# Лекция 3. Алгоритмы и функторы

# Стандартные алгоритмы

- Обогащают работу с контейнерами. И не только с ними.
- Используют обобщенное программирование, т.о. легко могут применяться к широкому спектру типов.
- Предоставляют интерфейс через итераторы. Если возвращают итератор, то того же типа, что и принимают.
- Объявлены в пространстве имен std, хидера: алгоритмы - <algorithm>, функторы - <functional>.
- Если контейнер обладает аналогичной функцией, используйте ее, т.к. она эффективнее (e.g. `map::find`).

# Описание алгоритмов

- При описании стандартных алгоритмов часто используются сокращения:
  - In, Out, For, Bi, Ran – типы итераторов по функциональности.
  - Op, BinOp – операции с одним или двумя аргументами.
  - Pred, BinPred – предикаты с одни или двумя аргументами. Те же операции, но всегда возвращают `bool`.

# Немодифицирующие операции

- `for_each` – применить для каждого элемента
- `find`, `find_if`, `adjacent_find` – поиск по условию
- `count`, `count_if` – количество элементов
- `mismatch`, `equal` – равенство/неравенство последовательностей
- `search`, `find_end` – поиск одной последовательности в другой

# Модифицирующие операции

- `transform` – изменение, часто inplace
- `copy`, `copy_if`
- `swap`, `iter_swap`
- `replace`, `replace_if`
- `fill`, `fill_n`, `generate` – заполнение значением, генератором
- `remove_if`, `remove_copy` – перенос элементов в конец (это не удаление элементов!)
- `unique`, `unique_copy` – оставить только уникальные элементы (аналогично, не удаление)
- `reverse`, `reverse_copy`, `rotate` – инвертировать, циклически сдвинуть
- `random_shuffle` – перетасовать

# Сортировка и поиск ©

- `sort`, `stable_sort` – сортировка быстрая или стабильная
- `lower_bound`, `upper_bound`, `equal_range` – бинарный поиск, возвращает итератор(ы)
- `binary_search` – бинарный поиск (есть/нет)
- `merge`, `inplace_merge` – слияние
- `partition`, `stable_partition` – разделение по предикату

# Разное (1)

- Множества
  - `includes`
  - `set_union`, `set_intersection`, `set_difference`
- Кучи
  - `make_heap`
  - `push_heap`, `pop_heap`
  - `sort_heap`

# Разное (2)

- МинМакс
  - `min`, `max` – для двух значений
  - `min_element`, `max_element` – в последовательности
  - `lexicographical_compare`
- Перестановки (лексикографические)
  - `next_permutation`
  - `prev_permutation`

# Контейнеры vs итераторы (1)

- Алгоритмы принимают итераторы, а не контейнеры, т.к. можно
  - передавать лишь часть контейнера;
  - делать итератор на другие, нежели контейнер структуры;
  - использовать итераторы «сквозь» несколько контейнеров (e.g. линеаризация вложенных контейнеров)

```
1.  typedef vector<string>                vstr;
2.  typedef istream_iterator<string>      in_str;
3.
4.  vstr col;
5.  copy(in_str(cin), in_str(), back_inserter(col));
```

# Контейнеры vs итераторы (2)*

```
1.  typedef vector<string>              vstr;
2.  typedef vstr::const_iterator        vstr_cit;
3.
4.  template<class In, class T>
5.  T accumulate(In first, In last, T init);            // std (1)
6.
7.  template<class In, class T, class BinOp>
8.  T accumulate(In first, In last, T init, BinOp op);  // std (2)
9.
10. template<class Con, class T, class BinOp>
11. T accumulate(Con const& con, T init, BinOp op);     // ours(3)
12. //...
13.
14. vstr_cit find_some_item(vstr const& v);
15. //...
16.
17. vstr v;
18. // What function to call? std(1) or ours(3)?
19. accumulate(find_some_item(v), v.end(), 0);
20.
21. // use special wrapper template<class Con> seq_t;
22. accumulate(seq(v), 0);
```

