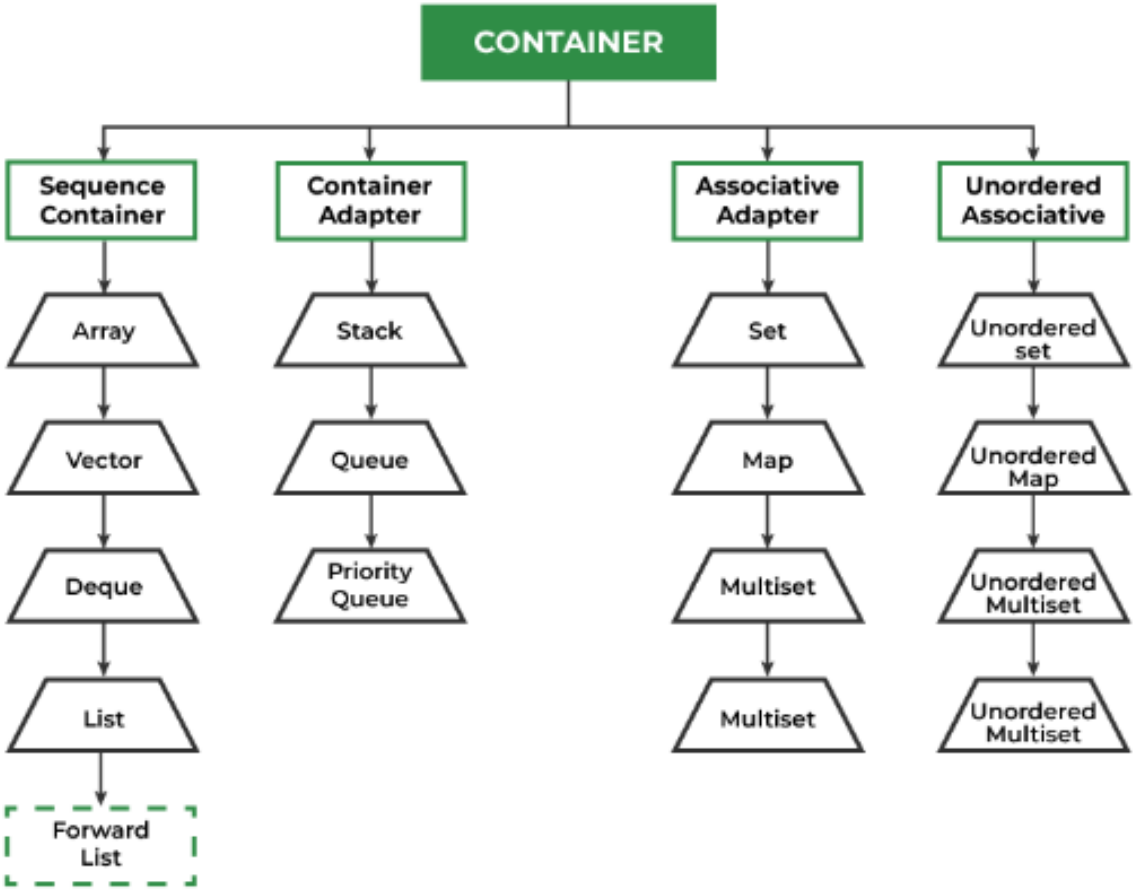
Итератор (англ. *iterator*) — обеспечение средств доступа к содержимому контейнера.

Алгоритм (англ. *algorithm*) — определение вычислительной процедуры.

Адаптер (англ. *adaptor*) — адаптация компонентов для обеспечения различного интерфейса.

Функциональный объект (англ. *functor*) — сокрытие функции в объекте для использования другими компонентами.

Содержимое STL



Историческая справка

Примечание

Реализация стандартной библиотеки C++ майкрософт часто называется библиотекой *шаблонов STL или* Standard. Хотя *стандартная библиотека C++ является официальным именем библиотеки* , как определено в ISO 14882, из-за популярного использования STL и "Стандартной библиотеки шаблонов" в поисковых системах, мы иногда используем эти имена, чтобы упростить поиск нашей документации. С исторической точки зрения , "STL" первоначально ссылается на стандартную библиотеку шаблонов, написанную Александром Стефановым. Части этой библиотеки были стандартизированы в стандартной библиотеке C++ вместе с библиотекой среды выполнения ISO C, частями библиотеки Boost и другими функциями. Иногда "STL" также используется для ссылки на контейнеры и алгоритмы стандартной библиотеки C++, адаптированной из STL Стефанова. В этой документации стандартная библиотека шаблонов (STL) ссылается на стандартную библиотеку C++ в целом.

