Прикладная разработка на C++
Лекция 2

Стандартная библиотека шаблонов
STL

Шаблон

Шабло́н (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Инстанцирование шаблона – процесс создания нового экземпляра.

Шаблоны функции

```
#include <iostream>
using std::cout;
using std::endl;

template <typename T>
void swap(T* val1, T* val2) {
  typename T tmp = *val1;
  *val1 = *val2;
  *val2 = tmp;
}

int main() {
  int i1 = 5;
  int i2 = 7;
  cout << i1 << " " << i2 << endl;
  swap(&i1, &i2);
  cout << i1 << " " << i2 << endl;
  return 0;
}
```

STL

STL (Standard Template Library “Стандартная библиотека шаблонов”).

Александр Степанов и Менг Ли

STL

В состав STL входят:

- **Контейнеры** (containers) – это объекты, предназначенные для хранения других элементов.
- **Итераторы** (iterators)
- **Алгоритмы** (algorithms) выполняют операции над содержимым контейнера.

Итераторы

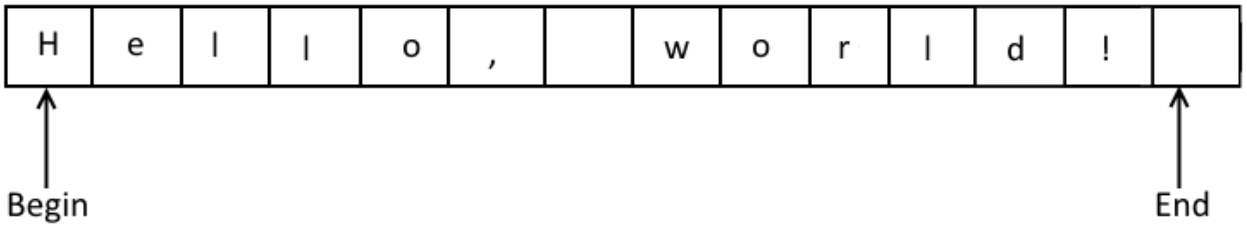
Итераторы (iterators) – это объекты, которые по отношению к контейнеру играют роль указателей. Они позволяют получить доступ к содержимому контейнера примерно так же, как указатели используются для доступа к элементам массива.

Типы итераторов

1. **итератор ввода** (input iterator);
2. **итератор вывода** (output iterator);
3. **однонаправленный итератор** (forward iterator);
4. **двунаправленный итератор** (bidirectional iterator);
5. **итератор произвольного доступа** (random access iterator);

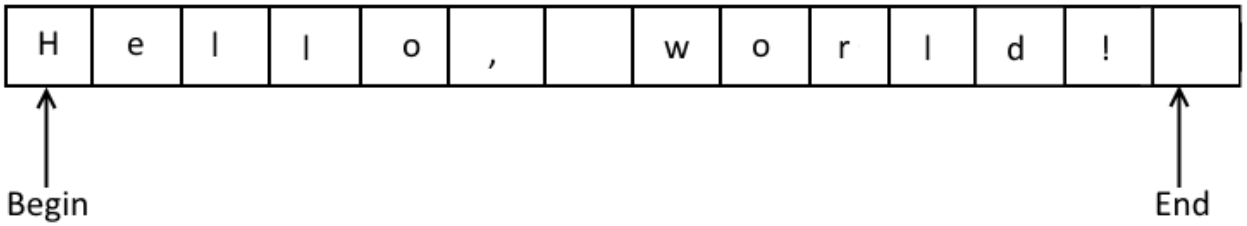
Пример кода

```
#include <iostream>
#include <cstring>
#include <algorithm>
#include <iterator>
using namespace std;

int main() {
  const char* c_str = "Hello, world!";
  int size = strlen(c_str);
  const char* end = c_str + size;
  for (const char* it = c_str; it != end; ++it) //вывод 1 способ
    cout << *it << " ";
  cout << endl;
  for (int i = 0; i < size; ++i)  //вывод 2 способ
    cout << c_str[i] << " ";
  cout << endl;
  // алгоритм копирования
  copy(c_str, c_str + size, ostream_iterator<char>(cout, "'"));
  cout << endl;
  return 0;
}
```