# Стандартные функторы

- Предикаты
  - `equal_to`, `not_equal_to`
  - `less`, `greater`, `less_equal`, `greater_equal`
  - `logical_and` (or, not)
- Арифметические
  - `plus`, `minus`, …
  - `modules`, `negate`, …
- Адаптеры
  - `not1`, `not2`

```
1.  sort      (v.begin(), v.end(), greater<string>());
2.  transform(a.begin(), a.end(), b.begin(),
3.                back_inserter(res), multiplies<int>());
```

# Что может быть функтором?

- Свободная функция.
- Объект с оператором operator().
- Результат `std::bind` (C++11).
- `boost::lambda`.
- Встроенные lambda функции (C++11).
- Deprecated: функции на основе связывателей `bind2nd`, `mem_fun`, `ptr_fun`. Вытеснены `std::bind`.

# Примеры функторов

```
1.  void offset_x_0_y_10(point& p) { p += point(0, 10); }
2.
3.  //
4.  template<class T>
5.  struct make_offset_t // : unary_function<T, void>
6.  {
7.      make_offset_t(T const& offset) : off_(offset){}
8.      void operator(T& x) const { x += off_; }
9.
10. private:
11.     T off_;
12. };
13.
14. template<class T>
15. make_offset_t<T> make_offset(T const& off)
16. { return make_offset_t<T>(off); }
17.
18. //...
19. point p(0, 10);
20.
21. for_each(v.begin(), v.end(), offset_x_0_y_10);
22. for_each(v.begin(), v.end(), make_offset_t<point>(p));
23. for_each(v.begin(), v.end(), make_offset(p));
24.
25. for_each(v.begin(), v.end(), _1 += p);
26. for_each(v.begin(), v.end(), bind(&point::operator+=, _1, p));
27. for_each(v.begin(), v.end(), [&p](auto& arg){ arg += p; }); // C++14
```

# Функции с предикатом и без

- Многие алгоритмы имеют вид *func* и *func*_if (find, copy, remove, …), что позволяет настраивать работу функции предикатом.

```
1.  auto num1 = count   (v.begin(), v.end(), 42);
2.  auto num2 = count_if(v.begin(), v.end(),
3.                       [](int v){ return v > 10 && v < 20; });
```

# Копирующие функции

- Другая разновидность модифицирующих функций *func* – копирующие *func*_copy (replace, remove, reverse, …).
- Цель – не изменять текущую последовательность, а сделать на ее основе новую с нужными свойствами.

```
1.  remove_copy_if(
2.      s.begin(), s.end(), back_inserter(res),
3.      [](auto str){ return str.size() > 8; }); // since C++14
```

# Удобно?

# Boost.range. Range concept

- Функции std, оперируя итераторами, предоставляют большую гибкость, однако весьма “монструозны” в использовании.

```
1.  using namespace boost::adaptors;
2.  std::vector<int> vi = ...;
3.  boost::copy(vi | filtered(fn) | reversed,
4.                ostream_iterator<int>(cout));
```

- Концепция Range «легче» контейнеров:
  - не обязательно владеет элементами, к которым имеет доступ;
  - не обязательно обладает семантикой копирования.

# Boost.range Алгоритмы и адапторы

- Алгоритмы дублируют (и даже расширяют) алгоритмы из STL.
  - возвращают снова Range для последовательного вызова.
- Адаптеры:
  - ленивы, нет лишних копий и аллокаций;
  - гибки, предоставляют последовательное обращение;
  - нет больше комбинаторного взрыва от _copy, _if.

# Примеры

```
1.
2.  // 1
3.  boost::push_back(vec,
4.                   rng | replaced_if(pred, new_value) | reversed);
5.
6.  // 2
7.  std::map<int,int> input;
8.  boost::copy(input | map_keys,
9.               ostream_iterator<int>(cout, ","));
10.
11. // 3
12. std::vector<int> input;
13. boost::copy(input | filtered(fn) | reversed,
15.              ostream_iterator<int>(cout, ","));
16.
```

# Вопросы?