Стандартная библиотека STL: последовательные контейнеры

Александр Смаль

Академический университет
13 февраля 2014
Санкт-Петербург

STL: введение

- STL = Standard Template Library
- STL описан в стандарте C++, но не упоминается там явно.
- Авторы: Александр Степанов, Дэвид Муссер и Менг Ли для HP, а потом для SGI.
- Основан на разработках для языка Ада.
- Основные составляющие:
  - контейнеры (хранение объектов в памяти),
  - итераторы (доступ к элементам контейнера),
  - алгоритм (для работы с последовательностями),
  - адаптеры (обёртки над контейнерами)
  - функциональные объекты, функторы (обобщение функций).
  - потоки ввода/вывода.
- Всё определено в пространстве имён `std`.

Общие сведения о контейнерах

Контейнеры библиотеки STL можно разделить на четыре категории:

- последовательные,
- ассоциативные,
- контейнеры-адаптеры,
- псевдоконтейнеры.

Требования к хранимым объектам:

1. copy-constuctable
2. assignable
3. “стандартная семантика”

Итераторы — объекты для доступа к элементам контейнера с синтаксисом указателей.

Общие члены контейнеров

Типы (typedef-ы или вложенные класс):

1. `C::value_type`
2. `C::reference`
3. `C::const_reference`
4. `C::pointer`
5. `C::iterator`
6. `C::const_iterator`
7. `C::size_type` = size_t

Методы:

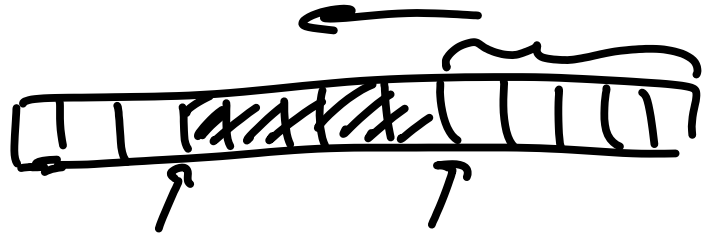
1. Конструктор по умолчанию, конструктор копирования, оператор присваивания, деструктор.
2. `begin()`, `end()`
3. Операторы сравнения: `==`, `!=`, `>`, `>=`, `<`, `<=`.
4. `size()`, `empty()`. empty() ⇔ (size() == 0)
5. `swap(obj2)`
6. `clear`.

Сложность?

Последовательные контейнеры

Общие члены

1. Конструктор от двух итераторов
2. Конструктор от count и defVal
3. Двух итераторный erase
4. `push_back, pop_back, back`
5. `front`
6. `assign` от двух итераторов
7. `assign` от count и val
8. `insert` от итератора и val
9. `insert` от итератора, n и val
10. `insert` от трёх итераторов

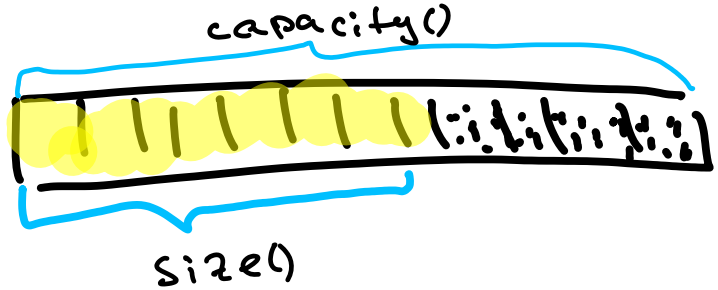


C(size_t n)

C(size_t n, T defVal = T())

С-подобный динамический массив произвольного доступа с автоматическим изменением размера при добавлении элементов.

1. operator[], at
2. resize
3. capacity, reserve



Разработан для работы со старым кодом.

```
#include <vector>

void legacy_function(int * m, int size);

std::vector<int> v;
...
legacy_function(&v[0], v.size());
```

deque

Контейнер похож на vector, но с возможностью быстрой вставки и удаления элементов на обоих концах за $O(1)$. Реализован в виде двусвязанного списка линейных массивов.