Пример кода

```
#include <iostream>
#include <string>
#include <algorithm>
#include <iterator>
using namespace std;

int main() {
  std::string str("Hello, world!");
  for (string::iterator it = str.begin(); it != str.end(); ++it)
//вывод 1способ
    cout << *it << " ";
  cout << endl;
  for (int i = 0; i < str.size(); ++i) //вывод 2способ
    cout << str[i] << " ";
  cout << endl;
  // алгоритм копирования
  copy(str.begin(), str.end(), ostream_iterator<char>(cout, "'"));
  cout << str2 << endl;
  for (char ch : str) // C++11  вывод 3 способ
    cout << ch << " ";
  cout << endl;
  return 0;
}
```



Контейнеры

В каждого контейнере определен набор методов для работы с ним, причем все контейнеры поддерживают стандартные базовые операции. Такие методы имеют одинаковое определение.

Например,

функция size() возвращающая текущий размер контейнера,

функция push_back(T val) помещает элемент в конец контейнера.

Но если метод не может быть эффективно реализован для какого-то контейнера его обычно исключают из общего набора операций.

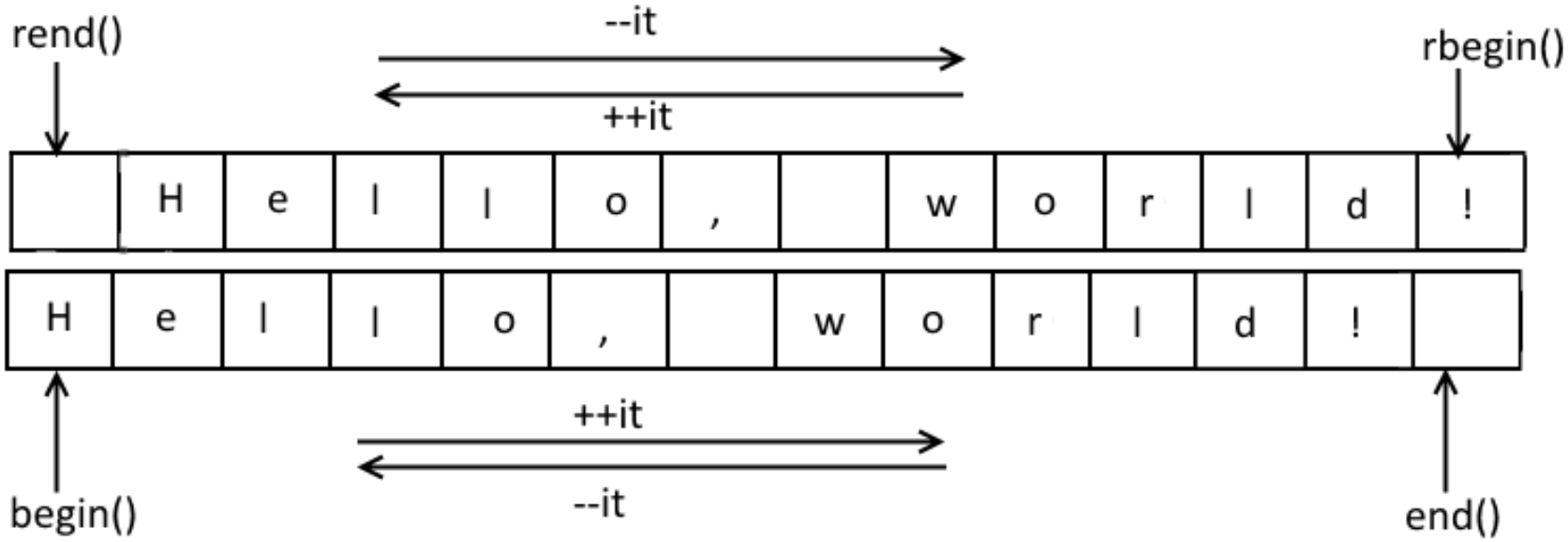
Итераторы в контейнерах

В контейнерах есть ряд методов позволяющие производить проход по элементам контейнера:

- begin() – возвращает **итератор** указывающий на первый элемент в последовательности;
- end() – возвращает **итератор** указывающий на элемент, следующий за последним элементом в последовательности;