Источник: https://learn.microsoft.com/ru-ru/cpp/standard-library/cpp-standard-library-overview?view=msvc-170

Контейнеры

Контейнер — это класс STL, реализующий функциональность некоторой структуры данных, то есть хранилища нескольких элементов. Примеры разных контейнеров: vector, stack, queue, deque, string, set, map и т.д.

Различные контейнеры имеют различные способы доступа к элементом. Например, vector и deque предоставляют так называемый "произвольный доступ" ("random access"), позволяющий работать с любым элементом контейнера, обращаясь к нему по индексу, между тем как stack и queue позволяют обращаться только к крайним элементам контейнера.

Типы контейнеров

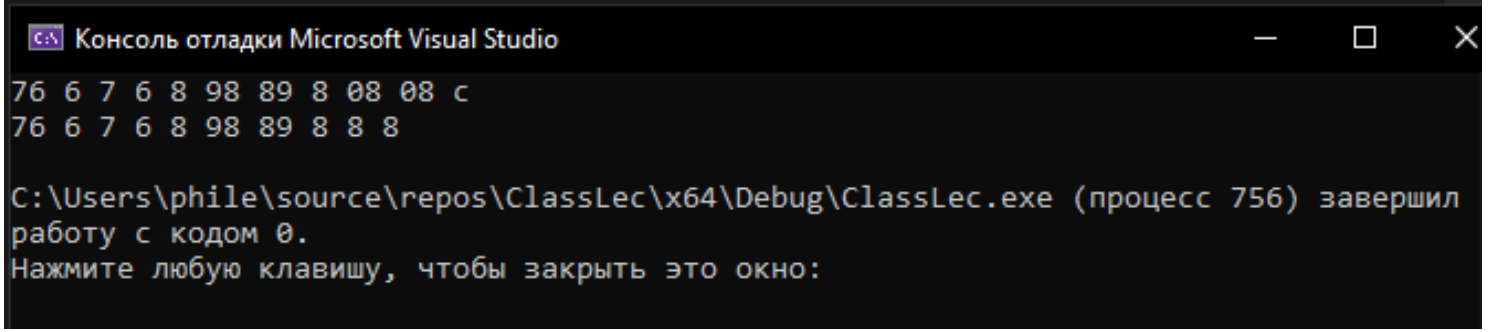
Последовательные контейнеры	Адаптеры контейнеров	Ассоциативные контейнеры
array: коллекция фиксированного размера	**stack<>**: представляет структуру данных "стек"	**set**: хранит уникальные элементы в отсортированном порядке.
vector: коллекция переменного размера	**queue<>**: представляет структуру данных "очередь"	**map:** хранит пары ключ-значение в отсортированном порядке.
deque: двусторонняя очередь	**priority_queue<>**: также представляет очередь, но при этому ее элементы имеют приоритеты	
list: двухсвязный список		
forward_list: односвязный список		

Итераторы

Очень важное понятие в реализации динамических структур данных — итератор. Неформально итератор можно определить как абстракцию, которая ведет себя как указатель, возможно, с какими-то ограничениями. Строго говоря, итератор — более общее понятие, и является объектной оберткой для указателя, поэтому указатель является итератором.

Пример использования итераторов. Input итераторы

```cpp
#include <iostream>
#include <vector>
#include <iterator>

int main() {
    std::vector<int> data = { 1, 2, 3, 4, 5 };
    std::istream_iterator<int> start(std::cin), end; // чтение данных из стандартного ввода
    std::vector<int> numbers(start, end); // инициализация вектора считанными числами

    for (int n : numbers) {
        std::cout << n << " ";
    }
    std::cout << std::endl;

    return 0;
}
```



