SECON'2017

#SECONRU

IX МЕЖРЕГИОНАЛЬНАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Повседневный C++: алгоритмы и итераторы

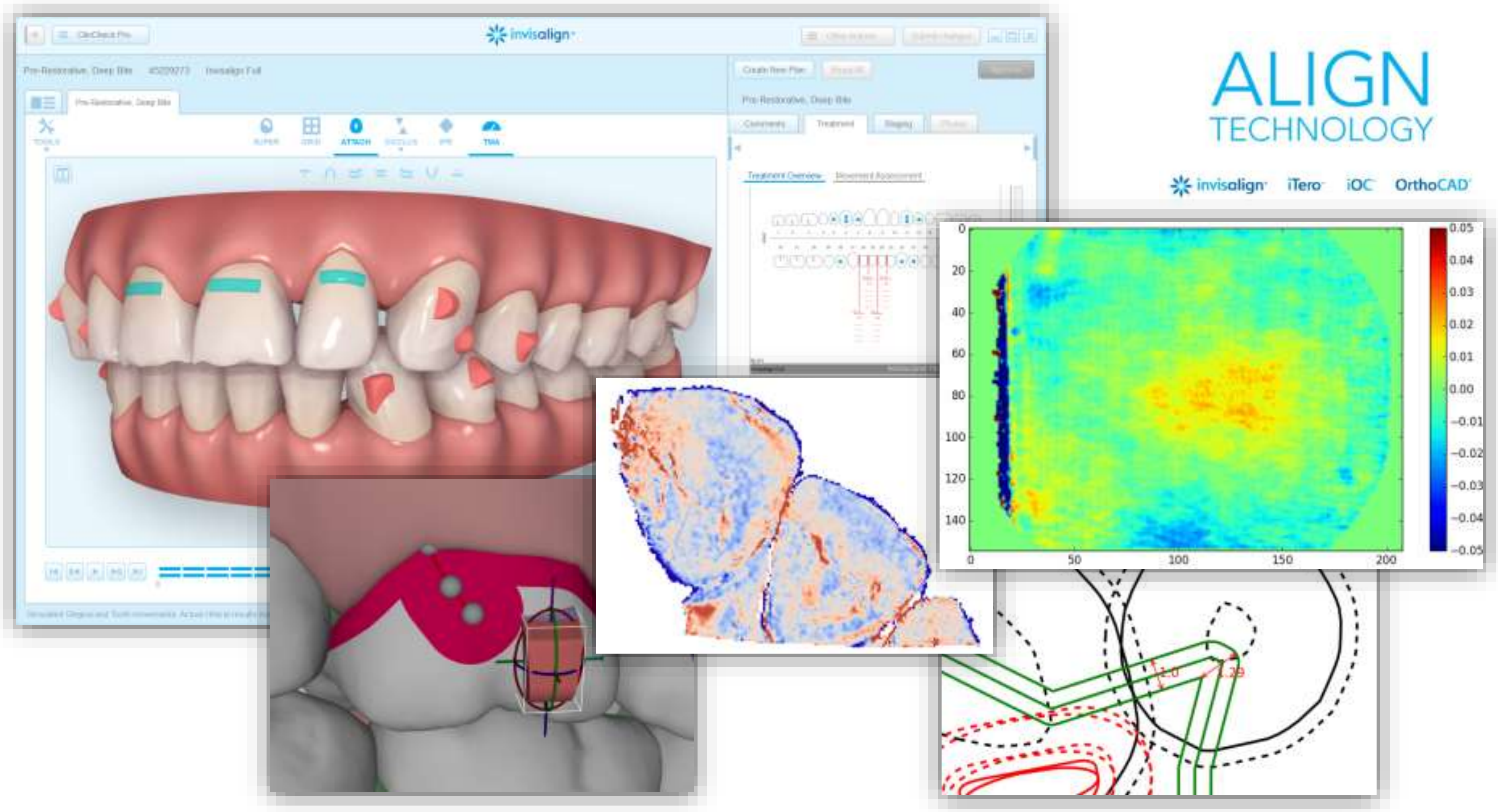
Матросов Михаил

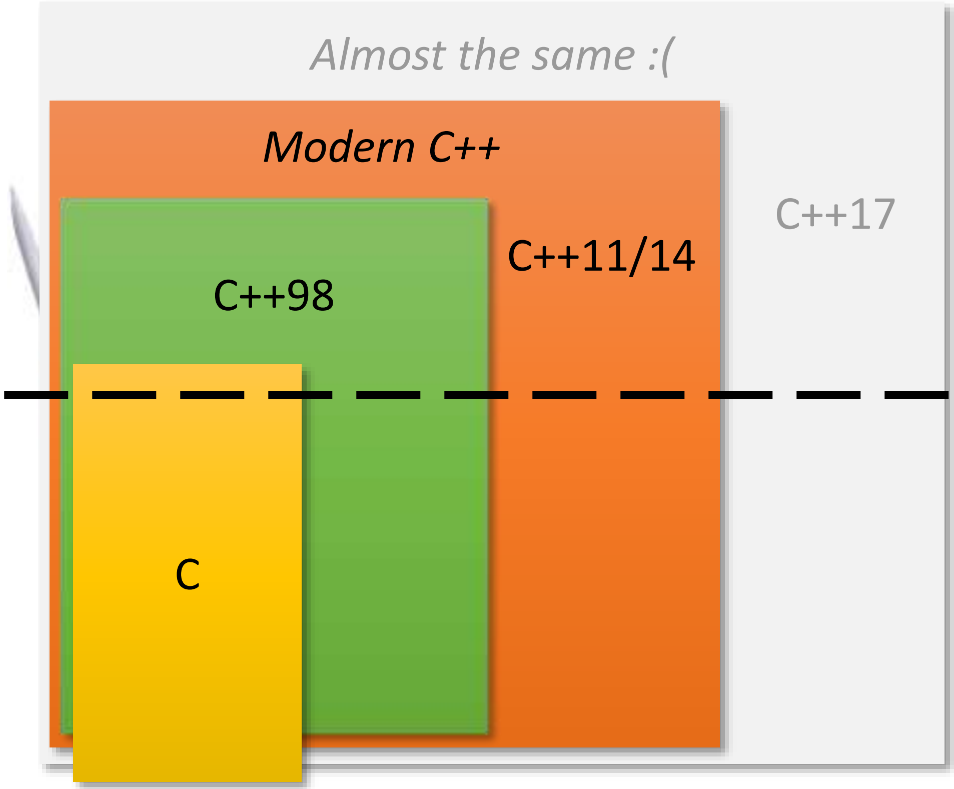
Технический менеджер, Align Technology

Москва

ALIGN
TECHNOLOGY

invisalign iTero iOC OrthoCAD





Almost the same :(

Modern C++

C++17

C++11/14

C++98

C

High level

Expert level

“Within C++ is a smaller, simpler, safer language struggling to get out”

Bjarne Stroustrup

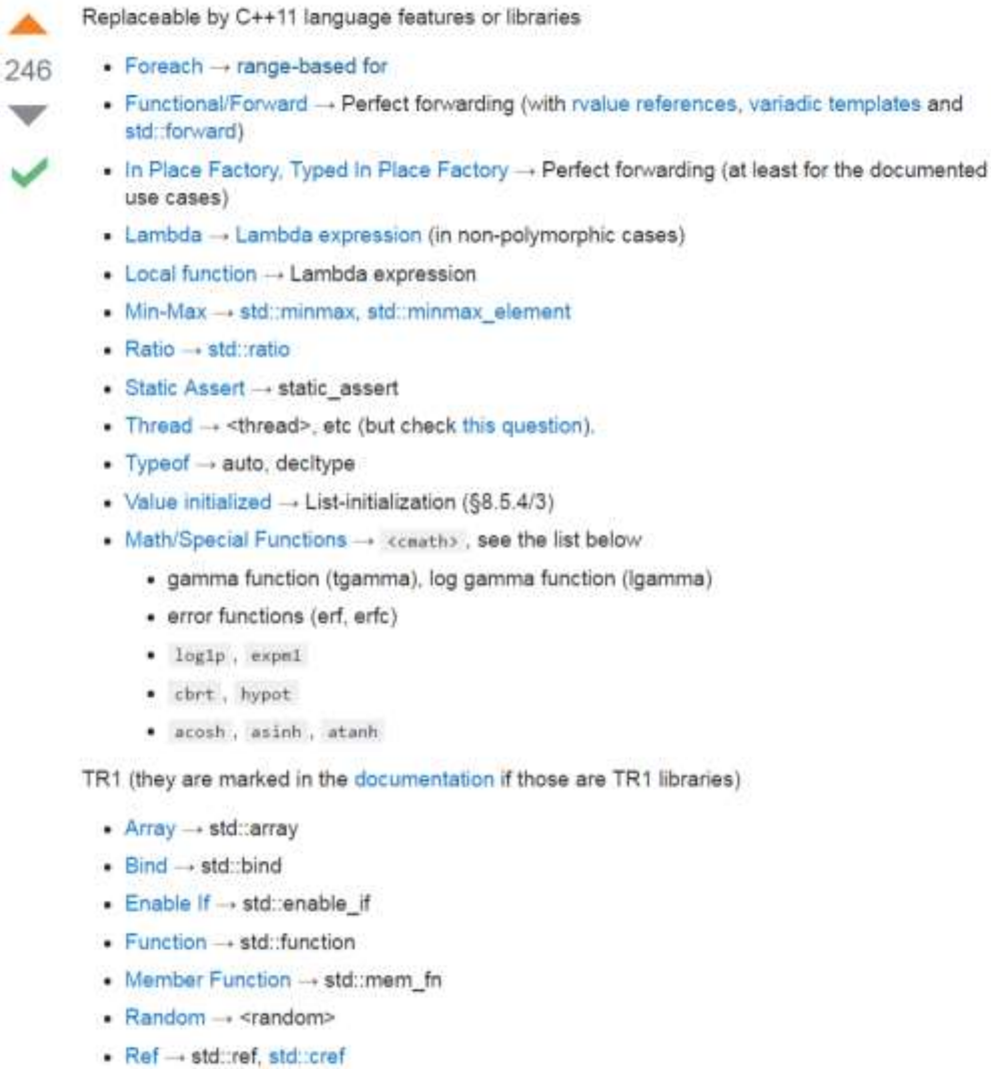
High level:

- Парадигма RAII и исключения (exceptions)
- Семантика перемещения
- λ-функции
- Классы и конструкторы
- Простые шаблоны
- STL
- Утилиты и алгоритмы boost

Expert level:

- Операторы new/delete, владеющие указатели
- Пользовательские операции копирования и перемещения
- Пользовательские деструкторы
- Закрытое, защищённое, ромбовидное, виртуальное наследование
- Шаблонная магия
- Все функции языка Си, препроцессор
- «Голые» циклы

Which boost features overlap with C++11?



Replaceable by C++11 language features or libraries

- Foreach → range-based for
- Functional/Forward → Perfect forwarding (with rvalue references, variadic templates and std::forward)
- In Place Factory, Typed In Place Factory → Perfect forwarding (at least for the documented use cases)
- Lambda → Lambda expression (in non-polymorphic cases)
- Local function → Lambda expression
- Min-Max → std::minmax, std::minmax_element
- Ratio → std::ratio
- Static Assert → static_assert
- Thread → <thread>, etc (but check this question).
- Typeof → auto, decltype
- Value initialized → List-initialization (§8.5.4/3)
- Math/Special Functions → `<cmath>`, see the list below
  - gamma function (tgamma), log gamma function (lgamma)
  - error functions (erf, erfc)
  - `log1p`, `expm1`
  - `cbrt`, `hypot`
  - `acosh`, `asinh`, `atanh`

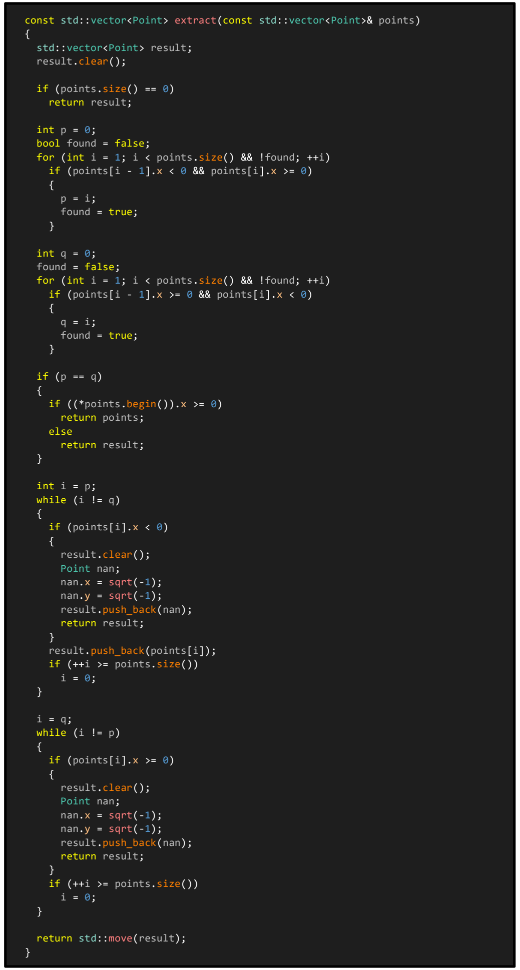
TR1 (they are marked in the documentation if those are TR1 libraries)

- Array → std::array
- Bind → std::bind
- Enable If → std::enable_if
- Function → std::function
- Member Function → std::mem_fn
- Random → <random>
- Ref → std::ref, std::cref

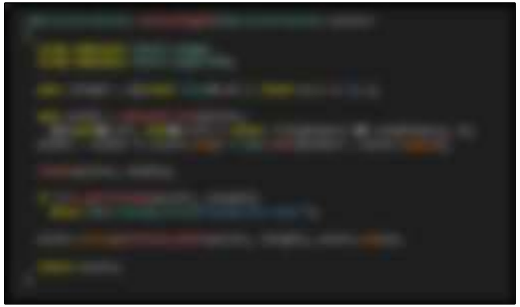
План действий

- Рассмотрим пример реального кода
- Вместе выполним рефакторинг
- Перепишем с нуля
- Обобщим решение

```cpp
const std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
  result.clear();

  if (points.size() == 0)
    return result;

  int p = 0;
  bool found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      found = true;
    }

  int q = 0;
  found = false;
  for (int i = 1; i < points.size() && !found; ++i)
```



```cpp
const std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
  result.clear();

  if (points.size() == 0)
    return result;

  int p = 0;
  bool found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      found = true;
    }

  int q = 0;
  found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x >= 0 && points[i].x < 0)
    {
      q = i;
      found = true;
    }

  if (p == q)
  {
    if ((*points.begin()).x >= 0)
      return points;
    else
      return result;
  }

  int i = p;
  while (i != q)
  {
    if (points[i].x < 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    result.push_back(points[i]);
    if (++i >= points.size())
      i = 0;
  }

  i = q;
  while (i != p)
  {
    if (points[i].x >= 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    if (++i >= points.size())
      i = 0;
  }

  return std::move(result);
}
```



```
const std::vector<Point> extract(const std::vector<Point>& points)
{
```

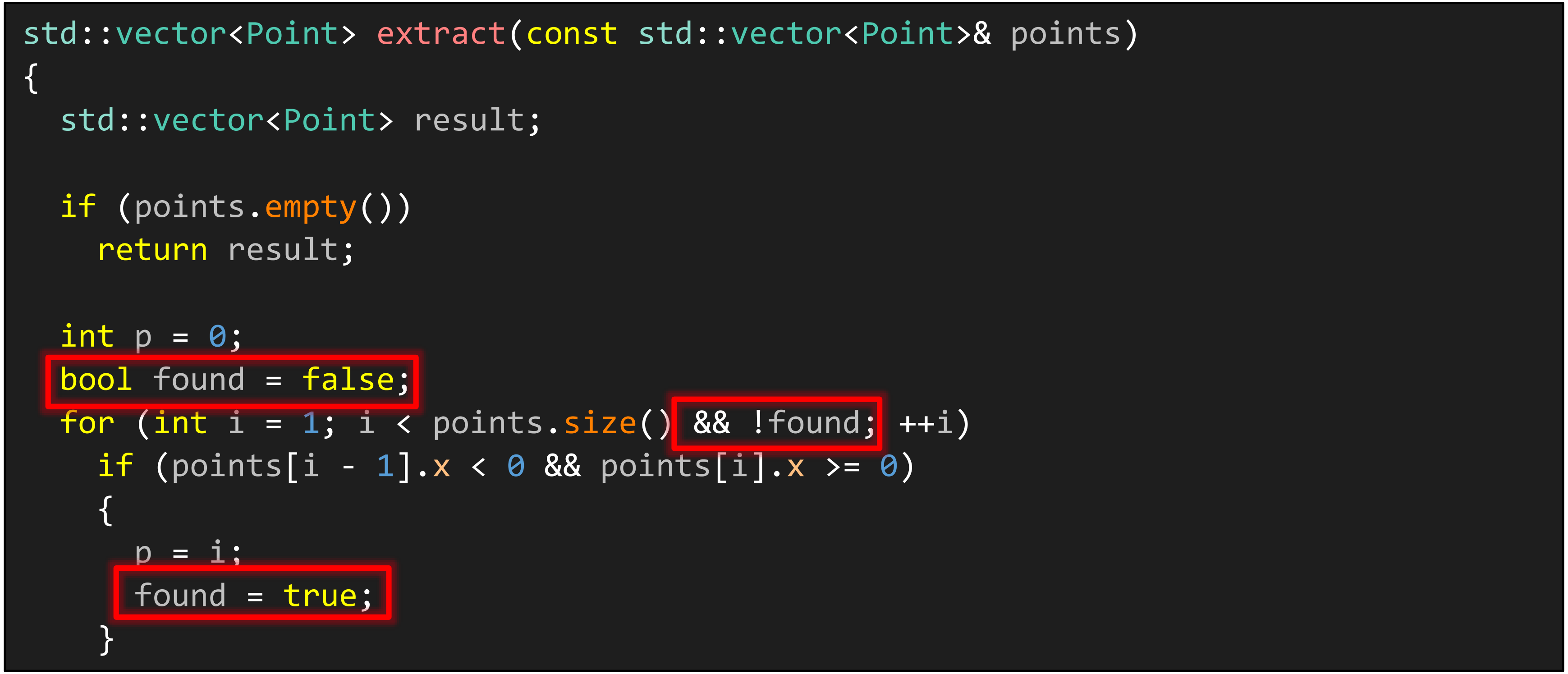
```
std::vector<Point> extract(const std::vector<Point>& points)
{
```

```
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
  result.clear();
```