Пример

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

int main() {
  std::string str("Hello, world!");
  // string::iterator it;
  for (auto it = str.begin(); it != str.end(); ++it)
    cout << *it; // Hello, world!
  cout << endl;
  // string::reverse_iterator rIt;
  for (auto rIt = str.rbegin(); rIt != str.rend(); ++rIt)
    cout << *rIt; // !dlrow ,olleH
  cout << endl;
  return 0;
}
```


rend()
--it
rbegin()
++it
H e l l o , w o r l d !
H e l l o , w o r l d !
++it
begin()
--it
end()

Вектор

Название: Динамический массив (std::vector)

Тип контейнера: последовательный доступ

Тип итератора: итератор произвольного доступа

Заголовочный файл: <vector>

Вектор

Методы доступа к элементам:

- at(size_type ind) – доступ к элементу ind с проверкой индекса;
- operator[size_type ind] – доступ к элементу ind без проверки;
- front() – доступ к первому элементу;
- back() – доступ к последнему элементу;
- data() – прямой доступ к внутреннему содержимому;

Вектор

Вместимость:

- empty() – проверка на отсутствие элем.;
- size() – кол-во элементов в контейнере;
- max_size() – макс. допустимое кол-во элем.;
- reserve() – зарезервировать память;
- capacity() – кол-во зарезервированной памяти;
- shrink_to_fit – уменьшить использ. памяти;

Вектор

Методы модификации контейнера:

- clear() – очистка контейнера;
- insert() – вставка элемента;
- erase() – удаление элемента;
- push_back() – добавление элем. в конец;
- pop_back() – удалить послед. элемент;
- resize() – изменить размер контейнера;
- swap() – обменивает содержимым между контейнерами;

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

int main() {
  std::vector<int> vec({5, 7, 1, 3, 2});
  vec.front() = 9;                 // 9 7 1 3 2
  vec.back() = 6;                  // 9 7 1 3 6
  vec.push_back(15);               // 9 7 1 3 6 15
  vec.erase(++vec.begin());        // 9 1 3 6 15
  vec.insert(vec.begin(), 5); // 5 9 1 3 6 15
  vec[4] = 2;                      // 5 9 1 3 2 15
  cout << "размер = (" << vec.size() // 6
       << ") vect1 пуст? (" << vec.empty() << ')' << endl;
// 0
  vec.resize(2);                 // 5 9
  vector<int> vec2;
  cout << "vec2 пуст? " << vec2.empty() << endl; // 1
  vec.swap(vec2); // vec (пуст), vec2 (5 9)
  return 0;
}
```

Ассоциативные контейнеры

pair

std::pair является шаблоном структуры, который представляет возможность хранить два разнородных объекта, как единое целое.

```
#include <utility>
```

Пример:

```
std::pair<string, string> author("Ivan", "Ivanov");
std::pair<int, string> count_word(15, "Ivan");
```

pair

- Значения пары могут быть получены с помощью открытых свойств first и second:

```
if (author.first == "Ivan" &&
    author.second == "Ivanov")
cout << "Ivan Ivanov" << endl;

author.first = "Vlad";
author.second = "Romanov";
```

Шаблонная функция std::make_pair

```
#include <utility>

// std::pair<string, string>
std::pair<string, string> author("Ivan", "Ivanov");
// std::pair<const char*, const char*>
auto author = make_pair("Ivan", "Ivanov");

// std::pair<int, string>
std::pair<int, string> count_word(15, "Ivan");
// std::pair<int, const char*>
auto count_word = make_pair(15, "Ivan");
// std::pair<int, string>
auto count_word = make_pair(15, string("Ivan"));
```

Ассоциативные контейнеры

- set<TKey, Cmp> - коллекция уникальных ключей
- multiset<TKey, Cmp>- коллекция ключей
- map<TKey, TVal, Cmp> - коллекция уникальных пар ключ-значение (отношение 1 к 1)
- multimap<TKey, TVal, Cmp> - коллекция пар ключ-значение (отношение 1 к M)

Данные коллекции являются сортированными контейнерами. По умолчанию сортируются с помощью оператора <.

Ассоциативные контейнеры реализованы на основе красно-черного дерева (бинарное дерево).

set & multiset (множества)

std::set контейнер содержащий в себе уникальные отсортированные элементы.

std::multiset контейнер содержащий в себе отсортированные элементы.