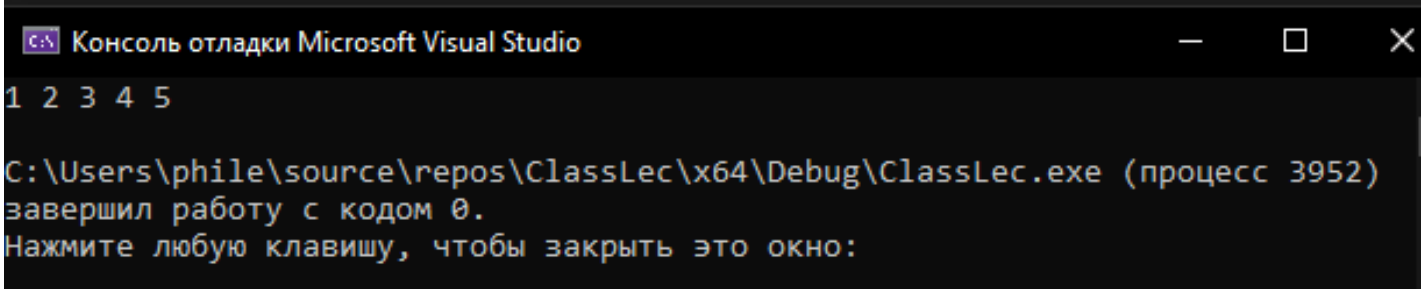
Пример использования итераторов. Output итераторы

```cpp
#include <iostream>
#include <iterator>
#include <algorithm>
#include <vector>

int main() {
    std::ostream_iterator<int> out_it(std::cout, ", ");
    std::vector<int> data = { 1, 2, 3, 4, 5 };

    std::copy(data.begin(), data.end(), out_it); // вывод данных в стандартный вывод
    return 0;
}
```

```
Консоль отладки Microsoft Visual Studio
1, 2, 3, 4, 5,
C:\Users\phile\source\repos\ClassLec\x64\Debug\ClassLec.exe (процесс 1748)
л работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Пример использования итераторов. Forward итераторы

```cpp
#include <forward_list>
#include <iostream>

int main() {
    std::forward_list<int> flist = { 1, 2, 3, 4, 5 };

    for (auto it = flist.begin(); it != flist.end(); ++it) {
        std::cout << *it << " ";
    }
    std::cout << std::endl;

    return 0;
}
```



Методы контейнеров

empty — определяет, пуста ли коллекция;
size — возвращает размер коллекции;
begin — возвращает прямой итератор, указывающий на начало коллекции;
end — возвращает прямой итератор, указывающий на конец коллекции, т.е. на несуществующий элемент, идущий после последнего;
rbegin — возвращает обратный итератор на начало коллекции;
rend — возвращает обратный итератор на конец коллекции;
clear — очищает коллекцию, т.е. удаляет все ее элементы;
erase — удаляет определенные элементы из коллекции;
capacity — возвращает вместимость коллекции, т.е. количество элементов, которое может вместить эта коллекция (фактически, сколько памяти под коллекцию выделено);