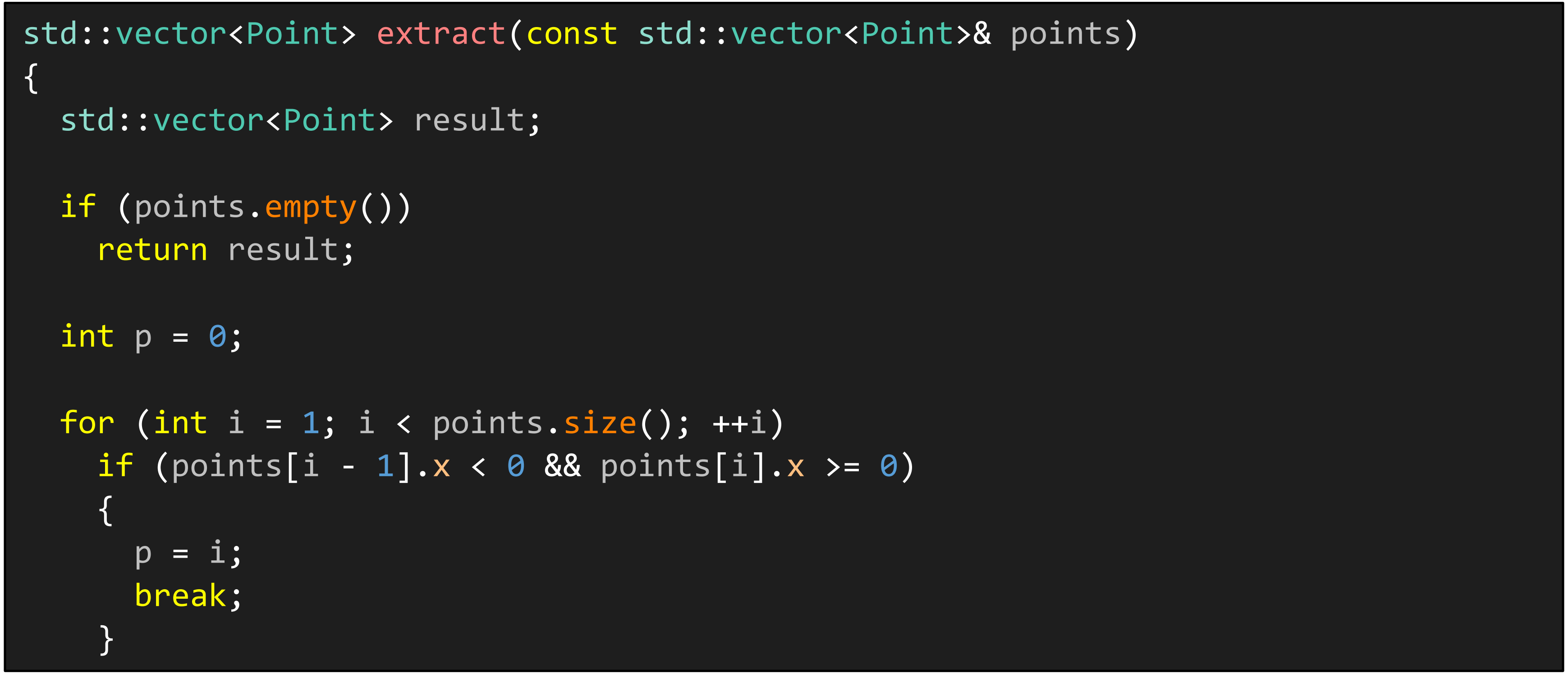
```cpp
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
```

```
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.size() == 0)
    return result;
```

```cpp
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.empty())
    return result;
```

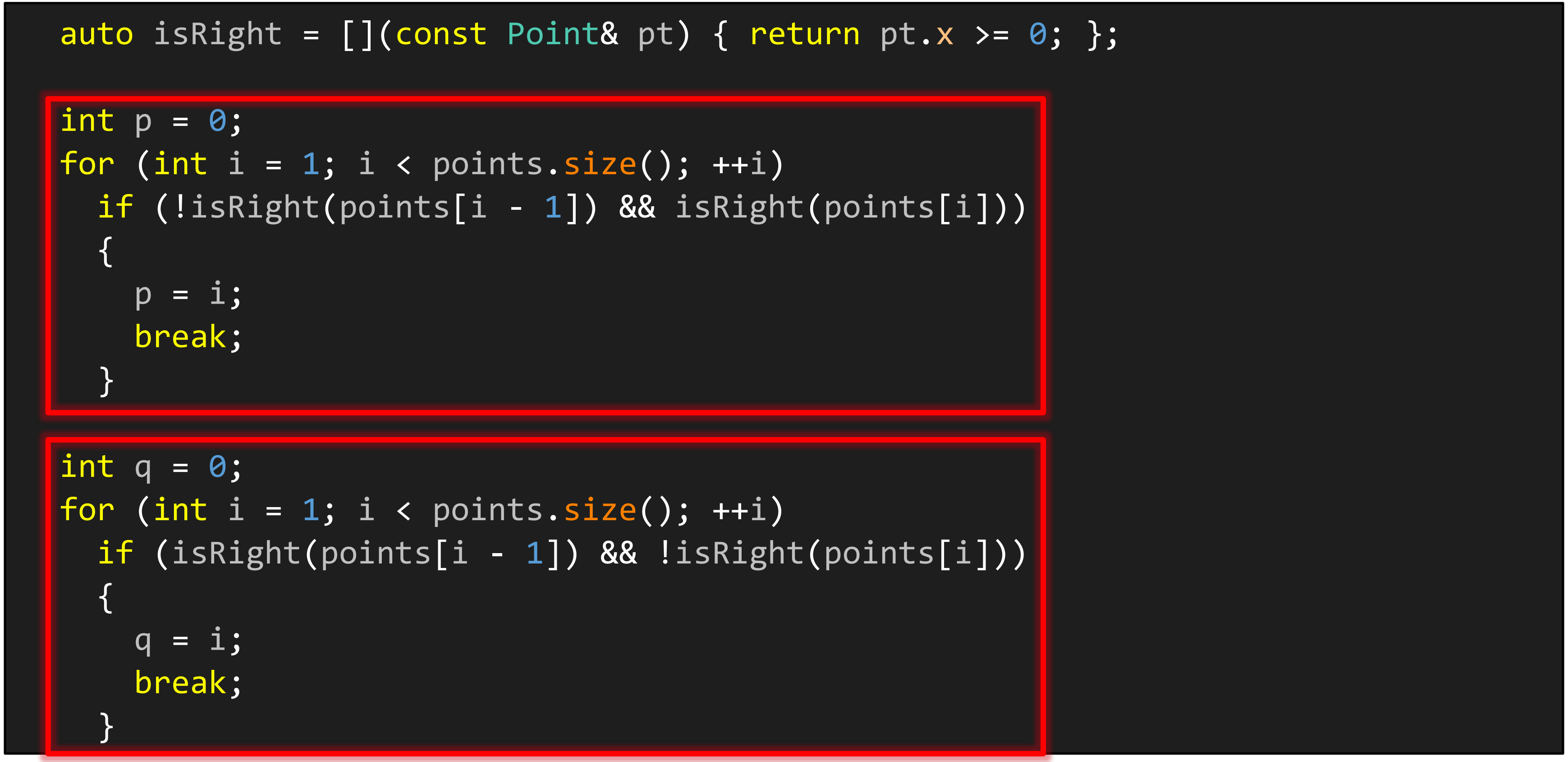


```cpp
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.empty())
    return result;

  int p = 0;
  bool found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      found = true;
    }
```



```
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.empty())
    return result;

  int p = 0;

  for (int i = 1; i < points.size(); ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      break;
    }
```