Тип итератора константный двусторонний итератор.

```
#include <set>
```

set & multiset & map & multimap

Итераторы:

- begin(), cbegin() – итератор на первый элемент ;
- end(), cend() – итератор на элемент, следующий за последним;
- rbegin(), crbegin() – обратный итератор на первый элемент;
- rend(), crend() – обратный итератор на элемент, следующий за последним;

Вместимость:

- empty() – проверка на отсутствие элем.;
- size() – кол-во элементов в контейнере;
- max_size() – макс. допустимое кол-во элем.;

set & multiset & map & multimap

Методы модификации контейнера:

- clear() – очистка контейнера;
- insert(…) – вставка элемента;
- erase(Key) – удаление элемента;
- swap() – обменивает содержимым между контейнерами;

set & multiset

```
std::set<int> Set;
std::multiset<int> mSet;

Set.insert(4);
Set.insert(4);
mSet.insert(4);
mSet.insert(4);

// mSet -> 4, 4
// Set  -> 4
```

typedef & using

```
typedef std::set<int> set_t;
typedef set_t::iterator iter_t;
// C++11 Type alias
//using set_t = std::set<int>;
//using iter_t = type_set::iterator;
set_t Set;

//Set.insert(4);
std::pair<iter_t, bool> pair_v;
pair_v = Set.insert(4);
if (pair_v.second)
{
  cout << "Значение было добавлено!" << endl;
  cout << *(pair_v.first) << endl;
}
```

set & multiset

```
#include <set>
#include <string>
#include <iostream>

using namespace std;

int main()
{
 std::set<std::string> S;

 S.insert("January");
 S.insert("February");
 S.insert("March");
 S.insert("April");
 // April February January March
 cout << S.size() << endl; // будет выведено 4
 cout << S.max_size() << endl;
 cout << *(S.begin()) << endl; // April
```

```
 //cout << *(S.end()) << endl;  // выход за границу
 cout << *(S.rbegin()) << endl; // March
 //cout << *(S.rend()) << endl; // выход за границу
 cout << S.count("January") << endl; // 1
 cout << *S.lower_bound("January") << endl; // January
 cout << *S.upper_bound("January") << endl; // March
 cout << *(S.equal_range("January")).first << endl; //
January
 cout << *(S.equal_range("January")).second << endl; //
March
 cout << *S.find("January") << endl; // January
 S.erase("January");
 cout << S.count("January") << endl; // 0
 cout << S.size() << endl; // 0
 S.clear();
 cout << S.size() << endl; // 0

 return 0;
}
```

set & multiset

map & multimap (ассоциативный массив)

std::map контейнер содержащий в себе уникальные пары, отсортированные по ключу.

std::multiset контейнер содержащий в себе пары, отсортированные по ключу.

Тип итератора: константный двусторонний итератор.

```
#include <map>
```

```cpp
#include <iostream>
#include <string>
#include <map>
using namespace std;

int main()
{
  // явная инициализация map
  map <string,int> myMap = {{ "Beta", 2 },
                            { "Alpha", 1 },
                            { "Gamma", 3 }};
  // присвоение элементам map новых значений
  myMap.at("Alpha") = 233;
  myMap.at("Gamma") = -45;
  myMap["Beta"] = 7;

  // добавление элементов
  myMap["Omega"] = 13;
  myMap.insert ( make_pair("Alpha2",221) );
```

```
// Если элемента нет в контейнере,
// то итератор будет равен (myMap.end())
auto it = myMap.find("Alpha");
if (it != myMap.end())
  myMap.erase(it);
myMap.erase("Alpha2");

cout << "myMap contains:" << endl;
for(auto pair_v : myMap)
{
  cout << pair_v.first << " : "
       << pair_v.second << endl;
}

multimap <char,int> myMultimap;//объявили multimap

//myMultimap.at('a') = 15; // error
//myMultimap['a'] = 15;    // error

//заполняем myMultimap
myMultimap.insert ( make_pair('q',111) );
myMultimap.insert ( make_pair('u',201) );
myMultimap.insert ( make_pair('h',301) );
```

```
cout << endl << "myMultimap contains:" << endl;
for (auto pair_v : myMultimap)
{
  cout << pair_v.first << " : "
       << pair_v.second << endl;
}
myMap.clear();
myMultimap.clear();

//новая инициализация myMap
myMap = {{ "Mike", 40 },
         { "Walle", 999 },
         { "Cloude", 17 }};

//новая инициализация myMultimap
myMultimap.insert ( make_pair('q',222) );
myMultimap.insert ( make_pair('u',223) );
myMultimap.insert ( make_pair('h',221) );
```

```
  // создаем итератор itMap на первый элемент myMap
  auto itMap = myMap.begin();
  // создаем итератор на первый элемент myMultimap
  auto itMultimap = myMultimap.begin();
  cout << endl << "myMap after clear contains: "
       << "\t myMultimap after clear contains:" << endl;
  // вывод на экран myMap и myMultimap
  while(itMultimap != myMultimap.end())
  {
    cout << '\t' <<  itMap->first << " : "
         << itMap->second << "\t\t\t\t\t\t"
         << itMultimap->first << " : "
         << itMultimap->second << endl;
    itMap++;
    itMultimap++;
  }

  return 0;
}
```