```
vector<int> vec;
cout << "Real size of array in vector: " << vec.capacity () << endl;
for (int j = 0; j < 10; j++)
{
    vec.push_back (10);
}
cout << "Real size of array in vector: " << vec.capacity () << endl;
return 0;
```

Алгоритмы STL

Разработчики библиотеки STL ставили перед собой гораздо более серьезную задачу, чем создание библиотеки с набором шаблонных структур данных. STL содержит огромный набор оптимальных реализаций популярных алгоритмов, позволяющих работать с STL-коллекциями. Все реализованные функции можно поделить на три группы:

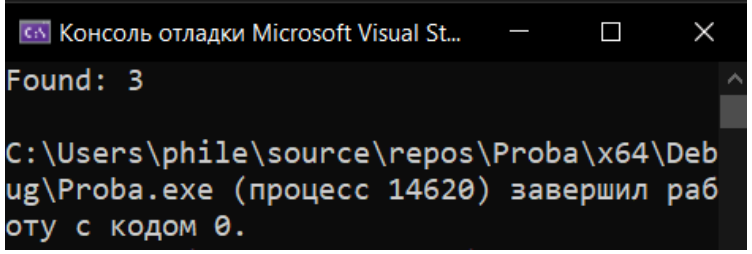
1. Методы перебора всех элементов контейнеров и их обработки;

2. Методы сортировки контейнеров;

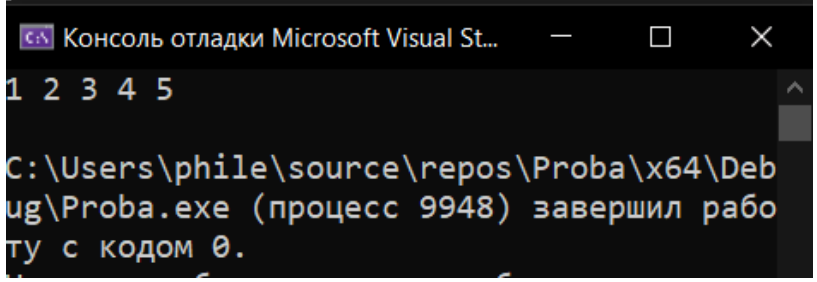
3. Методы выполнения определенных арифметических операций над элементами контейнеров

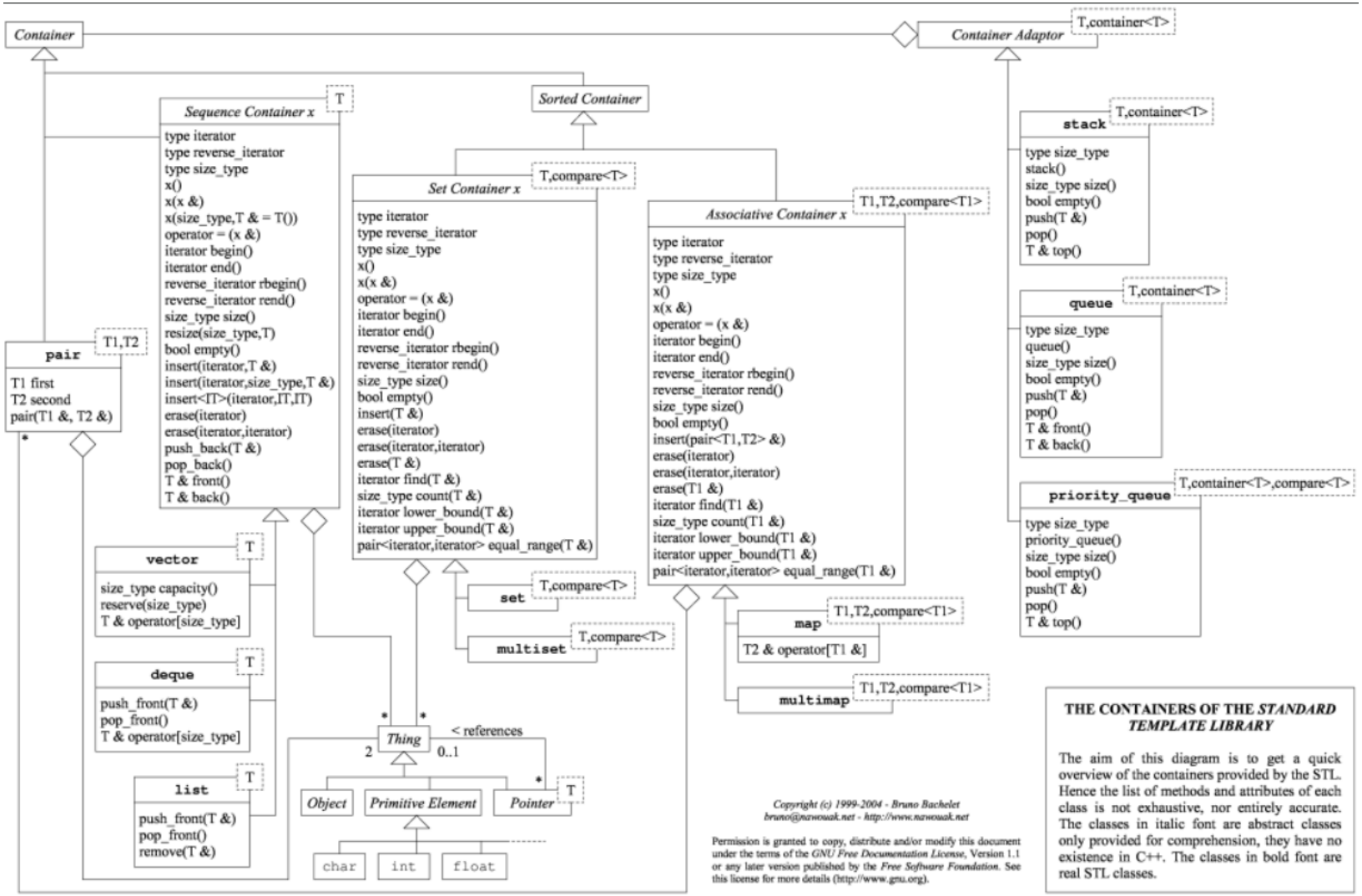
Пример алгоритмов STL

```cpp
#include <algorithm>
#include <vector>
#include <iostream>

int main() {
    std::vector<int> data = { 1, 2, 3, 4, 5 };
    auto it = std::find(data.begin(), data.end(), 3);
    if (it != data.end()) {
        std::cout << "Found: " << *it << std::endl;
    }
    else {
        std::cout << "Not found" << std::endl;
    }
    return 0;
}
```




```cpp
#include <algorithm>
#include <vector>
#include <iostream>

int main() {
    std::vector<int> data = { 5, 1, 4, 2, 3 };
    std::sort(data.begin(), data.end());
    for (int n : data) {
        std::cout << n << " ";
    }
    std::cout << std::endl;
    return 0;
}
```





Полезные ссылки

Большая шпаргалка:
https://hackingcpp.com/cpp/cheat_sheets.html

Много примеров:
https://habr.com/ru/companies/otus/articles/793278/