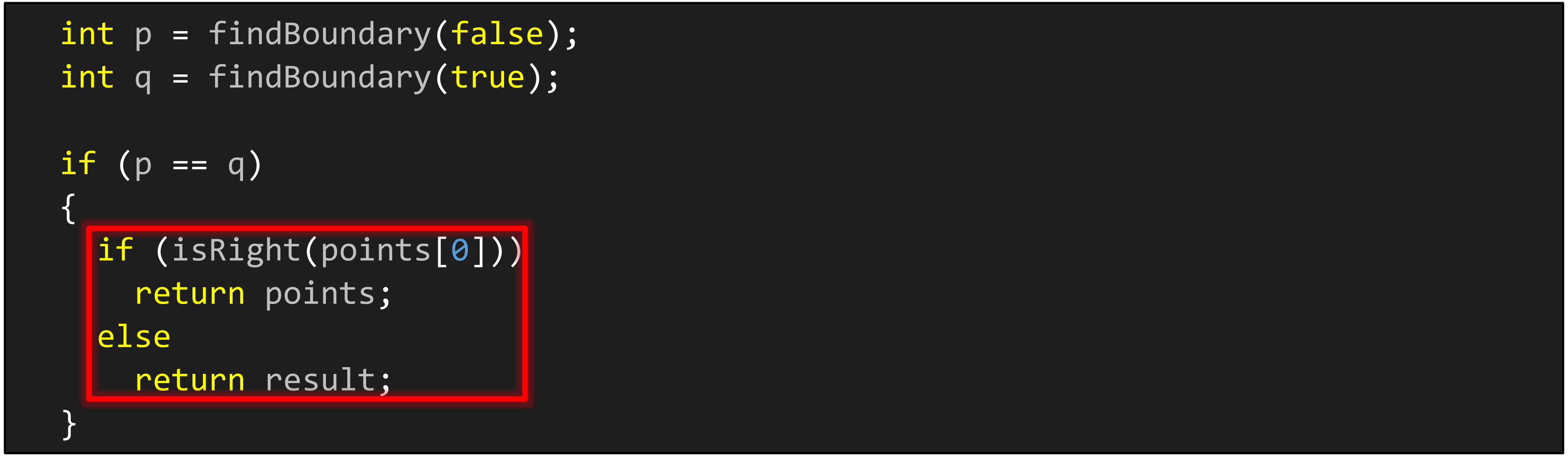
```
int p = 0;
for (int i = 1; i < points.size(); ++i)
  if (points[i - 1].x < 0 && points[i].x >= 0)
  {
    p = i;
    break;
  }

int q = 0;
for (int i = 1; i < points.size(); ++i)
  if (points[i - 1].x >= 0 && points[i].x < 0)
  {
    q = i;
    break;
  }
```



```
auto isRight = [](const Point& pt) { return pt.x >= 0; };

int p = 0;
for (int i = 1; i < points.size(); ++i)
  if (!isRight(points[i - 1]) && isRight(points[i]))
  {
    p = i;
    break;
  }

int q = 0;
for (int i = 1; i < points.size(); ++i)
  if (isRight(points[i - 1]) && !isRight(points[i]))
  {
    q = i;
    break;
  }
```

```cpp
auto isRight = [](const Point& pt) { return pt.x >= 0; };

auto find = [&](bool flag)
{
  for (int i = 1; i < points.size(); ++i)
    if (isRight(points[i - 1]) == flag &&
        isRight(points[i]) != flag)
      return i;
  return 0;
};

int p = find(false);
int q = find(true);
```

```
auto isRight = [](const Point& pt) { return pt.x >= 0; };

auto findBoundary = [&](bool rightToLeft)
{
  for (int i = 1; i < points.size(); ++i)
    if (isRight(points[i - 1]) == rightToLeft &&
        isRight(points[i]) != rightToLeft)
      return i;
  return 0;
};

int p = findBoundary(false);
int q = findBoundary(true);
int p = findBoundary(false);
int q = findBoundary(true);

if (p == q)
{
  if (isRight(*points.begin()))
    return points;
  else
    return result;
}
```



```
int p = findBoundary(false);
int q = findBoundary(true);

if (p == q)
{
  if (isRight(points[0]))
    return points;
  else
    return result;
}
```

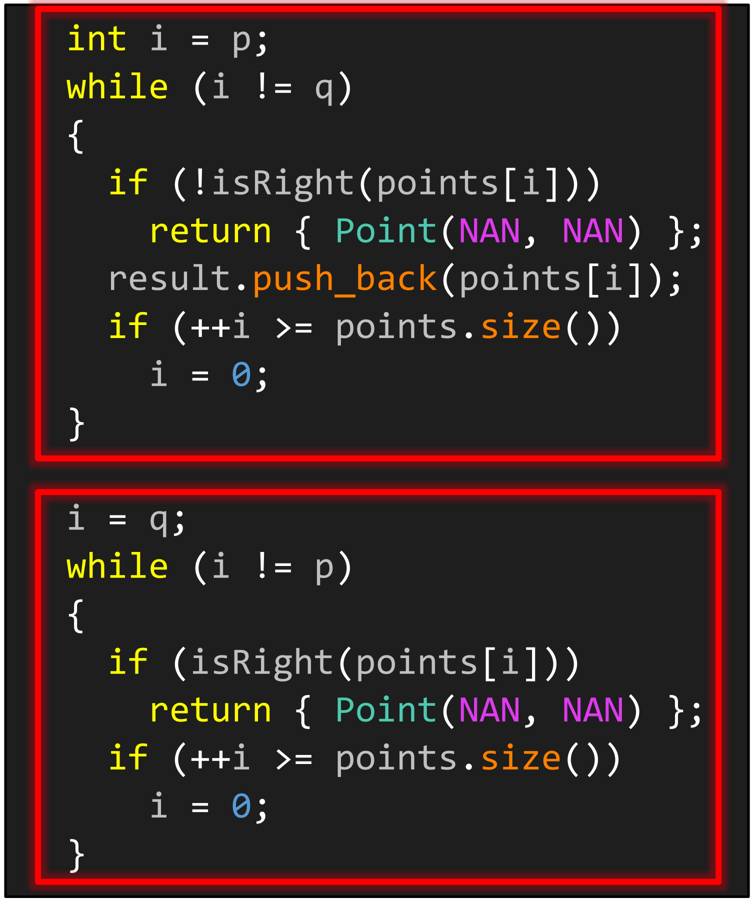
```
int p = findBoundary(false);
int q = findBoundary(true);

if (p == q)
  return isRight(points[0]) ? points : result;
```

```
if (p == q)
 return isRight(points[0]) ? points : result;

int i = p;
while (i != q)
{
 if (!isRight(points[i]))
 {
  result.clear();
  Point nan;
  nan.x = sqrt(-1);
  nan.y = sqrt(-1);
  result.push_back(nan);
  return result;
 }
 result.push_back(points[i]);
 if (++i >= points.size())
  i = 0;
}
```

```
if (p == q)
  return isRight(points[0]) ? points : result;

int i = p;
while (i != q)
{
  if (!isRight(points[i]))
    return { Point(NAN, NAN) };
  result.push_back(points[i]);
  if (++i >= points.size())
    i = 0;
}
```

std::numeric_limits::quiet_NaN() vs. std::nan() vs. NAN

```cpp
int i = p;
while (i != q)
{
  if (!isRight(points[i]))
    return { Point(NAN, NAN) };
  result.push_back(points[i]);
  if (++i >= points.size())
    i = 0;
}

i = q;
while (i != p)
{
  if (isRight(points[i]))
    return { Point(NAN, NAN) };
  if (++i >= points.size())
    i = 0;
}
```

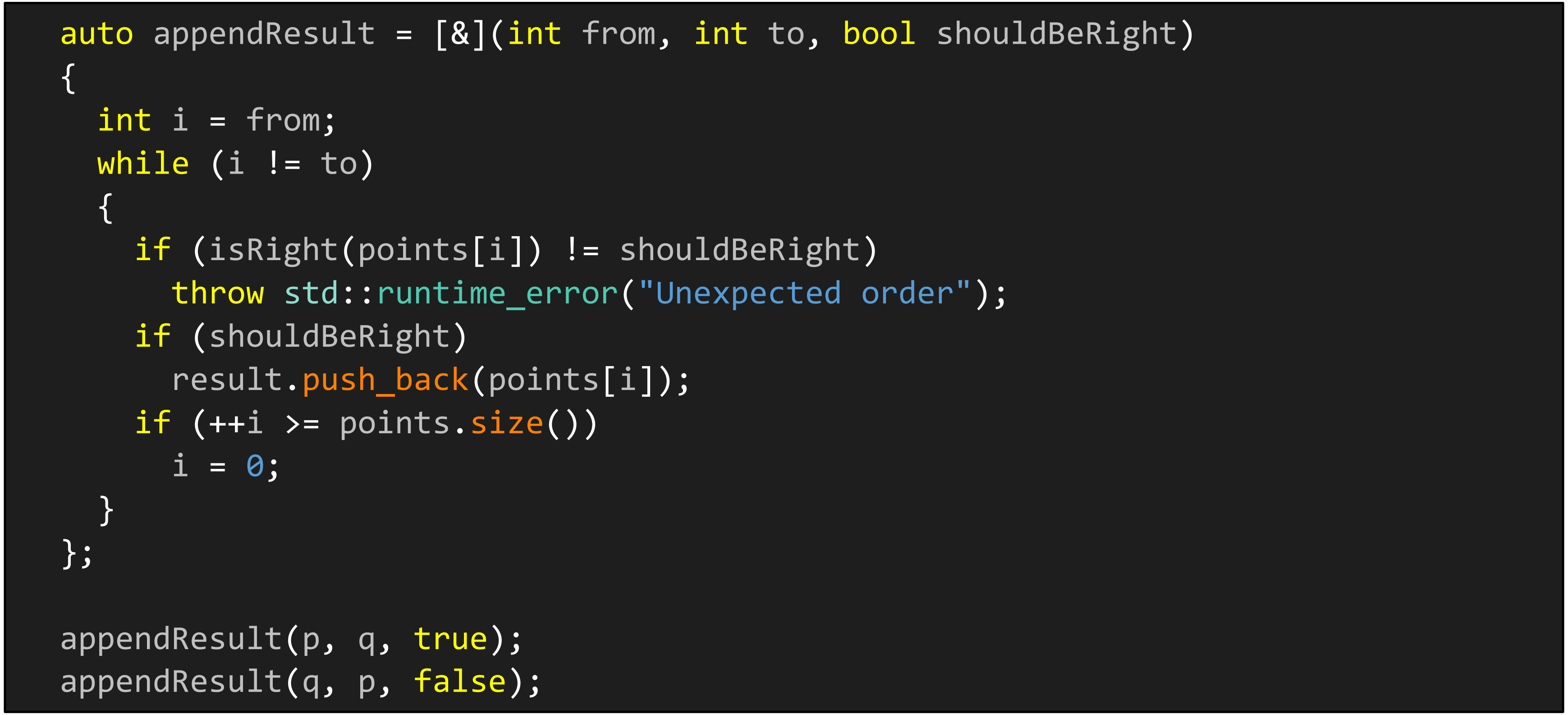


```
int i = p;
while (i != q)
{
  if (!isRight(points[i]))
    return { Point(NAN, NAN) };
  result.push_back(points[i]);
  if (++i >= points.size())
    i = 0;
}
```

```
i = q;
while (i != p)
{
  if (isRight(points[i]))
    return { Point(NAN, NAN) };
  if (++i >= points.size())
    i = 0;
}
```

