

Лекция 13. Стандартная библиотека шаблонов (STL)

Тема: Контейнеры (vector, list, map), итераторы, алгоритмы, лямбда-выражения.

1. Введение

Современное программирование в языке C++ невозможно представить без использования **Стандартной библиотеки шаблонов (Standard Template Library, STL)**.

STL представляет собой мощный набор готовых компонентов для работы с данными, включающий контейнеры, итераторы, алгоритмы и функции общего назначения.

Её основная идея — **обобщённое программирование (generic programming)**:

пишется один универсальный код, который можно применять к различным типам данных без повторного написания.

STL делает код более **понятным, надёжным, производительным и коротким**, а также полностью интегрируется со стандартом ISO C++.

2. Архитектура STL

STL построена на трёх взаимосвязанных компонентах:

1. **Контейнеры (Containers)** — структуры данных для хранения элементов.
2. **Итераторы (Iterators)** — объекты, обеспечивающие последовательный доступ к элементам контейнера.
3. **Алгоритмы (Algorithms)** — универсальные функции для обработки данных (поиск, сортировка, копирование и т.д.).

Дополнительно используются:

- **Функциональные объекты (Functors)**
 - **Лямбда-выражения**
 - **Адаптеры контейнеров** (stack, queue, priority_queue)
-

3. Контейнеры STL

Контейнеры делятся на три основных типа:

Тип	Примеры	Особенности
Последовательные (Sequence)	vector, list, deque	Хранят элементы в определённом порядке
Ассоциативные (Associative)	set, map, multiset, multimap	Организованы как сбалансированные деревья
Нестандартные/адаптеры (Container adapters)	stack, queue, priority_queue	Предоставляют ограниченный интерфейс доступа

3.1. Контейнер vector

vector — это динамический массив, который автоматически изменяет свой размер при добавлении новых элементов.

Особенности:

- Быстрый доступ к элементам по индексу ($O(1)$).
- Эффективное добавление в конец (`push_back()`).
- Перемещение данных при вставке в середину.

Пример:

```
#include <vector>
#include <iostream>
using namespace std;

int main() {
    vector<int> nums = {1, 2, 3};
    nums.push_back(4);

    for (int n : nums)
        cout << n << " ";
}
```

Вывод:

1 2 3 4

3.2. Контейнер list

list — это двусвязный список, где каждый элемент хранит ссылки на предыдущий и следующий.

Преимущества:

- Быстрая вставка и удаление в любом месте ($O(1)$).
- Нет необходимости перемещать элементы, как в vector.

Недостатки:

- Нет доступа по индексу ($O(n)$ для поиска).
- Большие накладные расходы на хранение указателей.

Пример:

```
#include <list>
list<string> names = {"Алия", "Бекзат", "Даурен"};
names.push_front("Айнур");
names.remove("Бекзат");
```

3.3. Контейнер map

map хранит пары **ключ–значение**, автоматически сортируя их по ключу. Каждый ключ уникален, что делает map аналогом словаря.

Пример:

```
#include <map>
map<string, int> scores;

scores["Айгерим"] = 95;
scores["Даурен"] = 88;

for (auto& p : scores)
    cout << p.first << " — " << p.second << endl;
```

Результат:

Айгерим — 95
Даурен — 88

Особенности:

- Вставка и поиск работают за $O(\log n)$ (используется красно-чёрное дерево).
- Элементы автоматически упорядочены.

4. Итераторы

Итератор — это обобщённый указатель, с помощью которого можно проходить по элементам контейнера.

Они обеспечивают единый интерфейс для всех типов контейнеров.

Типы итераторов:

1. **Input** — чтение элементов.
2. **Output** — запись элементов.
3. **Forward** — последовательное чтение/запись.
4. **Bidirectional** — движение в обе стороны (list, map).
5. **Random access** — произвольный доступ (vector, deque).