1. Конструктор от n
2. operator[], at
3. resize
4. push_front, pop_front

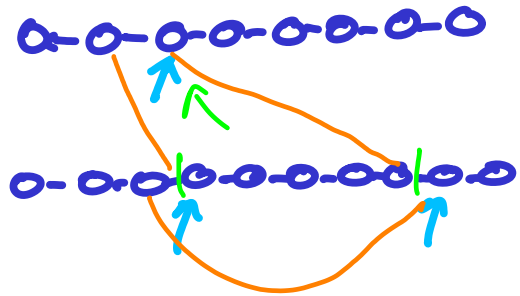
O(1)

```
#include <deque>

    std::deque<std::string> d;
    d.push_back(", world!");
    d.push_front("Hello");
    std::cout << d[0] << d[1] << std::endl;
```

Двусвязный список. В любом месте контейнера вставка и удаление производятся за $O(1)$.

1. merge, remove, remove_if, sort, unique, splice
2. splice
3. push_front, pop_front

```
#include <list>

    std::list<std::string> l;
    l.push_back(", world!");
    l.push_front("Hello");
    std::cout << l.size();
```



```
size_t i = 0;
list<int>::iterator it = l.begin();
for( ; i != l.size(); ++i, ++it)
    ....
```

```
char * s = ....,
for(int i = 0; i < strlen(s); i++)
    ....
```

string, wstring, basic_string

Контейнер для хранения символьных последовательностей.

1. Метод c_str() для совместимости со старым кодом:

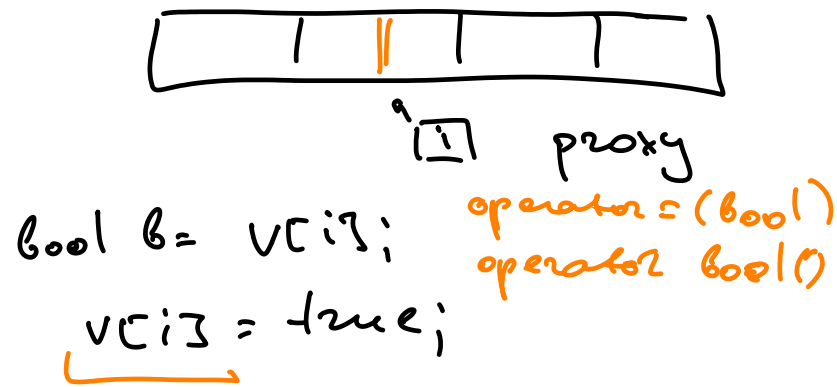
```
std::string res = "Hello";
...
printf("%s", res.c_str());
```

2. поддержка работы с строками в стиле C
3. множество алгоритмов вроде substr() (в терминах *индексов*),
4. string = basic_string<char>
5. wstring = basic_string<wchar_t>
6. могут быть реализованы как COW (Copy-On-Write).

Адаптеры и псевдоконтейнеры

Адаптеры:

1. `stack`
   Реализация интерфейса стека.
2. `queue`
   Реализация интерфейса очереди.
3. `priority_queue`
   Очередь с приоритетом на куче.



Псевдо-контейнеры:

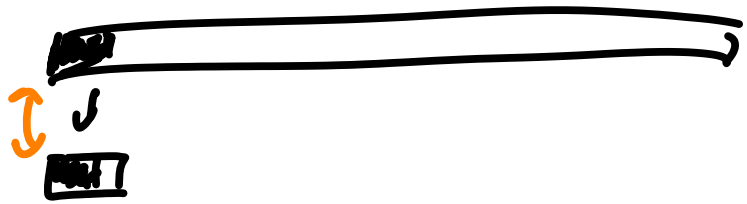
1. `vector<bool>`
   1. ненастоящий контейнер (не хранит bool-ы),
   2. использует proxy-объекты.
2. `bitset<32>`
   Служит для хранения битовых масок. Похож на `vector<bool>` фиксированного размера.
3. `valarray`
   Шаблон служит для хранения числовых массивов и оптимизирован для достижения повышенной вычислительной производительности.

Ещё о vector

vector — наиболее универсальный последовательный контейнер.

Идиомы

Сжатие и очистка:



```
std::vector<int> v;
...
std::vector<int>(v).swap(v); // compress

std::vector<int>().swap(v);  // clear
```

Использование reserve и capacity:

```
std::vector<int> v;
v.reserve(N); // N - upperbound for size
...
if (v.capacity() == v.size()) // reallocation
    ...
```

// size() = 0