Помоги Даше найти три отличия!

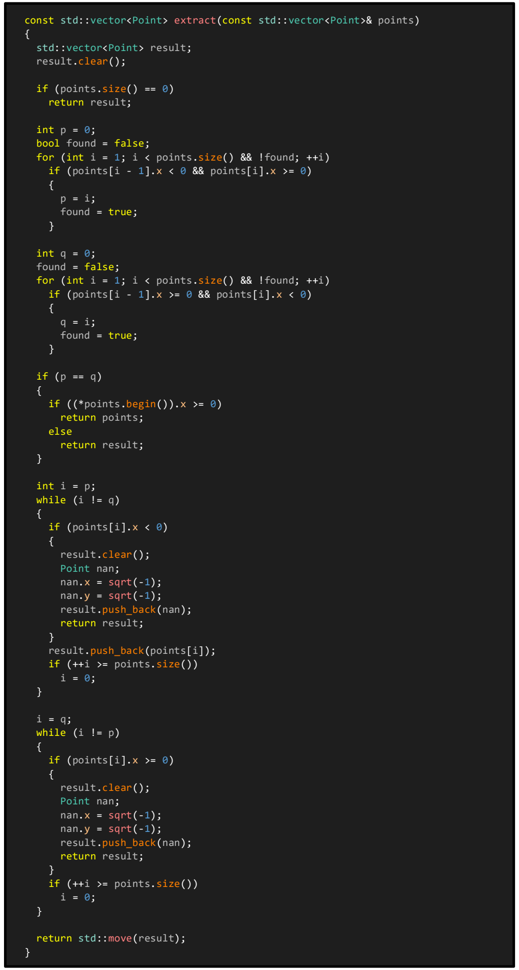
```cpp
auto appendResult = [&](int from, int to, bool shouldBeRight)
{
  int i = from;
  while (i != to)
  {
    if (isRight(points[i]) != shouldBeRight)
    {
      result = { Point(NAN, NAN) };
      return false;
    }
    if (shouldBeRight)
      result.push_back(points[i]);
    if (++i >= points.size())
      i = 0;
  }
  return true;
};

bool success = appendResult(p, q, true) && appendResult(q, p, false);
```



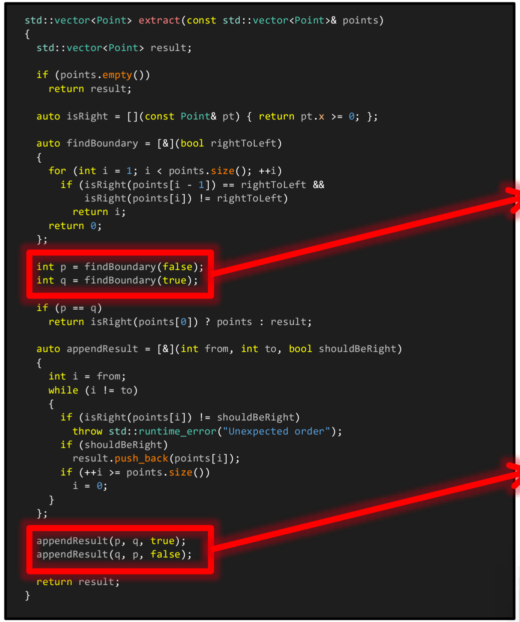
```
auto appendResult = [&](int from, int to, bool shouldBeRight)
{
  int i = from;
  while (i != to)
  {
    if (isRight(points[i]) != shouldBeRight)
      throw std::runtime_error("Unexpected order");
    if (shouldBeRight)
      result.push_back(points[i]);
    if (++i >= points.size())
      i = 0;
  }
};

appendResult(p, q, true);
appendResult(q, p, false);
```

```
  appendResult(p, q, true);
  appendResult(q, p, false);

  return std::move(result);
}
```

```
  appendResult(p, q, true);
  appendResult(q, p, false);

  return result;
}
```



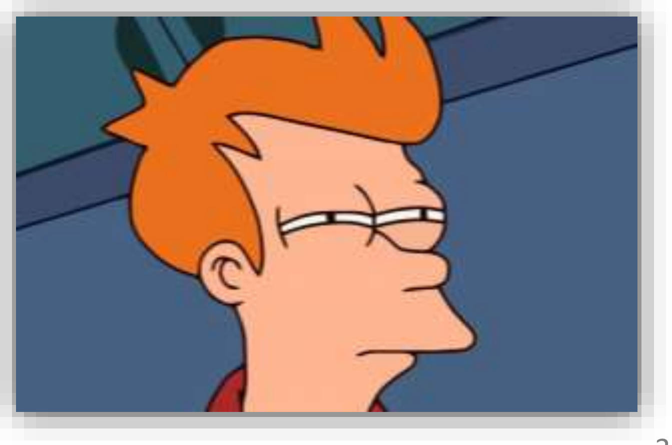
```cpp
const std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
  result.clear();

  if (points.size() == 0)
    return result;

  int p = 0;
  bool found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      found = true;
    }

  int q = 0;
  found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x >= 0 && points[i].x < 0)
    {
      q = i;
      found = true;
    }

  if (p == q)
  {
    if ((*points.begin()).x >= 0)
      return points;
    else
      return result;
  }

  int i = p;
  while (i != q)
  {
    if (points[i].x < 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    result.push_back(points[i]);
    if (++i >= points.size())
      i = 0;
  }

  i = q;
  while (i != p)
  {
    if (points[i].x >= 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    if (++i >= points.size())
      i = 0;
  }

  return std::move(result);
}
```