myMap contains:

Beta : 7

Gamma : -45

Omega : 13

myMultimap contains:

h : 301

q : 111

u : 201

myMap after clear contains:

Cloude : 17

Mike : 40

Walle : 999

myMultimap after clear contains:

h : 221

q : 222

u : 223

set & multiset & map & multimap

Операции:

- count(Key) – кол-во элементов соответствующих определенному ключу;
- find(Key) – поиск первого элемента по ключу;
- equal_range(Key) - возвращает набор элементов для конкретного ключа; (pair<iterator, iterator>)
- lower_bound(Key) - возвращает итератор на первый элемент *не менее*, чем заданное значение;
- upper_bound(Key) - возвращает итератор на первый элемент *больше*, чем определенное значение;

map

Доступ к элементам:

- at(Key) - предоставляет доступ к указанному элементу с проверкой индекса;
- operator[Key] - предоставляет доступ к указанному элементу;

std::getline();

```
// extract to string
#include <iostream>
#include <string>

int main ()
{
  std::string name;

  std::cout << "Please, enter your full name: ";
  std::getline(std::cin,name);
  std::cout << "Hello, " << name << "!\n";

  return 0;
}
```

Функция split()

```
std::vector<std::string> split(const std::string& s, char delimiter)
{
   std::vector<std::string> tokens;
   std::string token;
   std::istringstream tokenStream(s);
   while (std::getline(tokenStream, token, delimiter))
   {
      tokens.push_back(token);
   }
   return tokens;
}
```

Список литературы

- http://www.cplusplus.com/reference/string/vector/
- http://www.cplusplus.com/reference/string/map/
- https://www.fluentcpp.com/2017/04/21/how-to-split-a-string-in-c/
- Deitel P., Deitel H. C++ how to Program. – Pearson, 2016.

Типы, содержащиеся в контейнерах

В каждом контейнере существует список **typedef**, которые являются необходимыми для сокрытия типов от пользователя контейнера.

value_type – тип элемента;

allocator_type – тип распределителя памяти;

size_type – тип для индексации элементов;

iterator, **const_iterator** – прямые итераторы;

reverse_iterator, **const_reverse_iterator** – обратные итераторы;

pointer, **const_pointer** – указатель на элемент;

reference, **const_reference** – ссылка на элемент;

Итератор ввода

- Итератор способный считывать из указанного элементы;
- Поддерживаемые операции: операции копирования, a != b, *a, a->m, ++a, a++, *a++;
- Примеры: istream_iterator, istreambuf_iterator;

Итератор вывода

- Итератор способный записывать в указанный элемент.
- Поддерживаемые операции: копирующие операции, a++, ++a, *a = o, *a++ = o;
- Примеры: ostream_iterator, ostreambuf_iterator;

Однонаправленный итератор

- Для обхода элементов в одном направлении (от начало до конца).
- Поддерживаемые операции:
  - операции итераторов ввода/вывода;
  - Конструкторы по умолчанию;
- Пример: контейнер STL (forward_list);

Двунаправленный итератор

- Для обхода элементов в двух направлениях (от начало до конца, от конца к началу).
- Поддерживаемые операции:
  - операции однонаправленных итераторов;
  - a--, --a, *a--;
- Примеры: контейнеры STL (list, set, multiset, map, multimap)

Итератор произвольного доступа

- Позволяют получить доступ к любому элементу;
- Поддерживаемые операции:
  - операции двунаправленных итераторов;
  - арифметические: a += n, a -= n, a + n, n + a, a - n;
  - сравнения: a < b, a <= b, a > b, a >= b;
  - a[n], b - a;
- Примеры: контейнеры STL (vector, deque, string, array);