Пример:

```
vector<int> v = {10, 20, 30};
vector<int>::iterator it;
for (it = v.begin(); it != v.end(); ++it)
    cout << *it << " ";
```

5. Алгоритмы STL

STL предоставляет более 100 алгоритмов, которые можно применять ко всем контейнерам через итераторы.

Основные группы алгоритмов:

- **Поиск:** find(), binary_search()
- **Сортировка:** sort(), stable_sort()
- **Модификация:** copy(), transform(), fill()
- **Проверка условий:** all_of(), any_of(), none_of()
- **Суммирование и накопление:** accumulate() (из <numeric>)

Пример:

```
#include <algorithm>
#include <numeric>
vector<int> v = {1, 2, 3, 4, 5};

int sum = accumulate(v.begin(), v.end(), 0);
sort(v.begin(), v.end(), greater<int>());
```

```
for (int n : v)
    cout << n << " ";
cout << "\nСумма: " << sum;
```

6. Лямбда-выражения

Лямбда-выражения — это анонимные функции, которые можно передавать как аргументы алгоритмам.

Они делают код компактным и понятным.

Синтаксис:

```
[capture](parameters) -> return_type { body }
```

Пример:

```
vector<int> nums = {1, 2, 3, 4, 5};

for_each(nums.begin(), nums.end(),
    [](int n) { cout << n * n << " "; });
```

Результат:

1 4 9 16 25

Лямбды часто используются с алгоритмами `sort()`, `remove_if()`, `count_if()` и др.

7. Адаптеры контейнеров

Адаптеры изменяют интерфейс стандартных контейнеров, предоставляя доступ только к определённым операциям.

Адаптер	Основа	Назначение
stack	deque	Стек (LIFO)
queue	deque	Очередь (FIFO)
priority_queue	vector	Очередь с приоритетами

Пример stack:

```
#include <stack>
stack<int> st;
st.push(10);
st.push(20);
cout << st.top(); // 20
```

```
st.pop();
```

8. Пример комплексной программы с STL

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <map>
using namespace std;

int main() {
    vector<int> numbers = {5, 2, 8, 1, 9};
    sort(numbers.begin(), numbers.end());

    map<int, string> labels = {
        {1, "low"}, {5, "medium"}, {9, "high"}
    };

    for (int n : numbers)
        cout << n << " -> " << labels[n] << endl;

    return 0;
}
```

Результат:

```
1 -> low
2 ->
5 -> medium
8 ->
9 -> high
```

9. Преимущества STL

- **Надёжность:** минимизация ошибок за счёт использования проверенных структур данных.
 - **Гибкость:** совместимость с любыми типами данных.
 - **Унификация:** единый подход к обработке различных контейнеров.
 - **Производительность:** эффективные алгоритмы с асимптотикой $O(n \log n)$ и ниже.
 - **Совместимость:** входит в стандарт ISO C++ и поддерживается всеми компиляторами.
-

10. Заключение

STL является одним из главных достижений языка C++.

Её использование позволяет писать мощные, безопасные и лаконичные программы без изобретения собственных структур данных и алгоритмов.

Каждый программист на C++ должен уверенно владеть контейнерами (vector, list, map), понимать принципы работы итераторов и уметь применять стандартные алгоритмы с лямбда-выражениями.

Это не только ускоряет разработку, но и делает код профессиональным и универсальным.

Список литературы

1. Страуструп, Б. *Язык программирования C++*. — М.: Вильямс, 2013.
2. Шилдт, Г. *STL для профессионалов*. — М.: Вильямс, 2020.
3. Джосаттис, Н. *The C++ Standard Library*. — Addison-Wesley, 2017.
4. Лафоре, Р. *Объектно-ориентированное программирование в C++*. — СПб.: Питер, 2019.
5. Meyers, S. *Effective STL*. — Addison-Wesley, 2001.
6. Josuttis, N. *C++17: The Complete Guide*. — Leanpub, 2019.