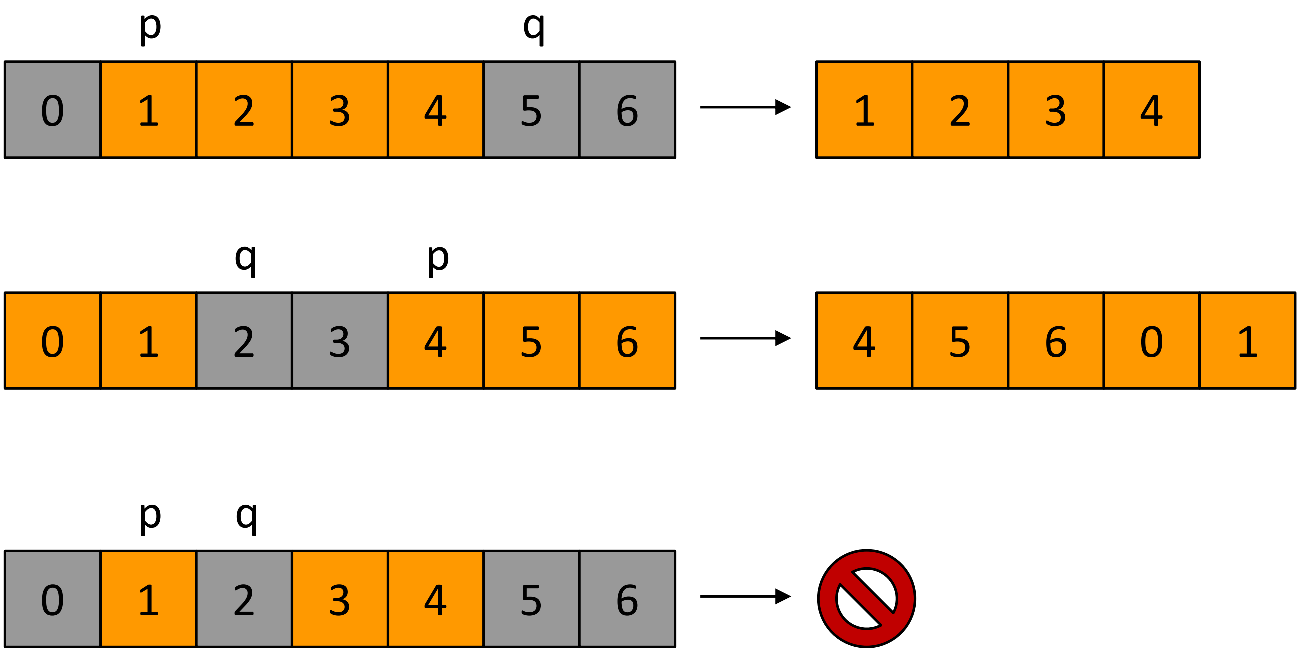


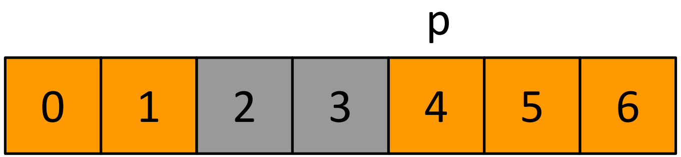
```cpp
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.empty())
    return result;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto findBoundary = [&](bool rightToLeft)
  {
    for (int i = 1; i < points.size(); ++i)
      if (isRight(points[i - 1]) == rightToLeft &&
          isRight(points[i]) != rightToLeft)
        return i;
    return 0;
  };

  int p = findBoundary(false);
  int q = findBoundary(true);

  if (p == q)
    return isRight(points[0]) ? points : result;

  auto appendResult = [&](int from, int to, bool shouldBeRight)
  {
    int i = from;
    while (i != to)
    {
      if (isRight(points[i]) != shouldBeRight)
        throw std::runtime_error("Unexpected order");
      if (shouldBeRight)
        result.push_back(points[i]);
      if (++i >= points.size())
        i = 0;
    }
  };

  appendResult(p, q, true);
  appendResult(q, p, false);

  return result;
}
```

```cpp
int p = findBoundary(false);
int q = findBoundary(true);
```

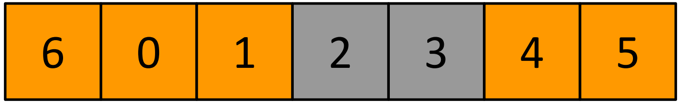
```cpp
appendResult(p, q, true);
appendResult(q, p, false);
```




p
q
0 1 2 3 4 5 6
1 2 3 4
q
p
0 1 2 3 4 5 6
4 5 6 0 1
p
q
0 1 2 3 4 5 6

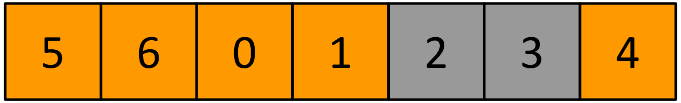


1. Найти первый элемент
2. Сдвинуть его в начало



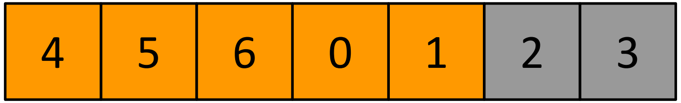
6	0	1	2	3	4	5

1. Найти первый элемент
2. Сдвинуть его в начало



5	6	0	1	2	3	4

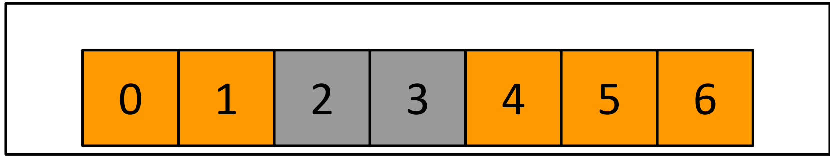
1. Найти первый элемент
2. Сдвинуть его в начало



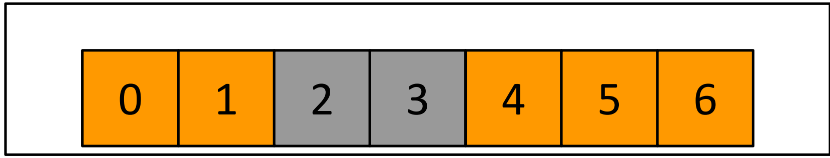
4	5	6	0	1	2	3

1. Найти первый элемент
2. Сдвинуть его в начало
3. Проверить структуру
4. Выкинуть хвост
5. Вернуть что осталось

```
std::vector<Point> extractRight(std::vector<Point> points)
{
```

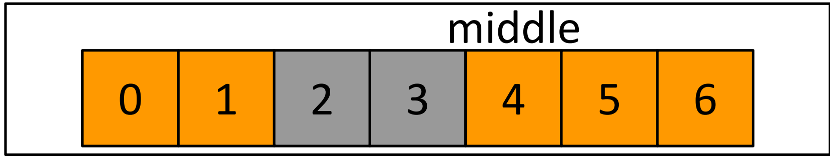


```cpp
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };
```

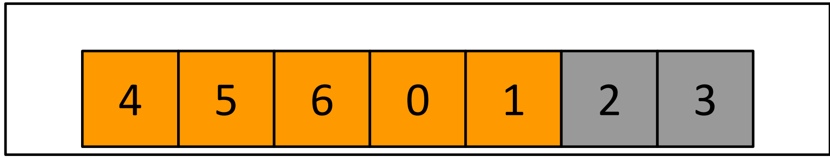


0	1	2	3	4	5	6

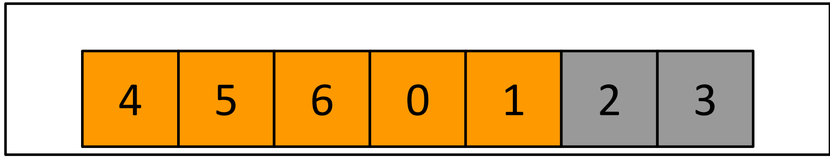
```
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });
  middle = middle != points.end() ? std::next(middle) : points.begin();
```



```
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });
  middle = middle != points.end() ? std::next(middle) : points.begin();

  rotate(points, middle);
```



```cpp
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });
  middle = middle != points.end() ? std::next(middle) : points.begin();

  rotate(points, middle);

  if (!is_partitioned(points, isRight))
    throw std::runtime_error("Unexpected order");
```

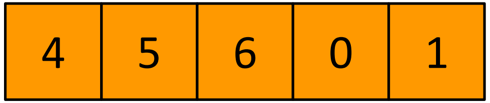


4	5	6	0	1	2	3

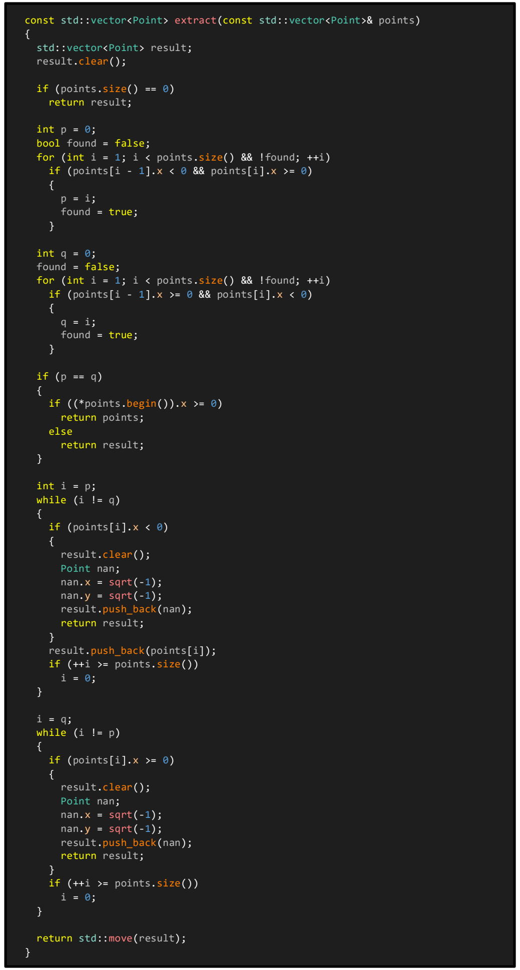
```cpp
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });
  middle = middle != points.end() ? std::next(middle) : points.begin();

  rotate(points, middle);

  if (!is_partitioned(points, isRight))
    throw std::runtime_error("Unexpected order");

  points.erase(partition_point(points, isRight), points.end());
```

4	5	6	0	1

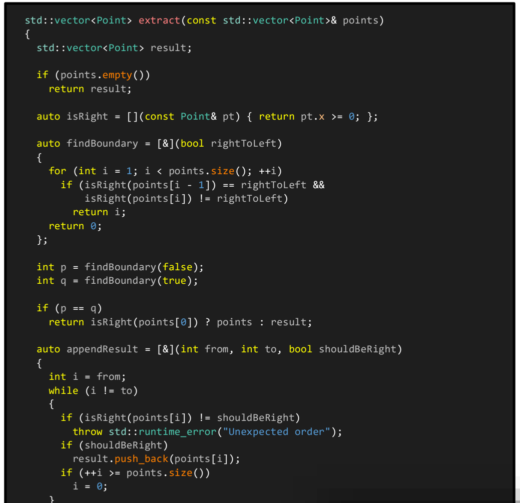
```
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });
  middle = middle != points.end() ? std::next(middle) : points.begin();

  rotate(points, middle);

  if (!is_partitioned(points, isRight))
    throw std::runtime_error("Unexpected order");

  points.erase(partition_point(points, isRight), points.end());

  return points;
}
```



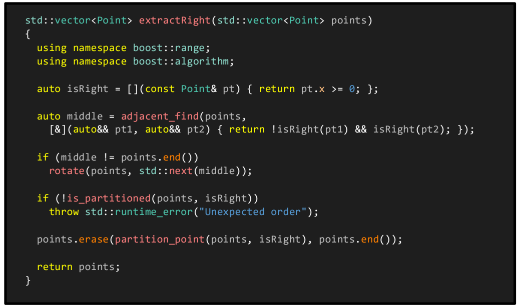
4	5	6	0	1



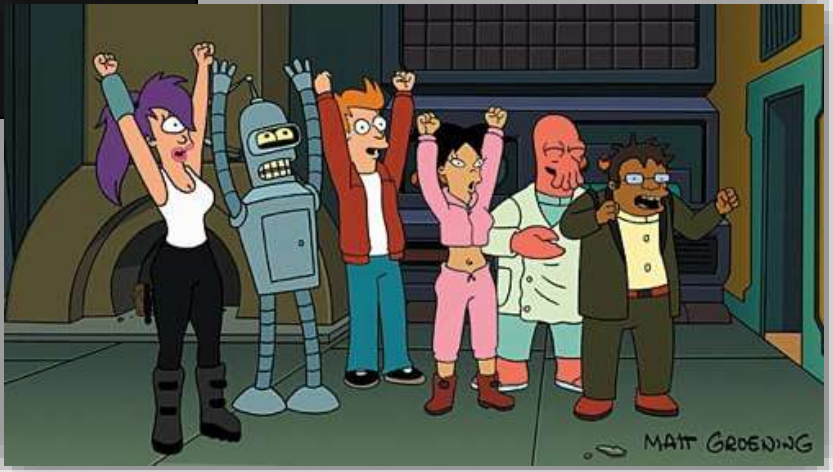
```
const std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;
  result.clear();

  if (points.size() == 0)
    return result;

  int p = 0;
  bool found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x < 0 && points[i].x >= 0)
    {
      p = i;
      found = true;
    }

  int q = 0;
  found = false;
  for (int i = 1; i < points.size() && !found; ++i)
    if (points[i - 1].x >= 0 && points[i].x < 0)
    {
      q = i;
      found = true;
    }

  if (p == q)
  {
    if ((*points.begin()).x >= 0)
      return points;
    else
      return result;
  }

  int i = p;
  while (i != q)
  {
    if (points[i].x < 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    result.push_back(points[i]);
    if (++i >= points.size())
      i = 0;
  }

  i = q;
  while (i != p)
  {
    if (points[i].x >= 0)
    {
      result.clear();
      Point nan;
      nan.x = sqrt(-1);
      nan.y = sqrt(-1);
      result.push_back(nan);
      return result;
    }
    if (++i >= points.size())
      i = 0;
  }

  return std::move(result);
}
```



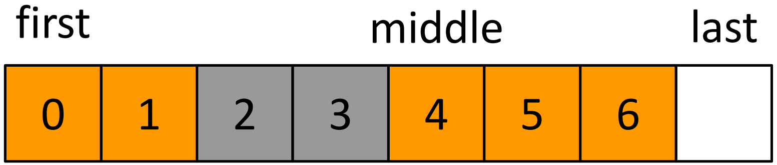
```
std::vector<Point> extract(const std::vector<Point>& points)
{
  std::vector<Point> result;

  if (points.empty())
    return result;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto findBoundary = [&](bool rightToLeft)
  {
    for (int i = 1; i < points.size(); ++i)
      if (isRight(points[i - 1]) == rightToLeft &&
          isRight(points[i]) != rightToLeft)
        return i;
    return 0;
  };

  int p = findBoundary(false);
  int q = findBoundary(true);

  if (p == q)
    return isRight(points[0]) ? points : result;

  auto appendResult = [&](int from, int to, bool shouldBeRight)
  {
    int i = from;
    while (i != to)
    {
      if (isRight(points[i]) != shouldBeRight)
        throw std::runtime_error("Unexpected order");
      if (shouldBeRight)
        result.push_back(points[i]);
      if (++i >= points.size())
        i = 0;
    }
  };

  appendResult(p, q, true);
  appendResult(q, p, false);

  return result;
}
```



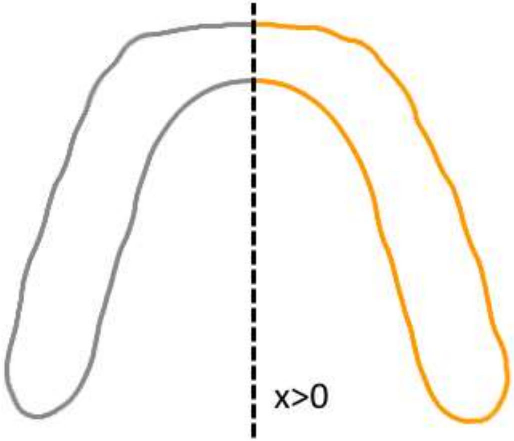
```
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) { return !isRight(pt1) && isRight(pt2); });

  if (middle != points.end())
    rotate(points, std::next(middle));

  if (!is_partitioned(points, isRight))
    throw std::runtime_error("Unexpected order");

  points.erase(partition_point(points, isRight), points.end());

  return points;
}
```



```cpp
// Two specialized overloads
std::vector<Point> extractRight(std::vector<Point>&& points)
{
  // Process points in-place...
  return points;
}
std::vector<Point> extractRight(const std::vector<Point>& points)
{
  std::vector<Point> result = points;  // Might be altered!
  // Process result...
  return result;
}

// One universal function
std::vector<Point> extractRight(std::vector<Point> points)
{
  // Process points in-place...
  return points;
}
```


first
middle
last
0
1
2
3
4
5
6

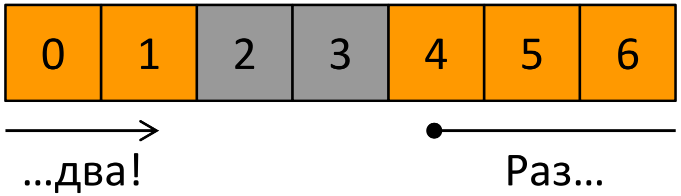
Sean Parent: C++ Seasoning (no raw loops)



x>0

Обобщаем!

- std::vector → последовательность (пара итераторов)
- isRight() → предикат
- Результат как копия → результат как подпоследовательность

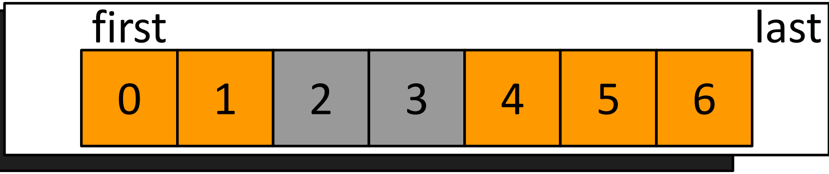


0	1	2	3	4	5	6

…два!

Раз…

```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
```

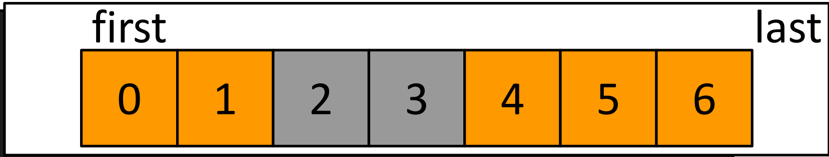


first

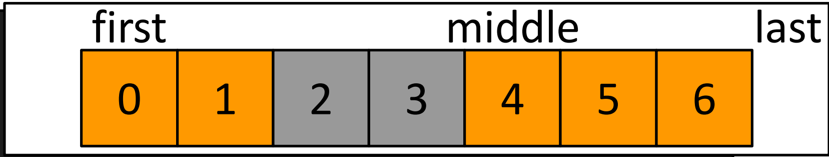
last

0 1 2 3 4 5 6

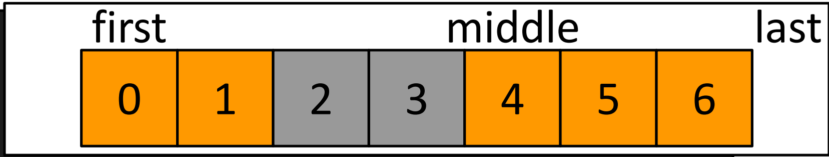
```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;
```

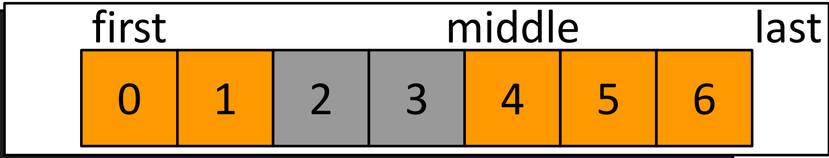


```cpp
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) { return !p(a) && p(b); });
  middle = middle != last ? std::next(middle) : first;
```

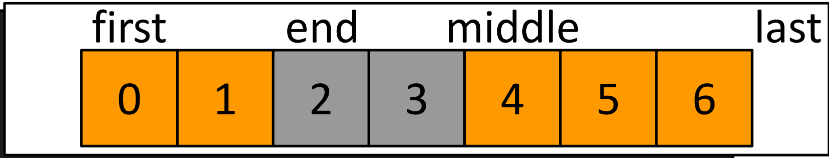


```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) { return !p(a) && p(b); });
  middle = middle != last ? std::next(middle) : first;

  auto rotated = boost::join(boost::make_iterator_range(middle, last),
                             boost::make_iterator_range(first, middle));
```



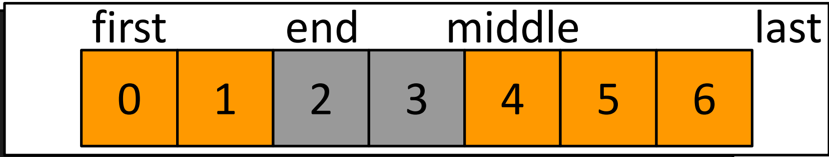


```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) { return !p(a) && p(b); });
  middle = middle != last ? std::next(middle) : first;

  auto rotated = boost::join(boost::make_iterator_range(middle, last),
                             boost::make_iterator_range(first, middle));

  if (!is_partitioned(rotated, p))
    throw std::runtime_error("Unexpected order");
```



```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) { return !p(a) && p(b); });
  middle = middle != last ? std::next(middle) : first;

  auto rotated = boost::join(boost::make_iterator_range(middle, last),
                             boost::make_iterator_range(first, middle));

  if (!is_partitioned(rotated, p))
    throw std::runtime_error("Unexpected order");

  auto end = partition_point(rotated, p);
```

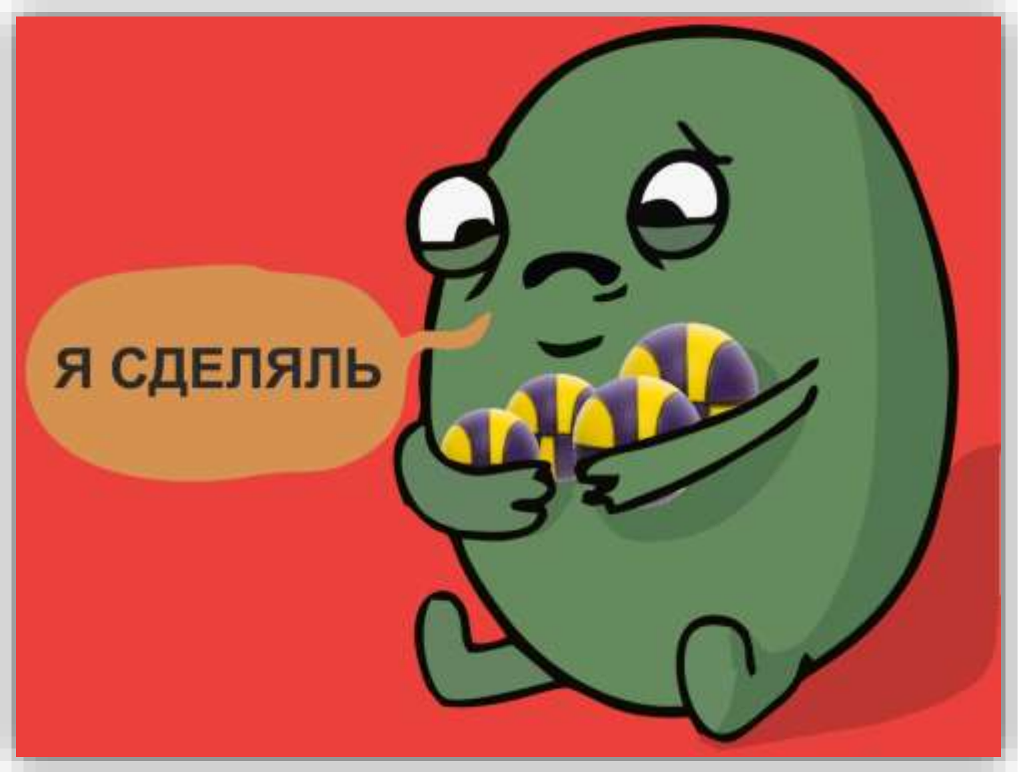
```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) { return !p(a) && p(b); });
  middle = middle != last ? std::next(middle) : first;

  auto rotated = boost::join(boost::make_iterator_range(middle, last),
                             boost::make_iterator_range(first, middle));

  if (!is_partitioned(rotated, p))
    throw std::runtime_error("Unexpected order");

  auto end = partition_point(rotated, p);

  return boost::make_iterator_range(rotated.begin(), end);
}
```



```
std::vector<Point> extractRight(std::vector<Point> points)
{
  using namespace boost::range;
  using namespace boost::algorithm;

  auto isRight = [](const Point& pt) { return pt.x >= 0; };

  auto middle = adjacent_find(points,
    [&](auto&& pt1, auto&& pt2) {
      return !isRight(pt1) && isRight(pt2);
    });
  middle = middle != points.end() ? std::next(middle)
                                  : points.begin();

  rotate(points, middle);



  if (!is_partitioned(points, isRight))
    throw std::runtime_error("Unexpected order");

  points.erase(
    partition_point(points, isRight), points.end());

  return points;
}
```

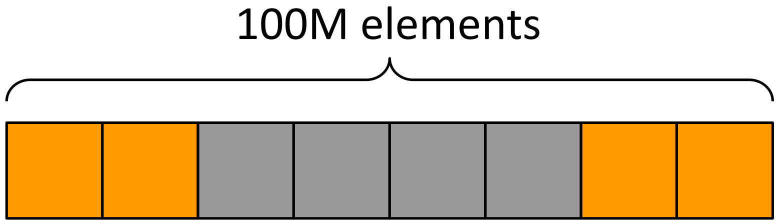
```
template<class It, class Predicate>
auto extractIf(It first, It last, Predicate p)
{
  using namespace boost::range;
  using namespace boost::algorithm;



  auto middle = adjacent_find(first, last,
    [&](auto&& a, auto&& b) {
      return !p(a) && p(b);
    });
  middle = middle != last ? std::next(middle)
                          : first;

  auto rotated = boost::join(
                   boost::make_iterator_range(middle, last),
                   boost::make_iterator_range(first, middle));

  if (!is_partitioned(rotated, p))
    throw std::runtime_error("Unexpected order");


  auto end = partition_point(rotated, p);

  return boost::make_iterator_range(rotated.begin(), end);
}
```

- Ответь на вопрос
- Получи мячик
- …
- PROFIT!



Performance?

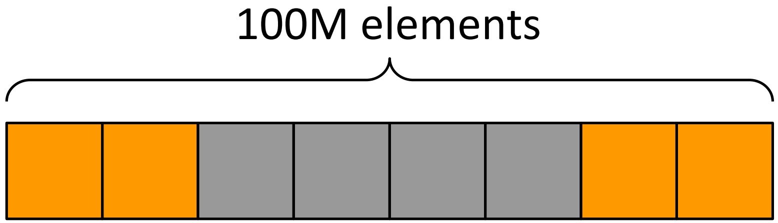
```
double sum = 0;
for (const Point& v : range)
  sum += v.x;
```



Range	Traverse time
std::vector	1.0
Joined range of two iterator ranges	

Performance?

```
double sum = 0;
for (const Point& v : range)
  sum += v.x;
```



Range	Traverse time
std::vector	1.0
Joined range of two iterator ranges	1.18

Спасибо за внимание!

- Мыслите в терминах алгоритмов
- Код должен ясно выражать намерение
- Знайте свои инструменты и используйте их к месту

```
// lambda functions (including generic)
// ternary operator
// exceptions
// transient parameters
std::next();
adjacent_find()
rotate()
is_partitioned()
partition_point()
```

```
// template parameters for iterators
// template parameters for predicates
// function return type deduction
boost::range
boost::algorithm
std::vector<T>::empty()
boost::make_iterator_range()
boost::join()
```

SECON'2017

IX МЕЖРЕГИОНАЛЬНАЯ КОНФЕРЕНЦИЯ РАЗРАБОТЧИКОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Матросов Михаил

Технический менеджер, Align Technology

mmatrosov@aligntech.com

8 926 381-61-64